DESENVOLVIMENTO DE UM CONTROLADOR DE *BOUNDARY SCAN* (IEEE 1149.1)

André Carvalho de Azevedo Silva Couto

Departamento de Engenharia Electrotécnica
Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

2014

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Disciplina de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: André Carvalho de Azevedo Silva Couto, Nº 1070187, 1070187@isep.ipp.pt
Orientação científica: André Vaz da Silva Fidalgo, anf@isep.ipp.pt

Departamento de Engenharia Electrotécnica
Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

Agradecimentos

Pelo tempo que este trabalho me tomou, devo agradecer a compreensão da minha ausência aos meus amigos e familiares, e seguros estejam que porquanto o trabalho me possa por vezes ter deixado afastado, de nenhuma forma me abala o apreço que lhes mantenho.

À Filipa Silva, não posso deixar de agradecer o especial suporte durante este período, que por certo teve uma maior influência na elaboração deste trabalho do que se poderá supor.

Ao Eng. André Fidalgo, estou grato de todo o acompanhamento a que se disponibilizou no desenvolvimento deste trabalho e por me ter garantido condições para trabalhar neste tema que muito me agradou.

Estimo todos os recursos e qualidade humana que vim experimentando e conhecendo ao longo do meu percurso académico nesta instituição, que, para além de garantir o meu ensino técnico, tanto contribuiu para a minha evolução pessoal. Por tudo, fico grato e certo da minha correcta escolha. Todas as amizades que levo não poderão senão manter-me ciente disso, e do quão agradável foi trabalhar e conviver, neste período, com todos vocês.

Resumo

O *Boundary Scan* consiste numa infra-estrutura de teste por varrimento periférico, com o objectivo de garantir a controlabilidade e observabilidade dos principais *inputs* e *outputs* dos vários circuitos integrados que compõem uma placa de circuito impresso. Esta metodologia está, desde *1990*, normalizada pelo *Institute of Electrical and Electronics Engineers* como o *IEEE 1149.1*, que permite a implementação facilitada de sistemas de teste integrados em placas de circuitos digitais. O presente trabalho incide no estudo da tecnologia, abordando-se a sua motivação, importante presença na indústria para a aplicação de testes estruturais, descrição técnica detalhada, e metodologias para o seu controlo; e o desenvolvimento de um controlador compatível capaz de actuar as infra-estruturas de *Boundary Scan*.

O controlador de *Boundary Scan*, aliado ao *software* desenvolvido, deverá possibilitar o teste de placas que suportem a tecnologia, garantindo a sua observação e controlo, estando adequado a uma utilização numa vertente didáctica. Outras funcionalidades com o intuito de melhor explorarem as capacidades do teste ou controlo por *Boundary Scan*, ou melhorar a sua utilidade como ferramenta no ensino de electrónica, foram também consideradas. A adaptação do sistema ao presente contexto tecnológico foi garantida pela análise de produtos comerciais e de distribuição livres, actualmente disponíveis.

O sistema desenvolvido é composto por três componentes: O controlador de *Boundary Scan* e o seu *firmware*; a biblioteca dinâmica para o seu controlo; e a aplicação gráfica que serve de interface com o utilizador. Esta modularidade do sistema permite que o controlador e biblioteca de controlo possam ser facilmente integrados em outros projectos, com outro *software* desenvolvido para o seu controlo. A aplicação gráfica criada disponibiliza um conjunto de ferramentas genéricas que tiram partido da tecnologia de *Boundary Scan*, destacando-se: Controlo e monitorização dos pinos físicos dos componentes alvo; Pesquisa e identificação automática dos componentes do circuito alvo; Programação de microcontroladores *Atmel ATmega*; Execução de código de descrição de testes segundo os *standards Serial Vector Format* (SVF) e *Xilinx Serial Vector Format* (XSVF); Aplicação automática ou manual de instruções de teste; entre outras.

O suporte à linguagem *Serial Vector Format* permite que o controlador seja eficientemente utilizado para o teste de circuitos, ou em parceria com um grande número de *softwares* de desenvolvimento, na execução de tarefas por *Boundary Scan* (utilizando os ficheiros *Serial Vector Format* que estes criam).

O protótipo funcional desenvolvido foi testado com recurso a uma placa de teste desenhada para o efeito, composta por um conjunto de componentes que dispõem de infra-estrutura de testes em conformidade com a norma *IEEE 1149.1*, que serve de alvo para o controlador. O teste ao sistema permitiu validar o correcto funcionamento das funcionalidades implementadas e optimizar o seu desempenho.

Considera-se que o controlador proposto respondeu às expectativas tanto no que respeita à sua funcionalidade, que excedeu os objectivos inicialmente traçados, de controlo e observação das placas alvo; como relativamente ao seu desempenho, que, pelo menos, enquadrado numa utilização didáctica, se entende muito satisfatório.

Palavras-Chave:

Boundary Scan; *JTAG*; *IEEE 1149.1*; *Serial Vector Format*; *Teste e Depuração*.

Abstract

The *Boundary Scan* is a test architecture with the aim of granting access to the input and output pins of the various components of an electric digital circuit, allowing for its observability and controllability. This technology was standardized, in the year *1990*, by the *Institute of Electrical and Electronics Engineers*, as *IEEE 1149.1*, which describes the implementation of the *boundary scan* as an integrated test system used for testing and debugging printed circuit boards. The present work focus on the study the *IEEE 1149.1* standard; understanding its motivation and important role in the industry in the structural testing of electronic products; describing a detailed technical overview including its control methods; and the development of a compatible controller for testing and debugging of circuit boards through the *boundary scan* interface.

The designed *Boundary Scan* controller, with its control software, should be able to test circuits with *Boundary Scan* capabilities, assuring its controllability and observability, while being suitable for use in a didactic environment. Other functionalities in order to better explore the *Boundary Scan* technology, or enhance its use as a tool for electronic teaching, were also considered. To better suit the developed controller to the current technology context, other commercially or freely available controllers were researched and taken in consideration.

The designed system has three main components: The *boundary scan* controller and its *firmware*, which links the computer to the target circuit board; a dynamic library for easier control; and the computer application that serves as the graphic user interface. This system modularity allows for an easy integration of the controller and library in other projects. The graphic application, as it was developed, has a number of generic applications that take advantage of *Boundary Scan* technology: Control and monitoring of the physical inputs/outputs pins of the target components; Automatic search and identification of the individual components of the target circuitry; Programming of *Atmel Atmega* microcontrollers; Execution of test descriptive languages according to *Serial Vector Format* (SVF) and *Xilinx Serial Vector Format* (XSVF) specification; Manual or automatic application of test instructions; among others.

The *Serial Vector Format* support enables the controller to be efficiently used for circuit testing, or to work in conjunction with a great number of development software, executing *boundary scan* operations by interpreting their *Serial Vector Format* files.

The assembled functional prototype of the controller was tested with a test board created for that purpose, which integrate a number of integrated circuit components compatible with the *IEEE 1149.1*, to be used as the target during the system testing. The conducted tests validated the correct functioning of the implemented functionalities and allowed to optimize its performance.

Considering the initial objectives of the project, the proposed controller responded well to both expectations regarding to its functionality, which far surpassed the initially proposed set, for controlling and observing the target boards; and with respect to its performance, that, at least as far as a didactic use is concerned, is considered very satisfactory.

Keywords:

Boundary Scan; *JTAG; IEEE 1149.1*; *Serial Vector Format*; *Debugging*.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ABSDL	*Analog Boundary-Scan Description Language*
ANSI	*American National Standards Institute*
AOI	*Automated Optical Inspection*
ASIC	*Application Specific Integrated Circuit*
ATAP	*Analog Test Access Port*
ATE	*Automated Test Equipment*
AXI	*Automated X-Ray Inspection*
BDM	*Background Debug Mode*
BGA	*Ball Grid Array*
BIST	*Bult-In Self Test*
BS	*Boundary Scan*
BSC	*Boundary Scan Cell*
BSDL	*Boundary Scan Description Language*
BSR	*Boundary-Scan Register*
BST	*Boundary Scan Test*
CLR	*Common Language Runtime*
COB	*Chip-on-board*
COP	*Common On-Chip Processor*
CPLD	*Complex Programmable Logic Device*
CSP	*Chip Scale Package*
DAC	*Digital-to-Analog Converter*
DFT	*Design for test*
DIR	*Device Identification Register*
ECL	*Emitter-coupled logic*
ECID	*Electronic Chip Identification*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
eTBC	*Embedded Test-Bus Controllers*
EXTEST	*External Test Instruction*
FPGA	*Field Programmable Gate Array*
FPT	*Flying Probe Test*

FST	*Functional Self Test*
FT	*Functional Test*
GCC	*GNU Compiler Collection*
GND	*Ground*
GPIO	*General-purpose input/output*
GPL	*General Public License*
HAL	*Hardware Abstraction Layer*
HDR	*Header Data Register* (SVF)
HIR	*Header Instruction Register* (SVF)
HSDL	*Hierarchical Scan Description Language*
HSIO	*High Speed I/O*
I/O	*Input/Output*
I2C	*Inter-Integrated Circuit*
IBIST	*Interconnect Built-In Self Test*
IC	*Integrated Circuit*
ICE	*In-Circuit Emulation*
ICT	*In-Circuit Testing*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
IJTAG	*IEEE P1687*
INTEST	*Internal Test Instruction*
IR	*Instruction Register*
ISC	*Internet Systems Consortium*
ISP	*In-System Programming*
ISR	*Interrupt Service Routine*
JEDEC	*Joint Electron Device Engineering Council*
JTAG	*Joint Test Action Group*; refere o *IEEE 1149.1*
LED	*Light-Emitting Diode*
LQFP	*Low Profile Quad Flat Package*
LSB	*Least significant bit*
LVDS	*Low-voltage differential signaling*
MCM	*Multi-chip module*
MCU	*Microcontroller Units*
MDA	*Manufacturing Defect Analyser*
MPSSE	*Multi-Protocol Synchronous Serial Engine*

MSB	*Most significant bit*
MS-DOS	*Microsoft Disk Operating System*
OCD	*On-Chip Debug*
PCB	*Printed Circuit Board*
PDIP	*Parallel dual in-line package*
PID	*Product Identification* (USB)
PLD	*Programmable Logic Device*
POSIX	*Portable Operating System Interface*
PTH	*Plated-Through Holes*
QFP	*Quad Flat Package*
RAM	*Random Access Memory*
RUNBIST	*Run Built-in Self-Test Instruction*
SDR	*Scan Data Register* (SVF)
SIR	*Scan Instruction Register* (SVF)
SMD	*Surface Mounted Devices*
SO	*Sistema Operativo*
SOC	*Systems-on-a-Chip*
SOIC	*Small Outline Integrated Circuit*
SPI	*Serial Peripheral Interface*
SRAM	*Static Random-Access Memory*
SRST	*System Reset*
STAPL	*Standard Test And Programming Language*
SVF	*Serial Vector Format*
SWD	*Serial Wire Debug*
TAB	*Tape-Automated Bonding*
TAP	*Test Access Port*
TCK	*Test Clock Input* (TAP)
Tcl	*Tool Command Language*
TDI	*Test Data Input* (TAP)
TDO	*Test Data Output* (TAP)
TDR	*Test Data Register*
TDR	*Trailer Data Register* (SVF)
THT	*Through-Hole Technology*
TIR	*Trailer Instruction Register* (SVF)
TMP	*Test Mode Persistance Controller*

TMS	*Test Mode Select* (TAP)
TRST	*Test Reset* (TAP)
TTL	*Transistor–Transistor Logic*
UART	*Universal Asynchronous Receiver/Transmitter*
UBRR	*USART Baud Rate Register* (MCU *Atmel ATmega*)
USB	*Universal Serial Bus*
VCC	*Positive power-supply voltage pin*
VCP	*Virtual COM Port*
VHDL	*VHSIC Hardware Descriptive Language*
VHSIC	*Very High Speed Integrated Circuit*
VID	*Vendor Identification* (USB)
VLSI	*Very-large-scale Integration*
WPF	*Windows Presentation Foundation*
XAML	*eXtensible Application Markup Language*
XSVF	*Xilinx Serial Vector Format*

1. INTRODUÇÃO

A testabilidade de um qualquer produto tem dois aspectos fundamentais: a sua controlabilidade e observabilidade. O teste de um sistema é geralmente conduzido colocando-o num estado pré-determinado, e fornecendo às suas entradas um conjunto de dados conhecidos (dados de teste), observando-se a sua resposta. Esta análise deverá permitir reconhecer se ele se comporta da forma prevista, tal como havia sido desenhado e produzido. Se o controlo e observação não puderem ser garantidos, não há forma empírica de se assegurar que o sistema funciona tal como previsto. Os procedimentos de testes são recorrentes durante todo o fluxo normal de produção de um produto, sendo um processo de demarcada significância[1].

O projecto para teste (*Design for test*, DFT) é um processo que incorpora regras e técnicas no projecto de um produto, vocacionadas para lhe garantir alguma testabilidade[1]. Este conceito não é recente, tendo sido utilizado no desenho de *hardware* desde há mais de cinquenta anos. A razão é simples: A pretender-se testar um circuito integrado durante as fases de criação e, mais tarde, na sua produção, este deve ser desenhado de forma a poder ser testado. A testabilidade deve ser criada com o produto, não pode ser simplesmente adicionada mais tarde. O DFT é um requerimento não funcional crítico, que afecta a maioria das áreas do desenho do circuito eléctrico dos produtos[2]. Este tem repercussão em todas as fases da vida do produto, desde o desenho do circuito eléctrico até ao serviço pós venda[1].

Se a testabilidade for considerada logo ao nível da lógica dos circuitos integrados (IC) utilizados nos produtos, os seus benefícios podem ser utilizados em todos os níveis da

produção, desde o funcionamento dos ICs elementares até ao nível do sistema final[1]. Esta perspectiva justifica o desenvolvimento e implementação de sistemas integrados de testabilidade, como o *Boundary Scan* (BS).

1.1. CONTEXTUALIZAÇÃO

Historicamente, à medida que os requisitos dos produtos foram justificando um cada vez maior número de componentes nos seus circuitos, a aplicação de equipamentos de teste foi tornando-se cada vez mais complicada. Consequentemente, os *layouts* das placas de circuito impresso (PCB) tornam-se cada vez mais densos, com ICs mais complexos. Estes avanços vieram dificultar o estabelecimento de acesso físico aos nós do circuito impresso, necessário para a verificação da integridade do PCB. Este acesso era tradicionalmente feito por pontas de prova, o que levou à crescente dificuldade no seu posicionamento nos PCBs, cada vez mais densamente preenchidos de componentes e, ao mesmo tempo, com dimensões tendencialmente mais reduzidas.

À parte das dificuldades de acesso, outras razões para o teste e depuração se verificarem cada vez mais difíceis e temporalmente dispendiosos existiam. Espelhando a modernização e evolução da tecnologia, circuitos que tendiam a necessitar de um PCB completo foram comprimidos em ICs compactos, com o incremento considerável do seu número de pinos. O seu teste singular não traria necessariamente uma grande dificuldade, no entanto, um PCB com vários desses ICs já seria muito mais problemático, até por razões de incompatibilidade física com os métodos de teste tradicionalmente utilizados, tornando a testabilidade num factor determinante para o tempo de desenvolvimento e produção. O custo dos equipamentos de teste para múltiplos PCBs era também demasiado significativo para ser opção para pequenas e médias empresas[3].

Os problemas resultantes da necessidade de teste de sistemas digitais cada vez mais complexos levaram assim, em *1985*, à formação do grupo *Joint Test Action Group* (JTAG), uma organização colaborativa para o desenvolvimento de metodologias de teste. Esta vem posteriormente a desenvolver o *Boundary Scan*, uma infra-estrutura de teste por varrimento periférico com o objectivo de melhorar a controlabilidade e observabilidade dos principais *Inputs/Outputs* (I/O) de ICs. Esta metodologia está, desde *1990*, normalizada pelo *Institute of Electrical and Electronics Engineers* (IEEE) como *IEEE 1149.1*, e permite a implementação facilitada de sistemas de teste em placas de circuitos digitais[1][4].

O *Boundary Scan Test* (BST) é assim uma das abordagens possíveis para o teste de PCBs. Esta metodologia prevê a adição de *hardware* dedicado, ao núcleo lógico de controlo do circuito impresso, constituído basicamente por várias células de *Boundary Scan*, que servem como pontos de prova, e um controlador. As funcionalidades deste circuito de teste são determinadas pelo conjunto de instruções implementadas no controlador[5].

Aplicar o BS no desenho de ICs permite o desenvolvimento de PCBs, mesmo com componentes de diversos fabricantes distintos, que suportam métodos padronizados de teste com aplicação em todo o ciclo de vida do produto. Actualmente, o teste por BS é adoptado globalmente por quase todos os fabricantes de PCBs. A principal razão é a contenção de custos, fazendo-se notar em diversas fases do ciclo de vida dos produtos: Na fase de desenvolvimento, permite a economia de tempo nas tarefas de teste e depuração durante a prototipagem do produto. Na produção, consegue um menor tempo de preparação de testes; diminuição do período de diagnóstico de falhas; e o equipamento utilizado é substancialmente mais económico que com outras metodologias de teste. Estes factores contribuíram para uma redução substancial dos custos, e o aumento do ritmo de produção. Os mesmos benefícios contribuem igualmente para uma maior eficiência nos serviços de assistência técnica. Desta forma, a metodologia de teste por BS originou uma redução global dos custos do teste na produção de circuitos impressos que atinge os *70%*, mesmo depois de considerados os custos extra da sua implementação nos circuitos[6].

Numa outra vertente, o BS poderá também ser utilizado como ferramenta pedagógica. Um aspecto desafiante no ensino de sistemas de electrónica envolve o teste de *hardware*, com dados reais, evitando a utilização de simuladores. Embora instrutivo para o aluno experimentar interactivamente com o *hardware*, esta actividade pode ser frustrante se não for conduzida de forma apropriada, pois é normal que os alunos que necessitem de aprender o funcionamento interno de uma plataforma, de forma a poder utilizá-la convenientemente, possam facilmente ficar sobrecarregados de detalhes. Idealmente, a plataforma de *hardware* utilizada para experimentação deverá estar prontamente acessível e ter uma fácil utilização. Ferramentas de alto nível, possivelmente baseadas em acesso via BS, que observem e controlem o *hardware* de teste, podem ter um papel significativo no aumento da produtividade dos alunos[7].

1.2. OBJECTIVOS

O trabalho previsto incide na tecnologia de *Boundary Scan*, e têm em vista o seu estudo; recolha de alternativas para a sua aplicação; e desenvolvimento de um controlador que satisfaça não só as necessidades de depuração de circuitos integrados e teste de PCBs, garantindo a sua controlabilidade e observabilidade, mas que também constitua uma alternativa didáctica a ser possivelmente utilizada no ensino de electrónica.

Espera-se que o estudo realizado possa facultar uma visão global sobre os sistemas de teste e depuração e a sua evolução. Uma abordagem com maior escrutínio será levada a cabo no que concerne especificamente à tecnologia da norma *IEEE 1149.1*, pretendendo-se um manual para a sua aplicação e utilização, de utilidade clara no actual projecto. As alternativas comerciais existentes para a sua aplicação, que se enquadrem nos objectivos definidos, serão também alvo de uma análise cuidada, auferindo-se as suas principais características e funcionalidades, e assim podendo optar-se pelas alternativas de desenvolvimento, para o actual projecto, que mais o beneficiem, de forma mais consciente e enquadrada com outras soluções actuais.

Perspectiva-se que o controlador de *Boundary Scan* desenvolvido reúna um conjunto de características preliminares que se descrevem:

O controlador de BS deverá ser vocacionado para actuar sobre a arquitectura BS de uma placa alvo, sob comando do computador. As funcionalidades de controlo que oferece devem ser essencialmente genéricas, permitindo a utilização da funcionalidade total da norma *IEEE 1149.1*. O controlador não é projectado com vista à depuração de nenhum sistema específico pré-determinado, mas para poder ser utilizado em conjunto com qualquer placa alvo a ser testada, desde que esta esteja em conformidade com a norma. Valendo-se dos benefícios da tecnologia utilizada, pretende-se disponibilizar, não exclusivamente, duas funcionalidades fundamentais: A monitorização do estado lógico das entradas/saídas de um circuito alvo compatível, que assegure a sua observabilidade; e o controlo desses mesmos estados lógicos, permitindo a controlabilidade do sistema.

Esperando-se assegurar apenas que o controlador permita assim a observabilidade e controlabilidade do circuito alvo, parciais ou totais, de acordo com o seu desenho, não se prevê o desenvolvimento de quaisquer procedimentos de teste, que não poderiam ser, de igual forma, genéricos e visar qualquer placa alvo.

O sistema desenvolvido deverá ser adequado a um ambiente pedagógico, prevendo-se que este tenha uma possível utilização por parte de utilizadores de conhecimentos reduzidos na área, pelo que se deseja poder actuar o controlador através de um ambiente gráfico de operação facilitada e esclarecedora dos processos envolvidos. Deverão ser equacionadas funcionalidades adicionais que permitam progredir nestes objectivos.

1.3. ORGANIZAÇÃO DO RELATÓRIO

Este documento está dividido em sete secções.

Inicia-se a exposição conduzindo-se, na seguinte secção *2* ("Boundary Scan no Teste e Depuração de Circuitos"), uma pequena abordagem à evolução das metodologias de teste, como ponto de partida para descrever a motivação para a criação do *Boundary Scan*, e justificação do sucesso observado na sua adopção actual. Este foi comparado com outros métodos de teste, fazendo-se o seu enquadramento com as outras soluções existentes. Seguidamente descreve-se a norma *IEEE 1149.1*, incidindo-se sobre a sua arquitectura e funcionamento técnico. Pretende-se, especialmente, dar a conhecer a forma como a norma deverá ser utilizada e os procedimentos técnicos para o conseguir. Nesse sentido são também abordadas as linguagens padronizadas de especificação de testes e descrição das implementações, utilizadas em conjunto com a norma.

O restante documento aborda a implementação prática do controlador de BS desenvolvido. Na secção *3* ("Planeamento do Controlador BS") descrevem-se as opções técnicas iniciais do desenvolvimento do controlador, no que concerne aos seus objectivos, agora mais determinísticos, e à sua componente física e de programação. Estas opções são apoiadas pela análise de soluções actuais do mercado, para o controlo de interfaces *Boundary Scan*, com objectivos similares ao sistema planeado ou que possam ser utilizadas no seu desenvolvimento.

As secções *4* ("Descrição da Componente Física") e *5* ("Descrição do Software") descrevem, respectivamente, as características técnicas do *Hardware* e *Software* do sistema proposto. Servem também como registo dos trabalhos envolvidos no desenvolvimento do protótipo funcional, e como importante guia para a sua utilização.

A validação do protótipo funcional desenvolvido segue-se na secção *6* ("Teste e Optimização"). Nesta verifica-se o seu funcionamento por meio de testes, e tenta-se

proceder à quantificação do seu desempenho nas tarefas a que se propõe. Alguns dos testes conduzidos visam verificar diferentes abordagens técnicas, que vieram dar origem à implementação de optimizações. Juntam-se também as limitações técnicas resultantes da implementação proposta.

Na secção *7* ("Conclusões") concluí-se quanto aos trabalhos conduzidos e ao controlador de BS proposto. Juntamente, exploram-se outras abordagens possíveis no seu desenvolvimento, e perspectiva-se sobre abordagens recomendadas para uma hipotética evolução futura.

2. BOUNDARY SCAN NO TESTE E DEPURAÇÃO DE CIRCUITOS

O *Boundary Scan* veio dar resposta a vários problemas existentes nos métodos de teste tradicionais, que, entre outras limitações, mostram alguma inércia na sua adaptação à implementação de novas tecnologias nos produtos sob teste. É conveniente conhecer a motivação para o surgimento e adopção do BS para entender a sua grande significância num plano industrial actual.

O corrente projecto justifica também uma abordagem técnica cuidada à norma *IEEE 1149.1* e às técnicas e ferramentas utilizadas para a sua implementação e utilização.

2.1. PROBLEMAS FREQUENTES NA PRODUÇÃO

Numa linha de produção, após os componentes serem montados no seu PCB, este pode ser testado segundo uma perspectiva estrutural ou funcional.

Os testes estruturais demonstram a conformidade de acordo com as especificações de produção, detectando problemas introduzidos durante o processo de produção. Estas falhas

podem ser causadas por defeitos na produção dos próprios componentes; defeitos na produção do PCB, que incluem componentes em falta, componentes errados ou montados com a orientação incorrecta, problemas de curto-circuitos ou circuito aberto na sua junção com a placa, curto-circuitos entre as pistas da placa, ou circuitos abertos por via de pistas quebradas; e por defeitos na produção do sistema, que podem ser causados por problemas como placas em falta, montagem de placas erradas ou incorrectamente encaixadas nos seus *sockets*, ou curto-circuitos/circuitos-abertos na estrutura que faz a ligação entre as placas do sistema.

Por outro lado, os testes funcionais pretendem demonstrar a conformidade do produto com as especificações do *design*. As ocorrências principais destes defeitos verificam-se devido a desgaste ou falhas por razões ambientais, como humidade alta, vibração mecânica, stress eléctrico, ou radiação. Estes problemas são mais difíceis de detectar e diagnosticar do que os problemas estruturais[8]. Como estes testes verificam se a placa funciona como esperado, são também capazes de acusar problemas estruturais caso estes interfiram no seu correcto funcionamento.

A experiência veio a mostrar que a maior parcela de falhas é de origem estrutural. Algumas estatísticas indicam que mais de *99%* das falhas têm origem não em componentes imperfeitos ou problemas de *design*, mas na montagem e soldadura dos componentes no PCB. Esta razão justifica que, pelo menos, as placas mais complexas sejam encaminhadas para testes que detectem problemas estruturais antes dos testes funcionais[9].

2.2. METODOLOGIAS DE TESTE E DEPURAÇÃO

O teste de placas de circuito impresso pode ser conseguido através de uma diversidade de técnicas a serem sumariamente abordadas. A maior parte destas têm hoje limitações importantes na sua adequabilidade ao teste de produtos modernos.

Os maiores destes problemas fazem-se sentir devido, especialmente, à miniaturização que se vem, desde há muito, verificando nos componentes electrónicos e PCBs, utilizando-se ICs progressivamente mais complexos, o que torna o teste das placas numa tarefa de dificuldade acrescida. As tecnologias de teste e depuração tradicionais tiveram que tentar acompanhar estes desenvolvimentos tecnológicos[5].

2.2.1. PROBLEMAS DA TESTABILIDADE

Durante anos, os gestores dos meios de produção viram a testabilidade apenas como algo que não adicionava valor aos produtos, embora os testes permitam distinguir efectivamente um produto imperfeito, evitando que este chegue dessa forma às mãos dos clientes. Várias razões existem para justificar esta imagem negativa dos processos de teste, embora a sua contribuição tenha demarcada importância:[10]

- A ênfase dada à afinação dos processos de produção, pois se estes fossem perfeitos dispensariam as tarefas de teste. A necessidade de sistemas de teste era um lembrete constante à incapacidade da produção;
- A aplicação de metodologias de teste necessitam de recursos (espaço; capital; pessoas; e tempo), embora o produto não seja afectado por este;
- O equipamento de teste era geralmente incapaz de se adaptar à introdução de novos produtos. Existe dificuldade dos sistemas acompanharem as novas tecnologias.
- Alguns sistemas de teste carecem da utilização de espaço na própria placa do produto, podendo causar também limitações de *design*.

Para além destas considerações negativas sobre a testabilidade, as rápidas mudanças características dos tempos actuais, quer a nível de tecnologia quer de necessidades de produção, colocam pressão acrescida nos departamentos de produção e teste. As mudanças tecnológicas têm acontecido desde que a produção de componentes electrónicos começou, no entanto, a frequência com que ocorrem e o seu impacto na produção, teste, e processos de inspecção, tem recentemente aumentado de forma dramática. Processos de teste convencionais, em tempos de tão marcada evolução tecnológica, simplesmente já não podem ser considerados viáveis.

Com a introdução de novas tecnologias, a complexidade dos produtos aumentou de forma acentuada. Esta complexidade acrescida leva a problemas de teste e inspecção, como a falta de acesso eléctrico ou visual. Juntamente, existe também a pressão contínua para se reduzirem os custos de produção sem prejuízo para a qualidade dos produtos[10].

Alguns dos referidos avanços tecnológicos que contribuíram para o aumento da complexidade de acesso de teste aos circuitos:[10]

- Novos tipos de circuitos integrados (*e.g. Ball grid array* – BGA; *Chip scale package* – CSP; *Chip-on-board* – COB; *etc.*);
- Adição dos componentes passivos aos circuitos integrados;
- *Layouts* de placas mais densos;
- Menos, e menores, pontos de teste possíveis.

Estas mudanças reduziram, com especial impacto, a viabilidade dos sistemas de teste baseados em pontas de prova (*probe-based test systems*). Na Figura 1 ilustra-se a evolução do tamanho do encapsulamento físico dos circuitos integrados:

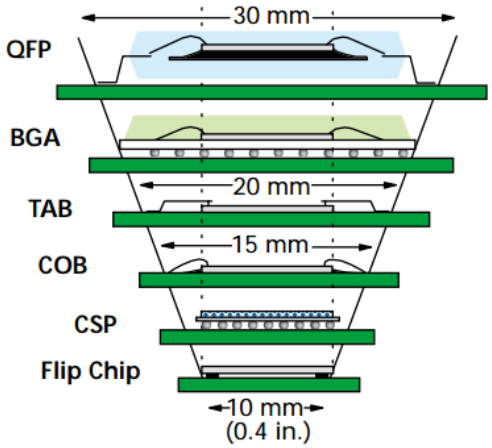


Figura 1 Evolução da dimensão dos encapsulamentos físicos dos ICs[10]

Tradicionalmente, na produção de componentes electrónicos, a automação dos testes era conseguida através de sistemas de *in-circuit testing* (ICT)[10].

2.2.2. IN-CIRCUIT TESTING (ICT)

Os sistemas de *In-Circuit Test* têm sido utilizados por mais de três décadas, para testar a correcta montagem dos componentes em PCBs. A razão para o desenvolvimento deste sistema foi a necessidade de ganhar acesso aos circuitos a nível das suas malhas, de forma a conduzir e ler sinais através dos pinos dos componentes do circuito, com o intuito de determinar se estes foram correctamente colocados e soldados na placa. Dada a corrente densidade das placas, e por vezes os seus requisitos em termos de frequência de operação, torna-se cada vez mais difícil conseguir acesso necessário para o teste dos circuitos[11].

Mesmo com acesso total, o equipamento de teste pode não ser capaz de testar todas as interligações de um PCB de grande dimensão e alta densidade, simplesmente devido ao

grande número de ramos na placa. Adicionalmente, à medida que mais lógica é integrada nos componentes, e a integridade do sinal começa a ser uma preocupação maior do que a dos problemas nas interligações físicas da placa, a relevância dos testes ICT tende a diminuir[11].

Com os sistemas ICT é necessário efectuar contacto com os pinos dos componentes e as linhas do PCB, através de algum contacto físico como pontas de provas. O método de contacto por "cama de agulhas" (*bed-of-nails*) é representativo das vantagens e problemas associados aos testes ICT (Figura 2). Através deste método de acesso podem ser aplicados sinais de entrada aos pinos dos componentes e examinadas as respostas nos seus *outputs*.

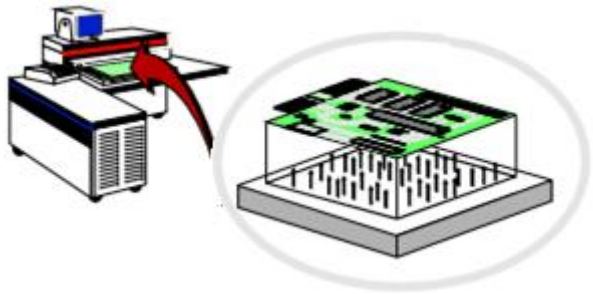


Figura 2 Tecnologia *bed-of-nails* para teste de PCBs[12]

É comum aos testes efectuados por *bed-of-nails* a sua célere execução assim como o relativamente rápido, segundo padrões tradicionais, desenvolvimento de programas de teste, o que se apresenta como os seus principais atractivos. Em contrapartida, contam com uma série de desvantagens:[13]

- O equipamento de teste, e a sua manutenção e armazenamento, são dispendiosos;
- Novos equipamentos podem ser necessários quando o desenho do produto se altera;
- Geralmente não é aplicável durante a prototipagem;
- Equipamentos demoram semanas a serem desenhados e produzidos;
- Problemas de acesso limitado em PCBs modernos.

O teste processa-se em duas etapas: primariamente conduzem-se os testes em modo *power-off*, seguidos dos testes em modo *power-on*. Os testes *power-off*, ou seja, sem alimentação da placa, verificam a integridade dos contactos físicos entre as agulhas e os pontos de prova no PCB, onde são depois conduzidos e recolhidos sinais que verificam circuitos abertos ou curto-circuitos, através da medição de impedâncias. Os testes que se seguem,

com o PCB alimentado, aplicam estímulos eléctricos a determinados componentes da placa, monitorizando a sua consequente resposta. Os outros componentes que estão electricamente ligados ao componente a ser testado são geralmente colocados num modo de segurança, de forma a não interferirem no resultado dos testes, processo denominado de "*guarding*". Desta forma, estes testes são capazes de verificar a presença, orientação e ligações de cada componente escolhido[12].

Este sistema é facilmente executado conjuntamente com placas de tecnologia *trough-hole*[1] (THT), mas a miniaturização referida veio criar os primeiros obstáculos a este sistema de teste[6]. Fundamentalmente, o sistema de contacto através de *bed-of-nails* precisa de acesso físico a todos os componentes da placa que se pretende testar. Nas placas THT este acesso é geralmente conseguido utilizando pontos de prova no lado da placa onde é efectuada a soldadura dos componentes, como sugerido na Figura 3[12].

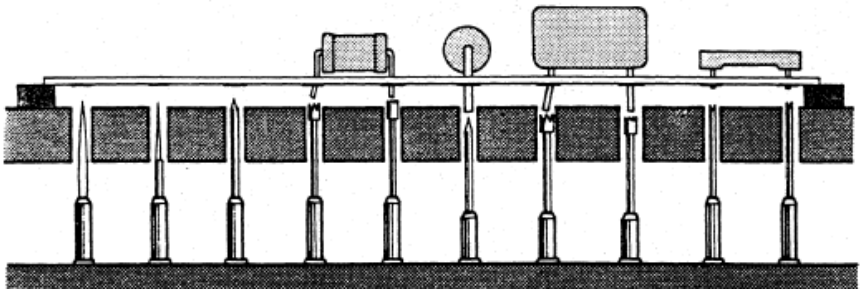


Figura 3 Agulhas de teste num sistema *bed-of-nails*[6]

Já a utilização de *surface mounted devices* (SMD), que possibilitam a colocação de componentes em ambos os lados do PCB, tornou a prova mecânica praticamente impossível[6]. A menor distância entre os pinos dos componentes veio reduzir proporcionalmente a área dos pontos de contacto destes com a placa, o que teve um grave impacto na capacidade de se colocar com precisão as agulhas de teste. A dificuldade de acesso veio ainda ser fortemente agravada com a introdução de placas multicamada[12].

Adicionalmente, uma das vantagens do ICT, a utilização de conjuntos de operações padronizadas de testes, com as quais os programas de teste podiam ser facilmente desenvolvidos, foi desaparecendo devido ao aumento do uso de ICs de aplicação específica (*Application Specific Integrated Circuit* – ASIC). Cada ASIC requer um conjunto separado

[1] *Through-hole technology*, ou a actualizada *plated-through holes* (PTH), são sistemas de montagem de componentes que envolve a inserção de pinos através de perfurações na placa de circuito impresso.

de testes que não está incluído nas bibliotecas ICT *standard*, e o fabricante dos sistemas ICT não consegue fornecer os testes específicos de cada componente. Conjuntos de testes para componentes complexos *Very-large-scale Integration* (VLSI) (*e.g.* microprocessadores) não estão disponíveis, ou têm um custo elevado[6].

As referidas dificuldades, cada vez mais pronunciadas, que se verificavam nos testes ICT, vieram a ser uma das principais razões para a motivação do desenvolvimento de metodologias de acesso para teste e depuração através de *Boundary Scan*. No entanto, o ICT não é a única ferramenta para aplicação industrial no teste de produtos. Outras metodologias implementam diferentes formas de acesso e verificação de falhas.

2.2.3. SISTEMAS DE TESTE E DEPURAÇÃO

Distinguem-se várias metodologias de teste de aplicação industrial:[13]

- *Manufacturing Defect Analyser* (MDA)

Ferramenta que utiliza técnicas ICT que lhe permite detectar defeitos de produção no PCB. Esta utiliza um equipamento de *bed-of-nails* (Figura 4) para aceder ao circuito sob teste, e permite validar o processo de produção no que concerne a problemas estruturais, *i.e.* a correcta colocação e soldagem dos componentes no PCB[14].

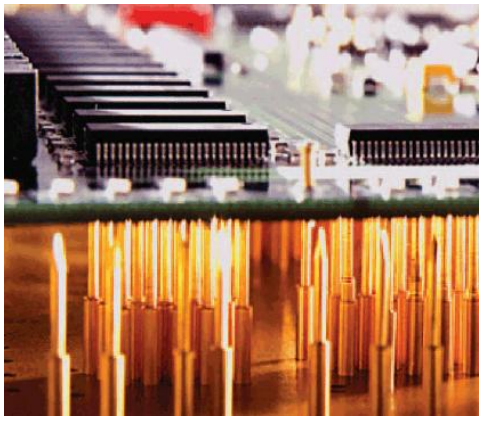


Figura 4 Acesso físico através de *bed-of-nails*

- *Flying Probe Test* (FPT)

Utiliza um equipamento de teste à base de pontas flutuante, que executa muitos dos mesmos testes utilizados por um equipamento de *bed-of-nails*. O FPT utiliza vários braços móveis, com pontas de provas, para aceder aos pontos de teste no PCB. Tipicamente, o PCB é colocado com o verso para cima no equipamento, e os vários braços movem-se pela placa, efectuando contacto nos pontos determinados. A sua grande vantagem é ser mais

flexível do que equipamentos que utilizem adaptadores *bed-of-nails*, adaptando-se a diferentes placas. Os testes são, no entanto, muito mais demorados devido ao movimento dos braços, e poderão haver dificuldades de acesso acrescidas, como se sugere na Figura 5[15].

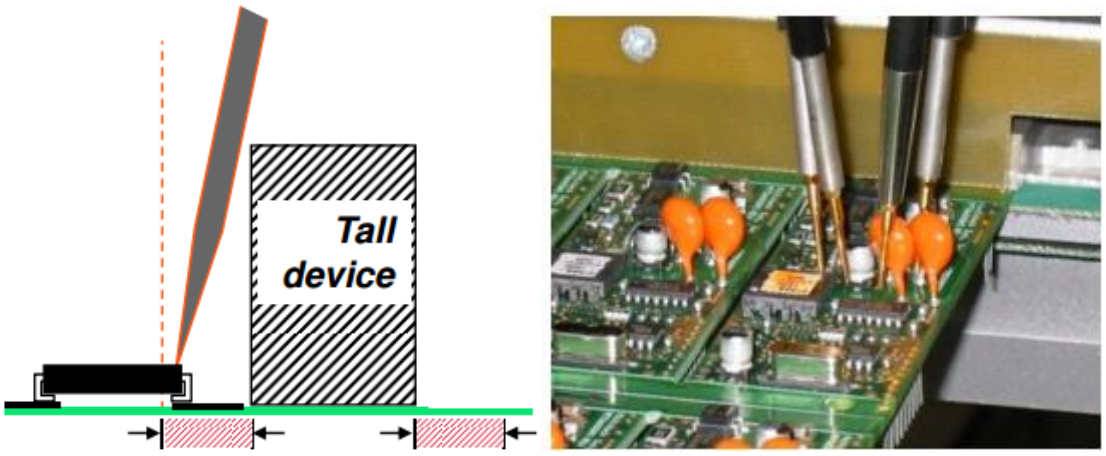


Figura 5 Acesso físico num equipamento FPT

- *Functional Test* (FT)

O teste funcional é geralmente conduzindo através da aplicação e observação dos sinais do PCB acessíveis nos seus conectores. Os sinais de teste devem ser próximos dos valores esperados num funcionamento normal do produto, que é testado desde um funcionamento estático até à velocidade máxima. Adicionalmente, em placas que contenham microprocessadores, os testes funcionais podem incluir que este execute *Functional Self Tests* (FST). Se não estiverem disponíveis quaisquer processadores, podem ser utilizadas técnicas de emulação, por exemplo *In-Circuit Emulation* (ICE), onde algum componente é substituído por um conector de teste que controla os seus sinais[6].

- *Automated Optical/X-Ray Inspection* (AOI/AXI)

Existem várias formas, de utilização industrial, para inspecção recorrendo a imagens, que podem ser ópticas, raio-X, ultrasónicas, térmicas, *etc.* As duas primeiras são frequentes na validação da produção de PCBs. A inspecção automatizada por raio-X tem uma vantagem importante em relação a outros testes estruturais: Os materiais absorvem os raios-X proporcionalmente com o seu peso atómico. A solda utilizada nas montagens electrónicas é feita de elementos pesados (estanho, chumbo, *etc.*), enquanto a maioria dos restantes materiais utilizados em PCBs são feitos de elementos mais leves. O raio-X tem assim uma vantagem única na geração de imagens das junções de solda, o que torna a sua análise facilitada. Exemplifica-se o referido na seguinte Figura 6: [16]

Figura 6 Imagem óptica (esq.) e de raio-X (dir.) capturadas para inspecção

É comum a utilização de uma combinação de vários métodos ao longo do processo de produção, como se sugere na Figura 7:

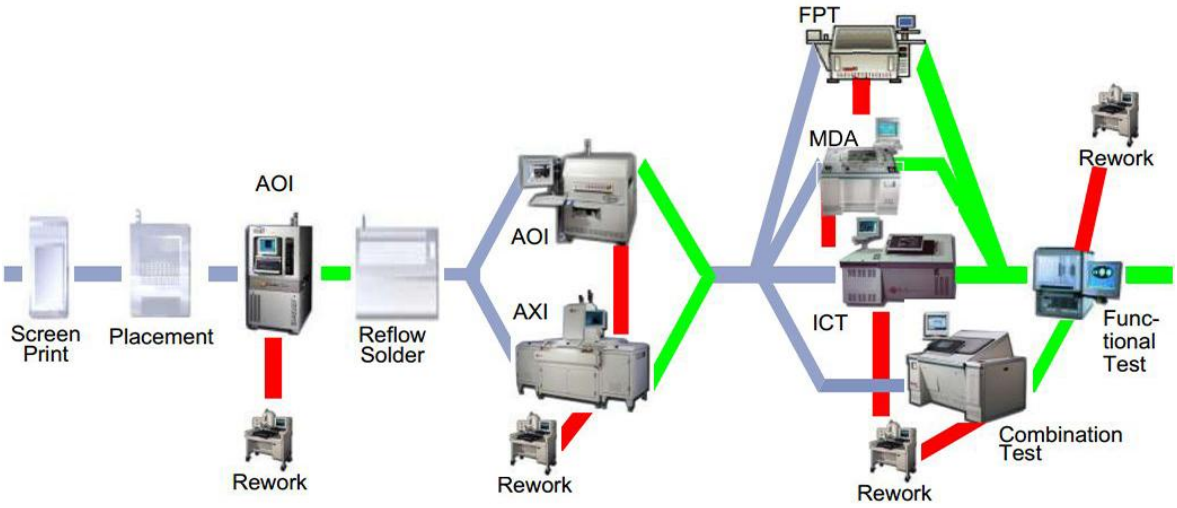


Figura 7 Várias combinações de métodos de teste num processo de produção[13]

Uma análise mais criteriosa a cada um destes métodos, abordando as suas vantagens e desvantagens, e a qualidade e cobertura de teste que proporcionam, é preterível no intento da contextualização das técnicas de teste por *Boundary Scan*, servindo o referido apenas para realçar as dificuldades das alternativas para acesso ao circuito dos PCBs para verificação da sua integridade.

2.3. BOUNDARY SCAN TEST VS. MÉTODOS ALTERNATIVOS

Sumariamente, a utilização de *Automated Test Equipments* (ATE) baseados nos sistemas descritos, considerando o seu acesso mecânico, a sua programação, e a qualidade de teste obtida, está a atingir o seu limite de viabilidade[6]. Estes métodos permanecem susceptíveis a vários problemas comuns, como: a dificuldade de acesso; o custo dos equipamentos; o tempo necessário para o desenvolvimento dos testes, a resolução de diagnóstico reduzida, *etc*[13].

Uma solução para estes problemas surgiu com a integração de tecnologias de teste nos próprios ICs. A utilização de tecnologias integradas de teste por varrimento veio ultrapassar considerável parte dos problemas comuns, e tornou-se uma boa solução para o teste de circuitos digitais. O enorme ganho em termos de observabilidade e controlabilidade que permitem é a base da tecnologia BS na sua aplicação ao teste de PCBs[6].

A metodologia de teste mais directamente comparável ao *Boundary Scan Test* é o ICT. O ICT prevê o acesso ao circuito da placa sob teste através de pontas de prova físicas. O BST implementou o que pode ser considerado como pontas de prova virtuais em cada um dos pinos dos próprios componentes, como sugere a Figura 8:

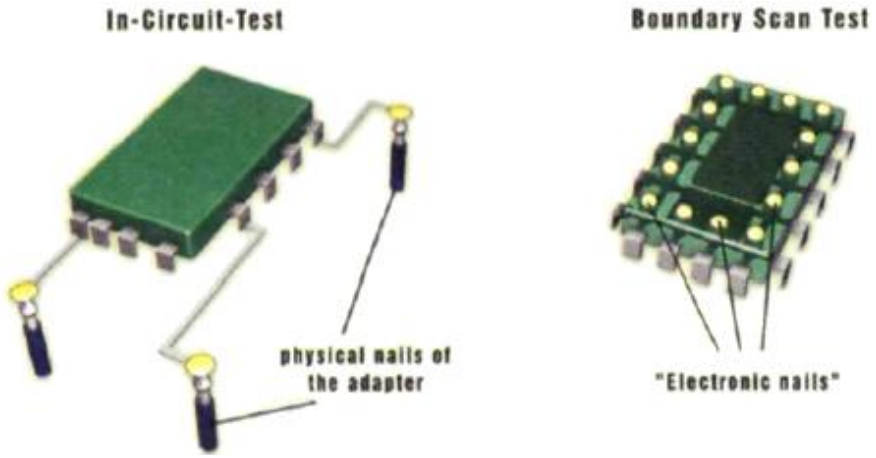


Figura 8 Comparação entre o acesso via ICT e BST[17]

Devido à eliminação do processo de desenho, criação, e verificação do adaptador *bed-of-nails* específico para a placa sob teste, num sistema ICT; e à reusabilidade da arquitectura do BST; o tempo e trabalho necessários para o teste de PCBs, assim como o custo associado, foram muito reduzidos. Para gerar os programas de teste segundo BST são apenas necessários alguns dias, ou até horas. No entanto, para um sistema ICT ou FT, o processo poderá tomar semanas ou meses[18].

Uma empresa especializada no desenho e produção de ATEs – *Teradyne* – estimou que a criação dos testes ICT para um microprocessador usual podia demorar até sete semanas: Uma semana para o estudo do dispositivo; quatro para o desenvolvimento dos testes; e duas semanas para os verificar no equipamento. Se esse microprocessador oferecer suporte para *Boundary Scan*, o tempo de desenvolvimento dos testes, com ferramentas actuais, pode ser inferior a duas horas[12]. Assim, o BS oferece um rápido diagnóstico ao nível dos

pinos dos componentes, evitando os altos custos da referida produção de um adaptador *bed-of-nails*, e também do seu posterior armazenamento e manutenção[18].

Na generalidade, o BST e os sistemas FT, ICT, e FPT oferecem a mesma cobertura abrangente na detecção de falhas estruturais, que, na prática, estará mais dependente de outros factores relacionados com o DFT. Comparado com qualquer uma destas metodologias, o BS oferece uma vantagem económica muito pronunciada[18].

Na seguinte Tabela 1 é comparada a aplicação do BST, ICT, e FPT, segundo várias áreas de análise:

Tabela 1 Comparação entre o BST e outras alternativas[13]

	ICT (*bed-of-nails*)	**FPT**	**BST (*IEEE 1149.1*)**
Velocidade de teste:	**+**	**––**	**0**
Teste de componentes analógicos:	**+**	**++**	**––** (*IEEE 1149.4*)
Teste de componentes digitais:	**0**	**0**	**+** (descr. funcional desnecessária)
Acesso aos pontos de prova:	**–** (mecânico)	**–** (mecânico)	**+** (barramento de teste)
Flexibilidade:	**––**	**+**	**++**
Custo por novo produto de teste:	**––** (adaptador)	**0** (criação do teste)	**0** (criação do teste)
Custos do equipamento de teste:	**–**	**0**	**++**

A anterior tabela considera PCBs onde os testes segundo ICT, FPT, e BST, sejam viáveis, o que depende do seu *design* e *layout*. Pelos motivos que foram sendo referidos, a aplicação dos métodos pode não ser garantidamente bem sucedida, obtendo-se diferentes coberturas do circuito da placa com cada método, dependendo das características do produto em análise.

Existem várias normas que prevêm a aplicação da tecnologia de BS, cujos objectivos principais e as suas características técnicas diferem entre si.

2.4. SUMÁRIO DAS NORMAS BOUNDARY SCAN

Para o desenvolvimento e adopção do BS, o IEEE constituiu um conjunto de grupos de trabalho que visam o desenvolvimento de diversos aspectos distintos da tecnologia. A sua separação é mantida pelas normas resultantes. A sigla JTAG é geralmente utilizada para referir a norma *IEEE 1149.1* − "*Standard Test Access Port and Boundary-Scan Architecture*"[19].

Apresentam-se sumariamente diversos grupos estabelecidos, e respectivas normas, para a aplicação do BS: [9][13][19][20][21]

- *IEEE 1149.1* (*Test Access Port and Boundary-Scan Architecture*)

  A norma original, que visa o teste de circuitos digitais e é abordada em detalhe nas próximas secções. Lançada em *1990*, veio a sofrer várias actualizações resultando nas seguintes revisões: *1149.1a-1993*; *1149.1b-1994*; *1149.1-2001*; e *1149.1-2013*.

- *IEEE 1149.4* (*Mixed-Signal Test Bus*)

  Aprovada em *1999*, descreve um *Boundary Scan* analógico para aplicação em componentes de sinais mistos para teste estrutural, paramétrico e interno. Este implementa dois pinos adicionais (*Analog-In* e *Analog-Out*). Serve para teste paramétrico de componentes passivos (resistências, condensadores, *etc.*), activos (díodos, transístores, *etc.*), e redes de impedância. A disponibilidade de dispositivos com suporte à tecnologia é limitada, não sendo muito utilizada.

- *IEEE 1149.6* (*Advanced Digital Networks*)

  Aprovada em *2003*, aumenta as capacidades do *IEEE 1149* para prever interligações de tensão diferencial e malhas ligadas segundo *AC coupling*[2]. É considerado o *Boundary Scan* para correntes alternadas, de aplicação no teste de pinos I/O tais como *low-voltage differential signaling* (LVDS). Define uma nova implementação no *hardware* e duas novas instruções.

[2] *AC Coupled* denomina malhas onde existem transferências de energia alternada e bloqueio dos sinais de corrente contínua pela introdução de algum elemento capacitivo.

- *IEEE 1149.7* (*Test Access Port and Boundary Scan Architecture: Reduced-pin and Enhanced-functionality* – cJTAG)

  Ratificada em 2009, define a porta da interface BS da próxima geração − uma versão de número reduzido de pinos dedicados, que passam de cinco para dois pinos multiplexados − e acrescenta mais funcionalidades de teste.

- *IEEE P1149.10* (*High Speed Test Access Port and On-chip Distribution Architecture*)

  Esta norma foi proposta com vista a definir uma porta de interface BS de alta velocidade, para a transferência de dados de teste; um formato de descrição de testes; e uma arquitectura de distribuição para conversão dos dados de teste para a estrutura de teste *on-chip*. Prevê a utilização de *High Speed I/Os* (HSIO) na implementação da porta da interface.

- *IEEE 1532* (*Boundary-Scan-based In System Configuration of Programmable Devices*)

  Prevê a normalização de uma metodologia de *In-System Programming* (ISP) de componentes digitais programáveis através de BS. Permite utilizar o BS para a programação de memórias *FLASH*, memórias integradas, *Complex Programmable Logic Devices* (CPLD), e *Field Programmable Gate Arrays* (FPGA). Pretende unificar a abordagem na programação de componentes de diversos fabricantes, programar concorrentemente vários componentes, e reduzir os custos associados ao ISP.

Alguns grupos de trabalho em aspectos específicos do BS foram descontinuados, tornaram-se obsoletos, ou foram completados e fundidos a outra norma (*e.g. IEEE 1149.2/3/5*). Outros novos grupos vêm desenvolvendo normas que visam ser utilizadas conjuntamente com alguma norma BS, num continuar da evolução da testabilidade e instrumentação integrada. Casos disso são, por exemplo, o *IEEE 1581* (*Static Component Interconnection Test Protocol and Architecture*), para aplicação em componentes de memória a serem testados por dispositivos com suporte de BS; ou o *IEEE P1687* (IJTAG), de implementação nos próprios componentes com suporte ao *IEEE 1149.1* a fim de lhes conferir funcionalidades específicas adicionais de instrumentação integrada[21].

O actual projecto de um controlador de BS prevê apenas a aplicação da norma *IEEE 1149.1*.

2.5. NORMA IEEE 1149.1

O BS é possivelmente a técnica de teste com maiores recursos, ao dispor de um acesso tão abrangente e facilitado, comparativamente a outros métodos. Enquanto similar ao ICT, esta não carece de contactos físicos ao implementar milhares de pontos de teste, só precisando de um barramento de teste de, no mínimo, apenas quatro linhas dedicadas (*IEEE 1149.1*)[5][17]. Essencialmente, *Boundary Scan* significa o teste na periferia ("*boundary*") dos ICs. Nestes, adicionalmente à lógica do seu núcleo, é implementada uma lógica de teste especial. Os pontos de teste, ou células de BS, encontram-se entre a lógica do IC e os seus pinos físicos. Esta tecnologia de teste veio a tornar-se no referido *IEEE 1149.1* em *1991*[18].

A ideia básica que motivou o BS foi possibilitar que seja acrescentado nos ICs um circuito de acesso que permita ler, ou estimular com sinais externos, pontos específicos. Os PCBs de grande densidade e múltiplas camadas podem assim deixar de ter um acesso tão dificultado[9].

A interface BS tem também outras possíveis aplicações, podendo ser utilizada para correr testes integrados nos componentes (*Bult-In Self Tests*, BIST) e recolher os seus resultados; como alternativa a ICEs; ou para a programação de componentes programáveis e memórias. Esta última função permite que ICs sem qualquer pré-programação sejam instalados no PCB e só programados antes do seu teste funcional[9].

2.5.1. ÁREAS DE APLICAÇÃO

A tecnologia BS possibilita a observação do estado das entradas dos componentes e o controlo das suas saídas independentemente da sua lógica interna. Através de testes simples consegue-se descobrir defeitos de produção como pinos desconectados, componentes em falta, componentes montados na orientação errada ou componentes estragados.

Enquanto é obvio que o BS pode ser aplicado à fase de produção para o teste estrutural dos produtos, a versatilidade do *IEEE 1149.1* permite que seja utilizado nas outras fases do ciclo de vida do produto, que são apresentadas na seguinte Figura 9[9][22].

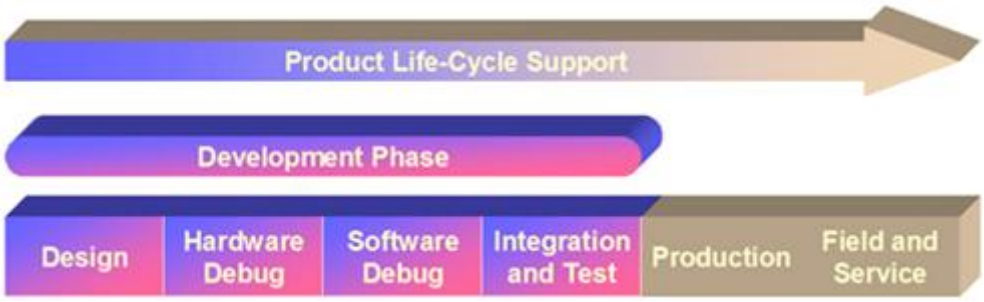


Figura 9 Áreas de aplicação do BS no ciclo de vida de um produto[22]

Porque os recursos BS são tão mais acessíveis que as anteriores alternativas, este pode ser actualmente também aplicado no desenvolvimento do produto, assistindo na depuração da prototipagem e teste do sistema; e nos serviços de assistência técnica. O BS pode ser utilizado antes do *layout* do PCB estar concluído, já que os testes por BS são desenvolvidos apenas com o esquemático do circuito. O *layout* físico dos PCBs não é necessário, como acontece com outros métodos de teste. O custo da implementação do BS pode assim ser amortizado ao longo de todo o ciclo de vida do produto, não se cingindo à sua produção[18][22].

Na produção, o BS tem aplicação no teste estrutural dos produtos, detectando eficazmente as falhas características. A Figura 10 sugere a integração do BS numa linha de montagem:[9]

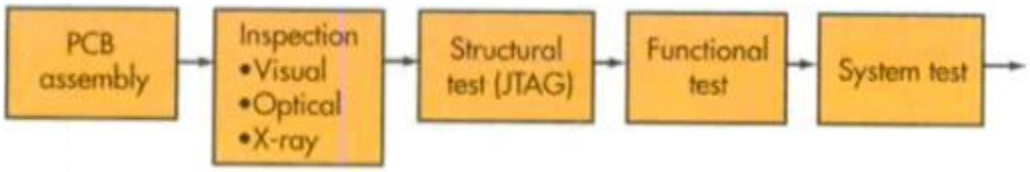


Figura 10 Integração do Boundary Scan (JTAG) num plano de teste[9]

O papel do BS não termina ainda com a finalização do produto. Este pode continuar a ser utilizado, nos centros de assistência técnica, para posteriores actualizações de *software*, actualizando os dados das memórias *FLASH* ou reprogramando os componentes programáveis; ou diagnósticos de falhas, que podem ser conduzidos com os mesmos programas de teste utilizados na produção. Não existe necessidade de equipamento de teste especializado, bastando um computador e controlador de BS para conclusão das tarefas descritas[22].

2.5.2. HARDWARE NECESSÁRIO

Actualmente, muitos, se não todos, os componentes de VLSI; *Ball-grid Arrays* (BGA); *Systems-on-a-Chip* (SOC); ASICs; FPGAs; e módulos *multichip* (MCM); têm um circuito de BS integrado[9]. No entanto, é evidente que nem todos os componentes serão compatíveis. Existem muitos tipos de componentes, frequentemente chamados de "*clusters*", que não estão em conformidade com qualquer norma IEEE de BS, por exemplo: componentes discretos como resistências ou condensadores; a maioria dos componentes analógicos; a maioria dos componentes de memória; *etc.* Estes componentes podem, ainda assim, ser acessíveis por BS, ao terem um ou mais pinos ligados directamente a um componente com BS. Da mesma forma, podem existir componentes não acessíveis, *i.e.* cujos pinos não estão directamente ligados a qualquer pino observável de um componente BS[8]. Para produtos cujo circuito contém partes que não são visíveis da cadeia de BS, como, por exemplo, conectores ou interfaces digitais, alguns fabricantes oferecem componentes de pinos I/O controlados por BS, que são uma alternativa de baixo custo para aumentar a testabilidade do circuito[22].

Quando múltiplos ICs com suporte BS são utilizados num mesmo PCB, os pinos de entrada/saída de dados de cada interface BS (de cada IC) são ligados em série entre si, e o resultado final será similar a uma única interface BS, naquilo que constitui uma cadeia de *Boundary Scan*[9]. A seguinte Figura 11 esquematiza a formação de uma possível cadeia de BS, e o seu acesso distinto, via pinos de I/O, a componentes sem suporte de BS.

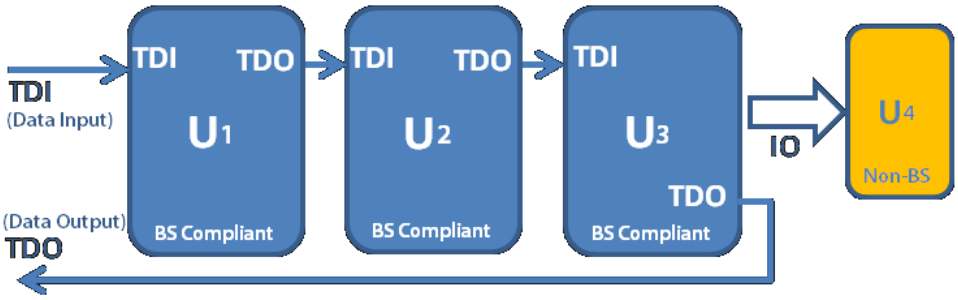


Figura 11 Esquematização de uma cadeia de BS de três componentes

Uma vez ligada em série, a frequência máxima de teste da cadeia será aquela que o seu dispositivo mais lento permitir. Embora seja a configuração tradicional, a implementação das cadeias não têm necessariamente de seguir a estrutura apresentada, dependendo das técnicas de DFT utilizadas. Uma prática usual é o particionamento da cadeia, que permite que certos componentes sejam operados a frequências superiores, resultado em

configurações diferentes da apresentada. Cadeias mais curtas têm o benefício de terem um funcionamento mais célere devido ao menor número de *bits* enviados no seu preenchimento[8].

As linhas de dados, assim como o restante barramento BS, são geralmente ligadas a uma interface, ou controlador, conectada a um computador. Esta interface pode conter memória destinada a ser carregada com vectores de *bits* de teste e a armazenar os resultados intermédios obtidos; e estabelece a comunicação com o computador, onde o *software* de teste é executado[9]. Existem também outras abordagens para a implementação do controlo das interfaces BS. Algumas reduzem a significância da interface/controlador que faz a ligação entre o computador e a cadeia de BS do PCB, passando a servir apenas para conduzir os sinais, ou, noutros casos, é possível até excluir totalmente do processo o computador, e implementar controladores de BS integrados. Sendo esta a área em desenvolvimento neste projecto, serão posteriormente abordadas em maior detalhe as alternativas para a criação de um controlador de BS na secção 3.2.

Qualquer alternativa de *hardware* utilizada como interface deverá poder gerar e adquirir os sinais necessários, mas apenas o *software* de teste, geralmente no computador, determina quais os valores que esses sinais devem tomar[23].

2.5.3. SOFTWARE DE SUPORTE

Parte da norma BS define o *Boundary Scan Description Language* (BSDL), a ser abordado em maior detalhe na secção 2.11.4, que descreve todas as funcionalidades e lógica implementada em cada componente com suporte BS. Ficheiros BSDL descritivos de cada componente são geralmente disponibilizados livremente pelos seus fabricantes. Os *softwares* de teste requerem essencialmente dois elementos para testar um PCB: o ficheiro BSDL de cada um dos IC da placa; e a descrição do seu circuito, que determina as interligações estabelecidas no PCB. Com estes dados, a ferramenta de geração de testes do *software* permite criar os procedimentos de teste para esse PCB[9]. Estas ferramentas desenvolvem todos os vectores de teste necessários para testar a placa e configurar os seus componentes programáveis. Adicionalmente, podem também sugerir estratégias de teste para *clusters* (componentes sem suporte de BS); determinar a cobertura de falhas dos testes; sugerir a adição de pontos de teste (sob a forma de componentes BS) que melhorem

a cobertura do teste; *etc*. O *software* pode então conduzir os testes gerados e fornecer o relatório sobre os seus resultados[9][23].

Outras ferramentas complementares ajudam no desenho de PCBs testáveis por BS, permitindo funções específicas para a visualização do esquemático e análise da cobertura do teste. O elemento determinante na eficácia dos testes por BS é a qualidade das práticas DFT no desenvolvimento do circuito do PCB, já que o *software* não pode ultrapassar as deficiências de um circuito inadequadamente desenhado[23].

2.5.4. LIMITAÇÕES DO BS (IEEE 1149.1)

A norma *IEEE 1149.1*, como referido, tem a sua aplicação melhor orientada para o teste de interligações de circuitos digitais[13].

Outras limitações, menos óbvias, do BS prendem-se especialmente com a sua velocidade inerentemente lenta: Mesmo ao ser actuado a uma frequência de teste próxima dos valores máximos verificados em controladores comerciais (*60 MHz*), o BS não permite um verdadeiro teste funcional a velocidades de operação normais. Consequentemente, os principais defeitos capturados pelo BS são os resultantes de problemas permanentes, como curto-circuitos e outros problemas estruturais. Basicamente, o BS tem dificuldade em executar funções de teste funcional, seja a pequena ou elevada velocidade. A utilização de testes BIST, iniciados por BS, é excepção. No entanto, estes testes servem apenas para diagnósticos internos dos componentes. Outras técnicas similares, como o *Interconnect Built-In Self Test* (IBIST), visam explorar as vantagens destes testes no diagnóstico de interligações, mas não são previstas no *IEEE 1149.1*[8].

Porque o BS utiliza um único canal de dados bidireccional em série, a largura de banda é limitada, o que para comunicações intensivas em aplicações de instrumentação, torna-se uma grande limitação. Da mesma forma, a natureza série do barramento de dados limita o número de componentes BS com que é viável comunicar, o que cria uma barreira limitadora do desempenho da comunicação em sistemas complexos (*e.g.* que utilizem SOCs). A interface do controlador integrado de cada componente BS também não tem quaisquer funcionalidades de segurança, gestão de energia, e outros factores actualmente importantes[24].

2.6. ARQUITECTURA DO BOUNDARY SCAN TEST

A arquitectura do sistema de teste previsto na norma *IEEE 1149.1* é apresentada na Figura 12:

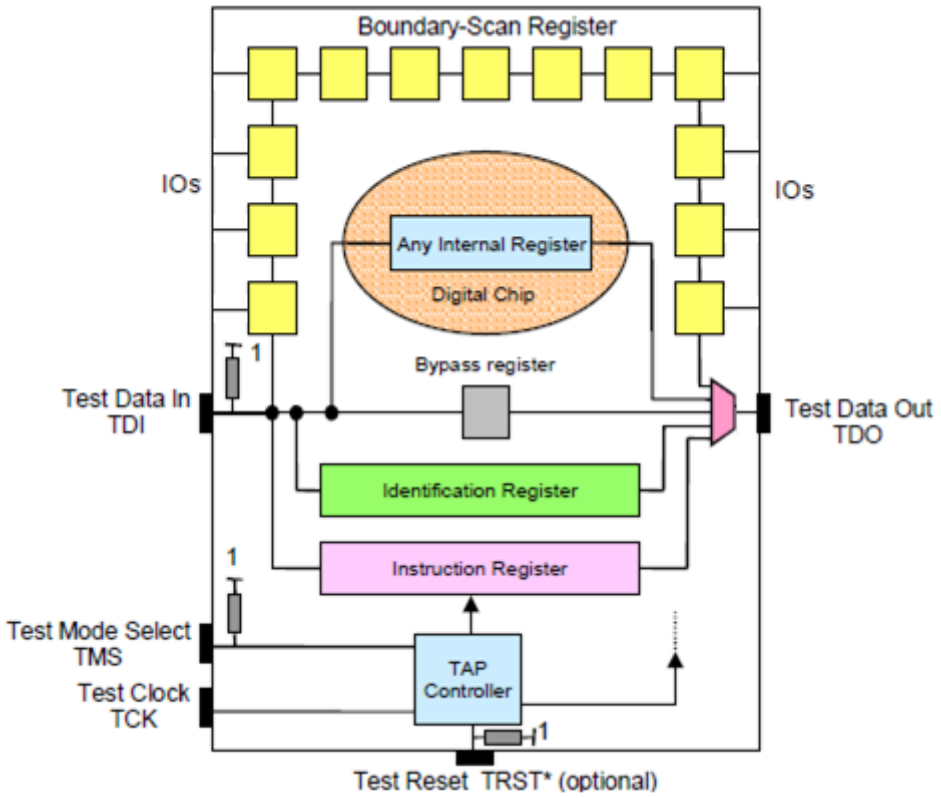


Figura 12 Arquitectura de um IC compatível com o *IEEE 1149.1* (adaptado de [12])

Esta distingue quatro elementos básicos:[5][6]

- A *Test Access Port* (TAP), conjunto de, pelo menos, quatro pinos para controlo dos testes;
- O controlador TAP;
- O registo de instruções (*Instruction Register* − IR);
- Um conjunto de registos de dados de teste (*Test Data Register* − TDR).

Os três primeiros elementos são, de acordo com a norma, indispensáveis para a sua aplicação. Quanto aos registos de informação de teste, apenas são estritamente necessários o registo *Boundary-Scan Register* (BSR) e o registo de *Bypass*[5][6]. O registo de *32 bit* de identificação, *Identification Register*, também presente na anterior Figura 12, é de carácter opcional, servindo para aceder ao código de identificação permanente do dispositivo.

Apenas um destes registos pode, num mesmo momento, estar conectado entre as portas de dados para ser acedido pelo utilizador. O registo TDR seleccionado é escolhido pela instrução introduzida no registo IR. A selecção e acesso ao registo IR fazem-se directamente, num procedimento a ser exposto mais tarde. Tal como os registos associados, algumas instruções são obrigatórias, como *EXTEST* (selecciona o registo de *Boundary Scan*), enquanto outras são apenas de carácter opcional (*e.g.* a instrução *IDCODE*, responsável pela selecção do registo de Identificação)[12].

Realça-se, ainda na Figura 12, a existência de (pelo menos) uma célula ligada a cada entrada/saída primária do componente, que são interiormente conectadas em série constituindo o registo BSR[12].

A aplicabilidade de cada instrução e respectivo registo será abordada mais adiante, progredindo-se na descrição de cada um dos elementos da arquitectura BS.

2.7. TEST ACCESS PORT

A porta TAP consiste apenas nos pinos respectivos do IC, responsáveis pelo controlo da interface de testes, sendo composta por um mínimo de três ligações de entrada e uma ligação de saída. Esta permite o acesso a todas as funções implementadas no controlador TAP. As ligações mínimas são as seguintes: [5][6][12]

- *Test Mode Select* (TMS);

Sinal de controlo com valor lógico padrão '*1*' que deve ser mantido quando não estiver a ser utilizado (geralmente este pino possuí uma resistência de *pull-up* interna). Os sinais lógicos recebidos são interpretados pelo controlador TAP com o fim de controlar as operações de teste. Estes são amostrados no flanco ascendente dos impulsos em TCK.

- *Test Data Input* (TDI);

Entrada em série da informação de teste, que pode ser orientada para os registos de IR ou TDR. Este deve também manter o sinal lógico '*1*' quando não estiver a ser accionado.

- *Test Data Output* (TDO).

Saída em série da informação contida no registo em utilização. O *output* de dados ocorre ao flanco descendente de TCK. Quando não existe informação a ser encaminhada para TDO, este pino mantém-se em estado de alta impedância.

- *Test Clock Input* (TCK);

Entrada para o sinal de relógio de teste dedicado. Os impulsos em TCK permitem temporizar a leitura ou devolução de dados nos restantes pinos da porta TAP, controlando os estados do controlador TAP e a movimentação de dados nas células dos registos. A leitura de informação através de TDI e TMS ocorre no flanco ascendente do impulso de TCK, e a saída de informação em TDO ocorre no flanco descendente do relógio.

O sinal de TCK é independente do relógio de funcionamento de cada componente, facilitando o teste dos vários componentes de uma placa mesmo que estes tenham diferentes frequências de funcionamento. Na sua actuação, pode ser utilizada a frequência que for mais conveniente, e não é necessário que esta se mantenha constante. Para se manter a independência entre os sinais de relógio, o sinal de TCK não pode interferir com nenhum outro relógio do sistema, mesmo que em alguns testes a sincronização de dois relógios possa ser necessária. Tipicamente usam-se frequências de TCK entre *2* e *60 MHz*.

Adicionalmente, pode existir uma quarta ligação de entrada, *Test Reset* (TRST), para inicialização assíncrona da lógica de teste: esta força a reinicialização do controlador e entrada no estado de *reset* ("*Test-Logic-Reset*") ao lhe ser aplicado o sinal lógico '*0*', independentemente dos sinais de TCK e TMS.

2.8. CONTROLADOR TAP

Todas as operações que visam os registos integrados são controladas pelo controlador TAP. Este controlador é composto por uma máquina de estados finita, de *16* estados, que recebe como entradas os sinais provenientes da porta TAP[6]. Através do accionamento sequencial da porta TAP tem-se acesso a todos os procedimentos de teste implementados. O bloco da Figura 13, representativo do controlador TAP, sugere o seu controlo:[12]

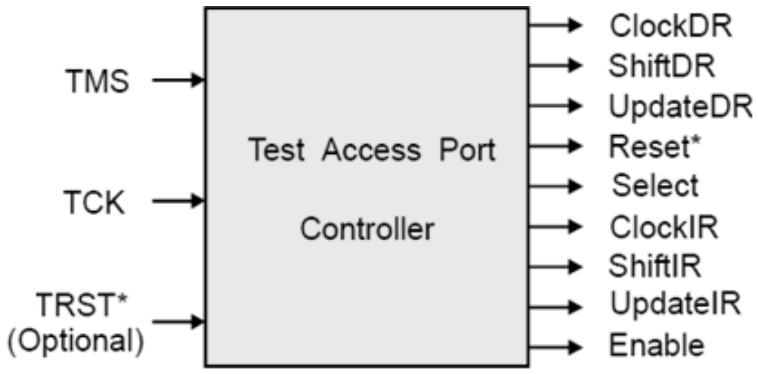


Figura 13 Esquematização do controlador TAP[12]

O diagrama de estados do controlador TAP é de análise imprescindível para o controlo das suas funcionalidades. Este é apresentado na Figura 14:

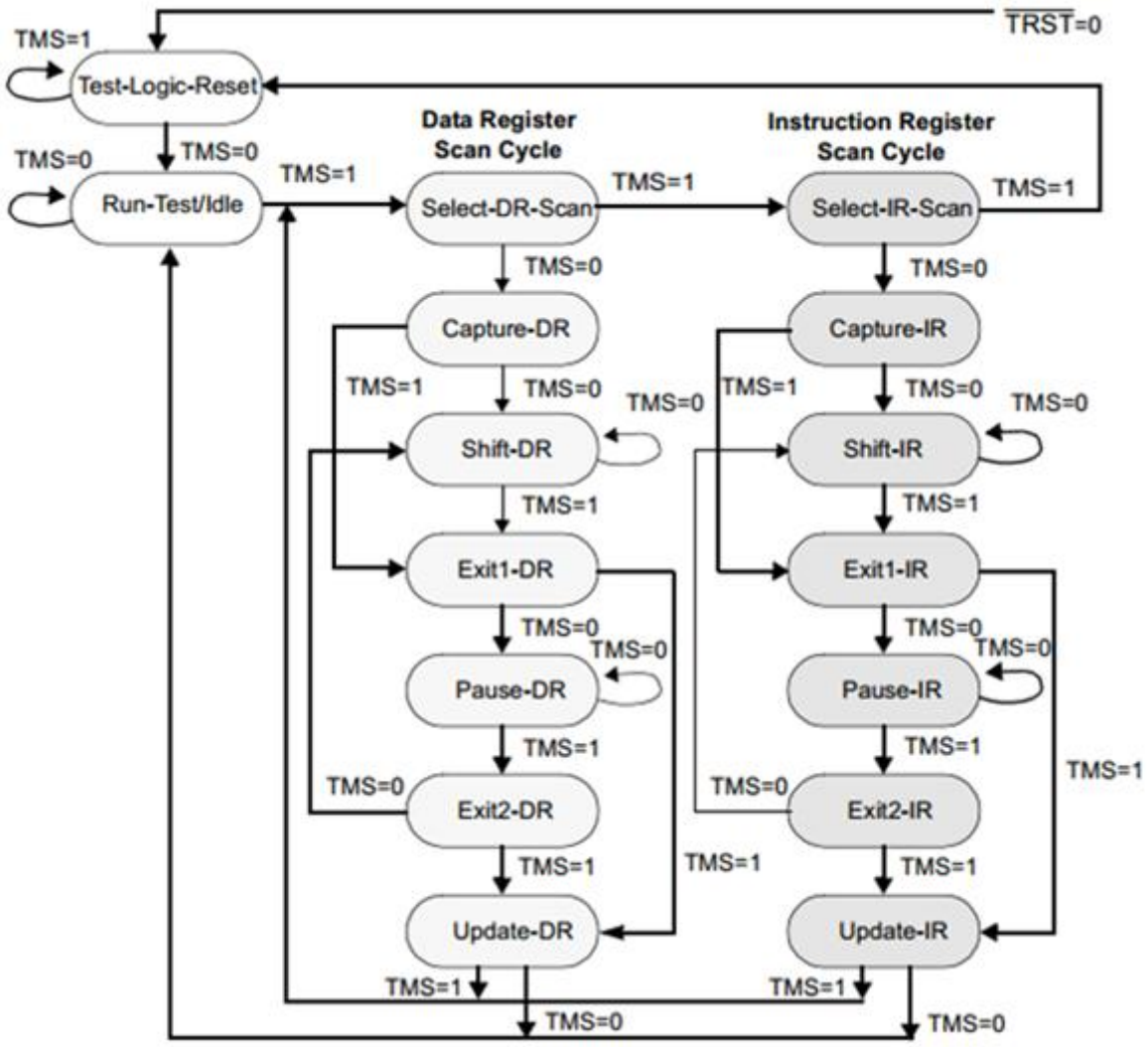


Figura 14 Diagrama de estados do controlador TAP[25]

Os valores lógicos apresentados ao longo dos estados do diagrama da Figura 14 são referentes ao sinal TMS, tal como indicado, no momento de ascensão de um impulso em TCK. Estas são as duas entradas responsáveis pelo fluxo da máquina de estados do controlador.

Na sua inicialização o controlador adopta o estado "*Test_Logic_Reset*", e permanecerá neste estado enquanto o valor de TMS permanecer em '*1*'. O valor '*0*' em TMS, acompanhado de um impulso de TCK, suscita o movimento para o estado "*Run-Test/Idle*", permanecendo neste enquanto não existir alteração do valor de TMS e novo impulso de TCK. O restante controlo do estado do controlador é efectuado da mesma forma e pode ser interpretado no diagrama apresentado.

Os estados especificados com a terminação "*-DR*" dizem respeito à utilização dos *Test Data Registers*, enquanto os estados "*-IR*" utilizam o *Instruction Register*. A função de cada um dos estados é equivalente em cada um dos registos alvo (TDR e IR). Aborda-se resumidamente a função de cada estado: o primeiro estado após selecção do registo – "*CAPTURE*" – marca o momento em que o controlador TAP captura um valor, que depende da instrução seleccionada, para o registo de dados de teste seleccionado. Em "*SHIFT*" é possível deslocar e obter os *bits* capturados, fazendo-se, ao mesmo tempo, a introdução do vector de *bits* de teste pretendido. O estado "*PAUSE*" permite que o controlador TAP se mantenha nesse estado, onde não executa qualquer função, mesmo continuando a receber ciclos no relógio de teste TCK (por exemplo útil para cadeias de BS que partilhem a linha de relógio de teste por todos os seus componentes mas tenham várias linhas de TMS distintas). Por último, em "*UPDATE*", aplicam-se os valores previamente deslocados para o registo em causa. Os estados "*EXIT1/2*" visam possibilitar mobilidades diferentes dentro da máquina de estados (permitem, por exemplo, efectuar pausas e resumir o decorrer do deslocamento de dados para o registo, que pode assim ser feito de forma parcial).

2.9. REGISTOS DO CONTROLADOR TAP

Os registos disponíveis, que já vêm sendo citados, são de conhecimento imprescindível no controlo de uma interface BS. Estes são os seguintes:[6][12][26]

- *Instruction Register* (IR);
- *Test Data Registers* (TDR):
    - *Bypass*;
    - *Boundary Scan Register* (BSR);
    - *Device Identification Register* (opcional);
    - Outros registos específicos dos fabricantes.

A selecção do registo TDR que se pretende utilizar é feita através da introdução da instrução correspondente no registo IR, que pode ser directamente acedido através dos impulsos sequenciais necessários na porta TAP, indicados anteriormente.

Na revisão *1149.1-2013* foi introduzido o conceito de segmento, e permite que segmentos de um registo possam ser excluídos ou incluídos. Esta medida foi introduzida para melhor suportar componentes que contêm partes que possam ser desligadas e ainda assim surgiam num registo TDR. Esses segmentos podem assim ser excluídos do registo quando as partes respectivas não estejam em funcionamento[27].

2.9.1. INSTRUCTION REGISTER (IR)

O registo de instruções, do tipo *serial-in/out parallel-in/out*, é composto por uma secção de "*shift*" (deslocação) que pode ser ligada em série entre TDI e TDO, e uma secção de "*hold*" (manutenção), como sugerido na Figura 15: [5][12]

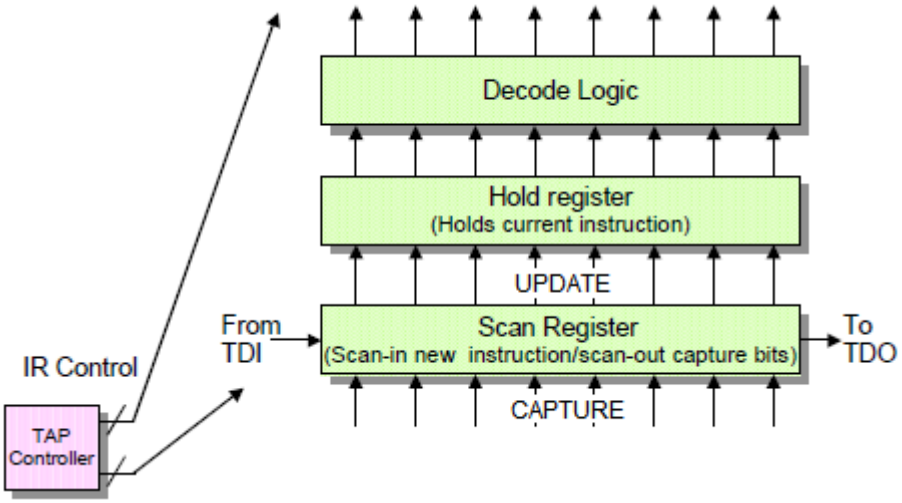


Figura 15 Diagrama do Registo de Instruções[12]

O registo de instruções permite que uma instrução seja para si deslocada, *bit* a *bit*, de TDI para TDO (Figura 16), a cada flanco ascendente de TCK.

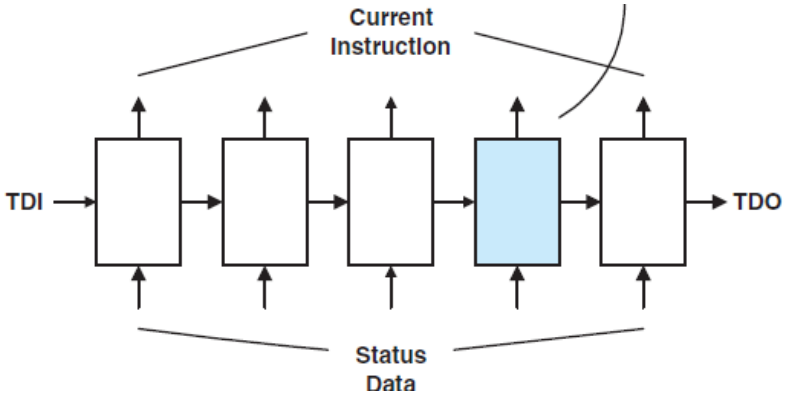


Figura 16 Composição do registo IR[26]

O accionamento sequencial dos sinais de controlo é responsável pelo deslocamento da codificação da instrução pretendida no registo IR, quando no estado *SHIFT-IR*, e pela transição paralela destes dados para a secção de armazenamento, na operação de actualização (*UPDATE-IR*). Nesta secção, os dados são interpretados pela lógica do controlador e é accionada a funcionalidade da instrução inserida.

É também possível a captura de valores no registo IR, movendo-os para a secção de deslocamento ("*Scan Register*" na Figura 15) após o estado *CAPTURE-IR*. No entanto, o valor obtido não será a instrução previamente inserida mas um valor pré-definido pelo fabricante.

A Figura 17 mostra a composição de uma célula do registo de instruções:

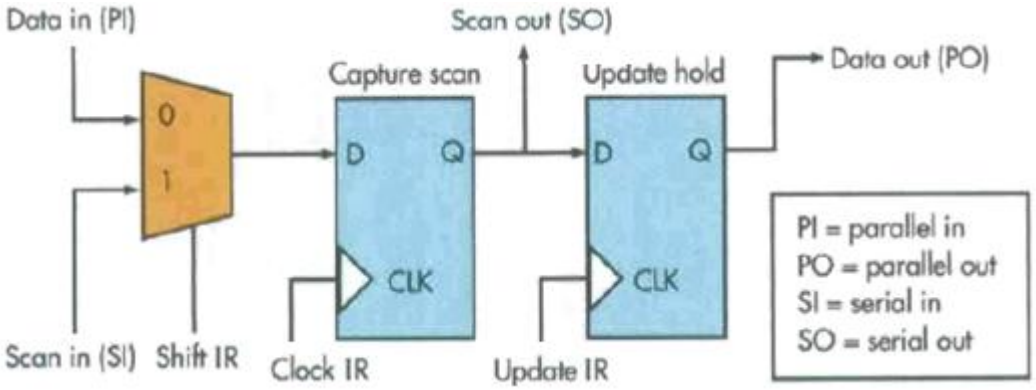


Figura 17 Constituição de uma célula do registo de instruções[9]

A informação actual da secção de deslocamento mantém-se armazenada no registo durante os estados: *EXIT1-IR*; *EXIT2-IR*; *PAUSE-IR* e *UPDATE-IR*. Uma "*latch"* em cada célula do registo de instruções (indicada como "*Capture Scan*" na Figura 17) é responsável por manter o valor lógico actual em cada célula, até que o controlador TAP entre no estado de *UPDATE*, no qual aplica a instrução; ou até ao *reset* do controlador: O estado de *Test-Logic-Reset*, para onde o controlador é remetido, origina o carregamento de uma instrução pré-definida, *IDCODE*, quando implementada. Caso esta não exista no controlador, é carregada a instrução de *BYPASS*. O *output* paralelo do registo (para aplicação da instrução) acontece ao flanco descendente de TCK[5][12].

Os registos de instruções são preenchidos sequencialmente em todos os ICs que estejam ligados em série. Um exemplo demonstrativo desta situação é apresentado na Figura 18:

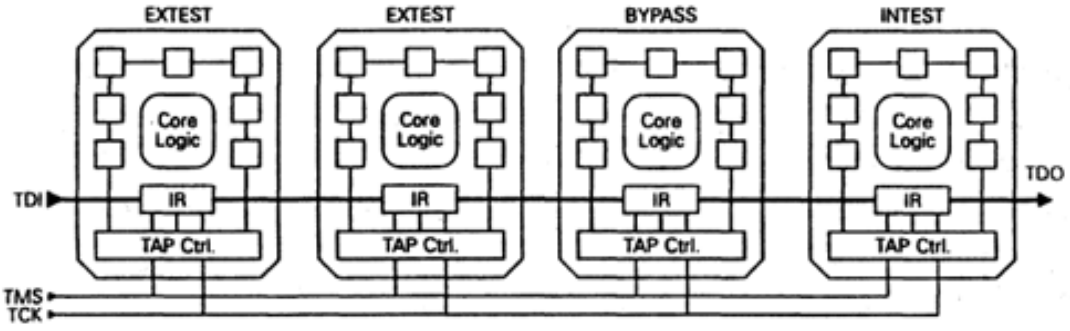


Figura 18 Programação do registo IR em múltiplos ICs em série[6]

De acordo com a instrução fornecida a cada um dos componentes, é possível atingir funcionamentos diferentes entre estes (por exemplo, seleccionando os componentes que são utilizados num teste e os que permanecem em estado de *BYPASS*).

2.9.2. TESTABILIDADE DA CAPTURA DO REGISTO DE INSTRUÇÕES

No modo de captura, os dois *bits* menos significativos (*Least significant bit* − LSB) têm, por imposição do *hardware*, de capturar o valor "*01*". O valor capturado nos restantes *bits* não é previsto na norma. Uma possível utilidade destes *bits*, livres para serem utilizados pelo fabricante, é fornecer informação adicional, *e.g.* um código de identificação informal, se o registo de identificação não estiver implementado[12]. Em casos onde não são utilizados para qualquer utilidade específica, é recomendado que estas células estejam desenhadas de forma a devolverem algum valor lógico fixo, seja '*0*' ou '*1*', no estado *Capture-IR* do controlador[6].

O valor "*01*", sempre disponível nos dois LSBs, é utilizado para testar o próprio sistema de BS, permitindo verificar se existe algum problema no sistema de teste antes de testar a placa. Este procedimento de teste passa pela aplicação da sequência de impulsos de controlo necessária, na porta TAP, para, após captura para o registo IR, movimentar os dados do registo − passando por todos os registos de instruções se considerando vários componentes − até TDO. Depois é possível conferir se a sequência "*01*" está presente nos últimos *bits* de todos os componentes (o comprimento dos valores capturados em cada componente corresponde ao tamanho do seu registo IR). Este teste permite verificar:[12]

- Se todos os sinais da porta TAP estão correctamente conectados desde a sua ligação na placa até à porta TAP de todos os componentes;

- Se cada TDO está correctamente conectado ao TDI do seu componente vizinho;

- Se cada controlador TAP consegue pelo menos responder correctamente a sequencias de TMS e TCK que causem deslocações e capturas no registo IR.

Os passos descritos representam um teste mínimo à integridade da infra-estrutura de *Boundary Scan*.

Outros testes adicionais podem ser executados, como carregar e executar instruções que seleccionem algum TDR, em todos os dispositivos da cadeia, para verificar o correcto funcionamento desses registos (fazendo percorrer um vector de *bits* na cadeia BS). No entanto, se o propósito for só verificar eventuais defeitos de produção na placa, o teste descrito ao registo IR é provavelmente suficiente[12].

2.9.3. CONJUNTO DE INSTRUÇÕES IR

O *IEEE 1149.1* define apenas quatro instruções obrigatórias: *BYPASS*; *SAMPLE; PRELOAD*; e *EXTEST*.

Porque são necessárias pelo menos estas quatro instruções, o comprimento mínimo do registo de instruções é de dois *bits*. O seu comprimento máximo não é definido. Qualquer instrução pode ter mais do que uma codificação correspondente, e todos os códigos não implementados são interpretados como *BYPASS*. Mas é de notar que o fabricante pode utilizar algum código particular para implementar instruções privadas, *i.e.* instruções cuja função não é tornada pública. Estes códigos reservados são geralmente publicados para que o utilizador não os utilize[12].

As instruções opcionais contempladas na norma, apesar de não terem de estar implementadas, têm de obedecer ao funcionamento especificado, mas não têm a sua codificação definida[6][12].

A seguinte Tabela 2 lista as instruções aplicáveis ao registo IR:

Tabela 2 Instruções do Registo de Instruções (IR)[6][26][27]

Instrução	Codificação	Observação
EXTEST [3]	***{000…0} / n.d.***	**Obrigatória**
SAMPLE [4]	**n.d.**	**Obrigatória**
PRELOAD [4]	**n.d.**	**Obrigatória**
SAMPLE/PRELOAD [5]	**n.d.**	Obsoleta
INTEST	**n.d.**	Opcional
USERCODE	**n.d.**	Opcional
IDCODE	**n.d.**	Opcional
HIGHZ [6]	**n.d.**	Opcional
CLAMP [6]	**n.d.**	Opcional
RUNBIST	**n.d.**	Opcional
BYPASS	***{111…1}***	**Obrigatória**

Os objectivos das instruções listadas são os seguintes, começando-se por descrever aquelas de carácter obrigatório: [1][6][12]

- *BYPASS*

A instrução de *Bypass* provoca a selecção do registo de *Bypass* entre TDI e TDO. Por definição, quando não existe a instrução *IDCODE*, esta é a instrução activada após a inicialização ou *reset* do controlador TAP do componente (estado *Test-Logic-Reset*).

- *SAMPLE, PRELOAD, ou SAMPLE/PRELOAD*

A instrução *Sample* e *Preload* não têm qualquer codificação definida, ficando ao critério do fabricante, e são responsáveis pela selecção do BSR para captura (*SAMPLE*) dos sinais lidos no componente, ou para pré-carregar (*PRELOAD*) valores específicos para os seus *outputs*. Os valores pré-carregados não se tornam efectivos até que seja utilizada outra instrução que accione o modo de teste (*e.g. EXTEST*).

[3] Até à revisão 1149.1-2001, a instrução EXTEST era necessariamente codificada por zeros em todos os *bits* do registo IR. Actualmente, a sua codificação não está definida.

[4] Instruções introduzidas pela revisão 1149.1-2001. Podem partilhar a mesma codificação, efectivamente funcionando como a preterida instrução SAMPLE/PRELOAD.

[5] Instrução utilizada nas revisões anteriores à 1149.1-2001.

[6] Instruções introduzidas pela revisão 1149.1a-1993.

- *EXTEST* e *INTEST*

A instrução *Extest* (*External Test*) é codificada com o valor '*0*' em todos os *bits* do registo IR, e é utilizada nos testes de interligações. *INTEST* (*Internal Test*) serve para testar o funcionamento do próprio IC. Ambas as instruções seleccionam o registo BSR. A sua aplicação prática é abordada na secção 2.10.

- *IDCODE* e *USERCODE*

Ambas as instruções seleccionam o registo de identificação mantendo o componente em modo funcional, *i.e.* não interferindo com o seu normal funcionamento. A primeira é utilizada para ler a identificação do componente, definida pelo fabricante. A segunda instrução, *Usercode*, é de carácter opcional e permite armazenar, no mesmo registo de identificação, informação que pode ser definida pelo utilizador.

- *HIGHZ*

Coloca todos os pinos de *output* do componente (incluindo *outputs* de apenas dois estados) em modo de alta impedância, e selecciona o registo de *Bypass*.

- *CLAMP*

Coloca todos os pinos de *output* do componente de acordo com os estados lógicos definidos no registo BSR, e selecciona o registo de *Bypass*. Antes de ser executada, os valores do BSR podem ser pré-carregados com a instrução *Preload*.

- *RUNBIST*

A instrução *RUNBIST* (*Run Built-in Self-Test*) inicia a execução de um teste automático pré-programado (BIST) no componente, para o seu teste interno. Os pinos I/O são mantidos num valor fixo, de forma a não interferirem com o circuito externo nem serem influenciados por este. Os testes BIST só são executados enquanto o controlador se mantiver no estado *Run-Test/Idle*, pelo que deverão ser temporizados para se poder estimar quando terão terminado. O resultado do teste é armazenado no registo TDR, e pode ser lido no estado *CAPTURE-DR*.

A revisão *IEEE 1149.1-2013* veio recentemente prever várias novas instruções opcionais, que se listam de seguida:[27][28]

- *CLAMP_HOLD*, *CLAMP_RELEASE* e *TMP_STATUS*

Instruções vocacionadas para o controlo de um novo controlador opcional, previsto na nova revisão da norma

, *Test Mode Persistance Controller* (TMP), que permite manter a infra-estrutura *IEEE 1149.1* em modo de teste mesmo quando a instrução IR activa não impõe o modo de teste.

- *ECIDCODE*

Instrução de identificação suplementar às já existentes *Idcode* e *Usercode*, que determina a selecção do *electronic chip identification register*. É utilizada para se obter o *Electronic Chip Identification* (ECID), valor que pode permitir provar a autenticidade e identificar várias características do componente, *e.g.* local de produção; gamas de frequências e temperaturas de operação; *etc*.

- *INIT_SETUP*, *INIT_SETUP_CLAMP* e *INIT_RUN*

Instruções para controlo do novo mecanismo opcional de inicialização do componente para preparação do procedimento de teste. Permite que os I/O programáveis sejam definidos antes da condução do teste à placa ou sistema, assim como efectuar alguma tarefa necessária para colocar a lógica do sistema num estado seguro para o teste. Prevêem os seguintes registos TDR: *initialization data register* e *initialization status register*.

- *IC_RESET*

Instrução que permite controlar as funções de *reset* do componente através do controlador TAP, utilizando o registo *reset_select*.

Outras instruções específicas, implementadas pelo fabricante do dispositivo, podem ainda existir. As futuras actualizações à norma *IEEE 1149.1* podem também prever novas instruções.

2.9.4. DEVICE IDENTIFICATION REGISTER (DIR)

Este registo é de carácter opcional e visa fornecer a identificação do componente BS. Todos os dispositivos BS iniciam o seu funcionamento com o registo de identificação seleccionado, que tem prioridade sobre todos os outros, se dispuser deste registo[12]. Assim, se incluído, a instrução *IDCODE* é forçada no *output* paralelo do registo de instruções quando o controlador TAP entra no estado *Test-Logic-Reset*. Esta forma de acesso à identificação dos dispositivos permite proceder a uma rápida listagem, após inicialização da cadeia BS, dos componentes aplicados ao PCB[6].

Os *32 bits* que o registo de identificação armazena informam sobre:

- Identificação do fabricante ("*Manufactor ID*", *11 bits*);
- Número do componente ("*Part Number*", *16 bits*);
- Versão do componente ("*Version*", *4 bits*).

Os códigos de identificação do fabricante são atribuídos, mantidos, e actualizados pela *Joint Electron Device Engineering Council* (JEDEC) no "*Standard Manufacturer's Identification Code*"[1].

A informação referida totaliza apenas *31 bits*, sendo o *bit* em falta destinado ao valor lógico '*1*', que ocupa a posição menos significativa do registo. A distribuição dos campos referidos segue a organização da Figura 19:

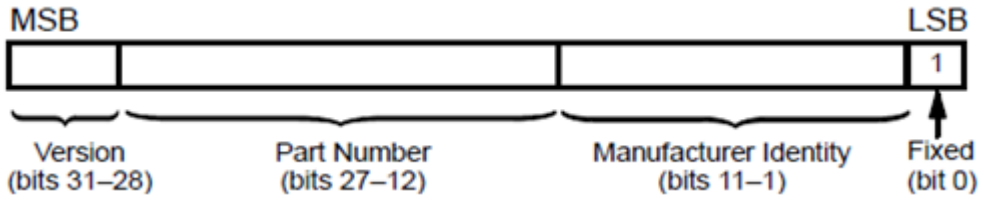


Figura 19 Ordenação do registo de identificação[1]

Uma vez capturados (estado *Capture-DR*), os *32 bits* do registo podem ser seguidamente deslocados através de TDO no estado *Shift-DR*. As células do registo de identificação não possuem qualquer *output* paralelo e, tal como com o registo de *Bypass*, o normal funcionamento do IC não é interrompido enquanto o registo de identificação está a ser utilizado[6].

O LSB do conjunto, de valor lógico predefinido a '*1*', permite uma flexibilidade particular na utilização do controlador TAP. Se o registo de identificação não estiver disponível no componente alvo, é esperado que o controlador TAP atribua o registo de *Bypass*, entre TDI e TDO, como resposta à execução da instrução *IDCODE*. Subsequentemente, se o primeiro *bit* a ser deslocado para TDO não estiver de acordo com o '*1*' lógico esperado, este é indicador que se encontra então, em substituição, activo o registo de *Bypass*. Esta verificação permite, numa cadeia de dispositivos *Boundary Scan*, deduzir que dispositivos têm registo de identificação, sabendo-se que os *31 bits* que se seguem ao primeiro '*1*' serão os códigos de identificação[6]. Numa situação como a descrita, em que se acede a uma placa desconhecida onde não se conhece o número de componentes com capacidade BS

existentes, o processo pode ser terminado introduzindo uma sequência de *bits* em TDI e esperando pela sua chegada, após circularem todos os componentes, em TDO. A sequência utilizada para este fim consiste em sete sinais lógicos '*1*', já que tal sequência é evitada pelo sistema de codificação JEDEC. É usual que esta sequência seja seguida de um '*0*' lógico, para verificação do caso do pino TDI estar preso a '*1*'[12].

2.9.5. BYPASS REGISTER

O registo de *Bypass*, de carácter mandatório, é composto por apenas uma célula e é seleccionado pela instrução *BYPASS*, ou ao inicializar o controlador TAP se este não incluir um registo de identificação. Este registo não tem qualquer *output* paralelo pelo que o estado *Update-DR* não provoca qualquer efeito, sendo indiferente o valor deslocado para o registo. A sua captura, no estado apropriado (*Capture-DR*), devolve um *bit* de valor lógico '*0*' que permite, como explicado na secção 2.9.4, a sua distinção do registo de identificação[12].

A utilização deste registo permite, considerando placas com vários dispositivos BS, abreviar o caminho da cadeia BS reduzindo o número de células de BS nos componentes que não estejam envolvidos no teste pretendido e que não seja necessário aceder aos seus registos[1]. Outro motivo para a sua utilização é deixar o componente funcional no decorrer do teste, já que, tal como o registo de identificação, permite a continuação do funcionamento normal do componente. A redução do comprimento da cadeia BS permite uma menor quantidade de informação redundante a circular e consequentemente consegue reduzir o tempo de execução dos testes, sendo enviados, recebidos, e armazenados na memória do sistema de teste, um possivelmente muito menor número de *bits*[6].

2.9.6. BOUNDARY SCAN REGISTER (BSR)

O outro *Test Data Register* obrigatório, *Boundary Scan Register* (BSR), consiste numa série de células de BS (*Boundary Scan Cells* − BSC) espalhadas ao longo da periferia da lógica do núcleo do IC[1][6]. Estas células estão geralmente colocadas no dispositivo nos seus pinos de entrada; saída; e no controlo de portas bidireccionais e de três estados. A ligação em série entre estas células é determinada pela proximidade física dos pinos do dispositivo ou outras restrições de *layout*[12].

As BSCs devem surgir associadas a todos os pinos do componente, com excepção dos pinos de alimentação. Cada pino do componente está directamente ligado à sua respectiva célula sem qualquer circuito intermédio, à excepção de algum possível *driver* de sinal ou outro circuito analógico[6].

A sua implementação permite a deslocação de vectores de teste sem provocar qualquer interferência na operação do componente. De forma similar aos restantes registos, este deslocamento é efectuado no estado *SHIFT-DR* do controlador TAP, sendo os dados apenas actualizados pelos *outputs* paralelos das células quando se atingir o estado *UPDATE-DR*. O circuito responsável pela retenção e actualização do valor das células faz parte da constituição de cada uma das BSCs[6].

Existem vários tipos de implementações das BSCs, de acordo com o tipo de pino no componente ao qual são aplicadas[12]. A seguinte Figura 20 mostra uma célula de *output* de dois estados, normalizada no *IEEE 1149.1* com a designação *BC_1*:[29]

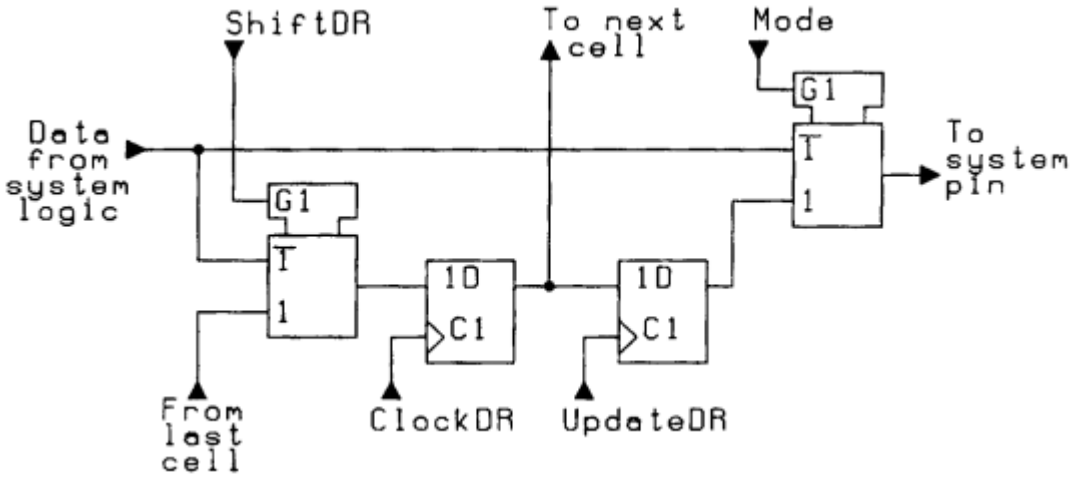


Figura 20 Célula *Boundary Scan* de *Output* (*BC_1*)[29]

O circuito apresentado mostra como esta célula permite capturar o valor da BSC anterior, quando no estado *SHIFT-DR*, e conduzir o seu valor actual para a célula seguinte. Se fora do estado *SHIFT-DR*, o valor capturado corresponde ao que a lógica do componente impõe ao seu pino respectivo, permitindo assim a sua leitura. No estado *UPDATE-DR*, o valor actual da célula é então deslocado para o seu *output*, sendo aplicado ao pino do componente se no modo de teste, determinado pelo valor lógico '*1*' no sinal "*Mode*". Este sinal depende da instrução IR aplicada no controlador TAP, de acordo com a seguinte Tabela 3:

Tabela 3 Geração do sinal "*Mode*" para a célula *BC_1*[29]

Instrução:	*Modo de Teste*:
EXTEST	*1*
PRELOAD	*0*
SAMPLE	*0*
INTEST	*1*
RUNBIST	*1*
CLAMP	*1*

Configurações de várias BSCs para um mesmo pino são possíveis e frequentes. Por exemplo, um pino responsável por um sinal de *output* de três estados carece de pelo menos duas BSCs para que se possa controlar todos os seus três estados possíveis. Na Figura 21 apresenta-se uma implementação, para o caso sugerido, onde é utilizada uma célula de controlo que define se o pino de *output* está activo ou inactivo (estado de alta impedância), em conjunto com uma outra célula que define o seu estado quando activo, conseguindo assim prever os seus três estados:[6]

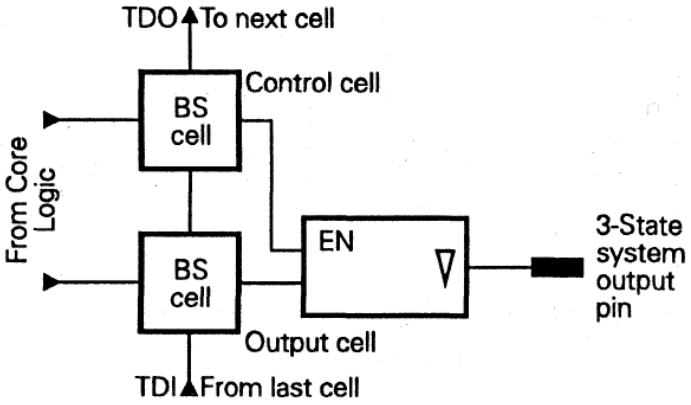


Figura 21 Implementação de duas BSC para o controlo de um *output* de três estados[6]

Outras configurações podem ser implementadas para possibilitar um controlo sobre outros tipos de pinos. As BSCs podem também ser utilizadas para controlar funções internas do componente. Assim, o número de células disponíveis num componente BS é variável e, em geral, bastante superior ao seu número de pinos.

O registo BSR é seleccionado pelas instruções *EXTEST*; *SAMPLE; PRELOAD*; e outras possíveis instruções opcionais ou não previstas na norma[12].

2.10. APLICAÇÃO NO TESTE DE PCBS

Uma estratégia genérica para o teste de PCBs segundo *Boundary Scan* consiste nos seguintes passos: Teste da cadeia de BS e seus componentes; teste das interligações entre componentes BS; e o teste de componentes sem suporte BS[30].

É comum começar por executar algum teste à própria infra-estrutura de teste, verificando a integridade e montagem dos seus componentes. Uma forma de o fazer é deslocar os valores capturados dos registos de instruções de todos os componentes da cadeia, que deverá retornar a sequência "*01*" na terminação do valor capturado de cada componente (mecanismo explicado na secção 2.9.2). Testes adicionais aos componentes da cadeia podem ser alcançados através de outras técnicas, como BISTs; testes internos (instrução *INTEST*); ou tão-somente conferindo os registos de identificação dos componentes (se disponíveis)[30].

Seguidamente, parte-se para a execução de testes externos, que visam as interligações entre os componentes BS. Estes testes são conduzidos aplicando-se estímulos nas ligações do circuito impresso e verificando a correcta leitura desses sinais no outro extremo da ligação, utilizando para tal os registos BSR dos componentes BS do PCB. Esta é a maior aplicação do BS, e permite diagnosticar nas interligações do PCB os tradicionais problemas estruturais de produção[30].

Por último, podem-se também conduzir testes a componentes sem suporte de BS (*clusters*). Muitos destes *clusters* podem ser denominados como componentes "*pass thru*", consistindo em *buffers*, *multiplexers*, *etc*. O teste à sua presença, orientação e soldagem é fácil de aplicar desde que estejam conectados a componentes BS[30]. Outros componentes mais complexos, como *flip-flops*; contadores; *shift registers*; *etc.*; vêm trazer maiores problemas, já que é provável que exista um acesso limitado. Componentes como memórias RAM (*Random Access Memory*), que não estão geralmente equipadas com funcionalidades de BS, podem também criar dificuldades no seu teste. Contando que os componentes BS tenham acesso às portas de controlo, dados, e endereço, dos componentes de memória, estes podem ser testadas através de BS, mas a execução do diagnóstico será expectavelmente lenta, comparativamente aos restantes testes[30].

A norma JTAG, no sentido de permitir os testes anteriormente descritos, prevê fundamentalmente a execução de testes externos, através da instrução *EXTEST*; e testes

internos, através de *INTEST*, cuja implementação é de carácter opcional. Recorda-se que estas não são as únicas ferramentas de teste de que o BS dispõe, tendo-se já referido outros tipos de teste possíveis (*e.g.* BIST).

O teste interno (*INTEST*) visa a utilização do registo BSR para testar a funcionalidade interna do próprio componente. Este prevê a imposição de sinais à sua lógica interna, processado no estado *UPDATE-DR*, de acordo com os dados já deslocados para o registo BSR; e posterior análise da sua resposta, após captura e deslocamento dos novos valores, obtidos no estado *CAPTURE-DR*. Este funcionamento é exemplificado na seguinte Figura 22:

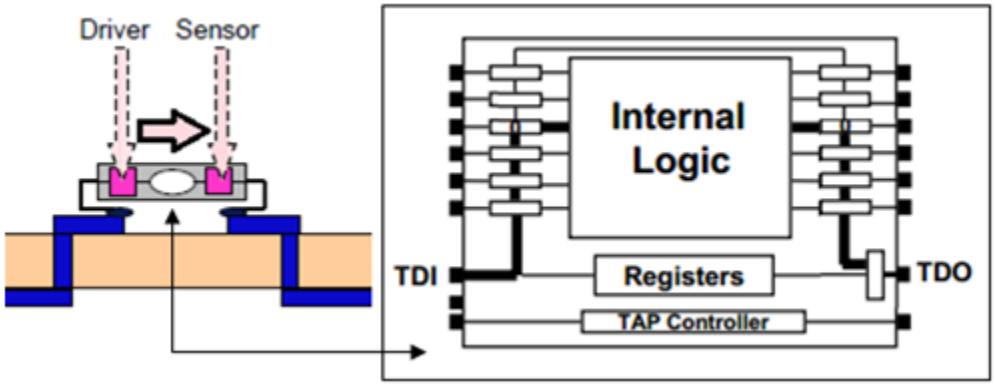


Figura 22 Cobertura dos testes no modo *INTEST* (*Internal Test*) (adaptado de [30])

Esta instrução de teste interno é geralmente só utilizada para efectuar testes muito limitados à funcionalidade dos componentes, identificando problemas como variações incorrectas dos componentes, ou algum defeito interno muito pronunciado[30].

O teste externo (*EXTEST*) já permite o teste das interligações do PCB, o que pode incluir o teste de *clusters*. Este processa-se através da aplicação de sequências de sinais lógicos conhecidos, nos pinos de componentes BS (que ocorre ao estado *UPDATE-DR*), e posterior captura dos sinais recebidos nos componentes aos quais deverá estar ligado. A Figura 23 sugere a aplicação descrita:

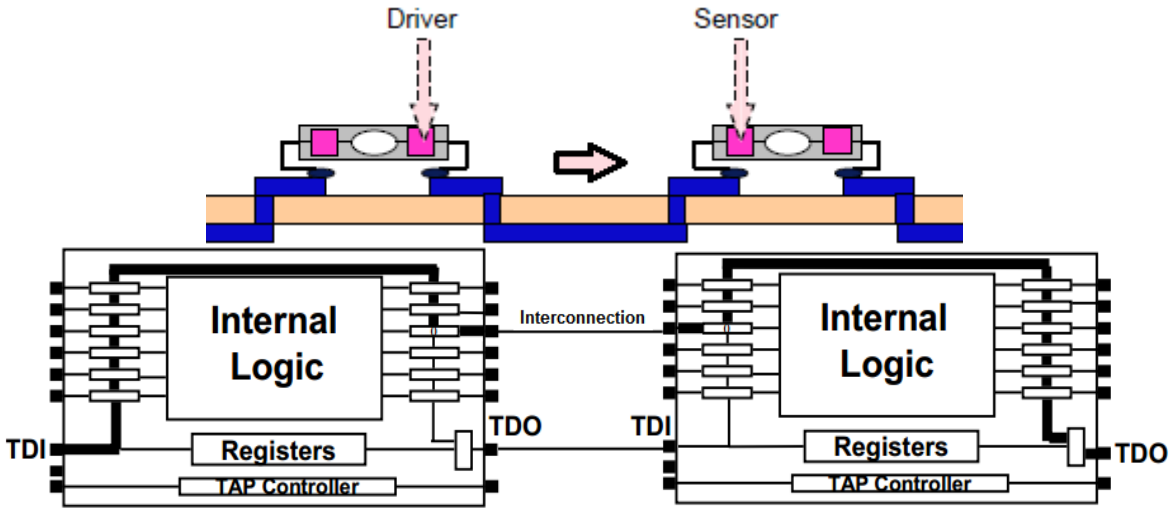


Figura 23 Procedimento dos testes no modo *EXTEST* (*External Test*) (adaptado de [30])

Os sinais lógicos impostos, ao serem verificadamente recebidos noutro componente, confirmam efectivamente a integridade da ligação e a correcta soldagem dos pinos dos ICs envolvidos. O método para a sua verificação não se altera mesmo que esta linha inclua *clusters* simples, como algum componente discreto, já que o sinal digital manter-se-á intacto, e a sua recepção confirmará também a presença e correcta montagem do *cluster*, que caso omisso resultaria num circuito aberto.

As verificações, consistindo na análise de sinais digitais, têm necessariamente de ser efectuadas recorrendo à aplicação de sequências de sinais em cada pino, para que possam ser identificados os problemas possíveis (*e.g.* circuitos abertos ou curto-circuitos). A Figura 24 exemplifica a definição de vectores de teste, a ser aplicados aos pinos de um componente, que permitem verificar a existência de problemas nas suas interligações:

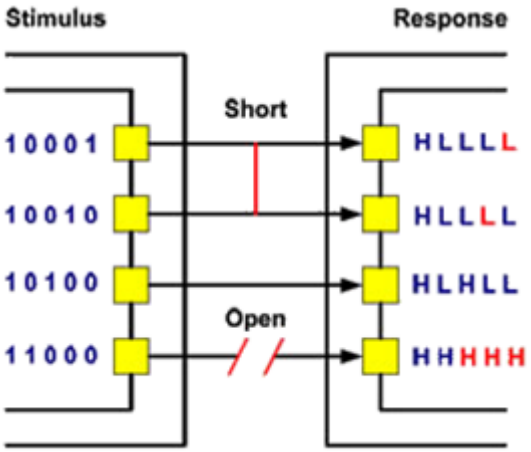


Figura 24 Identificação de falhas por *Interconnect Testing*[22]

Contando com circuitos moderadamente complexos, com cadeias consideráveis de componentes BS, entende-se que a análise ao circuito e descrição do processo de teste pretendido possam ser tarefas complexas. No entanto, estas são muito facilitadas pelas ferramentas de *software* existentes. Para dar o conveniente suporte a este tipo de ferramentas, existem vários formatos de dados com o intuito de facilitar a identificação dos componentes da cadeia BS e o estudo das suas arquitecturas de teste (*e.g.* o objectivo de cada célula dos registos BSR); e a descrição do processo de aplicação de vectores de teste e análise do seu resultado, entre outras tarefas.

2.11. LINGUAGENS/FORMATOS PARA DESCRIÇÃO DE TESTES E COMPONENTES

Reúnem-se nesta secção vários formatos de dados de interesse relacionado com a aplicação da norma *IEEE 1149.1*, tenham o intuito do controlo das operações de teste por *Boundary Scan* (*e.g.* SVF); a descrição da implementação do BS no componente (*e.g.* BSDL); entre outros. O estudo das duas linguagens referidas foi efectuado com maior critério em relação às restantes por virem a ter uma participação fundamental no desenvolvimento prático do corrente projecto.

2.11.1. SERIAL VECTOR FORMAT (SVF)

O "*Serial Vector Format*", comummente referido como SVF, foi desenvolvido pela parceria entre as empresas *Texas Instruments* e *Teradyne* em *1991*. A motivação para o seu desenvolvimento deveu-se à necessidade de um formato de descrição de testes *IEEE 1149.1* que fosse independente dos fabricantes dos componentes BS, capaz de ser utilizado e partilhado por uma grande selecção de *softwares* de simulação e teste de equipamento[1]. Esta reusabilidade e portabilidade foram conseguidas aumentando o nível de abstracção nas descrições dos testes BS, tornando-o num formato neutro, que pode ser utilizado como recurso de várias aplicações compatíveis com a norma BS[6]. A portabilidade pretendida levou a algumas restrições nas capacidades do SVF, o que se torna mais visível comparando-a à, menos simplificada, linguagem *Standard Test And Programming Language* (STAPL), descrita na secção 2.11.3[31].

Os ficheiros SVF, para além da sua aplicação no teste e depuração, são também considerados adequados para a programação de FPGAs; *Electrically Erasable Programmable Read-Only Memories* (EEPROM); e CPLDs; evitando algoritmos de

programação potencialmente complicados para o utilizador. Estes são facilmente interpretados e executados utilizando-se algum "*SVF Player*", interface que traduz o formato SVF nos devidos sinais JTAG[32]. É comum a disponibilização, pelos fabricantes de componentes com suporte BS, de ferramentas de *software* que permitem criar ficheiros do tipo SVF para a programação dos seus dispositivos. Tal acontece, por exemplo, nos conhecidos pacotes "*Xilinx Impact*" ou "*Altera Quartus II*"[33].

O SVF permite a troca de descrições de operações de alto nível no controlador TAP. Em geral, estas consistem em operações de "*Scan*", onde é enviada e recebida informação do dispositivo alvo através de TDI e TDO; e operações de movimento entre os diferentes estados estáveis do controlador. O SVF não descreve explicitamente o estado do controlador a cada impulso de relógio (TCK)[31]. Em alternativa, em lugar de descrever o estado do controlador, as declarações SVF contemplam apenas os dados envolvidos no processo que descrevem − por exemplo o *scan* de algum registo – e o estado estável que se pretende após a conclusão da tarefa. Os estados intermédios "*Capture*"; "*Update*"; "*Pause*"; *etc.*; são deduzidos em vez de explicitamente representados. É, ainda assim, contemplado também um comando ("*STATE*") que permite a navegação determinística nos estados TAP, quando necessário[1].

Formalmente, um ficheiro SVF é um documento codificado segundo o *American Standard Code for Information Interchange* (ASCII), composto por um determinado conjunto de declarações SVF. Cada linha pode ter um número máximo de *256* caracteres, embora uma declaração SVF possa ocupar várias linhas. Cada declaração consiste num comando e parâmetros associados, e é terminada por ponto-e-virgula. O formato não é sensível às capitulações e pode conter comentários, indicados pelos caracteres "!" ou "//" que os precedem. Os dados numéricos dentro das declarações são expressos em hexadecimal e são sempre apresentados entre parêntesis.

A linguagem é composta por catorze comandos, nem todos com parâmetros obrigatórios. Para minimizar o tamanho dos ficheiros SVF, alguns parâmetros opcionais, como: *MASK*; *SMASK*; *TDI*; entre outros, são recordados dos comandos anteriores até que sejam novamente alterados ou invalidados. Estes parâmetros são armazenados separadamente para os comandos: *SIR*; *SDR*; *HIR*; *HDR*; *TIR*; e *TDR*. Os estados do controlador TAP, referidos como parâmetro em alguns comandos, têm nomes específicos, ligeiramente diferentes do nome original dos estados (*e.g.* estado *Test-Logic-Reset* toma simplesmente a

denominação "*RESET*"). A especificação SVF prevê os seguintes estados estáveis: *IRPAUSE*; *DRPAUSE*; *RESET*; e *IDLE*. As operações descritas anteriormente como operações de *scan* têm de terminar num destes quatro estados[31].

Em seguida expõe-se sumariamente a descrição de alguns comandos que se consideram fundamentais, tendo em conta a sua importância na utilização da linguagem:[31][34]

- *Scan Data Register* (SDR) e *Scan Instruction Register* (SIR)

São os comandos encarregues do varrimento dos registos TDR e IR, respectivamente, da cadeia de BS alvo: enviam um vector predefinido (argumento "*TDI*" e "*SMASK*") e verificam se o valor capturado e recebido no decorrer do envio coincide com o esperado (argumentos "*TDO*" e "*MASK*"). Estes comandos devem ser acompanhados pela especificação do número de *bits* de comprimento do vector a enviar. A ordem pela qual os *bits* descritos são enviados e devolvidos é consistente com a definição da norma BS, sendo o LSB enviado em primeiro lugar para TDI e o primeiro *bit* a ser devolvido em TDO. O seguinte excerto exemplifica a utilização destes comandos:

```
SDR 64 TDI(0) TDO(0123456789ABCDEF) MASK(0FFFFFFFFFFFFFFF);
```

Esta declaração obtém os *64 bits* provenientes do *Data Register* do dispositivo, enviando *64 bits* de valor lógico nulo, e compara os valores recebidos com os especificados no parâmetro "*TDO*". O valor de "*MASK*" utilizado indica que, nessa comparação, todos os *bits* são significativos à excepção dos quatro primeiros (de valor lógico nulo). Adicionalmente, pode existir também o parâmetro "*SMASK*", que indica, com o valor '*0*', os *bits* a enviar cujo valor é indiferente ("*don't care*").

- *Header Data/Instruction Register* (HDR/HIR) e *Trailer Data/Instruction Register* (TDR/TIR)

Estes quatro comandos definem o número de *bits* a serem ultrapassados antes e depois de qualquer operação de *scan*. Estes comandos são úteis quando se pretende efectuar algum procedimento em um ou vários componentes consecutivos, de uma cadeia de BS, ignorando os seus restantes elementos. O número de *bits* a ultrapassar, e os dados que serão aplicados a tais elementos da cadeia, podem deixar de ser contemplados nas instruções SDR e SIR após a utilização destes comandos.

Os conceitos de *header* e *trailer* pretendem distinguir o número de células antes e depois da área alvo da cadeia, tal como sugerido na Figura 25:

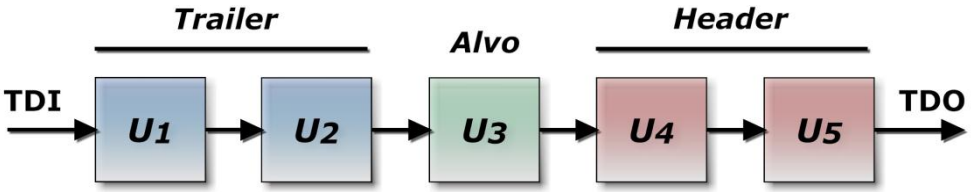


Figura 25 Localização do *trailer* e *header* em relação ao componente alvo

O código seguinte é um exemplo de uma declaração de um *header* de *32 bits* a ser considerado na execução de qualquer *Scan* a um registo TDR:

```
HDR 32 TDI(00000010) TDO(81818181) MASK(FFFFFFFF)
SMASK(0);
```

Este contempla os parâmetros que, há semelhança dos utilizados nas instruções de *scan*, definem os *bits* a enviar para os componentes considerados no *header* e os valores de *TDO* esperados.

Os restantes comandos disponíveis podem ser consultados na especificação SVF, referida na documentação bibliográfica[31].

2.11.2. XILINX SERIAL VECTOR FORMAT (XSVF)

Para possibilitar a funcionalidade dos ficheiros SVF num formato binário mais compacto, o fabricante *Xilinx* criou o formato XSVF. Os ficheiros XSVF estão optimizados para a execução de operações de BS em componentes *Xilinx* e destinam-se, não exclusivamente, a ser utilizados em aplicações integradas.

A diferença fundamental entre o SVF e XSVF é que o segundo formato possibilita uma melhor compressão dos dados, produzindo ficheiros mais pequenos. O formato adoptado num ficheiro XSVF pressupõe a utilização de instruções de *1 byte* seguida de uma quantidade variável de argumentos. Os valores binários de cada instrução ("*XSIR": 0x02*; "*XSDR": 0x03*; *etc.*), e dos seus argumentos é disponibilizado para a escrita dos ficheiros, mas torna a sua interpretação mais difícil para o utilizador[32].

Como referido, este formato torna-se assim mais adequado para utilização em sistemas integrados, podendo ser lido e executado em microprocessadores, e ser guardado na própria memória do sistema.

Os ficheiros XSVF podem ser criados através da ferramenta *Xillinx SVF2XSVF v5.02*, que traduz ficheiros SVF, de mais fácil escrita para o utilizador, para o formato XSVF. No

entanto não é possível converter a totalidade da linguagem SVF, já que nem todos os seus comandos são suportados no formato XSVF[32].

2.11.3. STANDARD TEST AND PROGRAMMING LANGUAGE (STAPL / JAM)

Outro formato para a descrição de acções através de uma interface BS é o STAPL, actualmente standardizado pela JEDEC como *JESD-71*. Comparativamente ao SVF, este parece-se mais com uma linguagem de programação, dispondo de funções de ciclos, execução condicional, entre outras. Este formato cria ficheiros do tipo JAM[33].

O STAPL foi desenhado para suportar a programação de componentes programáveis e o teste de sistemas electrónicos através da norma *IEEE 1149.1*. Este formato suporta a programação e teste de sistemas com ou sem interface de utilizador (suporta sistemas integrados). Um só ficheiro JAM pode conduzir vários procedimentos diferentes, como programar, verificar, e apagar a memória programável do componente.

A formatação STAPL disponibiliza as suas diferentes funções de alto nível através de declarações do tipo "*ACTION*", que correspondem aos controlos do utilizador num sistema interactivo. Em sistemas integrados, sem interface de utilizador, os ficheiros podem ser filtrados removendo estas declarações e correspondente procedimento, quando não necessárias (um ficheiro JAM pode suportar muitas declarações "*ACTION*"), reduzindo o tamanho do ficheiro e tornando-o mais adequado a sistemas com recursos de memória limitados. Para além disso, o STAPL pode ser implementado como uma linguagem interpretada (como o SVF), onde um programa interpretador executa directamente os procedimentos descritos nas declarações (ficheiros "*.jam*"; "*.stapl*"; "*.stp*"; *etc.*); ou como uma linguagem compilada, cujos ficheiros costumam tomar a extensão "*.jbc*" (*JAM Byte-Code*). No último caso, o código tem primeiro de ser pré-processado, compilado num ficheiro binário, e só depois executado[35].

Um ficheiro descrito em linguagem STAPL pode conter as seguintes componentes:[35]

- Dados sobre *design*, identificando o(s) componente(s) alvo;
- Informação sobre o algoritmo para programação *In-system programming* (ISP) do(s) componente(s) alvo;
- Configuração da(s) cadeia(s) de BS;

- Vectores de teste;
- Qualquer instrução baseada no funcionamento do *IEEE 1149.1*;
- Instruções extra-*IEEE 1149.1*.

Um ficheiro para programação de um componente programável deve conter os dados a serem programados e o algoritmo de programação para o fazer. A configuração da cadeia BS pode ser especificada através de uma interface gráfica, utilizando as instruções STAPL respectivas, ou lidas de um ficheiro descritivo, utilizando o *Standard for Chain Description File (EIA/JESD32)*. O STAPL oferece também um conjunto adicional de instruções, de implementação opcional, que permitem aceder e controlar sinais do *hardware* do sistema alvo que não fazem parte da interface *IEEE 1149.1*[35].

Formalmente, os ficheiros JAM consistem numa sequência de declarações. Cada declaração, que ocupa geralmente uma linha, contém uma identificação opcional, uma instrução, e os seus argumentos, terminando com o caractere ";"[35].

2.11.4. BOUNDARY SCAN DESCRIPTION LANGUAGE (BSDL)

A partir de *1990*, quando o *IEEE 1149.1* foi aprovado, à medida que este vinha a ser mais utilizado, foi notória a necessidade de um método padronizado de descrição dos componentes compatíveis. O mesmo grupo de trabalho envolvido na normalização do BS voltou a intervir no desenvolvimento de uma linguagem descritiva que respondesse ao referido[12].

A *Boundary Scan Descriptive Language* (BSDL) é um formato de dados padronizado, que é subconjunto da *VHSIC Hardware Descriptive Language* (VHDL), e descreve a implementação do *IEEE 1149.1* nos componentes suportados. Esta linguagem foi aprovada como um suplemento à norma original, em *IEEE 1149.1b-1994*[6][36]. A informação que se descreve não contempla as alterações implementadas na última revisão da norma (*IEEE 1149.1-2013*).

O propósito do BSDL é servir de linguagem comum para descrição da implementação do BS nos diferentes componentes de diversos fabricantes. É esperado que fabricantes de componentes que suportem a norma BS disponibilizem os ficheiros BSDL aos seus clientes[36]. As capacidades de cada componente BS são definidas nesses ficheiros BSDL,

que descrevem: o comprimento e estrutura dos seus registos BS, a codificação binária das suas instruções, o seu código de identificação, a disponibilidade do pino opcional TRST, a localização física da porta TAP, *etc*[32]. Realça-se que o BSDL não tem intenção de descrever qualquer parte da lógica interna do componente para além das propriedades dos seus registos BS e respectivas ligações físicas[6][12].

Os ficheiros BSDL podem ser utilizados para:[36]

- Descrição da lógica de teste;
- Sintetização da lógica de teste;
- Geração de testes para um circuito;

O terceiro item é o mais relevante: em conjunto com ferramentas de *software* especializadas, os ficheiros BSDL permitem determinar as sequências de *bits* necessárias para a geração de testes diagnósticos de falhas, através de BS, pela estimulação e leitura das entradas/saídas dos componentes. Esta é uma das maiores utilizações dadas aos ficheiros BSDL, que vêm viabilizar a automatização dos testes a PCBs, permitindo que as ferramentas que suportem o BSDL consigam controlar convenientemente a porta TAP dos componentes ao saberem como o BS foi implementado em cada um destes[6][12][36].

A descrição BSDL de um componente consiste nos seguintes elementos:[6][12]

- Descrição da entidade (*Entity description*)

  A declaração da entidade identifica o nome do dispositivo e engloba todo o conteúdo do ficheiro BSDL. *e.g.:*

```
entity componente is
{       <Restantes elementos descritivos>     }
end componente
```

- Parâmetro genérico (*Generic parameter*)

  Parâmetro que identifica o mapeamento físico utilizado, referindo o seu nome no elemento "*Pin Mappings*". É obrigatório mas pode ser declarado "*undefined*". *e.g.:*

```
generic (PHYSICAL_PIN_MAP : string := "TQFP");
```

- Descrição das portas lógicas (*Logical port description*)

  Incluí a descrição dos pinos do componente, atribuindo-lhes um nome simbólico e definindo as suas características, tais como o seu modo de funcionamento, agrupamento lógico, *etc.* A sua ordenação não tem significância. *e.g.:*

```
port (EN            : in bit;
Y                   : out bit_vector(1 to 3);
A                   : in bit_vector(1 to 3);
GND, VCC            : linkage bit;
TDO                 : out bit;
TMS, TDI, TCK       : in bit);
```

- Declarações de uso (*Use statements*)

Este elemento identifica os pacotes VHDL necessários com a declaração dos atributos, tipos, constantes, entre outros, que são utilizados na descrição. *e.g.:*

```
use STD_1149_1_1994.all;
```

- Mapeamento dos pinos (*Pin mapping*)

Mapeamento dos sinais definidos, na descrição das portas lógicas, no seu respectivo pino físico nos vários encapsulamentos disponíveis. Para o mapeamento é utilizada uma única declaração "*attribute*"; e uma "*constant*" para cada tipo de encapsulamento disponível. Os parêntesis agrupam vectores de *bits*, e o caractere '*&*' (concatenar) pode ser utilizado para dividir a *string* por várias linhas. *e.g.:*

```
attribute PIN_MAP of componente : entity is
PHYSICAL_PIN_MAP;
constant TQFP:PIN_MAP_STRING:=
"EN : 1, Y : (2,3,4), A : (5,6,7), GND : 8," &
"VCC : 9, TDO : 10, TDI : 11, TMS : 12, TCK: 13";
```

- Identificação da porta TAP (*Scan port identification*)

Elemento que define a porta TAP do componente. Os sinais referidos (*TDI, TDO, etc.*) devem constar da descrição dos pinos. O valor booleano dos atributos é arbitrário, só necessário para o associar ao sinal. O atributo de *TCK* refere a sua frequência de operação máxima suportada; e, no segundo campo, "*BOTH*" ou "*LOW*" identifica o estado em que *TCK* pode ser parado sem que haja perda de dados. *e.g.*:

```
attribute TAP_SCAN_IN of TDI : signal is TRUE;
attribute TAP_SCAN_OUT of TDO : signal is TRUE;
attribute TAP_SCAN_MODE of TMS : signal is TRUE;
attribute TAP_SCAN_CLOCK of TCK : signal is
(8.0e6, BOTH);
```

- Descrição do registo de instruções (*Instruction Register description*)

Identifica características do registo de instruções do componente: comprimento em *bits*; codificação das instruções suportadas; valor devolvido na sua captura; instrução para desactivação do componente; instruções privadas de acesso não recomendado; e informação adicional sobre a utilização de alguma instrução. *e.g.:*

```
attribute INSTRUCTION_LENGTH of componente :
entity is 2;
attribute INSTRUCTION_OPCODE of componente :
entity is "BYPASS (11), EXTEST (00), SAMPLE (10)";
attribute INSTRUCTION_CAPTURE of componente :
entity is "01";
```

- Descrição do acesso a registos (*Register access description*)

Indica que registo é colocado entre *TDI* e *TDO* para cada instrução. As instruções obrigatórias, definidas na norma, não precisam constar desta listagem, já que o registo que cada uma selecciona é já conhecido. *e.g.:*

```
attribute REGISTER_ACCESS of componente : entity
is "BOUNDARY (EXTEST, SAMPLE), BYPASS (BYPASS) ";
```

- Descrição do registo de *Boundary Scan* (*Boundary Register description*)

Incluí a listagem e descrição das células do registo BSR e o seu comprimento. A norma define os tipos de célula "*BC_0*" a "*BC_10*" (definidos em "*STD_1149_1_1994.all"*), que distingue as várias implementações físicas. *e.g.:*

```
attribute BOUNDARY_LENGTH of componente : entity
is 7;
attribute BOUNDARY_REGISTER of componente :
entity is "0 (BC_1, Y(1), output3, 0, 6, 0, Z),"&
" 1 (BC_1, Y(2), output3, 0, 6, 0, Z),"&
" 2 (BC_1, Y(3), output3, 0, 6, 0, Z),"&
(...)
```

A descrição das células obedece à seguinte formatação:

```
<núm> (<tipo>, <porta>, <função>, <estado_seguro>,
<cél_controlo>, <valor_cél_ctrl>, <resultado>)
```

O BSDL pode ser utilizado num sistema totalmente compatível com VHDL, onde esta informação BSDL é conduzida para um analisador VHDL e sintetizada numa biblioteca de *design*, de onde os dados sobre a arquitectura BS do componente são extraídos através da referência aos respectivos atributos; ou num sistema VHDL parcial, onde apenas um conjunto limitado da sintaxe do ficheiro passa por um processo de *parsing*. Este é efectuado por um *software* que interpreta o conteúdo do ficheiro BSDL como uma *string* de símbolos, e constrói uma estrutura de dados com as informações retiradas[6].

2.11.5. HIERARCHICAL SCAN DESCRIPTION LANGUAGE (HSDL)

A *Hierarchical Scan Description Language* (HSDL) visa descrever como o *IEEE 1149.1* está implementado numa placa ou sistema. Esta continua para além do conseguido num

ficheiro BSDL: descreve atributos adicionais da norma e como os componentes estão ligados à sua placa, módulo, e sistema[12].

Os ficheiros HSDL servem assim três necessidades não contempladas no BSDL:[12]

- Descrição das interligações do barramento de teste (*IEEE 1149.*1) ao nível da placa ou módulo;
- Descrição de placas com arquitecturas dinâmicas e reconfiguráveis;
- Melhor depuração e verificação do *design* dos produtos, mais fácil e com menor risco de erros.

O HSDL suporta todas as funcionalidades do BSDL, mantendo a compatibilidade com este e com as ferramentas para geração automática de testes, validação, ou sintetização. O HSDL é estritamente um superconjunto do BSDL. Todas as declarações que fazem parte das entidades HSDL de um componente, mas que não são previstas no BSDL, são de carácter opcional, o que torna o BSDL aceitável num interpretador de HSDL. O HSDL pode ser aceite num interpretador de BSDL removendo-se todas as novas declarações, que causam erros de sintaxe[37]. As novas declarações foram adicionadas para descrever membros (podem ser módulos ou componentes); caminhos de varrimento dos módulos; e para simplificar o uso interactivo[12]. Todas as camadas acima do nível dos componentes (*i.e.* placas, MCMs, *backplanes*, sub-sistemas e sistemas) são identificadas como módulos. Para a descrição de um módulo, é criada uma nova entidade, à semelhança das entidades dos componentes nos ficheiros BSDL, mas a sua descrição difere substancialmente[12][37].

A descrição dos elementos do sistema completo é feita de forma hierárquica e define o posicionamento de cada membro neste. Mesmo que não se pretenda descrever todo o produto sob teste, para a geração de testes às suas interligações, o HSDL permite simplesmente descrever a cadeia de BS utilizada, permitindo conhecer a configuração adoptada na placa. A Figura 26 mostra exemplos de vários tipos de implementações de cadeias de BS:

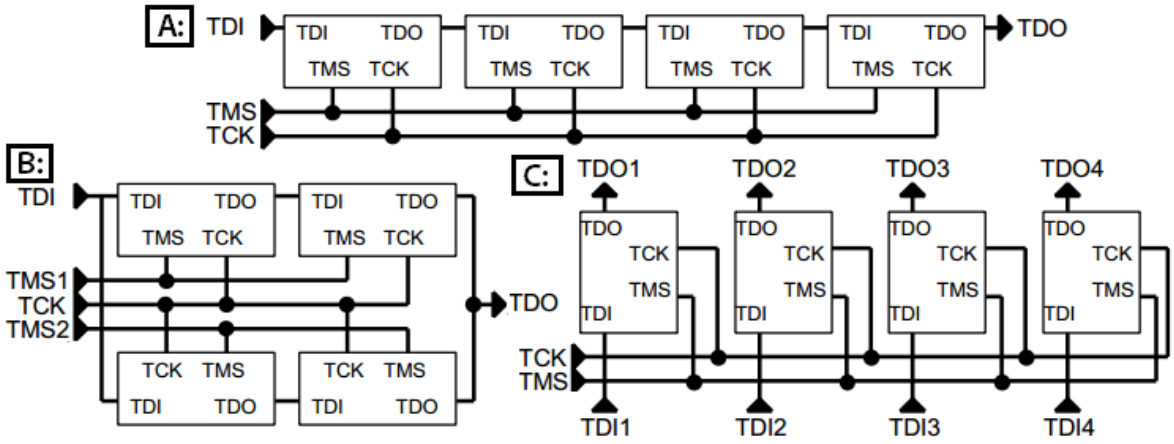


Figura 26 Várias configurações de uma cadeia de BS[37]

Uma distinção entre qualquer uma das implementações das cadeias de BS apresentadas, não é possível na linguagem BSDL, que apenas descreve cada componente. A linguagem HSDL, como se depreende, tem especial vantagem na descrição de placas mais complexas, facilitando a automatização da geração de testes para a placa e o controlo dos componentes por *Boundary Scan*.

2.11.6. PROCEDURAL DESCRIPTION LANGUAGE (PDL)

Recentemente definida no anexo C da revisão *IEEE 1149.1-2013*, é aplicada para documentar os procedimentos e requisitos de dados. Tem utilidade, por exemplo, para suportar algumas das novas instruções implementadas, que permitem a configuração de I/Os complexos antes de se entrar em modo de teste. Nestes casos, como os dados para a inicialização de cada componente podem variar com a sua aplicação, placa, ou sistema, existia a necessidade de uma nova linguagem para preenchimento dos registos TDR internos de cada componente para a configuração dos seus I/Os[27].

Esta linguagem é baseada na *Tool Command Language* (Tcl) e é uma extensão hierárquica ao BSDL. O PDL visa especialmente a redução dos custos industriais na preparação dos testes, facilitando a utilização de ATEs[38].

3. PLANEAMENTO DO CONTROLADOR BS

O desenvolvimento do sistema de controlo de infra-estruturas BS, seguindo intenções enquadradas no mercado actual de produtos para o efeito, está dependente da correcta aferição das alternativas de desenvolvimento possíveis, e a consciente opção pelas soluções mais favoráveis. Como tal, são referidos e analisados vários pacotes de *software* e *hardware* com objectivos similares a este projecto, e tecem-se considerações sobre as diversas abordagens encontradas para a implementação das componentes que compõem o sistema. Esta discussão, para além de expor as motivações que levaram ao sistema desenvolvido, serve também de indicação de outras alternativas que poderão ser úteis ao sugerirem formas de evitar problemas que venham a ser constatados no desenvolvimento e teste do protótipo funcional. Nesse sentido, não se reduz a análise às alternativas utilizadas, mas amplia-se o foco deste estudo prévio abordando-se também as tecnologias preteridas, quando a informação disponível o permita. Esta fase, por ser inicial e desencadeadora de todo o desenvolvimento, deverá ser já determinante para se atingir um sistema que venha a responder eficaz e eficientemente aos seus objectivos, pois todo o posterior trabalho assentará nas bases aqui traçadas.

Os pontos da eficácia e eficiência do sistema, embora perceptivelmente fundamentais nos intentos de um projecto satisfatório, têm, no entanto, a concorrência de outras razões que se fazem sentir muito nas opções tomadas, que visam a praticabilidade do sistema desenvolvido para o ambiente a que se destina. Nessa racionalização, destaca-se uma futura utilização em ambiente didáctico, contando com a possível utilização do sistema por parte de utilizadores inexperientes com o *IEEE 1149.1*. Para a utilidade do controlador desenvolvido, deverá assegurar-se que este possa ser construído sem dificuldades acrescidas, talvez com recurso a componentes facilmente obtíveis e de simples montagem, e que o custo da implementação do sistema não o torne inconveniente nem menos favorável que outros produtos. Espera-se que o resultado, para o qual se aponta nas opções tomadas, seja um sistema muito conveniente e com grande área de aplicação, ainda que possa não ter o melhor desempenho, que outras soluções técnicas, preteridas pelas razões citadas, lhe pudessem garantir.

3.1. SOLUÇÕES DE SOFTWARE DISPONÍVEIS

Nesta subsecção são seleccionados vários *softwares* com afinidade ao projecto a desenvolver, vocacionados para o teste e depuração de componentes ou circuitos, ou apenas como ferramenta didáctica. Todos eles, a não ser que enunciado o contrário, partilham o facto de serem disponibilizados gratuitamente. O *hardware* ao qual estão associados, para o controlo de BS, é referido e analisado se o for oportuno. Esta análise às várias soluções de *software* visa reunir as funcionalidades actualmente expectáveis neste tipo de produto de controlo de BS.

3.1.1. JTAGER

O *software JTager* (não confundir com "*JTAGer*", *software* nacional apresentado na secção *3.1.2*) é um sistema básico com o objectivo de possibilitar, gratuitamente, o teste e depuração por BS de microprocessadores *ARM*. Este corre em sistemas *Linux* e possibilita a comunicação com um único processador alvo, do tipo *ARM7TDMI* ou *ARM9TDMI*, para a execução de algumas tarefas simples de depuração: Ler ou escrever nos registos do processador ou de componentes externos da placa alvo; e ler ou escrever na memória RAM do processador ou memórias *FLASH*, do tipo *SST39LF* ou *MBM29LV650*[39]. O suporte de componentes é admitidamente limitado, cingindo-se aos quatro dispositivos referidos.

A interface do *JTager* é operada em linha de comandos. Analisando o conjunto limitado de comandos à disposição do utilizador percebe-se a sua grande orientação para o controlo da informação em memória, funcionalidade frequentemente desprezada noutros *softwares*. Resumidamente, os comandos disponíveis que visam o controlo JTAG são os seguintes:[39]

- *idcode* – Lê o registo de identificação do processador alvo;
- *halt* / *restart* – Interrompe / recomeça o funcionamento do processador alvo e entra / sai do modo de depuração;
- *reg* / *ice* – Lê ou escreve os registos do processador alvo;
- *memcpy* / *memcmp* – Copia / compara blocos de memória RAM;
- *memdump* / *memset* – Retorna / altera o conteúdo de um bloco de memória RAM;
- *memcsum* – Calcula o valor *MD5*[7] de um bloco de memória RAM;
- *memtest* – Testa se é possível ler e escrever a totalidade da memória RAM;
- *flprobe* – Analisa as memórias *FLASH* num dado endereço;
- *flerase* / *flwrite* – Apaga / escreve a memória *FLASH*.

Este *software*, tal como mais alguns dos referidos, não recorre a um verdadeiro controlador BS no sentido em que necessita apenas de uma interface que faça a condução dos sinais utilizados, fazendo a ligação entre o computador e a interface JTAG da placa alvo, sem qualquer processamento associado. A interface necessária para a operação deste *software* é o sistema "*OCDemon Macraigor Wiggler*", sendo recomendada a sua alternativa compatível de baixo custo de montagem, esquematizada na secção *3.2.1*.

3.1.2. JTAGER / TAPPER

O programa *JTAGer*, desenvolvido em *2002*, é uma actualização da anterior aplicação *Tapper*, que data do ano de *1994*, e com a qual partilha o conjunto de instruções de controlo JTAG. Para plataformas *Windows*, o *JTAGer* serve de interface para o controlo de

[7] MD5 é um código hexadecimal de *32* dígitos utilizado para verificação da integridade de dados.

até duas cadeias de BS. Esta aplicação implementa uma linguagem própria utilizada para as operações de BS[40]. Na Figura 27 apresenta-se o aspecto gráfico da aplicação:

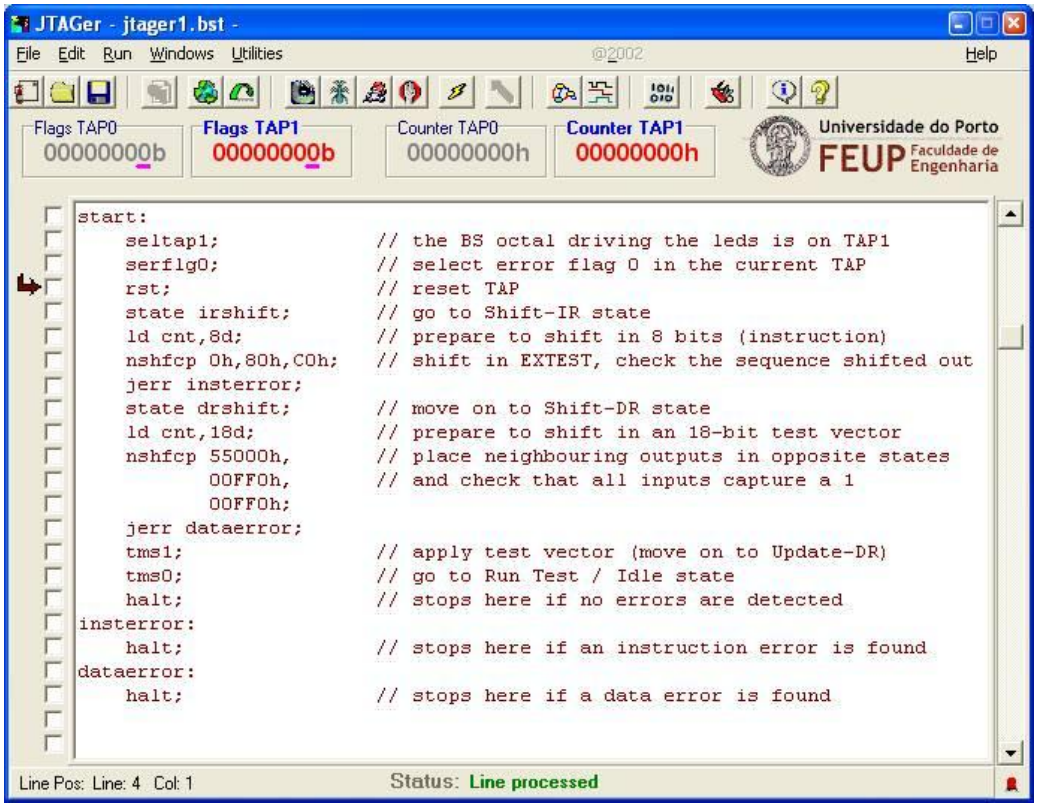


Figura 27 *Software JTAGer* e comandos para controlo do output do *IC 74BCT8244*[40]

Os comandos apresentados como exemplo servem para a atribuição de um *output* específico a um dispositivo BS (*74BCT8244*) ligado a vários *light-emitting diodes* (LED), tal como esquematizado na Figura 28:

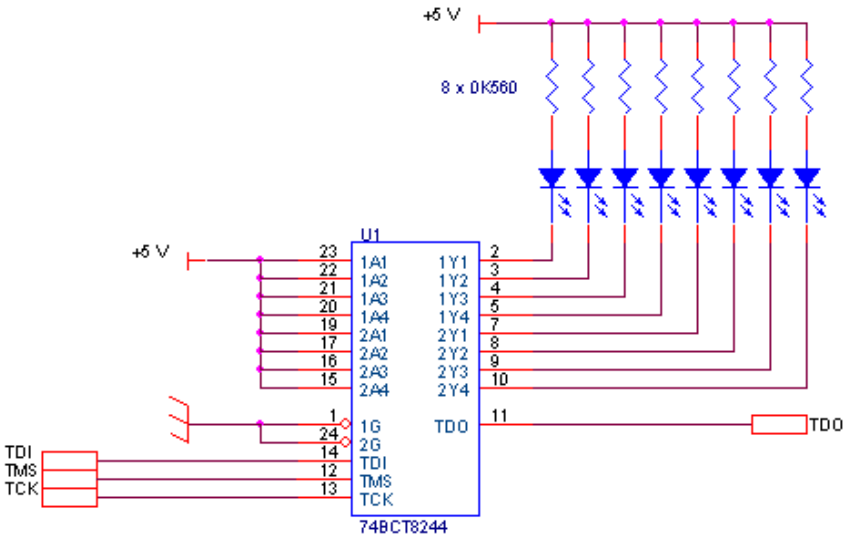


Figura 28 Circuito de teste baseado no IC *74BCT8244*[40]

Percebe-se que a implementação de uma linguagem de controlo JTAG própria causa dificuldades na operação deste programa, pelo menos inicialmente, comparativamente a *softwares* que utilizem linguagens *standard* já conhecidas, que são descritas na anterior secção 2.11. O controlo do estado dos LEDs que se exemplifica, e que deve ser considerada uma utilização de grande simplicidade, poderia ser efectuado com muito maior facilidade, de forma acessível até para um utilizador desconhecedor da norma *IEEE 1149.1* e qualquer das linguagens utilizadas no seu controlo, por exemplo na aplicação apresentada em seguida (*JTAGTest*, secção 3.1.3), fazendo-se valer da ajuda de uma interface gráfica capaz e intuitiva. É claro que este programa oferece uma alternativa que possibilita um maior controlo do que aquela a que se compara, mas a utilização de código SVF (possível em outras aplicações), ou outra linguagem descritiva dos procedimentos BS, oferecerá igual ou superior customização do comportamento pretendido. Este problema é assumido, ao considerarem que o conjunto de instruções tem um baixo nível de abstracção, o que o torna difícil de ser escrito e propenso a erros, se feito manualmente. Para tentar colmatar o descrito, existe um assistente que permite o controlo simplificado de algumas funções básicas por BS, apresentado na Figura 29:

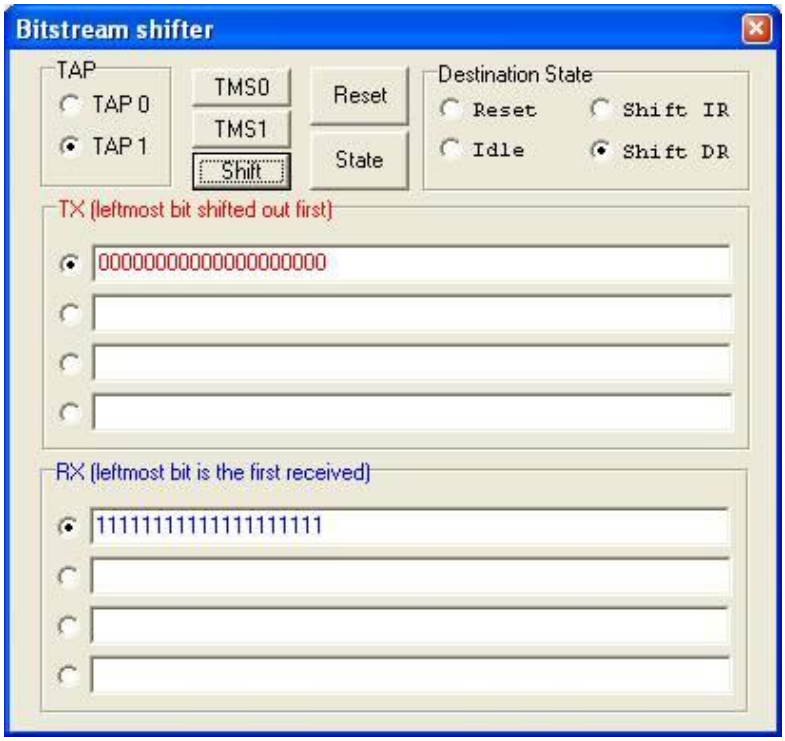


Figura 29 Ferramenta de envio de vectores BS (*JTAGer*)[40]

Nesta janela consegue-se escolher o estado estável dos controladores TAP alvo da cadeia de BS pretendida, e posterior introdução de vectores de dados, agora em formato binário

(estes devem estar em formato hexadecimal quando introduzidos como argumento de algum comando). São fornecidas também algumas outras ferramentas complementares. Na Figura 31, mostra-se, em cima, uma janela complementar à anterior, onde são apresentados os bits recebidos a cada ciclo de TCK em cada uma das cadeias BS:

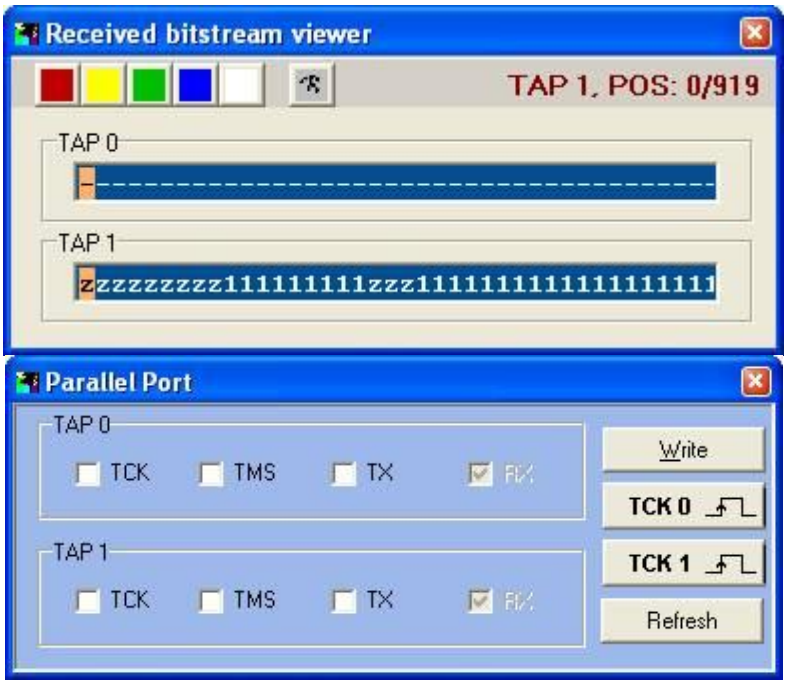


Figura 30 Ferramenta de registo dos *bits* recebidos (em cima) e de controlo directo do controlador TAP (em baixo) (*JTAGer*)[40]

Em baixo, surge a janela para controlo directo das linhas de cada um dos conectores JTAG. Na seguinte Figura 31, mostram-se outras duas ferramentas para, respectivamente, visualizar o estado TAP actual dentro do diagrama da máquina de estados definida na norma do BS, e a evolução gráfica de cada uma das linhas dos conectores JTAG:

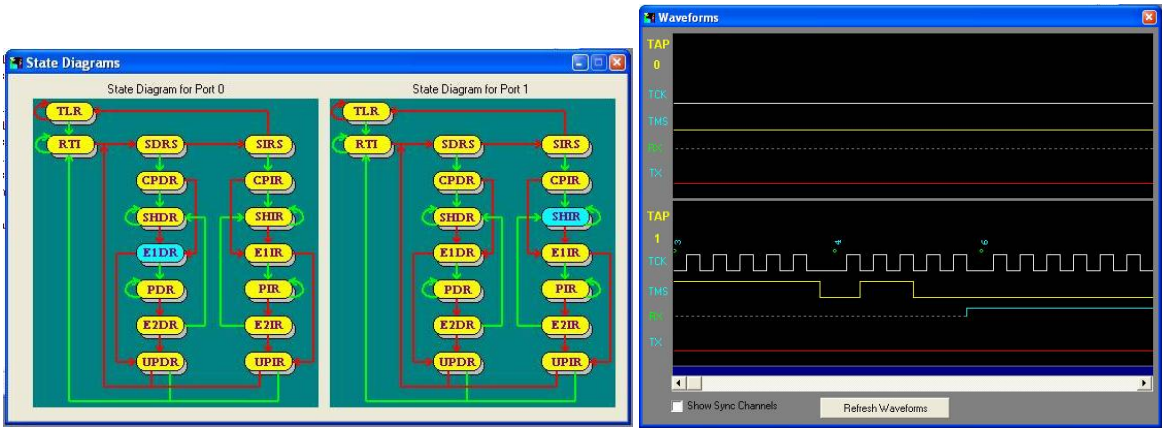


Figura 31 Ferramentas de visualização do estado TAP e evolução de sinais JTAG (*JTAGer*)[40]

Existem também janelas para o registo de cada evento e do valor lógico de cada pino da porta paralela, que é utilizada para o controlo JTAG. Um esquemático da interface JTAG necessária para a comunicação com a placa alvo não é fornecido, mas o sistema, que é o mesmo da aplicação *Tapper*, carece apenas da ligação de cada uma das linhas da porta paralela do computador à respectiva linha JTAG predefinida na aplicação (Figura 32), contando com a possível necessidade de utilização de um *driver/buffer* dos sinais (as portas paralelas funcionam a níveis TTL − *Transistor–transistor logic* −, no entanto, a corrente que permitem pode variar)[40][41].

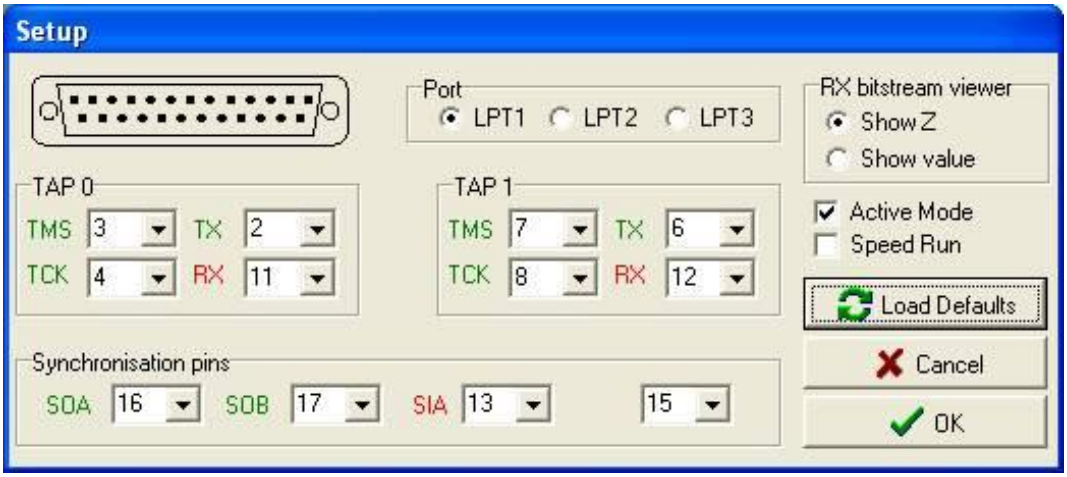


Figura 32 Configuração dos pinos da porta paralela *DB25* (*JTAGer*)[40]

3.1.3. SECONS JTAGTEST

O *software JTAGTest* é disponibilizado comercialmente em conjunto com a interface "*Via TAP JTAG-USB*", que constituem uma solução de custo acessível para o teste e apoio ao desenvolvimento de circuitos eléctricos utilizando o BS. Este par oferece funcionalidades de detecção automática das cadeias de BS e outras ferramentas de depuração que permitem verificar e controlar o estado de cada pino dos dispositivos alvo, podendo a sua evolução ser gravada simulando a utilização de um analisador lógico. É garantido o suporte completo das linguagens BSDL, HSDL, e SVF[42].

O aspecto da interface gráfica da aplicação, já após a detecção de uma cadeia de BS, é apresentado na Figura 33:

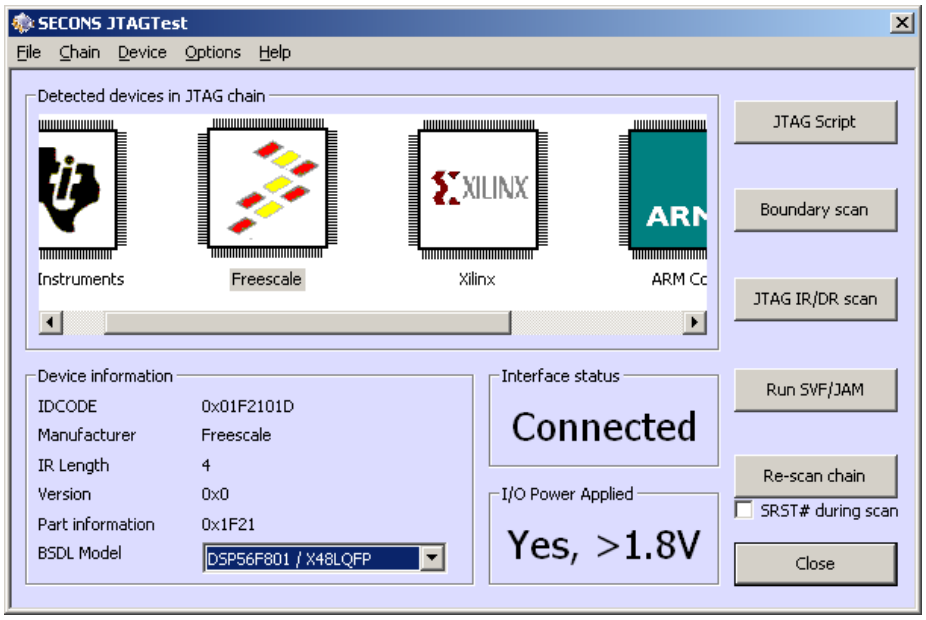


Figura 33 Aspecto da interface gráfica após detecção de dispositivos (*JTAGTest*)[42]

A detecção dos dispositivos da cadeia de BS faz a listagem de cada componente e seu fabricante, sugerindo, para cada um, o ficheiro BSDL mais indicado e os vários encapsulamentos físicos disponíveis. Nesta janela surgem as opções para execução de linguagens de teste (SVF e STAPL), assim como *scripts* próprios. Escolhendo um dispositivo, é possível aceder a outros controlos. Mostra-se, na Figura 34, o assistente para controlo dos pinos de cada dispositivo e restantes células do registo BSR:

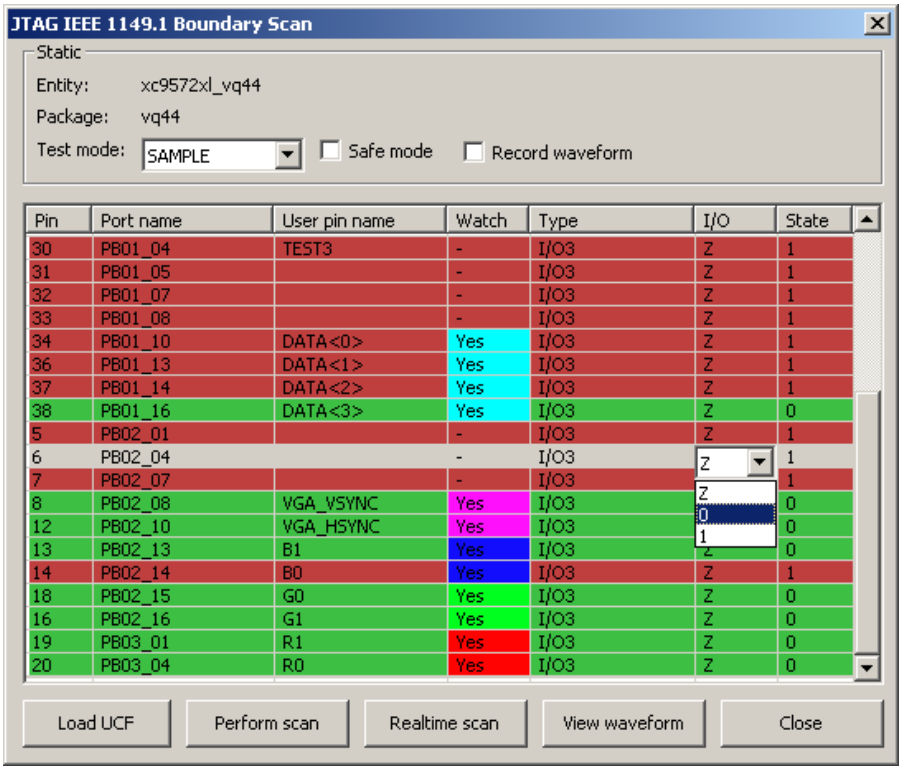


Figura 34 Ferramenta de edição dos registos BS do dispositivo alvo (*JTAGTest*)[42]

O valor de cada célula do registo BSR pode ser editado independentemente, não sendo necessário introduzir manualmente todo o vector de teste. A leitura das células é disposta na mesma organização, já associando cada *bit* recebido com a sua célula respectiva. A informação sobre a associação entre as células BSR e pinos do dispositivo é obtida no ficheiro BSDL do componente. Os valores de cada célula podem ser monitorizados de forma contínua e apresentados segundo uma representação gráfica, como na Figura 35:

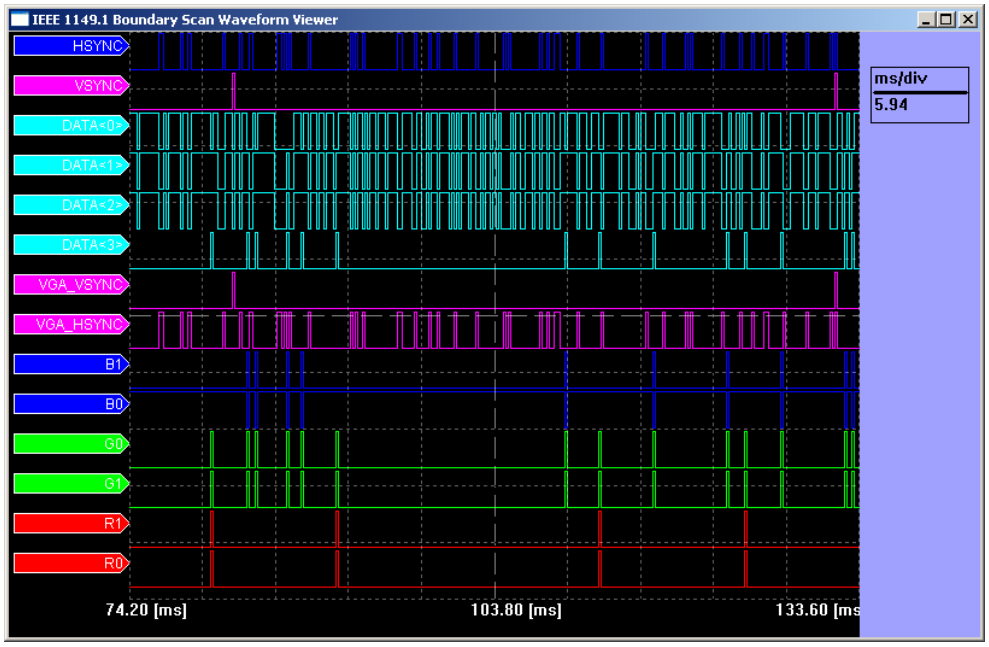


Figura 35 Ferramenta de visualização gráfica da evolução das células BSR (*JTAGTest)*[42]

O valor de cada pino do dispositivo pode também ser visualizado na representação do componente, segundo o encapsulamento seleccionado, como exemplificado na Figura 36:

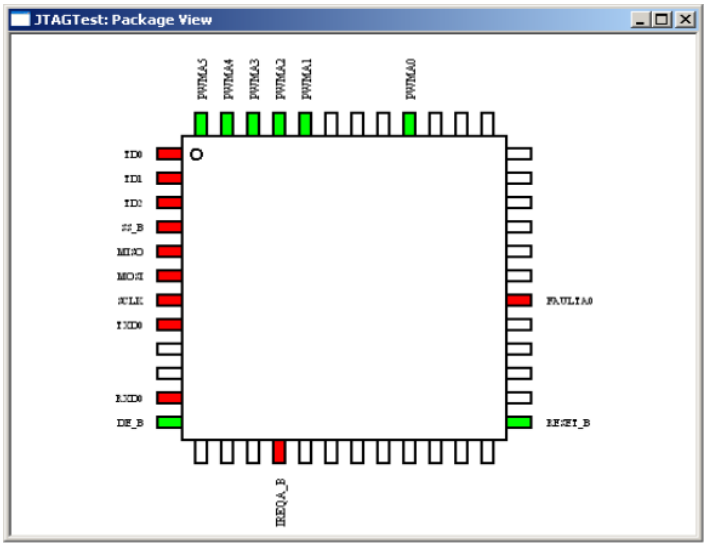


Figura 36 Ferramenta de visualização gráfica dos estados dos pinos (*JTAGTest*)[42]

Pretendendo-se efectuar outro tipo de operação, que não seja a edição ou monitorização do registo BSR, através dos modos *EXTEST*, *SAMPLE*, ou *PRELOAD*, do controlador TAP alvo, é necessário recorrer à ferramenta de controlo manual, onde o utilizador poderá efectuar as operações desejadas, sem qualquer assistência por parte da aplicação. Esta ferramenta é apresentada na Figura 37:

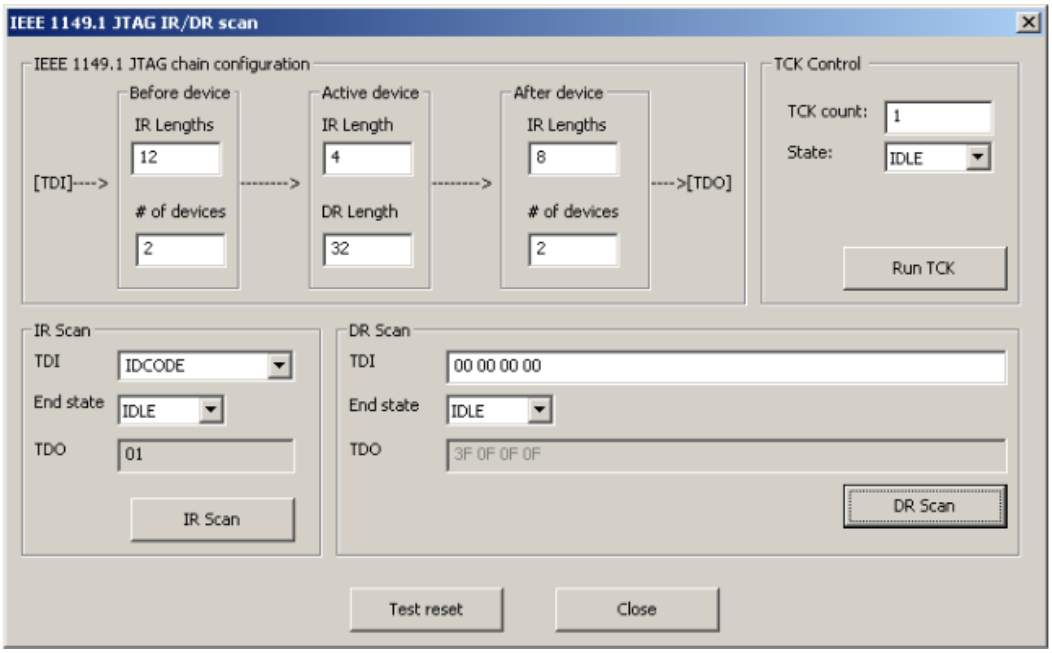


Figura 37 Ferramenta de varrimento de IR e TDR (*JTAGTest*)[42]

De notar que a selecção do dispositivo alvo já terá de ser feita manualmente, introduzindo os dados necessários sobre a cadeia BS. Os controlos disponíveis permitem: Alterar o estado dos controladores TAP alvos; Introduzir instruções e capturar o registo IR; e introduzir e capturar vectores do registo TDR.

A interface JTAG utilizada (Figura 38) recorre a uma ligação *Universal Serial Bus* (USB), e disponibiliza oito diferentes conectores para *flat cables*, conferindo-lhe compatibilidade, total ou parcial, com um grande número de conectores JTAG padronizados:[43]

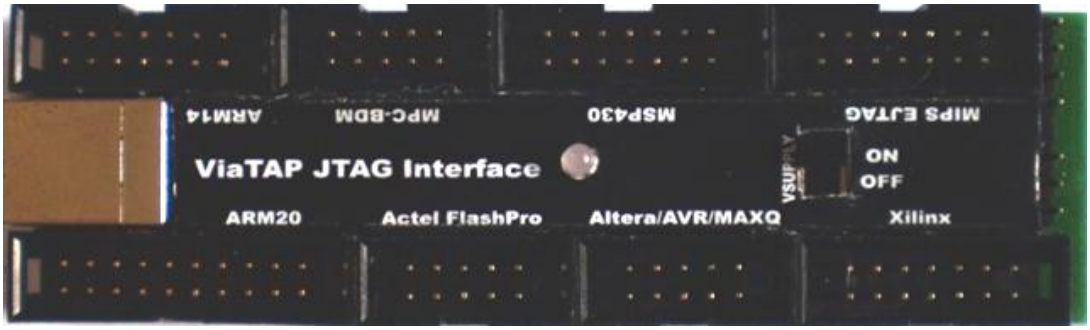


Figura 38 Interface *ViaTAP JTAG*[43]

Mais detalhes sobre o *hardware* utilizado na interface são desconhecidos. O valor comercial da licença do *software* e interface *ViaTAP* é de 90€[8].

3.1.4. OPENWINCE JTAG TOOLS / URJTAG

O projecto *Openwince* disponibiliza livremente várias ferramentas; módulos; aplicações; *drivers*; entre outros, para plataformas *Windows CE*, embora alguns destes possam ser utilizados noutras plataformas. O seu pacote de *software JTAG Tools* é uma solução que permite trabalhar com sistemas físicos que incorporem a norma *IEEE 1149.1* através de um adaptador JTAG. A sua arquitectura é aberta e modular, e oferece funcionalidades para implementações tanto de depuradores de placas como para programação de memórias *FLASH*[44]. A última versão deste *software* (*ver. 0.5.1*) data ainda do ano de *2003*, mas este veio mais tarde a ver o seu desenvolvimento prosseguido num novo projecto, com o nome de *UrJTAG*[33]. Esta aplicação, também gratuita e de código aberto, é compatível com sistemas *Windows* e *Linux*, disponibilizando suporte para linguagem SVF; STAPL; e BSDL.

Este *software*, que funciona em linha de comandos, oferece também a possibilidade de detecção automática dos elementos da cadeia BS. Esta operação é exibida na Figura 39:

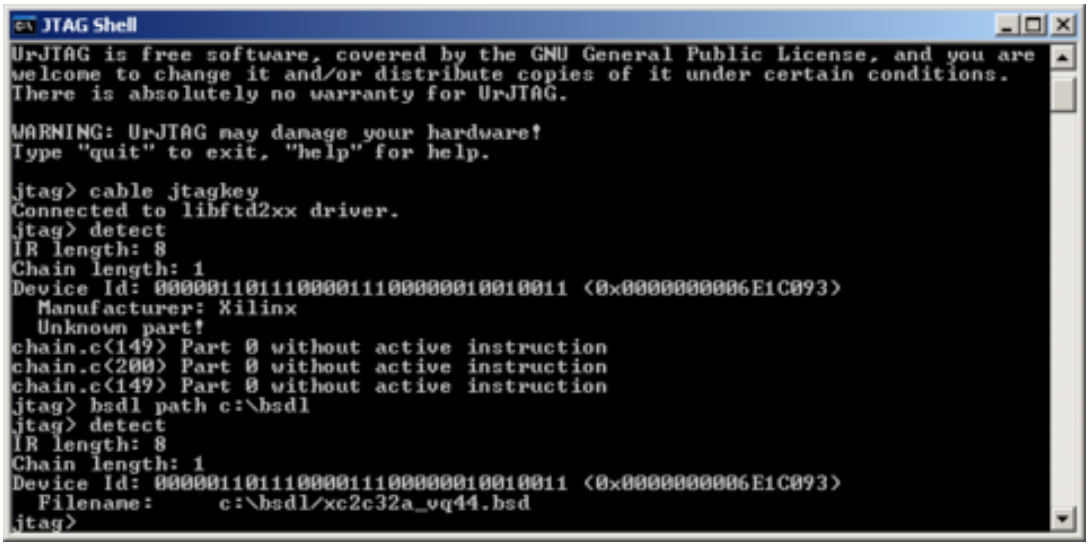


Figura 39 Detecção automática dos componentes da cadeia BS (*UrJTAG*)

[8] Valor indicado pelo fabricante, à data de 8 de Fevereiro de 2014, não incluindo custos de envio.

Outras funções disponíveis, já referidas em outras ferramentas, são a selecção de componentes na cadeia BS e edição do seu registo BSR, ou qualquer outro através da introdução manual dos vectores necessários, e a programação de memórias *FLASH*.

Uma das vantagens deste *software* é a sua compatibilidade com um grande número de interfaces JTAG, demasiado extensa para ser apresentada.

3.1.5. JTAG SCAN EDUCATOR / BOUNDARY SCAN COACH

Embora algo afastado do intuito do actual projecto, destaca-se ainda assim o simulador *JTAG Scan Educator Ver.2*, da *Texas Instruments*, devido à sua incidência sobre a vertente didáctica, desenvolvido para ser executado em ambiente *Microsoft Disk Operating System* (MS-DOS). Este tem como objectivo apenas a introdução do utilizador aos fundamentos da norma *IEEE 1149.1*, incluindo a sua arquitectura, método de controlo e o conjunto de instruções necessárias na sua aplicação, e fá-lo através de várias simulações de teste acompanhadas de bastante informação relacionada, guiando o utilizador na operação do controlador TAP através da manipulação das entradas da porta TAP. Este contempla situações com um ou vários dispositivos, onde se pretende analisar a observabilidade e controlabilidade do sistema[45]. Apesar de obviamente datado, este *software* mantém-se ainda assim uma das poucas alternativas para o que oferece, embora não o faça da forma mais intuitiva, como seria esperado de um *software* mais actual. A Figura 40 mostra um modelo interactivo básico:

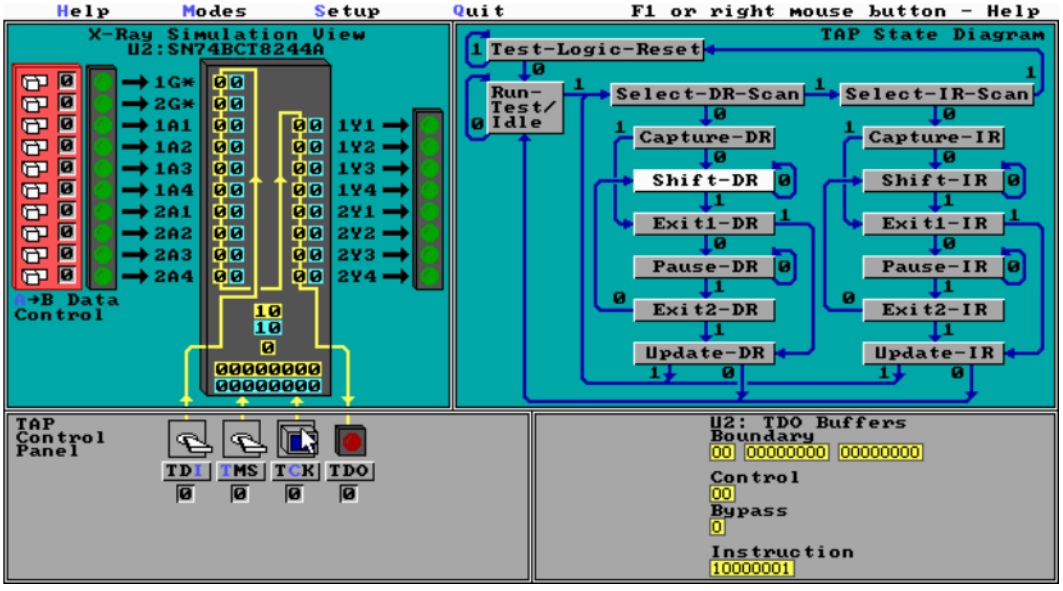


Figura 40 Aspecto gráfico da aplicação *JTAG Scan Educator Ver.2*

Outra desvantagem é o facto de existirem problemas de compatibilidade com sistemas operativos de *64 bits*, que impede a sua utilização em muitos casos.

O *software Boundary Scan Coach* é outra alternativa similar, mantida pelo fabricante *Goepel Electronics*, mas mais completa e actual, que permite a realização de tarefas tutoriais, desta vez, de forma interactiva recorrendo a uma placa de demonstração. Esta placa é de utilização é obrigatória, pelo que a aplicação não pode ser utilizada de forma independente, como acontecia com o *JTAG Scan Educator*[46].

A Figura 41 mostra a sua interface gráfica, que sobrepõe as tarefas tutoriais sob diagramas típicos do funcionamento da norma:

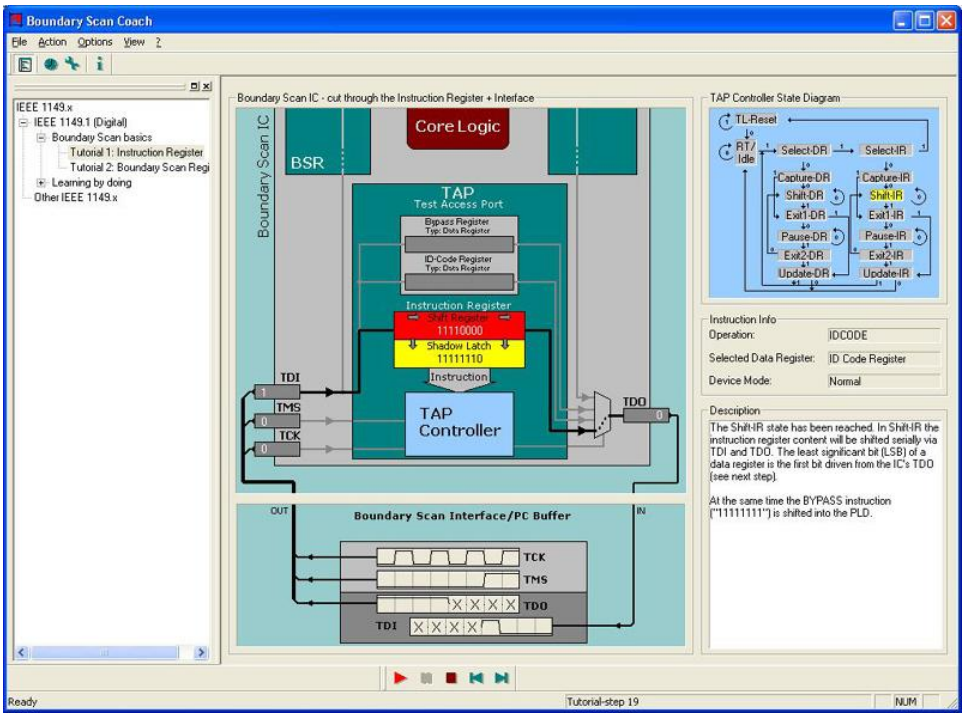


Figura 41 Interface gráfica do *software Boundary Scan Coach*

A placa de demonstração utilizada, na qual é possível verificar o resultado das tarefas conduzidas, é apresentada na Figura 42:

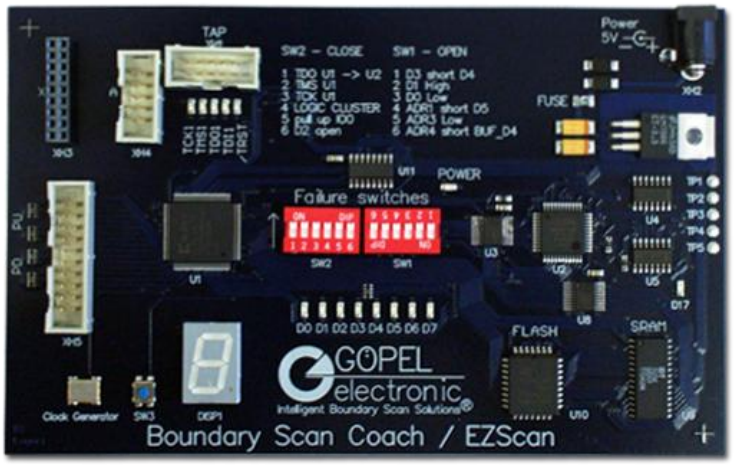


Figura 42 Placa de demonstração *Boundary Scan Coach / EZScan*[46]

Esta placa dispõe de dois *Programmable logic devices* (PLD); uma memória *Static random-access memory* (SRAM); uma memória *FLASH* EEPROM, um *buffer*, e outros componentes variados para experimentação das aplicações do *IEEE 1149.1*. Para o controlo por BS da placa são suportados todos os controladores JTAG do fabricante *Goepel*, assim como as interfaces de porta paralela para programação JTAG dos fabricantes *Altera*, *Lattice*, e *Xilinx*[46].

3.2. SOLUÇÕES DE HARDWARE DISPONÍVEIS

Qualquer *hardware* vocacionado para o controlo de uma interface BS tem apenas a obrigatoriedade de contemplar funções de baixo nível, que visam geralmente a actuação dos sinais de cada linha JTAG (*TDI*; *TMS*; e *TCK*), no controlador TAP alvo, para o acesso a registos BS; introdução de vectores de dados; captura e recepção de registos; *etc.* As funcionalidades de cada sistema estão especialmente dependentes do seu *software*, *i.e.* a finalidade para a qual utilizam a interface BS, já que qualquer que esta seja (*e.g.* programação; apoio à depuração funcional; teste estrutural; *etc.*) a actuação das linhas dos controladores TAP é similar. Nesse sentido, as funcionalidades esperadas de cada uma das alternativas visadas depende do seu emparelhamento com algum *software*, possivelmente já analisado na secção anterior (secção 3.1). Naturalmente, a camada de alto nível do *software* de controlo deverá poder ser implementado, pelo menos parcialmente, no próprio *hardware* dos controladores, mas tal não se verifica nas soluções livres, cuja análise do sistema é mais facilitada; e não é possível auferir com certeza nas soluções comerciais. Algumas das soluções de *hardware* analisadas nem tampouco efectuam qualquer

processamento, destinando-se apenas à condução dos sinais JTAG (*e.g. Wiggler JTAG*, secção 3.2.1).

As características fundamentais que distinguem as várias interfaces de controlo JTAG são:

- Interface física com o computador (*i.e.* por porta Paralela; USB; ou *Ethernet*);
- Interface física com a placa alvo: Tipo do conector JTAG, quantidade de cadeias BS simultâneas suportadas, e gama de tensões compatíveis;
- Velocidade de operação JTAG;
- Variações à implementação do *IEEE 1149.1* (*e.g.* inclusão do pino *RTCK* específico para actuação sobre componentes *ARM*);
- Disponibilização adicional de outros *standards* de depuração (*e.g.* o *Serial Wire Debug* – SWD, utilizado com componentes *ARM*);
- Disponibilização adicional de outros tipos de comunicação com a placa alvo (*e.g.* SPI ou I2C);
- Suporte para *software* e *hardware* específico (*e.g.* a operação do sistema de OCD integrado nos componentes *Atmel ATmega*, através de JTAG, utilizando o *software* do fabricante – abordado na secção 3.2.2);
- Outros recursos adicionais (*e.g.* GPIOs; ADCs; *etc.*).

Destes factores, os últimos cinco não têm relevância para a investigação conduzida, já que não se visa implementar qualquer uma dessas situações.

O conjunto de soluções de *hardware* seleccionadas e analisadas inclui controladores comerciais, alternativas compatíveis que estejam disponíveis livremente, e componentes singulares utilizados para a criação de interfaces de controlo JTAG. Foi analisado um produto representativo para cada forma de implementação do *hardware* encontrada.

A análise destes produtos pretende fornecer indicações acerca das alternativas de construção do controlador de BS, incluindo os componentes a que se poderá recorrer e as características técnicas actualmente esperadas neste tipo de sistema, não das suas funcionalidades. Os componentes destacados para a implementação do controlo JTAG

constituem alternativas reais para a implementação no corrente projecto (secção 3.2.5 e 3.2.6).

Todos os valores comerciais apresentados são indicados pelo respectivo fabricante, à data de 15 de Maio de *2014*, e não contemplam custos de envio e taxas de importação (como referência: *1$ = 0,73€*).

3.2.1. MACRAIGOR WIGGLER JTAG (INTERFACES PARALELAS SEM PROCESSAMENTO)

De acordo com o sistema oficial do fabricante *Macraigor Systems*, o seu adaptador ("*OCDemon Macraigor Wiggler"*, apresentado na Figura 43) é uma interface de baixo custo e aplicação generalizada, utilizada para *design*, depuração e programação de microprocessadores. Este deverá efectuar a ligação entre uma porta paralela de um computador com a porta de *On-Chip Debug* (OCD) de um sistema alvo (que pode ser: JTAG; EJTAG; ICE; *Common On-Chip Processor debug port* − COP; e *Background debug mode* − BDM). É indicado que este deverá suportar velocidades de transferência entre os *3* e os *35 KB/s*, e frequências de operação BS entre *60* e *380 KHz*[47].

Figura 43 Interface JTAG *OCDemon Macraigor Wiggler*

A lista de dispositivos alvo suportados é extensa. Embora seja descrito como um dispositivo de baixo custo e muito simples, que funciona apenas como um *buffer* entre os sinais da porta paralela do computador e o dispositivo alvo, o seu valor comercial recomendado é de *150$*[47].

Um circuito compatível, que se supõe semelhante ao sistema oficial, é apresentado na Figura 44, que atesta a favor da sua simplicidade:[48]

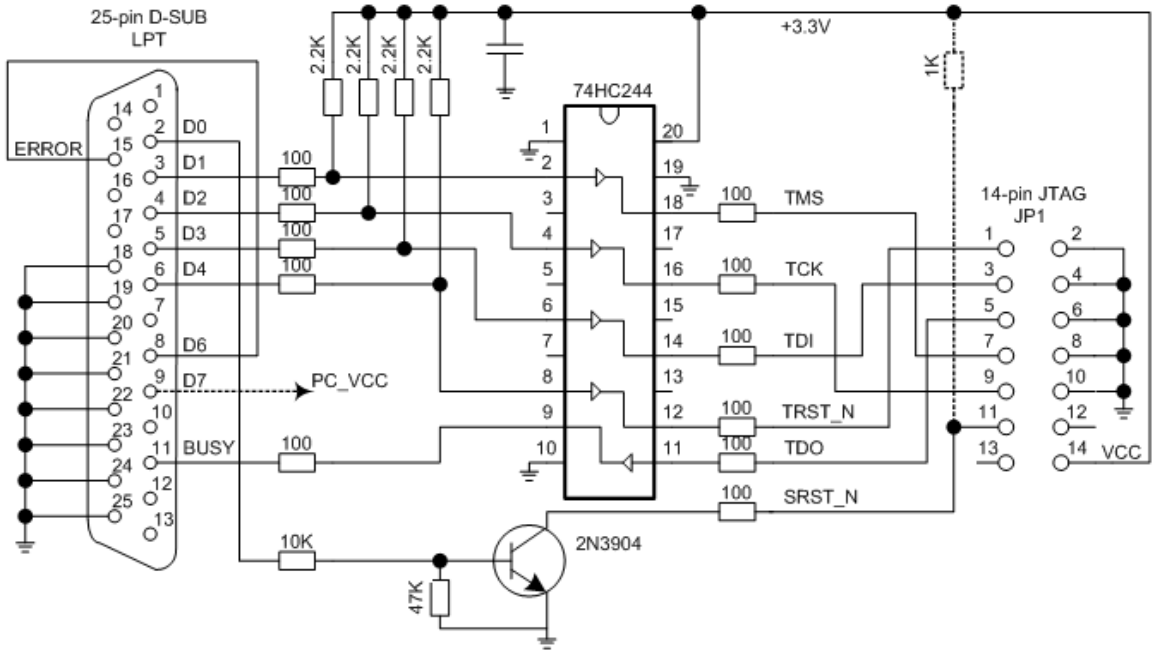


Figura 44 Interface JTAG compatível com o sistema *Macraigor Wiggler*[48]

Esta interface é recomendada, por exemplo, para utilização em conjunto com o *software JTager* (secção *3.1.1*). A sua utilização prevê todo o controlo a ser processado no computador, limitando-se a conduzir os sinais da porta paralela utilizada, directamente para o conector JTAG. A utilização do *line driver 74HC224* para accionamento das linhas da interface BS, e transístor *2N3904* para comandar a linha de *reset* do sistema alvo, apenas visa garantir o fornecimento de corrente necessário para a sua aplicação e a utilização de uma tensão de *3,3 Volt*, não tendo qualquer papel como lógica de controlo das operações de BS efectuadas.

Existem muitas outras interfaces com o mesmo tipo de implementação através de porta paralela. Um caso popular é o cabo *Xilinx DLC5*, cujo circuito também se encontra disponível livremente[49].

3.2.2. AVR JTAG ICE (INTERFACES BASEADAS EM MCU)

O *AVR JTAG ICE*, do fabricante *Atmel*, é um produto para controlo de interfaces BS vocacionado para aplicações de prototipagem e desenvolvimento.

A sua utilização com o *software AVR Studio* permite que seja utilizado para programar microcontroladores *AVR* compatíveis e, especialmente, para a posterior depuração do seu funcionamento através de *On-chip Debugging*. O produto original garantia ser suportado pelo *software AVR Studio*, mas actualmente a sua compatibilidade não está assegurada nas

versões mais recentes do *software*[50]. O *software AVR Studio*, por ser demasiado deslocado dos objectivos do sistema projectado, enquanto aplicação de controlo de interfaces JTAG, não foi analisado na secção respectiva (3.1). No entanto, é de fazer referência às potencialidades do OCD que oferece em parceria com o equipamento em análise, *AVR JTAG ICE*.

O *JTAG ICE* utiliza o *IEEE 1149.1* como interface com o sistema interno de OCD dos componentes *AVR*, e permite monitorizar e controlar a execução do programa no próprio componente. Este funcionamento difere assim de um sistema *In-Circuit Emulator* tradicional, que substitui o componente emulado controlando externamente os seus pinos. Em modo de funcionamento normal, a execução do programa é totalmente independente do computador, mas o controlador monitoriza continuamente o alvo à procura de condições de paragem (*break points*), definidas pelo utilizador. Quando acontecem, a execução do componente é parada e o seu sistema OCD lê todos os dados da execução do programa (*e.g.* contadores; registos; memórias; *etc.*) e transmite-os através da interface JTAG. Durante o período de paragem, o valor lógico dos pinos do componente são mantidos e não são interrompidas possíveis transmissões de dados que estejam a decorrer[50][51].

Este controlador recorre a uma comunicação com o computador através de uma porta série (*9-pin D-SUB*), como se esquematiza na Figura 45:

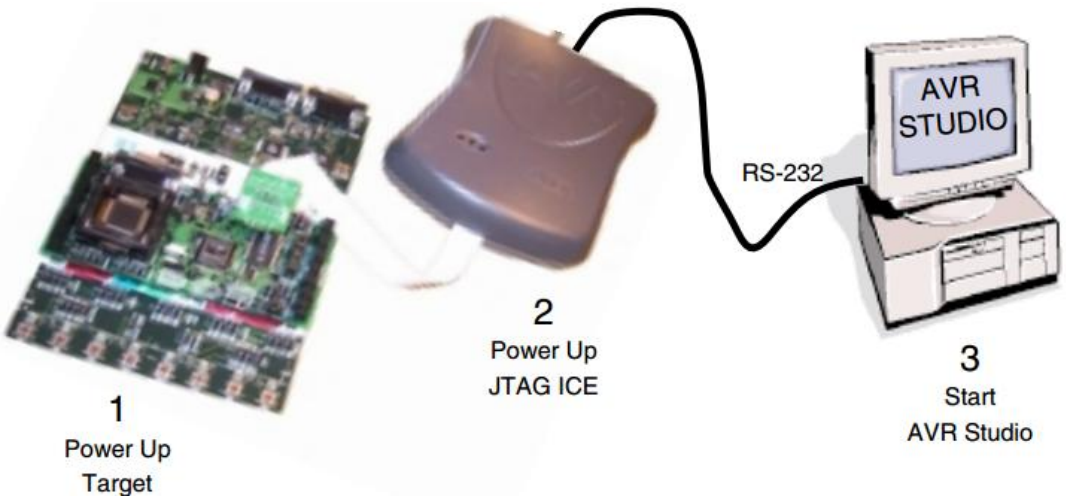


Figura 45 Ligação do controlador BS *AVR JTAG ICE*[50]

Não é possível a utilização de placas alvo com cadeias de BS de vários dispositivos. A cadeia à qual é ligado só deve conter um único microcontrolador *Atmel*. Os únicos dados

relativos ao desempenho deste controlador indicam que a comunicação série (*RS-232*) é efectuada a *115200 bits/s*, e a frequência de operação da placa alvo supera os *8 KHz*.

Agora descontinuado em função do lançamento de novas versões, o controlador foi temporariamente disponibilizado como *open source*. Desta forma, surgem *online* vários "clones" deste sistema ("*Aquaticus AVR JTAG*"; "*ISO JTAG ISP*"; *etc.*), totalmente compatíveis com as aplicações que suportavam o controlador original. Os "clones" disponíveis, embora baseados no *hardware* original, podem implementar algumas modificações. Na documentação original do fabricante *Atmel* era utilizado um MCU actualmente descontinuado. O *ATmega16* é um substituto totalmente compatível e opção obrigatória por razões de compatibilidade com o *firmware* disponível. Para além do MCU, os "clones" variam apenas na forma como implementam a comunicação com o computador, ou como gerem a alimentação do sistema e conector JTAG. No caso do controlador apresentado na Figura 46, optou-se por preterir a ligação série em favor da utilização de uma ligação USB através de um conversor Série-USB: [52]

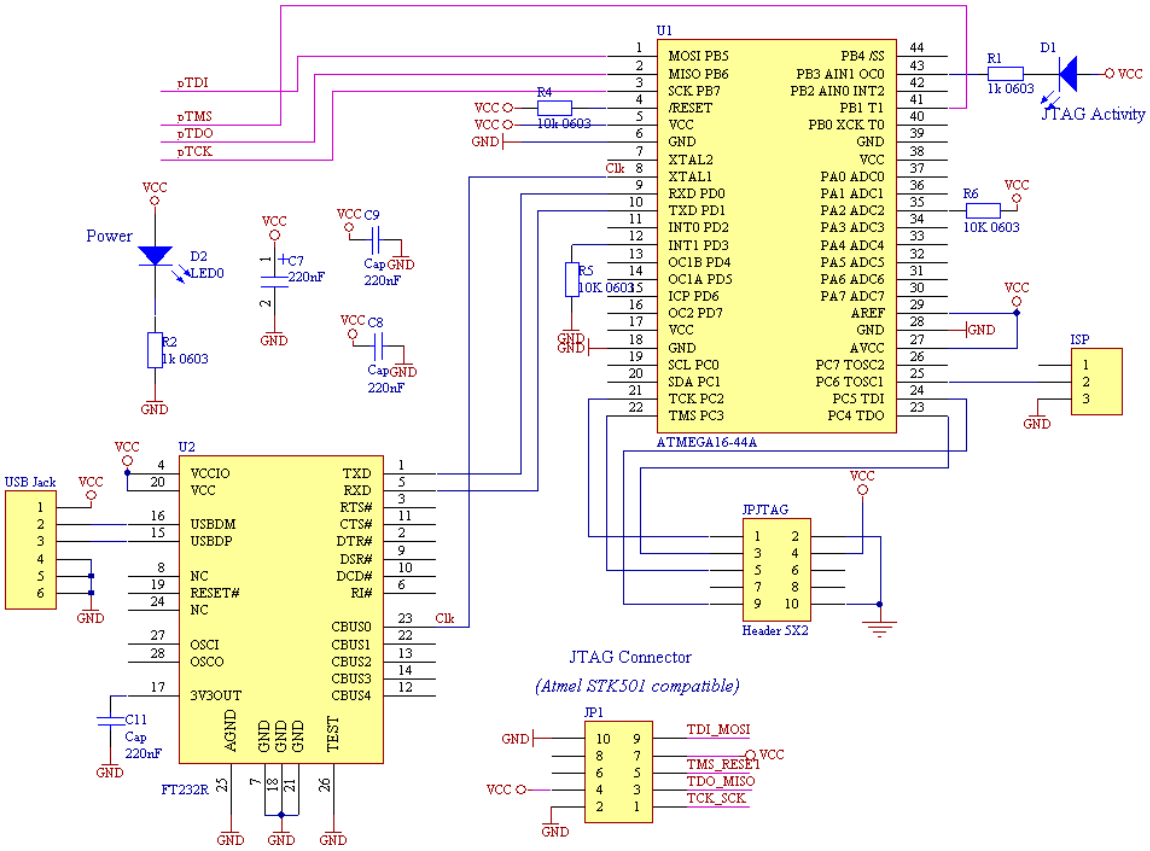


Figura 46 Esquemático do *ISO JTAG ISP* (derivado do *AVR JTAG ICE*)[52]

O conversor Série-USB utilizado neste caso, para substituir a ligação série original, foi o *FTDI FT232R*.

Os produtos actualmente comercializados pela *Atmel*, *JTAG ICE mkII* e *JTAG ICE 3*, não têm os seus circuitos disponíveis. Os seus custos comerciais são de *399$* e *99$*, respectivamente.

Outros exemplos de soluções baseadas num processamento similar são:[49]

- *Raisonance RLink Debugger Programmer* (*99€*): Este sistema está implementado, numa versão simplificada, no conector de depuração da conhecida placa de desenvolvimento *STM32 Primer*, onde foi utilizado um MCU *ST7* de *8 bits*.
- *Atmel SAM-ICE* (*100$*) e *Segger J-Link* original: São sistemas compatíveis, e utilizam o núcleo *ARM Atmel AT91SAM7(S)64*. Novas versões do *J-Link* prevêem funcionalidades adicionais como acesso através de cabo *Ethernet*.

3.2.3. GOEPEL PICOTAP (INTERFACES BASEADAS NO *FT2232X*)

O *PicoTAP*, da *Goepel*, é um controlador de interface USB com aplicação no desenvolvimento; produção; e assistência técnica, no diagnóstico de falhas e actualização do *firmware* dos produtos. Este permite operar uma cadeia de BS, com valores de tensão de *3,3 V*, a uma frequência máxima de *10Mhz*[53]. A Figura 47 mostra as suas dimensões reduzidas:

Figura 47 Controlador BS *Goepel PicoTAP*[53]

O esquemático desta solução está disponível livremente, juntamente com os respectivos *drivers*, em [54]. O *PicoTAP* utiliza o *FTDI FT2232HL* como gestor da norma JTAG e da comunicação USB, que é abordado na secção 3.2.6.

Pode ser utilizado com todos os *softwares* fornecidos pela *Goepel*, cuja gama cobre todas as áreas de aplicação do BS (*softwares Cascon Galaxy*; *Cascon Polaris*; e *TAP Checker*)[53].

Outros controladores populares baseados num núcleo *FT2232x* são, por exemplo, o *Olimex ARM-USB-TINY* (*39,95€*) e o *Amontec JTAGkey* (*129€*), ambos compatíveis com o *software UrJTAG* (secção 3.1.4)[33][49].

3.2.4. ALTERA USB BLASTER (INTERFACES BASEADAS EM CPLD)

Os produtos *USB Blaster* são uma alternativa para transferir dados de configuração entre o computador e dispositivos FPGA do fabricante *Altera*, através de BS.

A forma como estes efectuam as operações JTAG diferencia-se das outras opções apresentadas por utilizarem um CPLD como núcleo de processamento. Associado a este, têm acoplado uma interface USB desconhecida, que lhes permite a comunicação com o computador[55]. Na Figura 48 mostra-se a representação, segundo diagrama, do circuito eléctrico do *USB Blaster*:

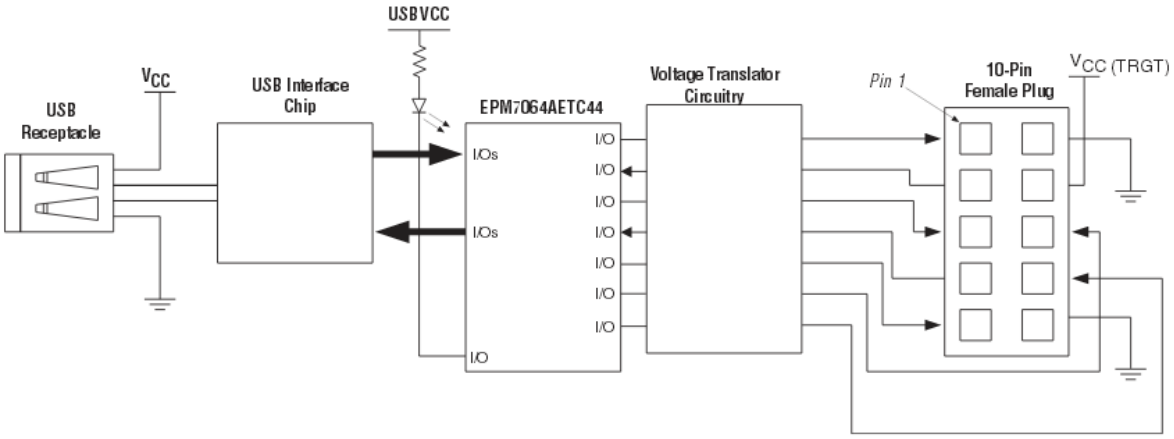


Figura 48 Representação do *hardware* do controlador *USB Blaster*[55]

Para além do referido CPLD e interface USB, é utilizado também um circuito para a adequação da alimentação dos pinos do conector JTAG. O CPLD utilizado é da família *Altera MAX7000*. O valor comercial desta solução é de *300$*, e pode ser utilizada para outros fins, além da programação de FPGAs, em conjunto com o *software UrJTAG* (secção 3.1.4).

É possível recriar este controlador, segundo o projecto *ixo.de USB JTAG Pod*[56], utilizando o CPLD *EPM7064* (*MAX7000*), com um oscilador externo de *24MHz*, associado

a uma interface USB *FT245R*. Este sistema, contando com a programação do *firmware* disponibilizado, pode ser utilizado com a mesma selecção de *software* suportada pelo *USB Blaster*[56]. O *USB Blaster II*, versão actualizada do controlador analisado, mantém uma arquitectura muito similar, mas utiliza um CPLD da família *Altera MAX II*.

3.2.5. TI EMBEDDED TEST-BUS CONTROLLER (ETBC)

O fabricante *Texas Instruments* disponibiliza uma gama de controladores integrados de teste (eTBC – *Embedded Test-Bus Controllers*), a qual incluí, por exemplo, o componente *SN74LVT8980A*. Esta família de dispositivos suporta a norma BS e visa facilitar o teste de circuitos, mas, não sendo, eles próprios, dispositivos controláveis por BS, servem para o controlo de portas TAP sob o comando de um microprocessador ou microcontrolador. Os comandos de alto nível e os respectivos dados são passados paralelamente para o eTBC através de uma interface genérica, composta por um barramento de dados de oito *bits* (*D7-D0*) e três *bits* de barramento de endereçamento. A captura destes dados é dependente do relógio de entrada (*CLKIN*), e o sinal *R/W* deve escolher se a tarefa é de escrita ou leitura. Através de *STRB* indica-se o intuito de iniciar ou concluir a tarefa. O sinal *RDY* indica quando o eTBC está pronto para proceder a qualquer operação de escrita ou leitura[57]. A forma de aplicação recomendada para este componente é apresentada na Figura 49, onde se indicam as linhas de controlo referidas:

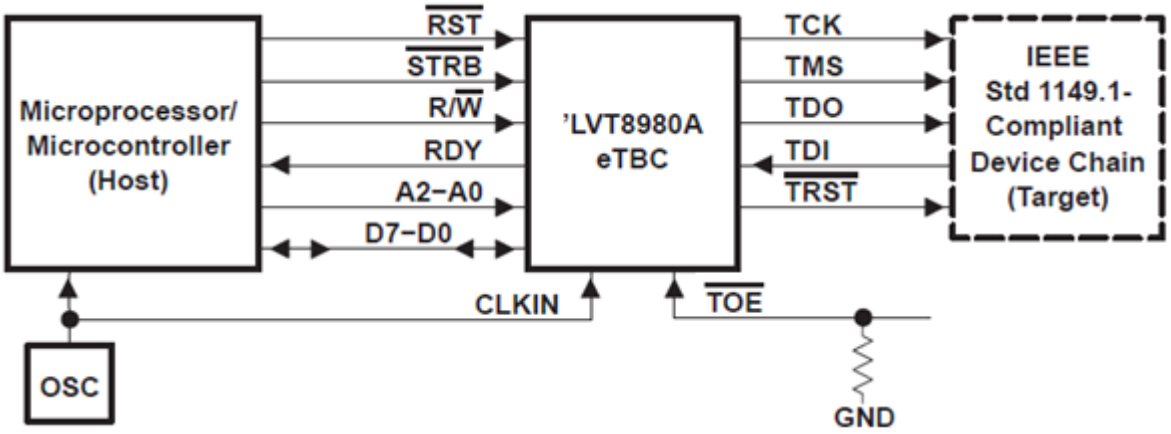


Figura 49 Aplicação típica de um *Embedded Test-Bus Controller*[57]

De acordo com o endereçamento *A2-A0*, e utilizando o barramento de dados *D7-D0*, é possível preencher as restantes configurações necessárias para cada tarefa, nos vários registos do controlador. Através destes pode-se também obter os estados de cada linha de controlo BS, dos controladores TAP dos dispositivos alvo, os valores de *TDO* recebidos, *etc.*[57].

Para estabelecimento da ligação com o computador, este componente é normalmente emparelhado com um MCU com capacidade de comunicação USB, mas deverá ser possível o seu controlo através da sua ligação a uma porta paralela.

3.2.6. FTDI MULTI-PROTOCOL SYNCHRONOUS SERIAL ENGINE (MPSSE)

A interface *Multi-Protocol Synchronous Serial Engine* (MPSSE) é uma alternativa utilizada em aplicações de comunicações série síncronas (como JTAG; *Serial Peripheral Interface* – SPI; e *Inter-Integrated Circuit* − I2C), disponível nos componentes *FT2232D/H/C*, *FT4232H* e *FT232H* do fabricante FTDI. Este modo de funcionamento permite velocidades sustentáveis de transferências de dados na ordem dos *0,7 MB/s* (*FT2232C*)[58].

Uma configuração física recomendada para a implementação deste sistema é mostrada na seguinte Figura 50:

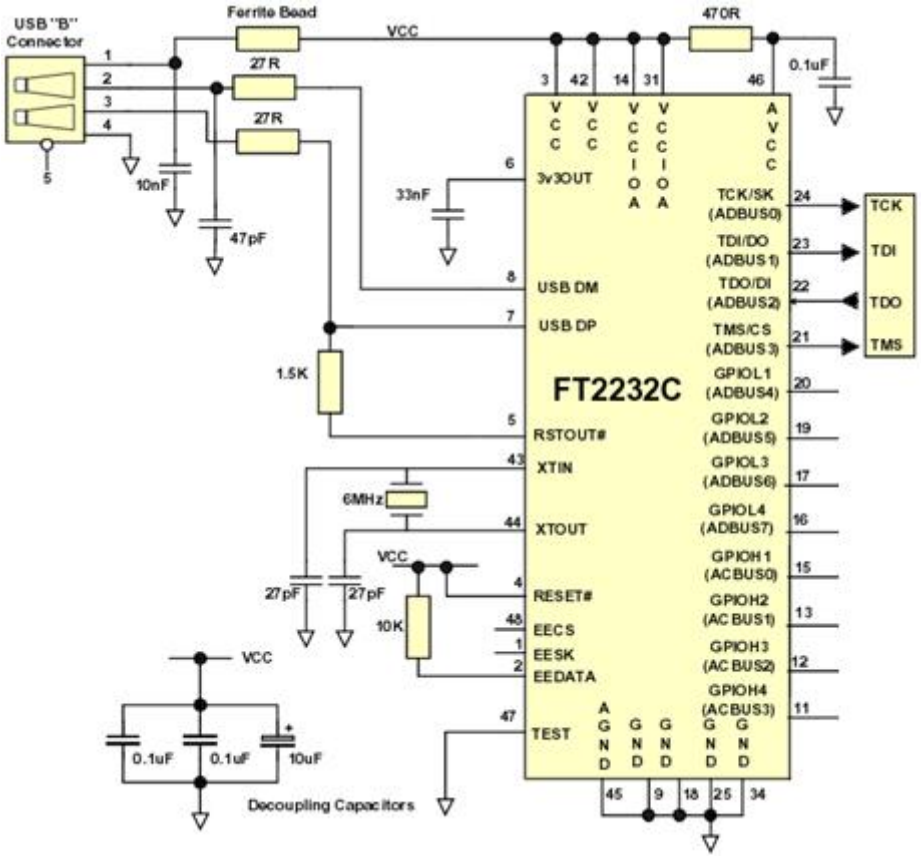


Figura 50 Esquema eléctrico da aplicação do *FT2232* para controlo JTAG (adaptado de [59])

Este caso diferencia-se do anterior ao ser prevista a utilização do controlador JTAG directamente sob ordens do computador, com o qual estabelece uma comunicação USB já integrada, não necessitando de componentes adicionais (excluindo oscilador e alguns

componentes discretos). Os pinos de *General-purpose input/output* (GPIO) são opcionais, não sendo necessários para o controlo por BS. Por ser controlado por USB, em detrimento de um microcontrolador como no caso dos eTBCs, não existem barramentos paralelos para transmissão de comandos e dados, e é necessário a aplicação de um cristal que mantenha o sinal de relógio do componente. A operação do componente é integralmente efectuada através de comunicação série, e permite as habituais funções de leitura e escrita dos diversos registos. Uma biblioteca dinâmica para SO *Windows* (*FTCJTAG DLL Ver. 2.0.0*) é fornecida para facilitar a interface com este dispositivo, tratando de todos os comandos associados ao MPSSE.

3.3. FUNCIONAMENTO BASE PRETENDIDO

O controlador desenvolvido deverá primariamente suportar uma linguagem de descrição de testes, e, com igual importância, ter a capacidade de gerir cadeias BS de múltiplos dispositivos. Estas são as características essenciais a dotar o sistema visto serem pontos-chave para aumentar a sua área de aplicação, ao permitirem: a sua integração com outros *softwares* que suportem a mesma linguagem de teste; que os utilizadores tenham maior facilidade na operação do controlador, já que a linguagem das operações é standardizada; e oferecer um suporte alargado para placas testáveis por BS. Outras funcionalidades que se pretendam acrescentar ao sistema serão secundárias, e estarão certamente dependentes das anteriores.

3.3.1. INTERPRETAR CÓDIGO SVF

Entende-se que para o controlador de *Boundary Scan* poder eficazmente efectuar as tarefas às quais o *IEEE 1149.1* dá resposta, é necessário dotá-lo da capacidade de interpretação de alguma linguagem padronizada que convenientemente as descreva. De outra forma, a execução de qualquer procedimento por BS, de um nível de complexidade equivalente ao verificado nos casos práticos, seria uma tarefa demasiado demorada e bastante susceptível a enganos por parte do utilizador, que tornaria facilmente inviável o procedimento com este controlador. Naturalmente, poderiam ser dadas ao utilizador funcionalidades de mais alto nível, *i.e.* em vez de controlar directamente a porta TAP com vista a alcançar algum estado, por exemplo, seria possível a criação de funções para executar autonomamente todos os procedimentos para o controlador atingir o estado pretendido; mas ainda assim, para que o controlador tenha a portabilidade desejada, que se pretende utilizável com

diversos sistemas, os procedimentos de BS não poderiam ser suficientemente simplificados. Uma parte do interesse num controlador como o proposto neste projecto é também servir de ferramenta que possa aplicar procedimentos de BS provenientes de outros pacotes de *software*, que descrevem, segundo alguma linguagem, o comportamento necessário.

Anteriormente foram descritas (secção *2.11*) diversas linguagens que servem como candidatas à implementação no sistema. Após consulta das suas características, verifica-se a adequação da linguagem SVF para este tipo de implementações devido especialmente a dois factores de grande importância: a sua ideologia de reusabilidade e portabilidade, tendo sido desenvolvida com o objectivo de ser realmente utilizada por grande número de *softwares* distintos e independentemente dos fabricantes dos dispositivos; e também por ser uma linguagem relativamente leve e assim adequada aos recursos limitados da implementação projectada.

No entanto, a utilização de uma linguagem que, embora aparentemente indicada, não seja convenientemente adoptada actualmente por outros pacotes de *software*, limitaria a utilidade prática do controlador, visto diminuir o número de *softwares* aos quais ofereceria suporte. A pesquisa realizada sobre a actual difusão das várias linguagens veio confirmar que não seria errada a escolha da linguagem SVF, já que é vastamente utilizada e promete ser uma mais-valia para o controlador desenvolvido.

Os fabricantes de dispositivos fornecem geralmente suporte, nos seus pacotes de *software*, para a criação de ficheiros descritivos dos procedimentos JTAG, cuja aplicação é depois da responsabilidade do controlador de BS.

Em seguida listam-se, na Tabela 4, alguns exemplos de diversos fabricantes e ferramentas, e o suporte que oferecem:

Tabela 4 Linguagens utilizadas por diversos pacotes de *software*[60]

Fabricante	Software	Formatos de saída possíveis:	Observações:
Xilinx	*iMPACT* (pacote *ISE*)	.stapl, ***.svf***	Preferência por output SVF
Altera	*Quartus II*	.jbc, .jam, ***.svf***	Preferência por output JAM
Lattice	*ispVM*[9]	***.svf***, .stapl	Preferência por output SVF.
Actel	*Designer/FlashPro* (*IDE Libero*)	.stp, ***.svf***	
Atmel	*AVRSVF*	***.svf***	Para MCUs ATmega *AVR*
Silicon Laboratories	*Hex2SVF*	***.svf***	Para MCUs Si-Labs
Cypress	*ISR*	.stapl, ***.svf***	Para PLDs Cypress

Destaca-se que o fabricante *Atmel* adopta apenas o formato SVF para a programação das suas *Microcontroller Unit*s (MCU) *AVR* por BS. Embora a definição inicial do projecto obrigue a uma maior preocupação em dar o conveniente suporte aos produtos *Atmel*, visto o controlador ter o objectivo principal de ser utilizado com o microcontrolador *ATmega16* como alvo, é satisfatório verificar que a opção pela linguagem SVF não compromete, muito pelo contrário, a usabilidade do controlador em conjunto com outros *softwares* e dispositivos alvo, ao estar presente como opção em todos os *softwares* listados.

Entendendo-se suficientemente dissipadas as dúvidas em relação à linguagem a ser implementada, opta-se pelo suporte, total ou parcial, da linguagem SVF, que permitirá padronizar as tarefas efectuadas pelo controlador, garantindo compatibilidade com a maioria das ferramentas.

3.3.2. SUPORTE PARA CADEIAS DE BS DE MÚLTIPLOS DISPOSITIVOS

A funcionalidade do sistema previsto não se limita a um "*SVF Player*", *i.e.* um *software* cujo único intuito é executar ficheiros SVF, interessando disponibilizar outras formas de controlo. Nesse sentido, é necessário ter em conta a possibilidade, recorrente na maioria dos casos reais, de existirem múltiplos dispositivos com suporte de BS na mesma placa alvo, ligados entre si formando uma cadeia de BS. A execução de um ficheiro SVF

[9] Esta aplicação utiliza uma extensão da linguagem SVF denominada *Lattice Extended Serial Vector Format*

adequado a uma determinada placa já deverá contemplar, nas operações em si descritas, a diferenciação dos seus dispositivos, com vista ao seu objectivo final. No entanto, tal terá de ser assegurado se, por outro lado, o utilizador quiser optar por efectuar, segundo quaisquer controlos adicionais facultados pelo sistema desenvolvido, funções específicas num dispositivo da placa alvo, já não podendo ser descorada a forma da cadeia de BS, da qual interessa essencialmente alguns factores descritivos, como a quantidade de dispositivos; a sua ordem na cadeia; e o comprimento, em *bits*, dos registos de IR de todos os dispositivos e do registo TDR que se pretenda editar.

Num caso prático, se, por exemplo, um utilizador pretender controlar as saídas de um determinado componente BS contido numa cadeia, é necessário que este envie um vector de *bits* que preencha os registos IR de todos os dispositivos dessa cadeia, sendo recomendável a atribuição dos modos de *BYPASS* a todos aqueles que não quer que sejam controlados e o modo *EXTEST* ao componente alvo. Para tal é necessário saber a ordenação dos componentes na cadeia e o comprimento de cada registo IR. O vector de *bits* seguidamente enviado, para a edição do registo TDR do componente alvo, necessita de contemplar a quantidade de componentes em modo *BYPASS*, e o tamanho do registo TDR alvo.

Embora este funcionamento prático não seja, *per se*, demasiado complexo, compreende-se que, especialmente tratando-se de muitos componentes e vectores consequentemente demasiado compridos, possa ser difícil para o utilizador fornecer sempre a informação necessária para a operação que pretenda efectuar, situação que seria também muito propensa a enganos na introdução manual dos valores.

Para que seja possível assistir convenientemente a manipulação de cadeias de múltiplos dispositivos, foi prevista a utilização de uma linguagem descritiva dos componentes, utilizando-se o formato BSDL (descrição presente na secção 2.11.4) para o efeito. O suporte ao formato HSDL (secção 2.11.5) que, para além de incluir as características do BSDL, engloba ainda outras funcionalidades, não se entende necessário ao sistema, pelo menos nesta fase inicial.

A interpretação de ficheiros BSDL permite que sejam facilmente reunidas as informações necessárias sobre um dado componente. Carregando-se o ficheiro BSDL de cada componente BS da placa alvo utilizada, será possível ter acesso a toda a informação

necessária para manipulação da cadeia. Valendo-se da interpretação destes ficheiros, abundantemente disponíveis para qualquer dispositivo BS, o sistema terá uma base sólida, no que concerne ao conhecimento da cadeia, para se tentar oferecer um suporte ao controlo de múltiplos dispositivos da forma mais intuitiva que se encontre, e suportará uma mais fácil adição de novas funcionalidades em desenvolvimentos futuros.

Embora se tenha já reconhecido a disponibilidade generosa destes ficheiros, para não limitar o sistema na eventualidade de poderem existir componentes cujo ficheiro descritivo BSDL não se encontre disponível, será também fornecido suporte para ficheiros próprios, criados pelo próprio sistema, de acordo com as informações fornecidas pelo utilizador. Estes ficheiros também deverão poder ser criados a partir de ficheiros BSDL, aproveitando-se para oferecer então a possibilidade de alteração dos seus valores e introdução de informação complementar (*e.g.* descrição dos *bits* do registo BSR).

A execução de tarefas complexas em cadeias de multi-dispositivos é, admite-se, mais eficientemente efectuada recorrendo a um ficheiro SVF. Ainda assim, serão, com recurso à referida interpretação de ficheiros descritivos, fornecidos vários controlos manuais assistidos, *i.e.* cuja operação permite ao utilizador descorar alguns procedimentos necessários, sendo estes levados a cabo automaticamente pelo *software* (identificação da ordem dos dispositivos, cálculo do *header/trailer* de cada componente, identificação do tamanho dos vectores, aplicação do modo de *BYPASS* a dispositivos não utilizados, *etc.*). Espera-se que o sistema final possa assim ser adequado, o melhor possível, a uma utilização por parte de utilizadores menos conhecedores da norma. Um controlo total, não assistido, deverá ser também disponibilizado.

3.3.3. OUTRAS FUNCIONALIDADES

Para além do suporte para execução de SVF e funções de distinção de dispositivos dentro das cadeias alvo, prevê-se a inclusão no sistema de outras funcionalidades que possam beneficiar a utilidade do sistema final, especialmente considerando o contexto didáctico em que se prevê que possa ser utilizado, e a sua utilização com MCUs *Atmel*. De acordo com as informações recolhidas sobre as funções disponíveis noutros programas, lista-se um conjunto de outras funcionalidades que se visam implementar no corrente projecto:

- Identificação de dispositivos alvo e da sua organização em cadeias de BS;
- Edição de registos (*e.g.* BSR) simplificada através de mecanismos da interface gráfica;
- Ferramentas de controlo directo ou assistido dos controladores TAP alvo;
- Ferramentas específicas para MCUs *AVR* com suporte de *Boundary Scan*;

A programação de MCUs *AVR* deve ser possível recorrendo à ferramenta do fabricante, referida na anterior Tabela 4, em conjunto com a capacidade de interpretação de código SVF de que se pretende dotar o sistema. As restantes funcionalidades seguem a tendência encontrada em outros *software*s similares de controlo por BS, de tentar facilitar a utilização do *IEEE 1149.1*, objectivo que se enquadra totalmente no corrente projecto. Outras funcionalidades poderão ser consideradas com este mesmo objectivo.

3.4. CONSIDERAÇÕES SOBRE O HARDWARE

Os recursos limitados do projecto excluem desta secção sistemas físicos demasiado complexos. Focando-se nas necessidades reais de *hardware* do controlador, é apenas estritamente necessário que este comunique com o computador e que possa operar mudanças no valor lógico de cada um dos pinos do conector que ligará à porta TAP dos dispositivos alvo. Todas as restantes funções (como, principalmente, a interpretação dos comandos SVF e a sua tradução na sequência de mudanças de estados dos pinos das portas TAP) podem ser deixadas a cargo do computador cuja superior capacidade de processamento e disponibilidade de memória tornam-no mais apto para a tarefa (como exemplo, um ficheiro SVF que faça programação e posterior verificação de um MCU *ATmega16* ultrapassa as *1300* linhas). A menor utilização do processamento na placa do controlador BS, que se pretende encarregue do controlo do conector JTAG, poderá também significar um desempenho mais célere nas suas poucas tarefas. No entanto, dever-se-á ter em consideração que o fluxo de dados entre o controlador e computador não seja excessivo, de forma a não aumentar o tempo dispendido em transferências.

Outras alternativas que pressuponham o processamento do código SVF no MCU, talvez com a adição de uma memória ou cartão de memória ao controlador, onde seriam armazenados os ficheiros SVF para posterior processamento autonomamente do computador, foram equacionadas mas, embora trouxessem outras vantagens, como um

melhor controlo das temporizações e frequência de TCK, por existir menor necessidade de sincronizações entre o computador e controlador, foram dispensadas em favor das razões descritas anteriormente.

Só se conseguindo ter a certeza das implicações, a nível de desempenho, desta abordagem após a sua implementação (foram conduzidos testes no sistema final cujas conclusões estão referidas na secção 6), crê-se sensato presumir, nesta fase, que a implementação do processamento de ficheiros SVF no controlador BS teria um desempenho bastante problemático, limitando a escolha do seu núcleo de processamento para alternativas bem mais poderosas do que as contempladas com a abordagem definida. Tendo em conta que é normal a operação JTAG com frequência, em *TCK*, de *10 MHz* (embora geralmente se utilizem por definição valores mais modestos de *1 MHz* e inferiores), entende-se que a utilização do controlador BS para, além de actuar o conector JTAG a essa frequência e comunicar com o computador, também interpretar o código SVF, traria uma exigência muito grande, pelo menos a nível da sua frequência de processamento. Os controladores de BS analisados (secção 3.2) estão também longe de permitirem a interpretação de qualquer linguagem completa de descrição das operações de BS, oferecendo apenas suporte para algumas funções que facilitem a sua operação, algo que não se coloca de parte no sistema actual, tendo de ser analisadas à medida que o sistema é desenvolvido.

Opta-se assim pela simplificação do *hardware* e suas responsabilidades, na esperança de que possa trazer uma maior eficiência ao controlador e, ao mesmo tempo, permitir manter um custo reduzido.

3.4.1. NÚCLEO DA PLACA DO CONTROLADOR DE BOUNDARY SCAN

Da pesquisa efectuada destacaram-se soluções de *hardware* que permitem ao computador controlar directamente a interface JTAG, através de uma ligação a uma porta paralela; e outras onde a interface JTAG é comandada por um núcleo de processamento, que estabelece, ele próprio, ou através de ainda outro componente, a interface de comunicação USB com o computador. A nível de *software*, estas abordagens diferenciam-se na implementação das funções de baixo nível para cumprimento das especificidades do *IEEE 1149.1*, que podem ser: programadas no computador (interfaces JTAG paralelas); ou serem efectuadas no próprio controlador (restantes soluções), já que estes terão de processar a informação série recebida do computador.

A primeira das soluções indicadas é muito simples, de fácil montagem, e deverá ser suficientemente rápida, ao ser controlada directamente pelo computador. No entanto, a utilização de portas paralelas está ultrapassada tanto que não estão disponíveis em todos os computadores, especialmente considerando computadores portáteis. Não se querendo implicar tal limitação ao controlador a desenvolver, esta alternativa foi rapidamente descartada.

O controlo pelo computador através de USB, situação de todo preferível, já requer algum tipo de componente que processe os sinais do computador e convenientemente proceda à actuação da interface JTAG, o que aumenta a sua complexidade e custo, mas não obrigatoriamente a sua velocidade, já que embora o barramento USB permita velocidades superiores, o tempo de processamento da interface utilizada para garantir o suporte do protocolo USB no controlador, assim como a velocidade de transmissão obtida, terão grande influência. O núcleo de processamento, encarregue da actuação da interface JTAG, segundo as especificações da norma, poderá ter já suporte nativo para USB ou necessitar de algum componente externo que ofereça tal suporte. Como a implementação de uma ligação ao computador é essencial, e não se pretende utilizar portas série ou paralelas, devido ao seu fraco suporte em computadores actuais, opta-se pela utilização de um barramento USB, e antevê-se a necessidade do seguinte sistema, apresentado na Figura 51:

Figura 51 Representação do Controlador de BS previsto

A parcela do controlo que irá recair ao computador e ao controlador BS deverá ser ajustada de acordo com o que melhor privilegia o desempenho do sistema. Um componente independente para garantir a interface USB é apenas necessário se o núcleo de processamento do controlador não tiver suporte nativo para esta comunicação. As opções consideradas para o processamento são: MCUs; CPLDs; eTBCs (secção 3.2.5); ou o sistema MPSSE (secção 3.2.6).

Embora os CPLDs ofereçam benefícios no melhor controlo dos tempos de actuação e, previsivelmente, na sua velocidade de operação, entende-se que estas vantagens percam significância com a necessidade de incluir um componente que sirva de interface com o computador, já que não é usual os CPLDs possuírem capacidades nativas de comunicação

com o computador. Para além disso, os encapsulamentos em que estes componentes são disponibilizados são menos práticos de montar, não sendo compatíveis com placas perfuradas ou *breadboards*. Considerações similares podem ser atribuídas também aos componentes eTBC.

Os componentes do fabricante FTDI com tecnologia MPSSE, ao terem suporte nativo para comunicação USB e efectuarem já a gestão das especificações da norma JTAG, aparentam ser a solução com melhor desempenho, embora não se encontrem dados quantitativos relativos ao seu desempenho enquanto controlador JTAG. Comparativamente aos MCU, este tipo de componentes permite uma muito menor flexibilidade e customização, ao não poder ser totalmente programado, e, tal como os anteriores, estar apenas disponível em encapsulamentos menos amigáveis (*48LQFP*). Embora os seus custos comerciais não sejam, por si só, elevados, ficando-se por *4,42€*[10] para o componente *FT232HL*, tem de ser também considerado os superiores custos de montagem, face a alguma alternativa de THT, devido à necessidade de criação de um PCB.

A escolha acaba assim por recair num MCU, especialmente por serem componentes programáveis, garantindo um maior controlo sobre o funcionamento do sistema, mesmo que possam não garantir o melhor desempenho. Serão ainda a alternativa mais prática, contando com a sua maior disponibilidade. Com as tarefas de processamento reduzidas que se pretende atribuir ao controlador, não são identificadas necessidades especiais a nível da sua unidade de processamento, tendo-se seleccionado o MCU de *8 bits*, *ATmega328P*. A sua opção face a produtos semelhantes mas de reduzida memória (*e.g. ATmega48/P*, *ATmega88/P*, *ATmega168/P*) é justificada pelas exigências de memória do *firmware* (referidas na secção *5.1*), mais adequadas às capacidades do dispositivo escolhido, e também pela pouco significativa diferença do seu valor comercial para com as alternativas inferiores referidas, como se mostra na Tabela 5:

Tabela 5 Listagem de preços dos MCU *ATmega48/88/168/328*

Nome:	Memória *FLASH*:	Memória SRAM:	Preço[10]:
ATMEGA48	*4 Kb*	*512 b*	*3,20€*
ATMEGA48P	*4 Kb*	*512 b*	*3,81€*
ATMEGA88PA	*8 Kb*	*1 Kb*	*3,00€*
ATMEGA88A	*8 Kb*	*1 Kb*	*2,74€*
ATMEGA168PA	*16 Kb*	*1 Kb*	*3,04€*
ATMEGA168A	*16 Kb*	*1 Kb*	*3,45€*
ATMEGA328P	***32 Kb***	***2 Kb***	***3,15€***

Sendo estruturalmente compatíveis entre si, os dispositivos apresentados diferem apenas na sua memória e outros aspectos mínimos, como o endereço de algumas interrupções. A diferença dos dispositivos indicados com a letra '*P*' é o seu consumo energético inferior ("*Picopower*"). O *porting*[11] de um programa entre estes dispositivos é por isso bastante facilitado[61][62]. Assim, antevê-se que seja possível a adaptação do sistema a uma placa que disponha de algum destes MCU, mas mediante uma possível redução do tamanho do código do programa.

O custo da implementação do *ATmega328P* em relação a outras alternativas de superior desempenho (*e.g.* dispositivos *ARM* de *32 bits*) é muito inferior. O valor comercial de um dispositivo com um núcleo *ARM7* de igual memória de programa, *i.e. 32 Kb*, como o *LPC2131FBD64*, que tem uma frequência de funcionamento três vezes superior (*60 MHz*) não comporta por si só um custo significativamente superior (*4,70€*[10]). No entanto, a solução escolhida dispensa a produção de uma placa de circuito impresso, podendo ser rapidamente soldada numa placa perfurada, obtendo-se um sistema muito mais económico, o que não acontece com as alternativas que não estão disponíveis em encapsulamento *Parallel dual in-line package* (PDIP).

A escolha de um MCU no formato PDIP possibilita também que o controlador possa ser rapidamente montado até numa *breadboard*, em conjunto com alguns outros componentes PDIP; e a utilização de uma placa de prototipagem muito difundida, o *Arduino UNO*, que

[10] Preços do distribuidor *Farnell*, em Portugal, à data de 16 de Dezembro de 2013, sem contabilização de portes de envio.

[11] *Porting* refere-se ao processo de adaptação do código de um programa para uma nova plataforma de outra forma incompatível.

também utiliza este MCU. Estas possibilidades para a rápida montagem do sistema são uma mais-valia para utilizações didácticas, ao permitir que os estudantes possam, sem dificuldade, montar o próprio controlador, ou utilizar a placa *Arduino*, e seguidamente programá-la com o *firmware* fornecido.

Uma característica determinante do MCU seleccionado é este não conter conectividade USB nativa, quando tal ligação será sempre indispensável neste projecto. A razão de não escolher um MCU com suporte USB prende-se mais uma vez com as razões do custo e praticabilidade dos seus encapsulamentos físicos.

3.4.2. CONECTIVIDADE USB

A utilização de um conversor Série-USB é a alternativa mais frequente para utilizar comunicação USB com este tipo de MCU, e não carece de qualquer programação. Mas é também possível não utilizar qualquer equipamento adicional e programar o MCU para emular o funcionamento do barramento USB (*DATA+* e *DATA-*). Uma terceira alternativa seria a utilização de um IC que gerisse o barramento USB (*e.g. USB2.0 physical layer transceiver*, ou outro componente com o mesmo objectivo), mas cujo controlo teria também de ser programado no MCU[63].

A alternativa da programação do barramento USB no MCU significaria um menor custo para a produção do controlador e a sua implementação não seria tão problemática quanto o conceito possa sugerir: Está disponível com licença *open source*, uma biblioteca (*V-USB*) compatível com o MCUs *ATmega* (*ATmega88* entre outros) com esta função, tendo inclusivamente sido utilizada indirectamente neste projecto, no programador utilizado (*USBasp*, equipamento descrito no Anexo A) para a programação ISP do MCU do controlador[64]. A aplicação da biblioteca ao *ATmega328P* não deverá suscitar qualquer problema[61][62].

As principais vantagens desta solução são a menor complexidade do *hardware* necessário (são apenas necessárias duas ou três linhas de I/O do MCU) e, consequentemente, o já referido menor custo que acarreta. As funcionalidades prometidas são:[64]

- Totalmente compatível com o *standard USB1.1* para dispositivos de baixa velocidade (*1,5 Mbit/s*), excepto gestão de erros de comunicação e especificações eléctricas;

- Compatível com sistemas *Linux*, *Windows* e *Mac OS X*;
- Suporta, por definição, blocos de transferência até *254 bytes*;
- Contém já identificadores USB − *Vendor-ID* (VID) e *Product-ID* (PID);
- Funcional em MCUs *AVR* com pelo menos: *2 Kbytes* de memória *FLASH*; *128 bytes* de SRAM; e frequência de *12 MHz*;
- Apenas *1150* a *1400 bytes* de código.

A maior desvantagem inerente a esta alternativa será a sua velocidade de transferência, que é expectável que seja muito inferior à obtida com a utilização de um conversor Série-USB, como é usual utilizar neste tipo de MCU.

Quanto à alternativa de utilização de um conversor Série-USB, que faz uso do suporte a comunicações série *Universal Asynchronous Receiver/Transmitter* (UART) do MCU utilizado, existem vários dispositivos que permitem a sua implementação com muita facilidade. As soluções usuais são normalmente de um dos seguintes fabricantes:

- *FTDI* (*e.g. FT232RL*, entre outros);
- *Silicon labs* (gama *CP210x*);
- *Prolific* (*e.g. PL2303*).

Não se encontrando comparações formais entre o desempenho real de cada um destes, é do consenso geral, considerando relatos pontuais, a superior fiabilidade das ligações estabelecidas com os componentes dos dois primeiros fabricantes listados. A diferença entre as especificações oferecidas pelos diversos componentes não deverá ser significativa para os comparar quanto ao seu desempenho real.

Infelizmente, os componentes com a função destacada estão disponíveis apenas em encapsulamento SMD. Todos estão difundidos em inúmeros cabos e adaptadores, de disponibilidade comercial, que permitem efectuar rapidamente a ligação entre a porta USB

do computador e as linhas de comunicação série UART dos MCU, comportando custos bastante reduzidos (a partir de *3€*[12] para um adaptador baseado no componente *CP2102*).

Existem também outros produtos que não são baseados em componentes com a função nativa de conversão Série-USB, como os listados, tais como o *Arduino USB Serial Light Adapter*, que utiliza o MCU *ATmega8u2* (também apenas disponível em encapsulamento SMD).

Tecidas estas considerações, optou-se pela utilização de um conversor Série-USB, por ser, depreende-se, muito menos exigente a nível da utilização de processamento do MCU, e permitir transferências expectavelmente mais céleres. Os conversores série são rapidamente reconhecidos e instalados pelos sistemas operativos (SO) recentes, não requerendo procedimentos adicionais por parte do utilizador. Reconhece-se ainda assim que seria proveitoso testar ambas as soluções, comparando o seu desempenho pela contabilização dos tempos de espera na realização de tarefas BS, já que a velocidade de transferência será previsivelmente um factor limitador do desempenho do sistema.

3.4.3. CONECTOR JTAG

A norma *IEEE 1149.1* não define um conector *standard* para a operação do sistema BS, apenas descreve quais pinos são obrigatórios e opcionais. Tal facto leva a que exista uma extensa lista de possíveis disposições destes pinos, que ficam ao critério de cada fabricante[65].

Estes conectores são normalmente machos de dupla coluna e espaçamento *standard* (*2,54 mm*), como esboçado na Figura 52. Os pinos opcionais de que dispõem dependem do fabricante mas em comum têm sempre cinco pinos obrigatórios: *TCK*; *TDI*; *TMS*; *TDO*; *GND*.

[12] Preços, à data de 18 de Fevereiro de 2014, em *website* de venda livre.

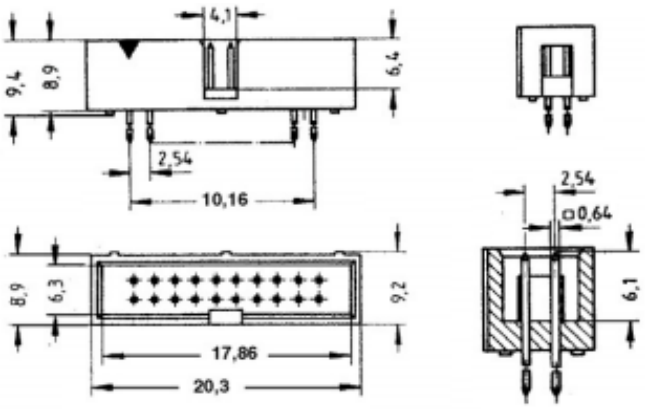


Figura 52 Dimensões genéricas nos conectores JTAG

Para além dos conectores conhecidos, em placas comerciais é muito frequente os conectores serem dissimulados de forma a dificultar o seu acesso por parte de terceiros, visto o enorme potencial que esta tecnologia oferece. A Figura 53 é um exemplo dos pinos JTAG descobertos num produto comercial (*router wireless*):

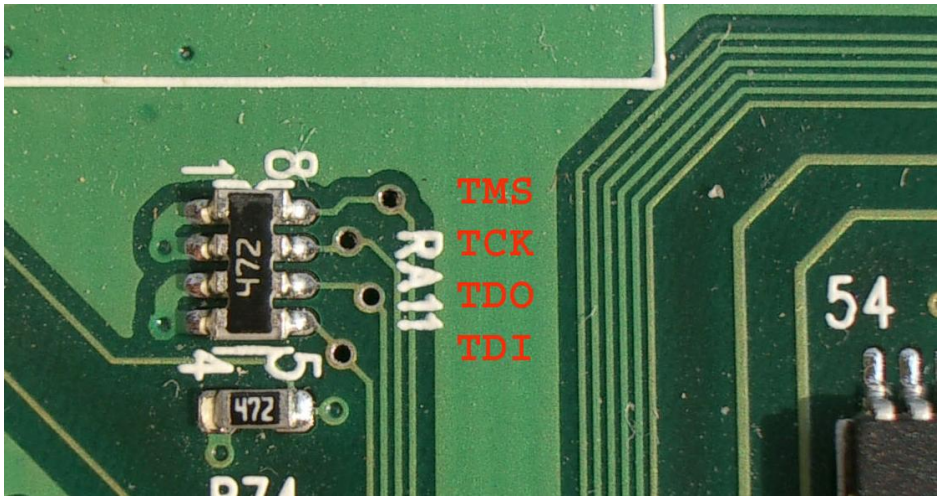


Figura 53 Pinos JTAG num produto comercial

A abordagem de alguns controladores de BS comerciais, tal como o *ViaTAP JTAG Interface*, enunciado na secção *3.1.3*, é a disponibilização de vários conectores distintos, no caso referido nove conectores, tentando assim aumentar a sua aplicabilidade[43].

Como o objectivo principal do controlador projectado é a operação do BS de um MCU *Atmel*, a decisão por utilizar o conector que esta disponibiliza nos seus produtos (sejam placas de prototipagem ou programadores/depuradores JTAG), deverá ser a que mais beneficiará a sua utilização futura. Outros conectores poderão ser também facilmente utilizados recorrendo a algumas ligações físicas que apenas reordenem os pinos do conector (Figura 54):

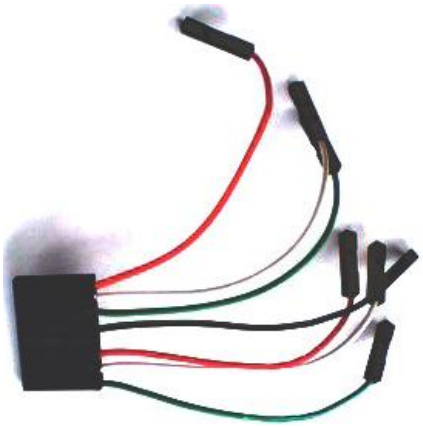


Figura 54 Adaptador para conector JTAG

3.4.4. PLACA ALVO PARA TESTES

A placa alvo deverá ser composta por componentes controláveis por BS. Não existindo nenhum objectivo de controlo pré determinado, relega-se qualquer definição mais pormenorizada da constituição e organização da placa alvo para momento posterior, de acordo com a necessidade de testar o sistema no decorrer do seu desenvolvimento.

A utilização de uma cadeia de BS com vários dispositivos é, no entanto, já planeada. Porque só se pretende formar uma cadeia de BS, todos os componentes controláveis implementados devem partilhar as linhas de TCK e TMS, enquanto TDI deverá formar a cadeia de dispositivos. Não se objectiva o controlo em simultâneo de mais do que uma cadeia.

Não se perspectiva também a realização de testes automatizados à placa alvo desenvolvida. Tais testes poderão ser levados a cabo através da execução de um ficheiro SVF para o efeito, mas a competência para a criação do código SVF apropriado para tal será da responsabilidade de outro *software*. Assim, será mais importante a disponibilização das entradas e saídas dos componentes, para verificação do correcto controlo e leitura dos seus valores lógicos, do que a criação de interligações entre os componentes para a realização de testes ("*Interconnect Testing*"), o que dispensa maiores considerações sobre o desenho do circuito da placa alvo (DFT).

3.5. CONSIDERAÇÕES SOBRE O SOFTWARE

A programação, para satisfazer a arquitectura do sistema a ser exposta na seguinte secção 3.6, e com vista a uma boa expansibilidade, divide-se em três partes fundamentais:

- *Firmware* da placa do controlador;
- Biblioteca para o seu controlo;
- Aplicação gráfica;

Havia-se já referido o intuito de aumentar a incidência do processamento das tarefas no lado do computador. No entanto, não se entende adequado a programação de uma aplicação totalmente responsável por este. A utilização de um terceiro elemento, uma biblioteca dinâmica (*.dll*), permite uma certa independência do controlador em relação à aplicação gráfica, que facilitará a implementação de suporte ao controlador em outros possíveis programas. Esta biblioteca deverá efectuar todo o processamento necessário e comunicação com o controlador, ficando a aplicação gráfica encarregue do controlo da biblioteca, sem aceder directamente à placa do controlador; formatação dos dados recebidos; e interacção com o utilizador.

Tecem-se de seguida as considerações preliminares sobre cada uma destas componentes.

3.5.1. PLANEAMENTO DO *FIRMWARE* DO CONTROLADOR E BIBLIOTECA DE CONTROLO

Para o desenvolvimento do código do *firmware* utiliza-se o *Integrated Development Environment* (IDE) da *Atmel*, destinado ao tipo de MCU utilizado: "*AVR Studio 6.1*". Este *software* está livremente disponível para *download* no *website* do fabricante e é assim a escolha mais imediata para a programação dos MCUs *AVR*. Este permite a utilização de linguagem *C* na programação dos MCU, opção que será escolhida pela maior facilidade com que permite implementar funções mais complexas, comparativamente à linguagem *Assembly*.

Para a programação da placa é utilizado o programador *USBasp*, sistema construído de acordo com o descrito no 0, em conjunto com o *software freeware* "*Khazama AVR Programmer*"[66]. Esta aplicação não permite a programação dos *fuses* do *ATmega328P*, pelo que é pontualmente utilizada uma outra aplicação *freeware* para esta tarefa: "*eXtreme Burner – AVR*".

O *firmware* desenvolvido responderá a comandos enviados pelo computador, não se antevendo a implementação de um funcionamento autónomo. Estes comandos poderão ser

provenientes de uma aplicação, no caso previsto utilizando a biblioteca de controlo programada, ou directamente introduzidos pelo utilizador, fazendo-o através de um terminal para comunicação série. Este modo de controlo visa, especialmente, servir para a depuração de eventuais problemas no controlador.

A biblioteca de controlo será compilada como uma biblioteca dinâmica para SO *Windows* ("*.dll*"), e limita a utilização do controlador a este tipo de SO, excepto se utilizado através de um terminal. Ambos os códigos, da biblioteca e *firmware*, foram escritos em linguagem *C*.

Pretende-se que o conjunto de *software* incluído na biblioteca, juntamente com a placa do controlador, constituam um sistema base já totalmente funcional. A sua operação deve ser parcialmente possível em linha de comandos *Windows* através da ferramenta de sistema "*rundll.exe*", ainda que de forma pouco amigável e prática[67]. A verdadeira vantagem que se antevê desta modularidade não será a possibilidade do utilizador dispensar a aplicação gráfica, mas a de poder ser programado com acrescida facilidade um novo *software* para funcionamentos específicos, que consegue assim utilizar todo o código fundamental para o controlo por BS, incluído na biblioteca, e também esquecer a comunicação com o controlador BS.

As funções primárias da biblioteca de controlo são:

- Comunicação com o controlador;
- Execução de operações SVF;
- Operação de procura e identificação de dispositivos na cadeia de BS da placa alvo;
- Interpretação de ficheiros descritivos da arquitectura BS dos componentes (BSDL e ficheiros próprios);
- Outras operações elementares (identificar controlador; activar/desactivar interface BS; ler/escrever pinos da porta TAP; etc.)

O comportamento da biblioteca é convenientemente descrito na secção *5.2*.

3.5.2. PLANEAMENTO DA APLICAÇÃO GRÁFICA

A aplicação gráfica destina-se a proporcionar, de forma simplificada, todas as funcionalidades de que se pretende dotar o sistema. Optou-se pela sua escrita em linguagem *C#*, tendo-se preterido o usual sistema de gestão da interface de utilizador *Windows Forms* (*WinForms*), em função do mais recente *Windows Presentation Foundation* (WPF).

O WPF foi introduzido no "*.NET Framework 3.0*" (actualmente na versão *4.5*) como alternativa ao tradicional *WinForms* para o desenvolvimento de interfaces de utilizador. O modelo de desenvolvimento do WPF é muito diferente do *WinForms*, o que dificulta a transição para o novo modelo, e está mais adequado à utilização do *hardware* gráfico moderno. O *design* gráfico da aplicação é escrito em *Extensible Application Markup Language* (XAML). As principais vantagens face ao anterior modelo são muitas, se, por vezes, subjectivas: a menor complexidade necessária para a construção de aplicações visualmente trabalhadas; maior flexibilidade; utilização de aceleração por *hardware*; utilização de linguagem XAML que simplifica a edição do aspecto da interface do utilizador; escalabilidade inata devido ao seu núcleo ser independente da resolução; superior customização do aspecto da aplicação; utilização tanto em aplicações *Windows* como aplicações *Web*; dispõe de novas técnicas de ligação entre a interface e o programa, como o "*Data Binding*"[13]; etc. O WPF, em contrapartida, poderá ainda não ter as suas *suites* de desenvolvimento tão desenvolvidas como o *WinForms*, sendo também menos suportado por aplicações de terceiros; cria dificuldades para os programadores aprenderem a utilizar o novo modelo, até porque ainda existe menos informação disponível comparativamente ao já tão estabelecido anterior modelo; é incompatível com SOs *Windows 2000* e anteriores; e necessita de placas gráficas compatíveis com, pelo menos, *DirectX 9*[51][68].

Na aplicação gráfica tenta-se aproveitar todas as funcionalidades do sistema, expondo-as de forma simplificada ao utilizador. Alguns *softwares* adicionais, de terceiros, poderão ser automaticamente executados por esta com o objectivo de aumentar as suas possibilidades. As funções de execução de operações JTAG em dispositivos específicos de uma cadeia de

[13] *Data Binding* é uma técnica que consiste em associar elementos da interface gráfica a alguma fonte de dados no programa.

BS deverão estar, na sua totalidade, implementadas na aplicação, pelo que actuando a biblioteca de controlo através de, possivelmente, outro *software* programado para o efeito, não será possível executar automaticamente a distinção do dispositivo alvo pretendido dentro de uma placa alvo com diversos componentes com BS. As operações JTAG executadas pela aplicação deverão recorrer a código SVF, enviado para ser interpretado pela biblioteca de controlo.

Assim, percebe-se que, embora a biblioteca e o controlador consistam num sistema já funcional, que pode ser implementado na componente física de algum outro projecto que careça de um controlador de BS, o seu funcionamento perde muitas funcionalidades se for comandado por outro *software* que não a aplicação gráfica programada. Apenas se entende recomendável, como anteriormente sugerido, a programação e utilização de outro programa em aplicações para as quais a aplicação gráfica possa não estar vocacionada.

Espera-se que a aplicação gráfica seja suficientemente intuitiva até para a sua utilização por parte de utilizadores alheios ao funcionamento do *IEEE 1149.1*, contando que, no entanto, algumas particularidades deste tenham sempre de ser conhecidas para garantir a sua correcta utilização.

3.5.3. LISTAGEM DO *SOFTWARE* DE DESENVOLVIMENTO NECESSÁRIO

Na Tabela 6 listam-se todos os pacotes de *software* necessários para o projecto, acompanhados da respectiva versão utilizada e de uma pequena descrição do seu intuito no desenvolvimento do sistema:

Tabela 6 Listagem dos *softwares* de desenvolvimento utilizados

***Software*:**	**Versão:**
• *AVR Studio 6.1* Ambiente para desenvolvimento do *firmware* do controlador de *Boundary Scan*	*6.1.2730 SP2*
• *Khazama AVR Programmer* Programação do *firmware* no MCU *ATmega328P*	*1.7.0*
• *eXtreme Burner – AVR* Programação dos *fuses* do MCU *ATmega328P*	*1.4.2*
• *Firmware* e *drivers USBasp* *Firmware* e *driver* do programador implementado para a programação do MCU *ATmega328P*	*28-05-2011*
• *Tera Term* Terminal para controlo e depuração do *firmware*	*4.69*
• *NetBeans IDE 7.4* Ambiente para desenvolvimento da biblioteca *libBS.dll*	*7.4*
• *MinGW* Pacote de ferramentas para a compilação da biblioteca *libBS.dll*	*4.8.1 (gcc)*
• *Microsoft Visual Studio Ultimate 2012* Ambiente para desenvolvimento da aplicação gráfica	*11.0.60610 Update 3*
• *Microsoft .NET Framework 4.5* Base de funcionamento da aplicação gráfica	*4.5.50709*

Alguns *softwares* foram incluídos no projecto com vista a garantir algumas funcionalidades específicas. Estes são o *ATMEL AVRsvf v.1.7*; *GOEPEL BSDL Syntax Checker v4.1*; e algumas bibliotecas de linguagem *C*, enunciadas na secção *5.2.1*. É de realçar que as bibliotecas referidas não se mantêm no seu estado original, tendo sido alvo de modificações consideráveis e indispensáveis ao seu funcionamento no actual sistema (processo descrito ao longo da secção *5.2*).

3.6. ARQUITECTURA DO SISTEMA

O controlador desenhado estará, numa aplicação prática, conectado a uma placa alvo e a um computador.

A representação da arquitectura do sistema, contemplando os módulos de *hardware* e *software* que foram sendo descritos, é apresentada na posterior Figura 55:

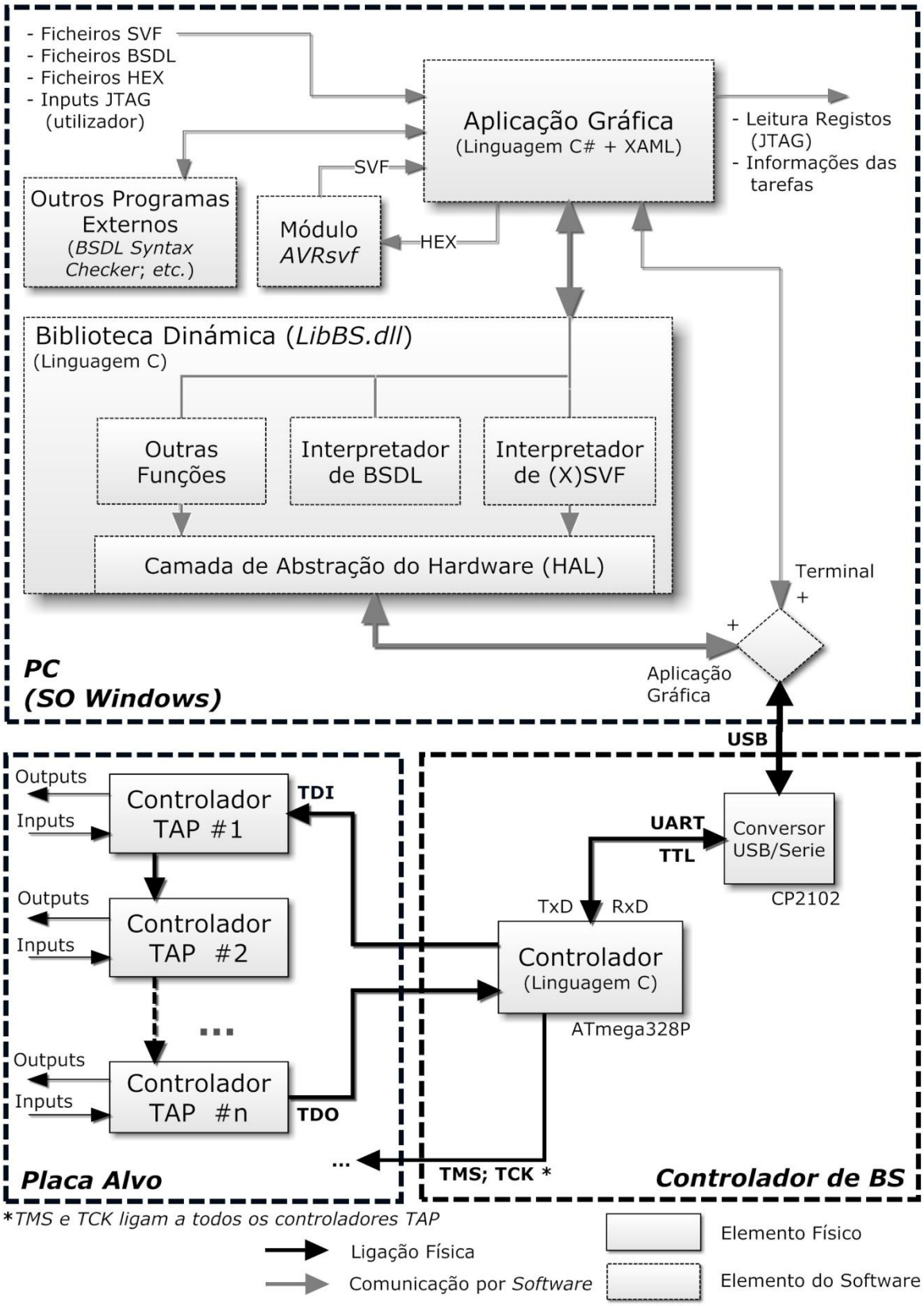


Figura 55 Arquitectura simplificada do sistema (*Hardware* e *Software*)

A esquematização apresentada, que é válida para o protótipo funcional desenvolvido, engloba elementos respeitantes ao *software* e *hardware* do sistema, que virão ainda a ser descritos, e sumariza as opções tomadas no desenvolvimento do projecto.

3.7. METODOLOGIA DE TESTE E DEPURAÇÃO DO PROTÓTIPO FUNCIONAL

O teste ao sistema desenvolvido (cujos resultados se descrevem na secção *6*) centra-se, no contexto dos objectivos definidos, em apenas duas vertentes: o correcto funcionamento das funcionalidades disponibilizadas e o seu desempenho, *i.e.* tempo dispendido na sua operação.

A integridade do funcionamento do sistema, característica prioritária, pode ser, em grande parte, verificada utilizando o controlador para tarefas complexas como a programação e verificação de um MCU *AVR* alvo, que deverá ser finalizada com sucesso, efectuando-se uma consequente aferição se o comportamento deste responde de acordo com o programa definido. Para tal será criado um programa de teste para o MCU alvo, com um comportamento bem definido que facilite a sua verificação. Descartados assim eventuais problemas na funcionalidade base do controlador, de operar correctamente o controlador TAP alvo de acordo com o código SVF utilizado, os restantes testes procurarão apenas verificar cada uma das funcionalidades específicas disponibilizadas. Todos os testes deverão ser também efectuados num dispositivo alvo específico de uma cadeia BS de múltiplos dispositivos, comprovando o correcto funcionamento do cálculo e aplicação dos *headers* e *trailers* do código SVF, necessários para operar individualmente os dispositivos da cadeia.

Prevê-se a implementação de mecanismos para testar e reunir possíveis erros no decorrer das operações de BS, que visem permitir uma maior confiabilidade na procura de eventuais problemas no funcionamento do protótipo funcional. Espera-se que estes dados, aliados ao próprio sistema de verificação dos valores esperados, presente no *standard* SVF (utilizando-se o parâmetro "TDO"), seja suficiente para a correcta avaliação do funcionamento obtido.

Relativamente ao desempenho do sistema, a sua avaliação será efectuada recorrendo à captura de informações sobre as operações executadas, que se pretende que, por si só, possam definir quantitativamente o seu desempenho e sirvam como referência de comparação (*e.g.* frequência de TCK; tempo total dispendido; *etc.*). Os testes efectuados ao

desempenho do sistema são especialmente importantes também no seu desenvolvimento, por poderem facultar indicações quanto às melhores abordagens, ao serem comparadas diversas opções técnicas com o objectivo de conhecer qual trará melhores resultados. Os testes documentados (secção 6.2) ao desempenho do sistema podem ser recriados executando os mesmos ficheiros SVF de teste sobre a placa alvo desenvolvida.

Pelo papel fundamental que tomam na metodologia de testes descrita, realçando-se também a importância que irão inclusivamente ter na optimização do sistema final, é então previsto o fornecimento ao utilizador de um conjunto detalhado de estatísticas sobre as operações executadas.

4. DESCRIÇÃO DA COMPONENTE FÍSICA

A componente física desenvolvida para o projecto consiste em duas placas. A primeira é o indispensável controlador de *Boundary Scan*, que deverá comandar fisicamente os controladores TAP dos dispositivos alvo. Para além desta, uma outra placa foi criada e serve como a placa alvo do sistema, composta por uma cadeia BS de alguns dispositivos.

Os circuitos esquemáticos apresentados nesta secção, relativos a ambas as placas, foram desenhados no *software* "*EAGLE 5.6.0 Professional Edition*".

4.1. CONTROLADOR DE BOUNDARY SCAN

Por ser baseado num conjunto básico de requisitos, existem várias alternativas para a montagem do controlador BS desenvolvido. No *firmware* do controlador existem parâmetros editáveis, que são referidos na posterior secção *5.1.2*, e que se destinam a adaptar o programa a mudanças no circuito eléctrico do *hardware*, pelo que o seguimento das soluções contidas nesta secção não é, salvo em algumas particularidades, imperativo para a recriação do sistema desenvolvido. Assim, para além da construção do protótipo funcional criado, destaca-se a possibilidade de implementar o sistema recorrendo a alguma placa de prototipagem de baixo custo, dispensando assim a sua construção. Para tal,

recomenda-se a placa "*Arduino Uno*", que vem já equipada com o MCU utilizado e um conversor Série-USB. No entanto, poderá ser necessária alguma alteração no *firmware* para a sua adaptação a essa placa de prototipagem, pelo menos para suportar a sua nova frequência de funcionamento (funciona a *16 MHz*).

O *hardware* criado para o este projecto, utilizado no teste e depuração da implementação proposta, ignora algumas protecções que serão referidas e se estimam recomendáveis na construção de um controlador destinado a ser utilizado por um maior número de utilizadores e em circunstâncias menos controladas.

4.1.1. ARQUITECTURA FUNDAMENTAL DO SISTEMA

Os requisitos fundamentais para a operação do sistema criado são apenas o seu núcleo de processamento, constituído pelo *ATmega328P*, ou similar mediante possível adaptação do *firmware*, e uma interface física que efectue a ligação entre a comunicação série do MCU com o barramento USB do computador. A implementação destes elementos foi conduzida de acordo com a sua aplicação já tradicional (Figura 56):

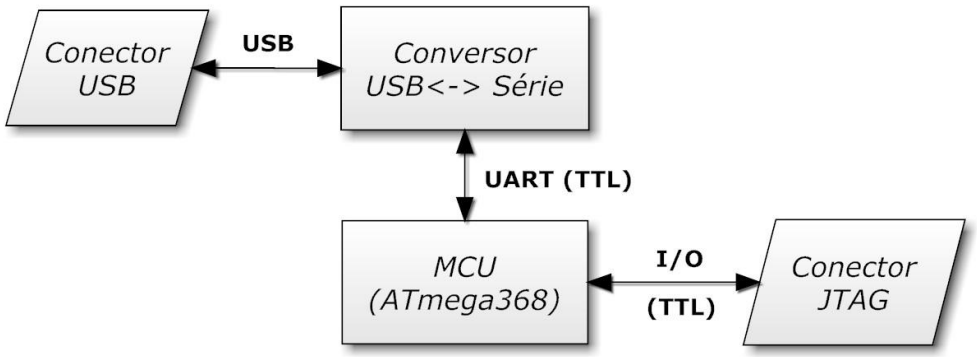


Figura 56 Diagrama dos requisitos mínimos de *hardware* (Controlador BS)

De seguida inicia-se a descrição do controlador pela indicação das características de funcionamento do MCU, seguindo-se das restantes considerações em relação a cada uma das partes que o compõem.

4.1.2. NÚCLEO DE PROCESSAMENTO

As características principais do MCU *ATmega328P* utilizado, seleccionadas de acordo com a sua importância para o actual projecto, são apresentadas na Tabela 7:

Tabela 7 Sumário de características do MCU *ATmega328*[69]

ATmega328P	
Descrição:	Microcontrolador *AVR 8-Bit*
Frequência:	Oscilador Interno: *1 - 8 MHz* Externo: ***0 - 20 MHz***
Memória de Programa (*FLASH*):	*32 Kb*
EEPROM:	*1 Kb*
SRAM:	*2 Kb*
Entradas/Saídas:	*23* linhas de I/O programáveis
Tensão de Alimentação:	***1,8 – 5,5 V***
Comunicações suportadas:	USART; ***UART***; SPI; I2C

A quantidade de linhas de I/O disponíveis é pouco relevante, visto só serem utilizadas as necessárias no conector JTAG. A tensão de alimentação do MCU, que deverá alimentar o restante sistema, depende da sua frequência de funcionamento; e a corrente consumida depende não só da sua frequência mas também da utilização dada aos pinos de I/O, entre outros factores. Como referência, o consumo previsto para este MCU, quando em estado activo a uma frequência de *8 MHz*, alimentado a *5 V*, é de tipicamente *5,2 mA*, não superando os *9 mA*. A utilização de um conversor Série-USB (*UART To USB Bridge*) pressupõe o recurso à comunicação UART do MCU.

4.1.3. INTERFACE DE COMUNICAÇÃO COM O COMPUTADOR

Embora seja descrita apenas a alternativa utilizada no protótipo funcional criado, para efectuar a ligação entre o MCU e o computador, relembra-se da existência de outras opções, como a implementação do sistema *V-USB* (referido na secção *3.4.2*) no *firmware* do controlador. A utilização de um outro conversor Série-USB deverá ser possível, porém recomenda-se a leitura da secção *6.2.1*, que alerta para as possíveis consequências da utilização de outros conversores.

A existência de disponibilidade para experimentar os produtos dos principais fabricantes de conversores Série-USB, permitiu que todos estes componentes (*CP2102*; *FT232R* e *PL2303*) pudessem ser testados com o protótipo funcional desenvolvido, para uma apreciação sobre o seu desempenho e fiabilidade, estando os resultados registados disponíveis no Anexo *D*. A opção que se mostrou mais proveitosa, a nível de velocidade de execução e fiabilidade, foi o componente *CP2102* da *Silicon Labs*. Este conversor, que

respeita a especificação *USB2.0 Full-speed* (*12 MB/s*), mostrou-se fiável em todas as situações e conseguiu o melhor desempenho do teste, ao introduzir os menores atrasos na comunicação. Na Tabela 8 mostram-se várias características eléctricas importantes deste componente:

Tabela 8 Condições eléctricas de operação (*CP2102*)[70]

Parâmetros:	Mín.	Tip.	Máx.	Unidade
Tensão de alimentação recomendada (*VDD*)	*3,0*	*3,3*	*3,6*	*V*
Tensão em qualquer pino *I/O*, *VBUS*, ou *RST*	*- 0,3*	-	*5,8*	*V*
Corrente máxima entre *VDD* e *GND*	-	-	*500*	*mA*
Corrente de alimentação - USB *Pull-up*[14]	-	*200*	*230*	*µA*
Corrente de alimentação – Operação Normal[15]	-	*20*	*26*	*mA*
Corrente de alimentação – Operação Suspensa	-	*80*	*100*	*µA*

Este componente dispõe de um regulador de tensão integrado, através da sua entrada *REGIN*, que regula a tensão de *5 V* do barramento USB para os *3,3 V* com os quais é alimentado. Embora assim operado a *3,3 V*, este componente pode trabalhar em conjunto com o MCU a *5 V* ao suportar tensões até *5,8 V* nos seus pinos. A utilização de sinais de nível TTL nas linhas de transmissão é preferível, por aumentar a imunidade ao ruído e consequentemente a fiabilidade da ligação. Realça-se que a corrente máxima consumida por esta interface é de apenas *26 mA*, importante para a contabilização da corrente consumida pelo sistema.

A comunicação UART do *CP2102* consiste nos pinos de transmissão (*TX*) e recepção (*RX*) de dados, assim como as diversas linhas para controlo de fluxo por *hardware* (*RTS*, *CTS*, *etc.*), que não foram utilizadas. A sua memória de *buffer* dispõe de *576 Bytes* para recepção e *640 bytes* de transmissão. A comunicação UART pode ser programada segundo as especificações apresentadas na Tabela 9:

[14] Corrente consumida nas resistências de *Pull-up*/*down* das linhas *D+* e *D-* do barramento USB.

[15] Corrente fornecida no pino *REGIN*. A este valor deve ser acrescentada a corrente consumida nas resistências de *D+* e *D-*.

Tabela 9 Formatos de tramas e *baud rates* suportados (*CP2102*)[70]

Configuração:	Opções Suportadas:
Data bits:	*5, 6, 7,* ou ***8***
Stop bits:	***1****, 1.5,* ou *2*
Parity type:	***None****, Even, Odd, Mark, Space*
Baud rates:	*300, 600, 1200, 1800, 2400, 4000, 4800, 7200, 9600, 14400, 16000, 19200, 28800, 38400, 51200, 56000, 57600, 64000, 76800, 115200, 128000, 153600, 230400, 250000, 256000, 460800, 500000,* ***576000****, 921600*

No protótipo funcional desenvolvido, para evitar o desenho de uma placa de circuito impresso para utilização deste IC (encapsulamento QFN de *28* pinos), foi utilizado um adaptador Série-USB comercial, apresentado na Figura 57:

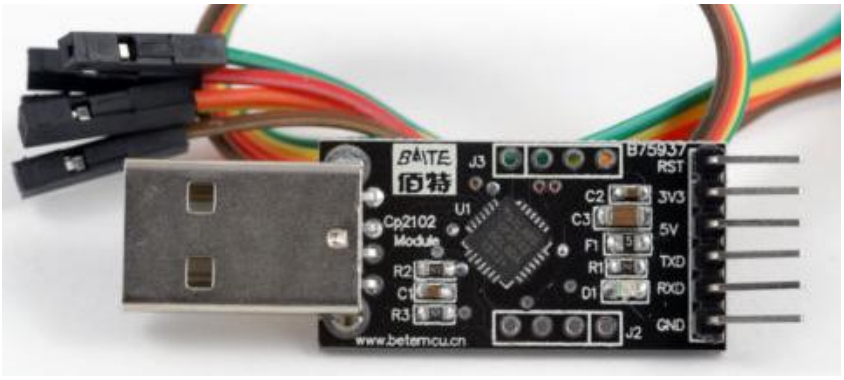


Figura 57 Adaptador Série-USB baseado no IC *CP2102*

Este permite a alimentação da placa do controlador através da energia do barramento USB (*5 V*).

4.1.4. FREQUÊNCIA DE FUNCIONAMENTO E VELOCIDADE DE TRANSMISSÃO

A velocidade de transmissão, numa comunicação série assíncrona (UART), depende da frequência de funcionamento do MCU de acordo com a seguinte equação:

$$BaudRate = \frac{F_{osc}}{Div \cdot (UBRR + 1)}$$

Onde o divisor "*Div*" toma o valor de *16*, em funcionamento normal, ou de *8*, no modo de duplicação da velocidade de transmissão; e "UBRR" é o valor do registo "*USART Baud Rate Register*", que tem de ser um número inteiro de *12 bits* (*0 - 4095*) e serve para determinar a *baud rate* pretendida.

Utilizou-se, em conjunto com o MCU, um oscilador externo de *18,432 MHz*, por estar próximo da frequência máxima suportada por este MCU (*20 MHz*) e permitir uma grande compatibilidade com muitos valores típicos de *baud rate*. Pois note-se, partindo da equação anterior, e considerando o modo de operação normal:

$$BaudRate = \frac{18.432 \cdot 10^3}{16 \cdot (UBRR + 1)}$$

$$UBRR = \frac{18.432 \cdot 10^3}{16 \cdot BaudRate} - 1$$

Os resultados, não arredondados e consequentemente com um erro nulo, de UBRR para diversas *baud rates* típicas são listados na seguinte Tabela 10:

Tabela 10 Valores de *baud rate* e respectivo UBRR com erro de *0%* (*18,432 MHz*)

Baud rate	***UBRR***
300	*3839*
4800	*239*
9600	*119*
19200	*59*
57600	*19*
115200	*9*
230400	*4*
570600	***1***

É prevista a utilização da velocidade de *570600 baud*, valor máximo permitido no modo de operação normal para esta frequência de relógio.

A referida frequência de funcionamento de *18,432 MHz* impede que o MCU opere a outros níveis que não TTL (*5 V*), como verificado na Figura 58:

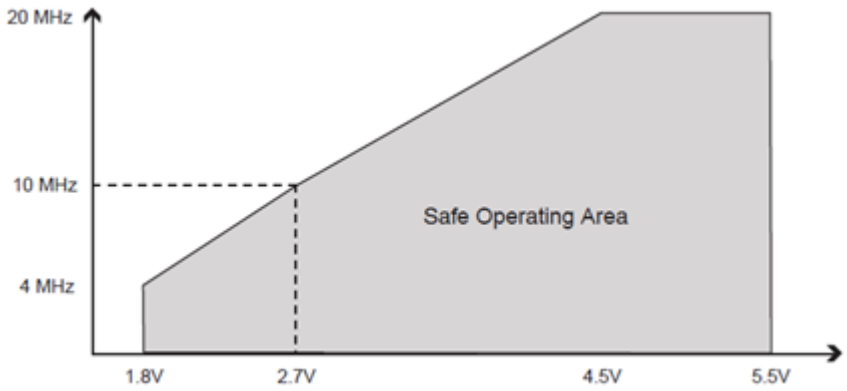


Figura 58 Tensão de alimentação x Frequência (*ATmega328P*)[69]

4.1.5. CONFIGURAÇÃO DO MCU

A modificação da configuração de fábrica dos *fuses* do *ATmega328P* é um requisito para que este funcione com o oscilador externo escolhido. Apenas o valor do "*fuse low byte*", que diz respeito à configuração da fonte de relógio do MCU, necessita de ser modificado, passando para o valor hexadecimal *0xF7*, que define um oscilador externo do tipo *Full-Swing* e um atraso na inicialização de *65 ms*. Os restantes parâmetros podem ser mantidos nos valores padrão, como se resume na Tabela 11:

Tabela 11 Valores da programação dos *fuses* do *ATmega328P*

***Fuse*:**	**Valor: (*hex*)**
Low Byte	*0xF7*
High Byte	*0xD9*
Extended Byte	*0xFF*

4.1.6. LIGAÇÃO JTAG

No conector JTAG foram implementadas as linhas: *TCK*; *TMS*; *TDI*; *TDO*; *SRST*; *VCC*; e *GND*. A linha *System reset* (SRST), que poderá ter passado desconhecida até este ponto, é de utilização usual em controladores de BS e serve para efectuar *reset* ao sistema alvo controlando a linha de *reset* do MCU, diferenciando-se de *TRST*, definido na norma JTAG, que apenas provoca o *reset* do controlador TAP do dispositivo alvo[65].

De notar a ausência da linha de *TRST*, opcional segundo a norma. Esta foi preterida na implementação do controlador pelo seu intuito ser tão facilmente atingido utilizando-se as linhas de *TMS* e *TCK* para efectuar o reset ao controlador TAP alvo (são necessários, em qualquer estado do controlador TAP alvo, um máximo de cinco ciclos de *TCK*, mantendo-se a linha *TMS* com o valor lógico '*0*'). Pretendendo-se acrescentar também a linha de *TRST*, poderá ser utilizado o pino *PD7*, actualmente disponível, e programada no *firmware* do controlador e biblioteca de controlo tal como a linha de *SRST*, servindo como referência as descrições do *software* conduzidas na secção *5.1* e *5.2*.

A disposição das linhas no conector foi organizada de acordo com o *standard* utilizado pela *Atmel*, como já havia sido indicado e justificado (secção *3.4.3*), que está de acordo com a Figura 59:

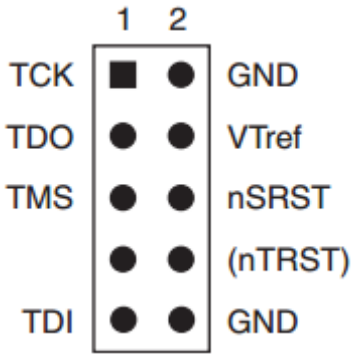


Figura 59 Conector *Atmel JTAG10PIN* [71]

No sistema desenvolvido, a linha de "*VTref*" permite alimentar a placa alvo. No caso de não se desejar alimentar a placa alvo, quer porque esta possa requerer mais corrente do que a disponibilizada no barramento USB ou por funcionar com níveis de tensão inferiores a *5 V*, esta linha deve ser obrigatoriamente interrompida, tendo sido colocado, no protótipo funcional, um *jumper* para o efeito.

4.1.7. ALIMENTAÇÃO DO SISTEMA

Ao longo desta descrição foram sendo referidos os valores dos consumos expectáveis de cada componente. Tendo em conta esses valores, para reduzir os componentes necessários e também tornar mais prática a utilização do controlador, a alimentação do *ATmega328P* será proveniente do barramento do USB.

Na Tabela 12 apresentam-se as características eléctricas do barramento USB, de acordo com a sua versão, no que concerne à sua capacidade de alimentação do controlador.

Tabela 12 Energia dos barramentos USB

Especificação	Corrente Máx.	Tensão	Potência Máx.
USB1.0	*150 mA*	*5 V*	*0.75 W*
USB2.0	*500 mA*	*5 V*	*2.5 W*
USB3.0	*900 mA*	*5 V*	*4.5 W*

Verifica-se que, ignorando os já raros barramentos *USB1.0*, é esperado que seja disponibilizada, pelo menos, até *500 mA* de corrente, cujo funcionamento do MCU não excederá por si só, nem em conjunto com o conversor Série-USB escolhido. A utilização desta fonte de alimentação pode ainda assim não ser a melhor opção ao pretender-se alimentar uma placa alvo, de consumo desconhecido, através da linha *VTref*. Assim, nesse

caso, o utilizador deverá ter os valores máximos de fornecimento de corrente em consideração.

O isolamento das tensões de alimentação do controlador e da placa de testes, algo preterido no protótipo funcional, poderia garantir um mais adequado suporte para placas com tensões de alimentação diferentes do controlador (que, relembre-se, actua com níveis TTL sobre as placas de teste). Para tal, poderá ser utilizado algum *buffer*/*driver* que isole os níveis da alimentação do controlador dos da placa alvo, como, por exemplo, através do componente *74ACT244* (*Octal Buffer/Line Driver* de três estados) ou *6N137* (*optocoupler* de um canal único, para cada uma das linhas JTAG). De acordo com as suas especificações, estes componentes deverão efectuar as transições dos seus estados lógicos de forma suficientemente célere para acompanhar os requisitos do sistema. O emprego de um circuito com esta função permitiria assim ao sistema actuar as placas alvo independentemente da sua compatibilidade com níveis de 5 *V*, e melhorar a sua imunidade ao ruído.

Todas as linhas de controlo são, tal como foram implementadas, directamente alimentadas pelos I/Os do MCU, pelo que nunca deverão fornecer mais do que *40 mA* (valor máximo suportado pelo *ATmega328P*)[69].

4.1.8. PROTÓTIPO FUNCIONAL

Reforçando-se a ideia de que esta não é, necessariamente, a única alternativa, nem a mais completa, para a implementação do sistema desenvolvido, o protótipo funcional do controlador construído, com o qual se procedeu aos testes finais do sistema, utiliza o esquema eléctrico da seguinte Figura 60:

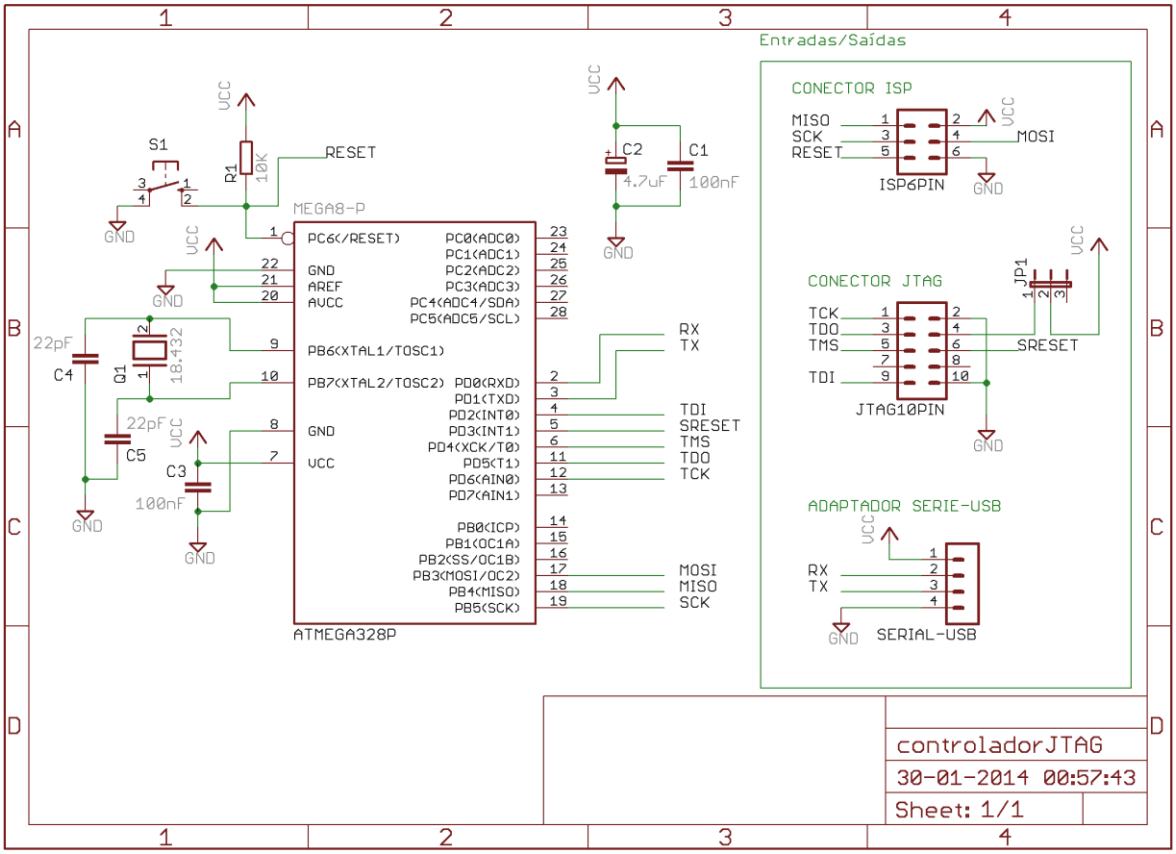


Figura 60 Circuito esquemático do controlador de *Boundary Scan*

Este circuito considera os pontos referidos ao longo deste capítulo assim como as recomendações do fabricante para a operação deste MCU. Na placa do controlador do protótipo funcional juntou-se, ao circuito apresentado, também um programador ISP – de nome *USBasp* – que está descrito no 0. A alimentação deste, proveniente do barramento USB do seu conector independente, foi mantida separada da do circuito do controlador, fazendo com que o programador apenas seja alimentado quando utilizado o seu respectivo conector. O conector ISP do controlador, apresentado no circuito da anterior Figura 60, está ligado directamente ao dito programador. Desta forma, a programação do *firmware* do controlador não necessita de qualquer equipamento adicional.

O material utilizado na placa do controlador, e referido programador, utiliza encapsulamento do tipo THT e foi montado numa placa perfurada. Na Figura 61 apresenta-se uma ilustração representativa do protótipo funcional desenvolvido:

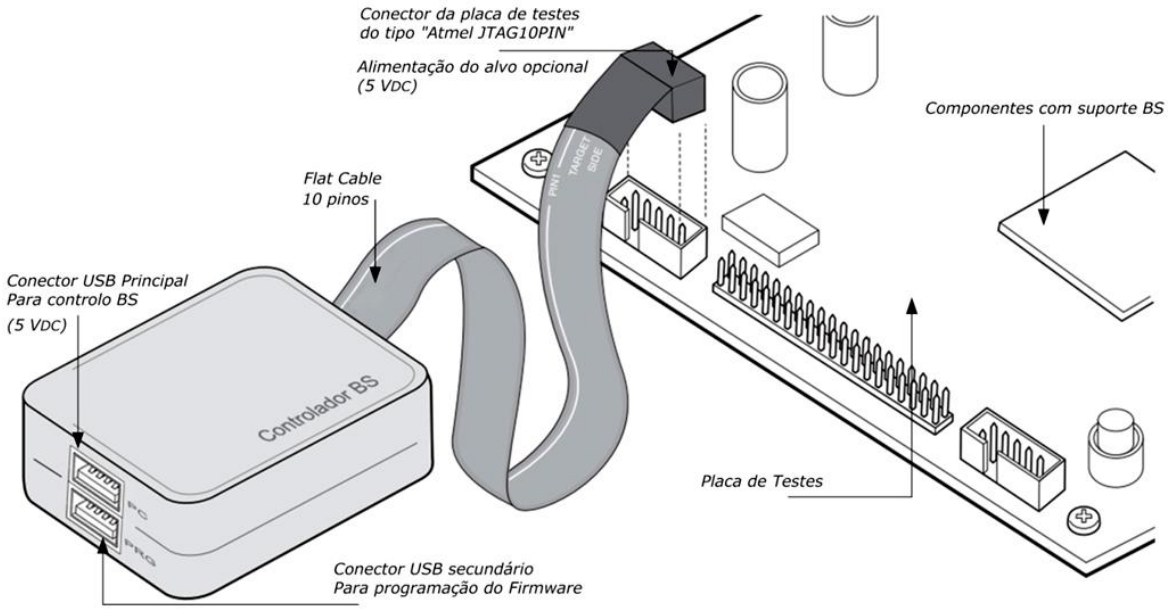


Figura 61 Representação do protótipo funcional do controlador BS

Imagens reais do protótipo desenvolvido, assim como uma descrição algo mais detalhada sobre a montagem efectuada, encontram-se remetidas para o 0.

Embora a interface USB para controlo BS tenha sido adquirida para este projecto, é possível o seu desenho e integração na placa do controlador de acordo com o esquemático da Figura 62:

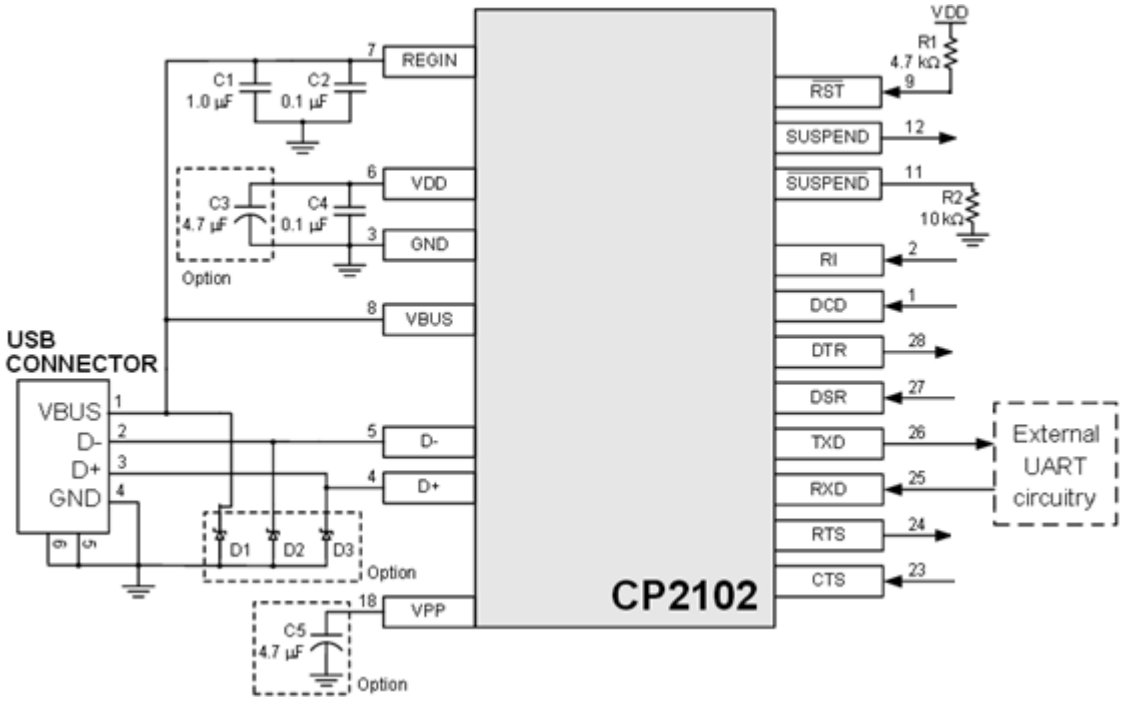


Figura 62 Aplicação recomendada do componente *CP2102* (interface USB) (adaptado de [70])

Os componentes destacados, circundados por linhas descontínuas, são de carácter opcional. Das linhas que comunicam com a placa do controlador, apenas foram utilizadas *RXD*, *TXD*, e pinos de alimentação (*VDD* e *GND*).

4.1.9. LISTAGEM DE MATERIAL

Na Tabela 13 listam-se os componentes utilizados no controlador de BS:

Tabela 13 Listagem do material utilizado no Controlador BS (sem programador)

Nome de Ref.:	Componente:	Unid.:	Valor p/unid.[16]:	Valor Total:
ATMEGA328P	*ATMEGA328P-20PU*	*1*	*3,15€*	*3,15€*
C1, C3	*100nF MLCC 50V*	*2*	*0,035€*	*0,07€*
C2	*4.7uF ELEC 50V*	*1*	*0,06€*	*0,06€*
C4, C5	*22pF MLCC 50V*	*2*	*0,025€*	*0,05€*
ISP6PIN (3x2), JP1 (3x1), JTAG10PIN (5x2), SERIAL-USB (4x1)	*Male Header 2.54mm (12pin)*	*2*	*0,55€*	*1,10€*
Q1	*Cristal Std. 18.432MHz HC-49*	*1*	*0,28€*	*0,28€*
R1	*10 KΩ 1/4 W*	*1*	*0,008€*	*0,008€*
S1	*Tactile SPST MC32828*	*1*	*0,105€*	*0,105€*
			Total:	*4,83€*

O material indicado diz apenas respeito aos componentes constantes do circuito do controlador, apresentado na Figura 60, e não contempla:

- Placa perfurada (*e.g. Roth Elektronik RE520-HP*, *100x160mm*, *4,08€*);
- Adaptador Série-USB baseado no *CP2102* (*CP2102 USB2.0 to UART TTL 6PIN Module, 1,88€*[17]);
- Programador *USBasp* – opcional (*5,63€*);
- *Flat Cable* para os sinais JTAG;

[16] Preços do distribuidor *Farnell*, em Portugal, à data de 28 de Fevereiro de 2014, sem contabilização de portes de envio.

[17] Preço em mercado online de venda livre (*Ebay*), incluindo portes de envio.

- Componentes opcionais preteridos, indicados na secção *4.1.7*.

Adicionalmente, disponibiliza-se no 0 o projecto de um PCB para a montagem do controlador, que incluí já a interface USB e dispensa assim o adaptador exerno.

4.2. PLACA DE TESTE

Na segunda placa desenhada, referida como placa de teste ou placa alvo, foi implementado o sistema controlável por BS utilizado durante todo o desenvolvimento e teste do controlador. Esta serve como apoio ao desenvolvimento e depuração do sistema, requerendo somente o estabelecimento de uma cadeia de BS de vários componentes controláveis, e da disponibilização das suas entradas/saídas para auferir o seu correcto controlo por BS.

4.2.1. DESCRIÇÃO DA CADEIA DE *BOUNDARY SCAN*

Inicialmente foi apenas utilizado o MCU *ATmega16* como dispositivo alvo, tal como havia sido previsto numa primeira fase do projecto, servindo este de único componente de teste. Com o avançar do seu desenvolvimento, e opção pela introdução de ferramentas para controlo de múltiplos dispositivos, foi necessária a adição de pelo menos outro componente controlável por BS, para a formação de uma cadeia de BS. Esta cadeia de BS ficou constituída por dois elementos: o MCU *ATmega16* e o dispositivo de teste *SN74BCT8245A*, e segue a arquitectura apresentada na Figura 63:

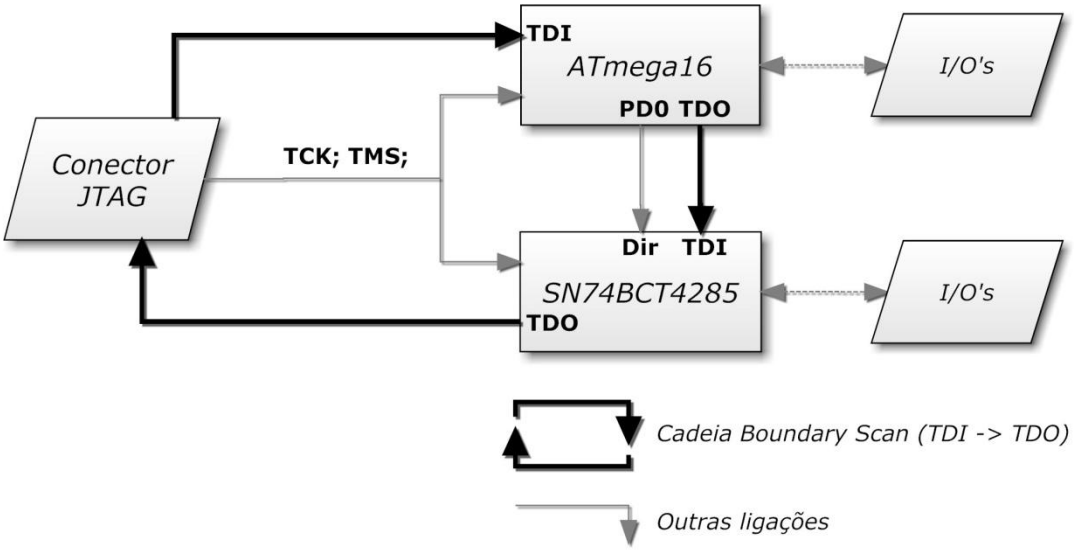


Figura 63 Diagrama da organização da placa de teste

Para a verificação do correcto funcionamento do controlador em cadeias de vários dispositivos não se entendeu ser preciso mais do que esta curta cadeia de dois elementos.

4.2.2. DESCRIÇÃO DOS COMPONENTES DE TESTE

Descrevem-se algumas peculiaridades relativas à operação de cada um dos controladores TAP dos componentes alvo utilizados:

O MCU *ATmega16*, para além do registo IR, que aceita instruções com um comprimento de *4 bits*, tem os seguintes TDRs: *Boundary-scan Register*; *Bypass Register*; *Reset Register*; e *Device Identification Register*.

O registo BSR tem um comprimento de *144 bits* e permite o varrimento da totalidade dos seus pinos. O registo de *reset* é um registo adicional de que dispõe, e é acedido através da instrução *AVR_RESET* ("*1100*"), permitindo fazer *reset* ao MCU de forma similar à activação da sua linha de *reset*. As restantes instruções aceites são: *EXTEST* (*"0000"*); *IDCODE* (*"0001"*); *SAMPLE/PRELOAD* (*"0010"*); e *BYPASS* (*"1111"*). Os códigos de identificação do MCU, segundo o registo de identificação, são apresentados na Tabela 14:

Tabela 14 Códigos de identificação JTAG do *ATmega16*[72]

Fabricante:	**Identificação do Fabricante: (*hex*)**
ATMEL	*0x01F*
Dispositivo:	**Identificação do dispositivo *AVR*: (*hex*)**
ATmega16	*0x9403*
Versão:	**Identificação da versão: (*hex*)**
Revision A – L	*0x00 – 0x0B*

A leitura dos valores indicados pode ser verificada no protótipo funcional, recorrendo por exemplo à função elementar para leitura de códigos de identificação de componentes, do *firmware* do controlador desenvolvido (como se mostra mais à frente, na Figura 67, pág. 121).

O componente *SN74BCT8245A* é um *octal transceiver*[18] com capacidades BS para apoio ao teste de circuitos. Este permite o controlo, por BS, das suas saídas, e leitura das suas entradas, de forma a apoiar a realização de procedimentos de teste no circuito da placa em que está inserido. É composto por oito canais, que ligam os pinos *Ax* ao seu respectivo par *Bx*. A cada um dos pinos físicos *Ax* e *Bx* está associada uma célula do registo de BS. A Figura 64 mostra a representação de um dos canais descritos:

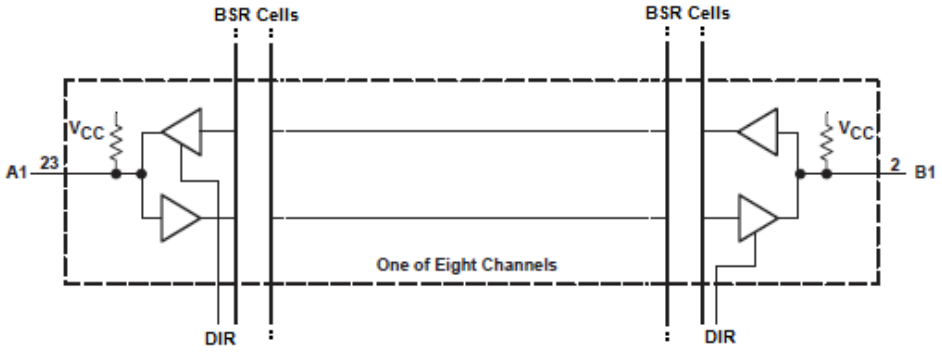


Figura 64 Esquematização de um canal do *SN74BCT8245A* (adaptado de [73])

A direccionalidade dos canais, e consequentemente quais dos seus pinos servem como entrada ou saída, é definida através da linha de direccionalidade (*DIR*), de acordo com a seguinte Tabela 15:

Tabela 15 Estado de operação do *SN74BCT8245A*[73]

Entrada:		Operação:
OE	***DIR***	
L	***L***	De *Bx* para *Ax*
L	***H***	De *Ax* para *Bx*

Este componente contém um registo de instruções de *8 bits* de comprimento, um BSR de *18 bits*, e não dispõe de registo de identificação.

4.2.3. CIRCUITO ELÉCTRICO DA PLACA DE TESTE

A alimentação desta placa foi efectuada recorrendo a um adaptador AC de *5 V* conectado através de um cabo USB, preterindo-se desta forma a utilização da alimentação proveniente do controlador BS. Ambos os componentes são alimentados directamente

[18] *Octal Transceiver* refere-se a um circuito integrado de recepção e transmissão de sinais digitais composto por oito canais.

através desta tensão de alimentação. O *ATmega16* foi implementado de acordo com a montagem recomendada pelo fabricante, utilizando o seu oscilador interno como fonte de relógio. Dos seus pinos I/O, escolheu-se deixar o porto *A* (*PA0-PA7*) fisicamente acessível para verificação do controlo de BS. No caso do componente *SN74BCT8245A*, os pinos *A1-A8*.

O circuito eléctrico da placa alvo corresponde ao representado na Figura 65:

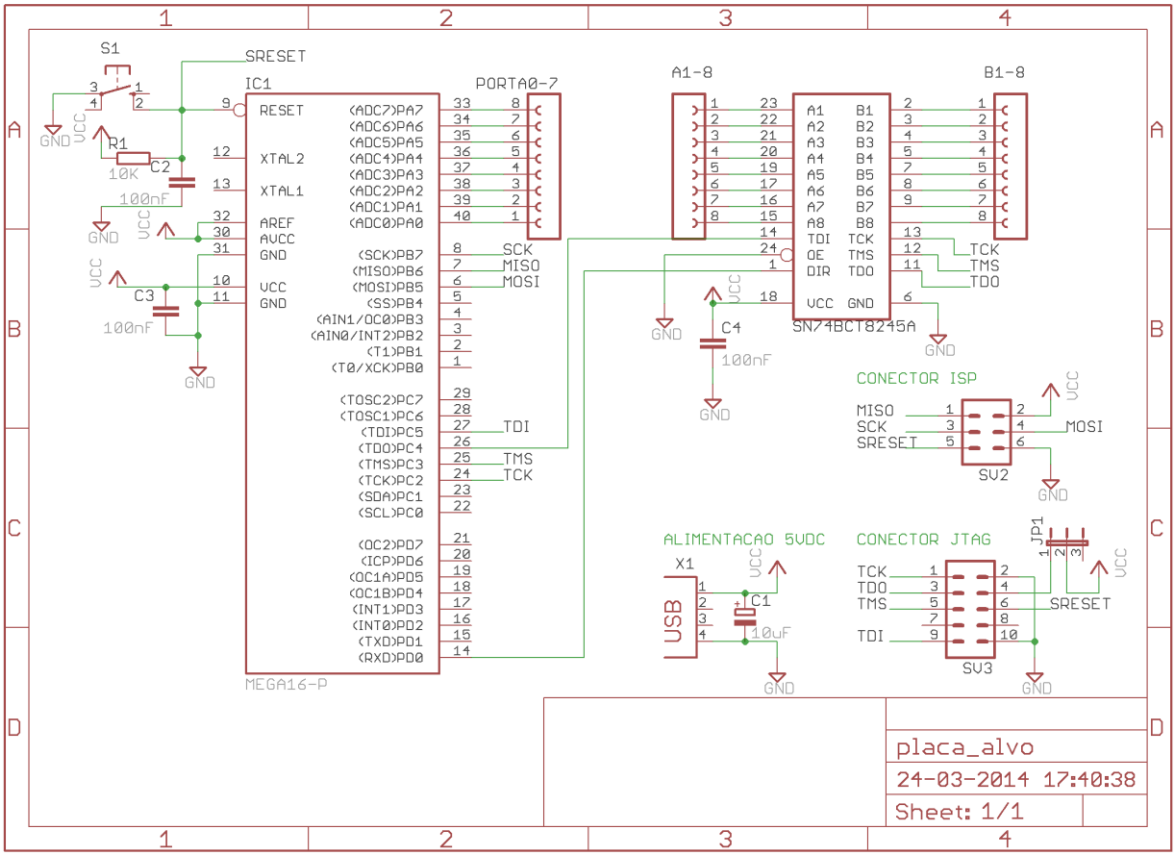


Figura 65 Circuito esquemático da placa de teste

De notar que a linha de *reset* do MCU está ligada ao sinal de *SRST* do conector JTAG, permitindo que o controlador possa impor o *reset* do sistema. O MCU *ATmega16* controla a direccionalidade (entrada *DIR*) do dispositivo de teste (*SN74BCT8245A*) através da actuação da saída *PD0*. Da forma como foram ligados, o valor lógico '*1*' deverá ser atribuído ao pino *PD0* quando o porto *A* se destinar à leitura de valores de entrada e o porto *B* (que suporta correntes superiores) actuar como saída. Foi utilizado, em conjunto com esta placa de teste, uma *breadboard*, que incluí um conjunto de LEDs, destinada à imposição de valores nos *inputs* da placa de teste e observação dos seus valores de *output*,

servindo-se dos conectores para os pinos de I/O do MCU e *SN74BCT8245A*. Os valores impostos deverão poder ser lidos recorrendo ao controlador de BS.

4.2.4. LISTAGEM DE MATERIAL

Todos os componentes utilizados foram escolhidos em encapsulamento THT e montados numa placa perfurada. Estes são listados na Tabela 16:

Tabela 16 Listagem do material utilizado na placa alvo

Nome de Ref.:	Componente:	Unid.:	Valor p/unid.:	Valor Total:[19]
C1	*10uF ELEC 50V*	*1*	*0,06€*	*0,06€*
C2, C3, C4	*100nF MLCC 50V*	*3*	*0,035€*	*0,105€*
IC1	*ATMEGA16A-PU*	*1*	*4,85€*	*4,85€*
R1	*10 KΩ 1/4 W*	*1*	*0,008€*	*0,008€*
S1	*Tactile SPST MC32828*	*1*	*0,105€*	*0,105€*
SN74BCT8245A	*SN74BCT8245A*	*1*	*n.d.*	*n.d.*
SV2 (3x2), SV3 (5x2), JP1 (3x1), A1-8 (8x1), B1-8 (8x1), PORTA0-7 (8x1)	*Male Header 2.54mm (12pin)*	*4*	*0,55€*	*2,20€*
X1	*Conector USB3.0 A THT*	*1*	*0,41€*	*0,41€*
			Total:	*7,74€*

O componente *SN74BCT8245A*, à data da compilação desta listagem, apenas se encontra disponível no formato *Small Outline Integrated Circuit* (SOIC), tendo-se utilizado o componente em encapsulamento PDIP. A esta lista de material deverá acrescer a placa perfurada onde os componentes listados foram montados, e o adaptador AC de entrada USB e respectivo cabo, que alimenta o sistema.

[19] Preços do distribuidor *Farnell*, em Portugal, à data de 28 de Fevereiro de 2014, sem contabilização de portes de envio.

5. DESCRIÇÃO DO SOFTWARE

Utilizando um *hardware* cuja complexidade é reduzida, servindo-se de uma solução simples e genérica, *i.e.* não vocacionada especificamente para controlo de BS, não surpreende que a maior incidência de trabalho no desenvolvimento do projecto se prenda na sua programação, a fim de se atingir as funcionalidades desejadas.

A descrição da componente de programação do projecto dá especial relevância a matérias das quais esteja dependente uma futura evolução do sistema proposto. Nesse sentido, tenta-se facultar informação sobre os passos necessários para a implementação ou modificação de funcionalidades, quando aplicável; assim como sobre as limitações do sistema actual, tecendo-se considerações sobre possíveis melhorias ou diferentes abordagens possíveis.

Inicia-se a exposição da programação pela descrição do *firmware* do controlador, seguido do programa da biblioteca dinâmica de controlo e, por último, do programa da interface gráfica. Para além das considerações aqui registadas, o código redigido para cada um dos programas está acompanhado por comentários que se espera que facilitem a sua interpretação.

5.1. *FIRMWARE* DO CONTROLADOR DE *BOUNDARY SCAN*

A programação do *ATMega328P* utilizado no controlador teve como objectivo principal a implementação de funções básicas de escrita e leitura do valor dos pinos do conector JTAG. Para além disso, não se descorou a inclusão de alguns outros comandos adicionais cuja utilidade se entendeu verificada, assim como algumas funções orientadas apenas para a depuração do funcionamento do controlador.

O código do *firmware*, escrito em linguagem *C*, resume-se ao conjunto de ficheiros apresentados na Figura 66:

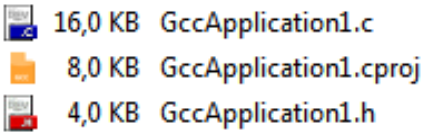


Figura 66 Listagem dos ficheiros do programa do *firmware*

Como parâmetros do compilador (*Atmel Studio 6.1*), é de referir apenas que foi utilizado o nível *2* de optimização do programa ("*Optimize More*"), embora tenha sido verificado (secção 6.2.1) que não existem diferenças significativas em termos de desempenho em relação aos outros níveis. Com esta optimização, o programa requer a seguinte memória:

```
Program Memory Usage:    6316 bytes  19,3 % Full
Data Memory Usage:        697 bytes  34,0 % Full
```

Embora estes valores estejam bem dentro das capacidades do MCU seleccionado e até de outras soluções inferiores, verificou-se que o programa não funcionava correctamente quando se aproximava dos *50%* de ocupação da "*Data Memory*", provocando *resets* espontâneos que indiciavam algum *overflow* de memória. Acredita-se que tal resulte, especialmente, da utilização de algumas funções complexas para tratamento de *strings* formatadas (*e.g.* "*printf*", "*strtok*", *etc.*), cujas necessidades de memória SRAM livre deverão ser consideráveis. Esta situação, que pode ser facilmente recriada, causa alguma apreensão quanto à possibilidade de se utilizar este programa em outros MCUs *AVR* de memória inferior.

5.1.1. FUNCIONAMENTO GERAL

O programa é orientado para a sua utilização preferencial através do *software* da biblioteca dinâmica programada para o efeito (secção 5.2), mas oferece também a possibilidade de ser directamente operado pelo utilizador através de um terminal. Este facto justifica a criação de, em certas situações, duas funções para o mesmo efeito, uma para que se

obtenha o melhor desempenho ao ser executada pelo *software* de controlo, e outra, complementar, que permita uma execução facilitada pelo utilizador através de uma ligação série UART por terminal. O controlo directo do controlador permite o seu *debug* de forma mais rápida por ser independente do *software* do computador e dos seus eventuais problemas. A seguinte Figura 67 exemplifica a utilização do controlador, através de um terminal, para procurar e listar os componentes da cadeia BS da placa alvo:

```
-==  Controlador Boundary Scan  ==-

 DEBUG Reset: 0

Continuar...

Comando: id
Registo de ID

  > LSB: 0
  -- Disp. encontrado s/IDCODE --

  > LSB: 1
  > Manufacturer_ID: 0x001F
  > Part_Number: 0x9403
  > Version: 0x0000
  Scan terminado
```

Figura 67 Leitura dos *IDCodes* da placa de teste via terminal

As funcionalidades implementadas no controlador são executadas como resposta a pedidos provenientes da comunicação UART com o computador. Na sua generalidade, têm como objectivo a alteração dos estados dos pinos do conector JTAG, tendo sido deixada para o computador, como já referido, a maioria das tarefas, mais complexas, que levam à necessidade de alterar cada um dos pinos da porta TAP. Distinguem-se quatro grupos de operações implementadas, que serão posteriormente descritos:

- Controlo elementar dos sinais JTAG;
- Operações automáticas no controlador TAP alvo;
- Depuração e controlo via terminal;
- Funções complementares requeridas pela biblioteca de controlo.

Esta selecção de funcionalidades, implementadas no controlador, pretende dar resposta ao funcionamento estabelecido na biblioteca de controlo, cuja descrição se encontra na posterior secção 5.2.2.

O funcionamento base do programa pode ser sumariamente descrito pelo fluxograma da Figura 68:

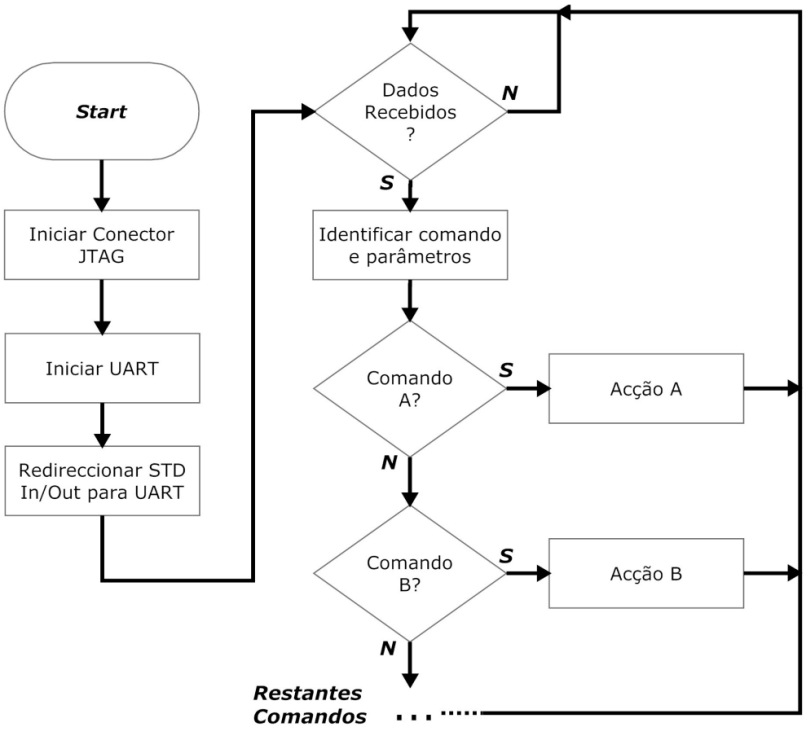


Figura 68 Fluxograma do funcionamento geral do *firmware* do controlador de BS

A descrição do *firmware* segue a ordem dos blocos enunciados. O objectivo de cada comando será especificado mais em diante (secção 5.1.5).

5.1.2. PARÂMETROS PARA ALTERAÇÕES DE HARDWARE

O ficheiro *header* "*GccApplication1.h*" contém várias definições que identificam as ligações físicas do MCU do controlador. Estas definições podem ser alteradas para comportar mudanças no esquema eléctrico de ligações do MCU, pelo que se entende de interesse enunciá-las.

```
// Porto e pinos da ligação UART:
#define PORTUART   PORTD
#define DDRUART    DDRD
#define PRXI       0
#define PTXI       1
// Porto e pinos do conector JTAG:
#define PORTTAP    PORTD
#define DDRTAP     DDRD
#define PINTAP     PIND
#define PTDI       2
#define PRESET     3
#define PTMS       4
#define PTDO       5
#define PINTDO     PINC5
#define PTCK       6
```

Estas definições respeitam o circuito eléctrico utilizado, apresentado na Figura 60 (pág. 110).

5.1.3. COMUNICAÇÃO UART

A comunicação série foi implementada utilizando as características que constam da seguinte Tabela 17:

Tabela 17 Características da comunicação série

Comunicação Série	
Tipo:	Assíncrono (UART)
Baud rate:	*576000 bd*
Data Bits[20]:	*8 bits*
Paridade:	Não
Stop Bits[21]:	*1 bit*

Esta configuração define o envio de *10 bits* (impulsos não necessariamente úteis, que incluem o *stop* e *start bit*[22]) para cada *8 bits* de informação útil enviada, o que significa, considerando a *baud rate* de *576000 bd*, a seguinte taxa de transferência útil:

$$BaudRate \cdot \frac{8\ bits}{10\ bits} = 460800\ bits/s$$

$$\frac{460800\ bits/s}{8 \cdot 1024} = 56{,}25\ Kbytes/s$$

O tempo dispendido nas transferências estará também dependente dos atrasos introduzidos pelo conversor Série-USB.

A *baud rate* definida pode ser alterada ainda no ficheiro "*GccApplication1.h*" editando as seguintes linhas de código:

```
// Frequência do MCU (Hz):
#define FREQ 18432000
// Baud rate do UART:
#define BAUD 576000
```

[20] *Data Bits* são os *bits* de informação útil transferidos em cada trama de transmissão.

[21] *Stop Bits* são os *bits* indicadores do final da trama de transmissão.

[22] *Start Bit* é o *bit* indicador do início da trama de transmissão.

A frequência de funcionamento do MCU, expressa em *Hertz* e denominada, no excerto apresentado, por "FREQ", é mandatória, pois participa no cálculo do valor de *USART Baud Rate Register* (UBRR). Recomenda-se, considerando a frequência de *18,432 MHz* utilizada no *hardware* proposto, a utilização de qualquer *baud rate* típica múltipla de *300* (*e.g. 76800*; *115200*; *230400*; *etc.*) que permitem um erro de *0%* no cálculo do valor de UBRR. A influência das diferentes *baud rates*, assim como o impacto dos tempos dispendidos na comunicação, no desempenho do sistema foi verificada nos testes descritos na secção 6.2.1.

Como as operações JTAG executadas pelo controlador não devem ser interrompidas, não é utilizada qualquer *Interrupt Service Routine* (ISR) associada à comunicação série. O programa apenas verifica a recepção de algum comando quando em estado de espera, pelo que o envio de uma rápida sucessão de comandos para o controlador levará à criação de uma fila de espera de comandos, que, no entanto, em todos os testes efectuados, nunca mostrou causar qualquer problema relacionado com o esgotamento da memória dos *buffers* de transmissão. Assim sendo, não se verificou qualquer necessidade de implementação de um sistema de controlo de fluxo, fosse por *software* ou *hardware* (a colecção de erros de transmissão no controlador dá segurança quanto à inexistência de comandos corrompidos nos referidos testes, indicando que não são perdidas ou corrompidas quaisquer tramas de transmissão). Como segurança adicional, no caso da biblioteca de controlo detectar algum erro na resposta esperada a algum comando, esta está preparada para aguardar um curto espaço de tempo, que serve para o esvaziamento de algum possível *buffer* esgotado (do MCU ou conversor Série-USB), e posteriormente reenviar o comando anteriormente falhado (mecanismo descrito em maior detalhe na secção 5.2.4).

A transmissão e recepção de informação série foram associadas ao *Standard Input* (*stdin*) e *Standard Output* (*stdout*) do programa:

```
(Em GccApplication.c)
static FILE mystdout =
FDEV_SETUP_STREAM(uart_putchar_stdout,NULL,_FDEV_SETUP_WRITE);
static FILE mystdin =
FDEV_SETUP_STREAM(NULL,uart_getchar_stdin,_FDEV_SETUP_READ);
(...)
stdout = &mystdout;
stdin = &mystdin;
```

Onde as funções "*uart_putchar*" e "*uart_getchar*" indicadas para o *stdout* e *stdin* estão encarregues de efectuar a transmissão e recepção, respectivamente, da informação. Deste

modo, todo o envio e recepção de dados ao longo do programa puderam ser efectuados recorrendo a funções típicas, no caso: "*printf*"; "*puts*"; "*putchar*" e "*getchar*".

5.1.4. IMPLEMENTAÇÃO DE COMANDOS

A implementação da recepção e interpretação dos pedidos recebidos no *firmware* suporta comandos de comprimentos variáveis, com ou sem argumentos. O comprimento máximo de um comando, assim como a quantidade máxima de argumentos que poderão vir associados, podem ser facilmente controlados recorrendo à alteração das seguintes linhas de código:

```
(Em GccApplication.h)
#define TAM_MAX_LINHA 255 // Tam. máx. cmd + args
#define CMD_MAX_CHAR 4    // Núm. máx. caracteres
```

```
(Em GccApplication.c)
static const unsigned int NUM_MAX_ARG = 10;
```

Os comandos utilizados são indicados numa lista, tal como apresentado no seguinte excerto (ficheiro "*GccApplication1.c*"):

```
typedef enum {
   cycle_TCK,
   escrever_TDI_0,
   (...) //Restantes comandos

   RETURN,          // Caso de comando vazio
   Numero_opcoes    // Tamanho da lista
} Lista_cmds;
```

Novos comandos deverão ser adicionados a esta lista, mas nunca deverão tomar a sua última posição, que é utilizada para conhecer a quantidade total de comandos.

Respeitando a mesma ordenação da lista anterior, é mantida uma matriz que associa a cada comando a sua codificação:

```
static const char
instrucoes_cmds[Numero_opcoes][CMD_MAX_CHAR + 1]
= {
   [cycle_TCK] = "K",
   [escrever_TDI_0] = "P",
   (...) //Restantes comandos

   [RETURN] = "",
};
```

Como o algoritmo de interpretação de comandos tem de procurar sequencialmente na matriz a tradução de cada comando recebido, ambas as listas, que, como referido, têm de partilhar a mesma ordem, foram ordenadas de forma a que os comandos com maior

prioridade, *i.e.* cuja velocidade de execução mais influencie o desempenho final do sistema, ocupem os primeiros lugares. Foram também utilizados comandos menores, de apenas um caracter, como os apresentados no excerto anterior, na codificação dos comandos mais vezes utilizados, com o intuito de reduzir o fluxo de informação a circular na comunicação entre o controlador e o computador. Ambas as medidas referidas foram implementadas numa tentativa de prioritização de comandos críticos para o desempenho do sistema, e o resultado conseguido é comentado na secção 6.2.1.

O código a executar em cada um dos comandos é introduzido como um caso do "*switch*" principal do programa. Neste já não existe necessidade de respeitar a ordenação anterior, mas foi mantida a opção de colocar as funções prioritárias em primeiro lugar.

```
if (n != -1 && comando <= Numero_opcoes) {
        switch (comando) {
              case cycle_TCK:
              (...)
                    break;
              (...) //Restantes comandos
              }
```

Sobre o algoritmo de leitura dos comandos e colecção dos argumentos, que se pensa de descrição dispensável, refere-se apenas que necessita que todos os comandos sejam, obrigatoriamente, terminados pelo caractere '\r' e que todos os argumentos sejam numéricos. Os comandos não são *case-sensitive*. Os argumentos de cada comando são armazenados no vector declarado como:

```
int arg[NUM_MAX_ARG];
```

A implementação de novos comandos no código, dependentes ou não de qualquer argumento, carece apenas do seguimento dos passos indicados.

5.1.5. LISTAGEM DOS COMANDOS E FUNÇÕES

No controlador foram implementados o seguinte conjunto de comandos, listados na Tabela 18, aos quais se junta uma descrição sumária:

Tabela 18 Listagem dos comandos implementados no *firmware*

Nome	Comando	Argumentos	Descrição
Escrever_TAP	*WT*	*[pino#] [valor#] …*	Escreve pino seleccionado do conector JTAG
Ler_TAP	*RT*	*[pino#] …*	Lê pino seleccionado do conector JTAG
Reset	*RE*	*[valor]*	Controla linha de *System Reset*
Setup	*SE*		Activa conector JTAG
Desligar	*DE*		Coloca conector JTAG em modo de alta impedância
Terminal	*CO*	*[estado]*	Activa/Desactiva modo de Terminal
Teste01	*TE*		Confere o valor capturado no registo IR (inserindo a instrução *BYPASS*)
Erros	*ER*	*[reset] (opcional)*	Retorna ou limpa a contagem dos erros de transmissão
Help	*HELP*		Lista os comandos disponíveis
Versão	*VS*		Lista a versão do *firmware*
Test_Logic_Reset	*LR*		Coloca o controlador TAP alvo no estado *Test_Logic_Reset*
Device_Id_Register	*ID*		Lê os *IDCODEs* dos dispositivos alvo
Instrucao_IR	*IR*	*[instrução] [nº bits]*	Introduz instrução IR
Cycle_TCK	*K*	*[req]*[23]	Efectua um ciclo de *TCK*, escrevendo *TMS* e TDI, e lendo *TDO*
Escrever_TDI_0	*P*		Define valor lógico '*0*' em *TDI*
Escrever_TDI_1	*O*		Define valor lógico '*1*' em *TDI*
Escrever_TMS_0	*M*		Define valor lógico '*0*' em *TMS*
Escrever_TMS_1	*N*		Define valor lógico '*1*' em *TMS*
Escrever_TCK	*C*	*[valor]*	Controla linha de *TCK*
Ler_TDO	*R*		Retorna leitura de *TDO*
ID_controlador	*RI*		Retorna um "*acknowledge*" para identificação

Todos os comandos listados necessitam de ver a sua introdução finalizada com o caractere '\r'. Os últimos comandos, realçados em itálico, são vocacionados para a sua utilização através do *software* da biblioteca de controlo, e nada acrescentam aos restantes comandos quando utilizados directamente através de um terminal. No entanto, a sua utilização através do *software* de controlo, em substituição aos restantes, consegue melhorar muito

[23] O parâmetro em questão é um *byte* descrito na secção 5.1.7.

substancialmente o desempenho do sistema (secção 6.2.1). Descreve-se, de seguida, a implementação e funcionalidade dos diversos comandos.

5.1.6. CONTROLO ELEMENTAR DOS SINAIS JTAG

As funções destinadas ao controlo directo do valor dos sinais do conector JTAG, a ligar à placa alvo, constituem a parte fundamental das funcionalidades do controlador. A sua implementação é simples e facilmente perceptível analisando-se o código do *firmware*, pelo que se omite qualquer descrição criteriosa. A este grupo de funções pertencem: "***Setup***"; ***Desligar***"; "***Escrever_TAP***"; "***Ler_TAP***"; e "***Reset***".

A função "***Setup***" ("*SE*") destina-se à activação do conector JTAG, *i.e.* colocando os seus pinos em modo de *output* (*input* no caso de *TDO*) e mantendo o sistema alvo activo – não activando o seu *reset* – colocando os controladores TAP da placa alvo no estado *Test_Logic_Reset* através da operação das linhas de *TMS* e *TCK*.

A função "***Desligar***" ("*DE*"), de utilidade antagónica à anterior, efectua apenas a mudança do estado dos pinos para o modo de alta impedância, mantendo um *pull-up* interno activo no pino do *System Reset* (activo com valor lógico "0").

O comando de "***Reset***" ("*RE [valor#]*") carece já da utilização de um parâmetro e visa o controlo da linha de *System Reset*. O valor lógico da linha do *reset* do sistema é colocado de acordo com o argumento recebido, que deverá tomar o valor '*0*' ou '*1*'.

Restam as funcionalidades de escrita e leitura das principais linhas de controlo JTAG. Estas operações são efectuadas recorrendo a comandos com múltiplos argumentos. Os parâmetros de "***Ler_TAP***" ("*RT [pino#] ...*", de *Read Tap*) indicam os pinos dos quais se pretende receber o valor. Embora, de acordo com o funcionamento do BS, a linha de TDO seja a única do conector JTAG padrão com o intuído de ser lida pelo controlador, é possível utilizar esta função para conferir o valor de todas as outras. As linhas estão numeradas, em todos os comandos que as têm como argumento, segundo a lista definida no código do *firmware*, que se expõe em seguida:

```
typedef enum {
   TCK,        // 0
   TDI,        // 1
   TDO,        // 2
   TMS,        // 3
   SRESET      // 4
} Lista_pinos;
```

Assim, por exemplo o comando "*RT 0 1 2 3 4*" retorna o valor de todo o conector JTAG. A forma de devolução dos valores pedidos é sequencial, sem separadores, e segue a ordem dos argumentos fornecidos, dispensando quaisquer caracteres adicionais, excepto se estiver activo o modo de terminal (referido na secção 5.1.9), onde toma a apresentação da Figura 69:

```
Comando: rt 0 2 3
 > TCK tem o valor: 0
 > TDO tem o valor: 1
 > TMS tem o valor: 1
```

Figura 69 Execução do comando *RT* em modo de Terminal

O comando de "***Escrever_TAP***" ("*WT [pino#] [valor#] ...*" de *Write Tap*) funciona com pares de argumentos, onde o primeiro indica o pino alvo e o segundo o valor pretendido ('*0*' ou '*1*'). Para, por exemplo, serem colocadas todas as linhas do conector com o valor lógico '*0*', pode ser utilizado o comando "*WT 0 0 1 0 2 0 3 0 4 0*" (que totaliza o número máximo definido de dez argumentos). Os comandos para escrita de *TDO* são ignorados. Este comando actua nas linhas pedidas pela ordem que são introduzidas e apenas retornará qualquer resposta se estiver activo o modo de terminal, que indicará os pinos alterados e os seus novos valores, tal como exemplificado na Figura 70:

```
Comando: wt 3 0 1 1 0 1 4 1
 > Valor TMS -> 0
 > Valor TDI -> 1
 > Valor TCK -> 1
 > Valor SRST -> 1
```

Figura 70 Execução do comando *WT* em modo de Terminal

Especialmente estas duas últimas funções, de leitura e escrita, são vocacionadas para o controlo feito directamente através de uma consola de comunicação série com o controlador, por serem mais práticas ao evitarem a necessidade de diversos comandos para ler ou alterar cada uma das linhas JTAG, mas são preteridas para utilização por *software*, embora naturalmente possam ser também utilizadas.

5.1.7. FUNÇÕES COMPLEMENTARES PARA UTILIZAÇÃO VIA SOFTWARE

Como alternativa às funções de escrita e leitura anteriores, foram implementadas as funções "***Escrever_TDI_0***" ('*P*'); "***Escrever_TDI_1***" ('*O*'); "***Escrever_TMS_0***" ('*M*'); e "***Escrever_TMS_1***" ('*N*'); que, sem recorrer a qualquer parâmetro, colocam a linha respectiva no valor lógico indicado. Estas funções requerem apenas do envio de dois *bits*, o

comando e o consequente '\r', o que permite reduzir os tempos dispendidos em transmissões de comandos.

Para o controlo de *TCK*, a função "***Escrever_TCK***" ("*C [valor]*") requer o parâmetro indicativo do valor lógico pretendido. Para obter o valor actual da linha de *TDO* é utilizada "***Ler_TDO***" ('*R*') sem qualquer argumento.

Restam ainda duas funções de descrição necessária. A primeira, "***ID_Controlador***" ("*RI*"), oferece um método de confirmação de que se está a comunicar com o dispositivo correcto e que este consegue receber correctamente os comandos enviados, respondendo com o caractere '*ACK*' – *Acknowledge*, com o código ASCII hexadecimal *0x06* – aquando da recepção do comando respectivo.

Optou-se por esta alternativa de reconhecimento do controlador, embora mais lenta e menos prática, em detrimento da verificação dos códigos *Product Identification* (PID) e *Vendor Identification* (VID) do dispositivo USB por duas razões: A atribuição de códigos PID e VID próprios à interface *USB to UART* do *hardware* (*CP2102*) iria requerer *drivers* próprios e a programação inicial das identificações pretendidas no componente. Embora a criação destes *drivers* fosse facilitada por um assistente disponibilizado pelo próprio fabricante, os *drivers* criados teriam que ser difundidos com o restante material do projecto, e obrigatoriamente instalados pelos utilizadores. Tal como implementado, ignorando-se os códigos PID e VID, o utilizador pode instalar qualquer *driver* compatível com a interface USB utilizada, que será prontamente sugerido pelo próprio SO, o que evitará problemas futuros. Outra vantagem desta identificação é ser possível verificar, para além do estabelecimento da comunicação com a placa correcta, do controlador de BS, também o seu estado operacional. Num caso prático onde, por exemplo, o controlador esteja inactivo, em estado de *reset*, a ligação poderá ser estabelecida correctamente, mas o controlador não responderá a qualquer comando.

Por último, "***Cycle_TCK***" ("*K [req]*") é a função base de apoio ao funcionamento implementado no *software*. De modo a se obter os melhores resultados possíveis ao nível da velocidade de transmissão na execução desta função fundamental, este comando é composto apenas por um caractere associado a um argumento de igual dimensão, e não é necessária a sua conclusão com o caractere '\r', totalizando assim apenas o envio de dois *bytes*. Como argumento, é esperado um *byte* organizado de acordo com a Figura 71:

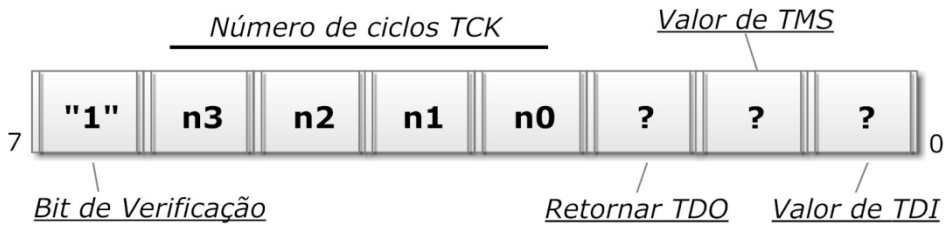


Figura 71 Argumento ("*req*") da função principal de actuação JTAG

Se o MSB não corresponder a '*1*', o comando é interpretado como corrompido, não se executando qualquer operação JTAG. Os seguintes *bits* determinam respectivamente:

- O número de ciclos de *TCK* a efectuar (n_{3-0}). Estabeleceu-se o máximo de oito ciclos, mas este valor poderia ser superior ($2^4 = 16$ *bits*). Valores inferiores a um e superiores a oito ciclos são verificados e indicam que o comando está corrompido.
- Intuito de receber o valor de *TDO* − o valor '*1*' solicita o seu retorno −, o que implica que o *software* aguarde a sua recepção. Este processo pode ser utilizado para sincronização entre o computador e o controlador, desprezando-se o valor recebido. Só é enviado o *TDO* do último ciclo, se vários forem efectuados ($n_{3-0} > 1$);
- Valor lógico a atribuir à linha de *TMS*;
- Valor lógico a atribuir à linha de *TDI*;

O controlador executa os ciclos de *TCK*, aplicando os dados recebidos, de acordo com a seguinte ordem de tarefas, que satisfaz os momentos de escrita/leitura definidos na norma:

Algoritmo de execução da função ***Cycle_TCK*** (*firmware*):

1: **Se** MSB ≠ '*1*' ou $n_{3-0} > 8$ ou $n_{3-0} < 1$:
2: Aborta a execução e regista o erro.
3: Atribui o respectivo valor à linha de *TDI*;
4: Atribui o respectivo valor à linha de *TMS*;
5: **Por** n_{3-0} iterações:
6: Coloca linha de *TCK* com o valor lógico '*0*';
7: Coloca linha de *TCK* com o valor lógico '*1*';
8: **Se** *bit_TDO* = '*1*': (*bit 2*)
9: Envia pela comunicação série o valor actual da linha de *TDO*.

5.1.8. OPERAÇÕES AUTOMÁTICAS NO CONTROLADOR TAP ALVO

Apenas algumas funções foram implementadas para efectuar automaticamente tarefas no controlador TAP dos dispositivos alvo. Estas foram as seguintes:

- ***"Test_Logic_Reset***" ("*LR*")

  Efectuar *Reset* dos controladores TAP, *i.e.* entrada no modo *Test_Logic_Reset*, não utilizando a linha *TRST* mas através do controlo sequencial das linhas de *TMS* e *TCK*.

- "***Device_Id_Register***" ("*ID*")

  Leitura do registo de Identificação do dispositivo alvo, através do controlo de todas as linhas BS obrigatórias (*TCK*; *TMS*; *TDI* e *TDO*), devolvendo o valor hexadecimal de cada identificação (*IDCODE*), já se distinguindo cada uma das suas componentes: *Manufactor ID*; *Part ID*; e *Revision*. Esta função suporta convenientemente cadeias de BS, listando o *IDCODE* de cada um dos componentes encontrados, e indicando aqueles que não dispõem de registo de identificação. A implementação desta funcionalidade obedece à seguinte descrição:

  Algoritmo do funcionamento da função ***Device_Id_Register***:

  1: Execução da função ***Test_Logic_Reset*** (o *reset* do controlador define *IDCODE* como registo TDR seleccionado, se existir);

  2: Actuação de *TMS* e *TCK* para atingir o estado *DR-SHIFT*;

  3: Aplicar ciclo a *TCK*, mantendo *TDI = 1*, e colectar primeiro *bit* em *TDO*;

  4: **Se** *bit = 0*: (registo *IDCODE* inicia-se sempre em '*1*')

  5: Reportar dispositivo sem IDCODE.

  6: **Senão** (*bit = 1*):

  7: Executar *31* ciclos de *TCK* recolhendo os valores de *TDO*;

  8: **Se** (*valores_tdo = 0xFFFFFFFF*):

  9: Terminar procura de dispositivos BS.

  10: Reportar *IDCODE* do dispositivo.

  11: Saltar para passo '*3:*'.

Utilizada através de uma consola que comunique directamente com o *firmware*, em modo de Terminal, obtém-se o resultado já apresentado na anterior Figura 67 (pág. 121).

- "***Instrucao_IR***" ("*IR [instrução] [nº bits IR]*")

Deslocamento e activação de uma instrução do registo IR do controlador TAP alvo, actuando todas as linhas BS obrigatórias. No caso de existirem mais do que um dispositivo BS na cadeia alvo, é necessário introduzir ordenadamente o conjunto de instruções para todos os dispositivos, de acordo com a sua ordenação. A codificação das instruções deve ser inserida em base decimal, e deve fazer-se acompanhar do tamanho total dos registos IR conjuntos da cadeia, segundo argumento do comando.

5.1.9. DEPURAÇÃO E CONTROLO VIA TERMINAL

Sendo apenas para fins de teste ao controlador, a quantidade de funções específicas para assistir a sua utilização através de um terminal é bastante limitada.

Em primeiro lugar, refere-se a função "***Terminal***" ("*CO [estado]*") que é responsável por activar ou desactivar a informação adicional que se apresenta no terminal com o fim de guiar o utilizador nos procedimentos efectuados. Este modo de terminal vem sendo utilizado em todos os exemplos fornecidos até agora. Esta apresentação da informação nunca é utilizada quando o controlador é operado directamente por *software*, como a biblioteca de controlo desenvolvida, onde, nesse caso, são apenas devolvidos os caracteres indispensáveis sem qualquer informação adicional (*i.e.* utiliza o modo de terminal desactivado, tal como se inicializa o controlador). Assim, para utilização de um terminal de comunicação série com o controlador, é necessário que, primeiramente, o utilizador insira a instrução "*CO 1*" (de notar que esta ainda não será visível enquanto a escreve). Para voltar a utilizar o controlador através do *software*, é obrigatório desligar este modo com a introdução de "*CO 0*", e desconectar a ligação série estabelecida pelo terminal.

As seguintes funções destinam-se a serem utilizadas no modo de terminal:

- "***Help***" ("*HELP*")

Lista a informação descritiva das várias funções implementadas e respectivos comandos, como exposta na anterior Tabela 18.

- "***Versao***" ("*VS*")

  Lista a informação da versão do *Firmware* implementado no controlador.

- "***Teste01***" ("*TE*")

  De funcionamento similar ao da função "***Instrucao_IR***", descrito na secção 5.1.8, difere desta ao introduzir só alguns valores lógicos '*1*' no registo de instruções alvo, apenas para retornar o valor nele capturado, que serve para um método simples de teste à integridade do controlador TAP alvo, de acordo com o referido na secção 2.9.2, através da verificação dos seus dois *bits* menos significativos, cujo valor deverá corresponder a "*01*". Em caso de não se verificar a correspondência, uma mensagem de erro alertará para o facto. Esta função, embora vocacionada para ligações a um único dispositivo alvo, pode ser utilizada também em cadeias de múltiplos dispositivos, sob pena de apenas ser testado o último componente da cadeia. Serve também, por exemplo, para verificar se a interface JTAG já havia sido inicializada (comando "*Setup*").

Por último, e indubitavelmente a funcionalidade mais importante para a depuração de problemas no controlador, refere-se a implementação da contabilização de todos os erros ocorridos, na execução do programa, que estejam relacionados com a recepção de comandos ou argumentos que não respeitem o esperado. O registo de erros pode então ser enviado para o computador utilizando a função "***Erros***" ("*ER*"). Não se considerou necessário registar os comandos ou argumentos responsáveis por cada um dos erros. Esta estatística é usada para grande benefício nos testes efectuados ao sistema, facilitando a identificação de implementações problemáticas. Para repor o valor da contagem, útil antes do início de cada teste, é introduzido o argumento '*1*', *i.e.* "*ER 1*".

5.2. BIBLIOTECA DINÂMICA DE CONTROLO

A funcionalidade implementada no controlador é, por si só, de um nível demasiado baixo para que se possa antever uma utilização, em casos reais, sem recurso a qualquer outro *software* que efectue o seu controlo.

Para responder a esse problema programou-se, também em linguagem *C*, um pacote de *software*, compilado como uma biblioteca dinâmica para ambientes *Windows*, de nome "*LibBS.dll*", que possibilita a execução de um conjunto de tarefas bem mais complexas. Esta, tal como se havia referido, recomenda-se que seja utilizada por uma interface de

utilizador ou qualquer outro *software* desenvolvido para o efeito. A sua utilização directa, possivelmente recorrendo à ferramenta "*rundll.exe*" disponível nos SOs *Windows*, não é prevista e antevê-se que tal execução seja muito limitada, devido aos argumentos complexos que a biblioteca de controlo requer.

Os ficheiros listados na Figura 72 compõem o programa desenvolvido:

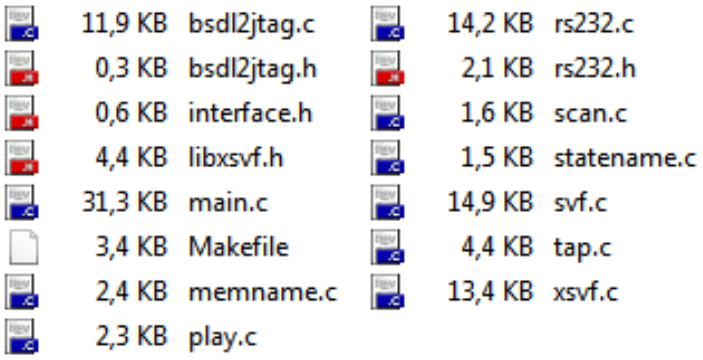


Figura 72 Listagem dos ficheiros do programa da biblioteca de controlo

Da compilação do referido código resulta o ficheiro "*LibBS.dll*" necessário para o correcto funcionamento da aplicação gráfica. O compilador utilizado foi o *software* "*MinGW*", associado ao IDE "*NetBeans 7.4*".

Diferentemente do que acontece no código do *firmware*, ou no funcionamento base da aplicação gráfica, nem todo o código utilizado foi integralmente desenvolvido para o corrente projecto. Algumas bibliotecas externas, livremente disponíveis, são utilizadas para garantir funcionalidade ao sistema. Estas sofreram, no entanto, derivado de diversas carências aquando da programação do sistema, consideráveis mudanças que foram registadas e são expostas nesta descrição.

5.2.1. IMPORTAÇÃO DE CÓDIGO EXTERNO

O *software* desenvolvido dispõe de algumas funcionalidades que lhe são conferidas pela inclusão dos seguintes pacotes de código de linguagem *C*:

- ***bsdl2jtag.c***, versão *1.4*, disponibilizada em [74] sob licença *General Public License v.2* (GPL);
  Código responsável por interpretar ficheiros BSDL, traduzindo-o para um formato mais acessível. Originalmente utilizado no descontinuado *software* "*JTAG Tools*" (abordado na secção 3.1.3). As alterações, limitações, e implementação deste código são descritas na secção 5.2.9.

- ***lib(x)svf.h***, disponibilizada em [75] sob licença *Internet Systems Consortium* (ISC); Biblioteca vocacionada para o desenvolvimento de aplicações de execução de ficheiros SVF ou XSVF (popularmente conhecidos como "*SVF Players*"). As alterações, limitações, e implementação deste código são descritas na secção 5.2.5.

- ***rs232.h***, actualização de *01/02/2013*, disponibilizada em [76] sob licença GPL *v.2*; Responsável por efectuar a comunicação com o controlador. As alterações, limitações, e implementação deste código são descritas na secção 5.2.4.

Embora estas bibliotecas tenham sido aglutinadas integralmente nesta biblioteca de controlo, as suas funções não se destinam a ser acedidas externamente, funcionando apenas como suporte às operações internas da biblioteca de controlo.

Listar-se-ão em momento oportuno, na esperança de facilitar um posterior desenvolvimento e de melhor dar conta dos trabalhos desenvolvidos no actual projecto, as alterações efectuadas em cada um dos códigos listados, assim como as suas limitações conhecidas que perdurem ainda. O código seleccionado para ser exposto e descrito nesta secção (incluindo todas as funções e estruturas apresentadas na listagem de funcionalidades da biblioteca de controlo), a não ser que seja expressamente referido o contrário, foi desenvolvido para o actual projecto, não pertencendo a qualquer uma destas bibliotecas.

5.2.2. FUNCIONAMENTO GERAL

A biblioteca desenvolvida não tem um funcionamento base bem definido, pois pretende facultar um conjunto de ferramentas, sem ligação entre si, de apoio à utilização do controlador, que, como se viu, tem apenas funcionalidades de BS elementares, cuja utilização para tarefas mais complexas tornar-se-ia complicada.

O sistema desenvolvido deve oferecer suporte aos vários comandos e respectivos parâmetros definidos na especificação SVF, o que torna a flexibilidade do funcionamento implementado num factor de relevância. Para o deslocamento de dados na interface JTAG, determinante no funcionamento do controlador, foi, por exemplo, considerada a inclusão no *firmware* de uma função para deslocamento de vectores, similar ao comando *Scan Data Register* (SDR) do SVF. No entanto, a sua implementação iria sugerir que o *firmware* pudesse autonomamente mover os controladores TAP alvo para o modo *SHIFT-DR*

quando lhe fosse enviada a instrução para deslocamento de algum vector. Os percursos das transições de estados dos controladores TAP estão integralmente definidos na especificação SVF, já que ao transitar entre dois estados poderão existir vários caminhos possíveis. Assim, seria necessário implementar a gestão dos estados TAP no próprio *firmware*, para que pudesse, em qualquer circunstância, efectuar automaticamente o deslocamento dos vectores pedidos. Mais ainda, o estado em que o deslocamento de dados deve terminar é mutável e está dependente do comando "ENDDR" do SVF (pode-se, por exemplo, actualizar os valores introduzidos, em *UPDATE-DR*, ou ficar em espera no estado *PAUSE-DR*), pelo que o controlador teria de aguardar pela indicação de qual o estado para onde encaminhar os componentes alvo, já que se tal fosse feito no computador e as linhas JTAG fossem directamente comandadas para atingir esse estado, seria mais difícil para o controlador seguir o estado actual dos controladores TAP, necessário para poder efectuar algum deslocamento futuro. A abordagem de implementar a referida gestão dos estados TAP no *firmware*, que se entende que fosse expectável na implementação de funções automática de varrimento de dados no controlador, chocaria contra o intento de se simplificar o seu processamento, e crê-se que fosse atrasar o funcionamento do sistema (a referida gestão dos estados dos controladores TAP é efectuada na biblioteca de controlo, estando implementada no ficheiro "*tap.c*").

A solução adoptada nesta implementação, que se entende que oferece maior flexibilidade, e tem em consideração as modestas capacidades de processamento do *hardware* do controlador, foi a de se efectuar o controlo, no computador, da interface JTAG actuando directamente os seus sinais elementares (escrita de *TMS*, *TDI*, e *TCK*). Embora tal controlo deva significar um maior fluxo de informação entre o computador e o controlador, a especificação SVF permite determinar valores de *TDI* e *TDO* sem significância ("*don't care*"), o que, associado à flexibilidade do controlo sugerido, vem permitir optimizações que favoreçam a diminuição das transferências de dados. A implementação de uma função de deslocação de vectores, no *firmware*, que não tivesse em conta qualquer estado (apenas definisse os valores de TDI e efectuasse os ciclos de TCK) seria uma possibilidade, mas o controlo directo dos sinais JTAG continuaria a ser fundamental.

O código da biblioteca de controlo, ao ser programado em linguagem *C* nativa e não recorrer ao *.Net Common Language Runtime* (CLR), é do tipo não gerido, *i.e.* "*unmanaged*", que é compilado para código máquina direccionado para a plataforma

utilizada (*x86*) e executado directamente pelo sistema operativo. Código gerido ("*managed*"), por outro lado, é interpretado e executado por um serviço do sistema, e corre num ambiente controlado, que gere, por exemplo, a sua utilização de memória ou linhas de processamento, limitando o acesso aos recursos físicos, como endereços de memória, *stacks*, *etc*. A aplicação gráfica desenvolvida, que utiliza o *.Net Framework*, é um exemplo de código deste tipo. Devido a estas diferenças, e por a execução de código *unmanaged* ser geralmente considerada um risco à segurança do sistema, são necessários alguns procedimentos especiais na utilização desta biblioteca[77]. O exemplo mais recorrente será a necessidade de proceder ao "*marshaling*" de variáveis que transitem entre códigos do tipo *managed* e *unmanaged*. As considerações necessárias para a utilização da biblioteca de controlo são referidas na secção 5.3.1.

Porque esta biblioteca será operada através de código *managed*, e para facilitar a comunicação entre estes, teve-se em conta que a memória dos programas não poderá ser partilhada entre si e que as linhas de *standard input* e *standard output* de ambos os códigos estarão isoladas. Assim, desenvolveu-se a biblioteca de controlo de forma a receber os seus *inputs* através dos argumentos da chamada das suas funções e do carregamento de ficheiros do computador. Os seus *outputs* cingem-se ao valor de retorno das funções chamadas e, igualmente, ao seu registo em ficheiros do computador.

Ao pretender-se, num mesmo momento, invocar funções diferentes da biblioteca, tal não poderá ser feito em simultâneo já que estas deverão partilhar o acesso aos mesmos ficheiros ou à mesma ligação série com o controlador BS.

Apresenta-se um exemplo da declaração de uma função que se destina a ser posteriormente acedida por outro programa:

```
__declspec(dllexport) void __cdecl
controlo_svf(svf conf)
```

As funções da biblioteca de controlo que se pretendem acedidas foram declaradas com a directiva "*__declspec(dllexport)*", que permitem o seu acesso externo através do método de "*dllimport*". As restantes funções do código são consideradas internas e são inacessíveis do exterior. A directiva "*__cdecl*", também presente, define a convenção de chamada da função. No entanto, esta é a convenção padrão em código *C* e *C++*, pelo que esta directiva seria desnecessária, tendo sido mantida na declaração para ser assim mais intuitiva a necessidade de, ao ser acedida por outra linguagem como *C#*, ter de se declarar esta forma

de chamada (*e.g.* em linguagem *C#* esta convenção deve ser escolhida na declaração da função externa, usando o método "*dllimport*", com definição de "*CallingConvention = CallingConvention.Cdecl*").

5.2.3. LISTAGEM DE FUNCIONALIDADES

São listadas, na Tabela 19, as funcionalidades base da *LibBS.dll*, com a sua respectiva função e argumentos necessários. Omite-se, para já, as diversas variações possíveis na execução de cada uma destas funcionalidades.

Tabela 19 Listagem de funcionalidades gerais da biblioteca *LibBS.dll*

Funcionalidade	Função utilizada	Argumentos
• Executar código SVF e XSVF	*controlo_svf*	Estrutura do tipo ***SVF***
Executa código SVF ou XSVF na cadeia BS alvo a partir de um ficheiro .SVF/.XSVF.		
• Procurar e listar dispositivos	*controlo_svf*	Estrutura do tipo ***SVF***
Procura dispositivos na cadeia de BS alvo e devolve ordenadamente os seus *IDCodes*.		
• Controlo das linhas JTAG	*controlo_tap*	Estrutura do tipo ***pc_tap_pins***
Escreve ou lê o estado de cada uma das linhas do conector JTAG (*TCK*; *TMS*; *TDI*; *TDO* (só leitura); e *SRST*).		
• Verificar ligação ao controlador	*check_connection*	*Port* e *Baud rate*
Tenta estabelecer ligação com o par *Porta:Baud* indicado. Se bem sucedido, verifica se comunica com o controlador correcto.		
• Interpretar linguagem BSDL	*read_BSDL*	Localização do ficheiro
Interpreta o ficheiro BSDL indicado, devolvendo a descrição do componente no ficheiro "*description.dat*".		

O modo de funcionamento de cada uma destas funções é descrito nas próximas secções.

As estruturas que servem de argumentos para as funções referidas na anterior Tabela 19 são importantes para a correcta operação da biblioteca de controlo. A criação destas estruturas foi preferida à utilização de extensos conjuntos de argumentos individuais. O ficheiro "*interface.h*" contém todas as definições destes e outros tipos de variáveis pelas quais se operam, através da interface gráfica, as funções implementadas na *LibBS.dll*.

Listam-se, na Tabela 20, os argumentos de entrada das já citadas funções:

Tabela 20 Listagem dos argumentos das funções da biblioteca *LibBS.dll*

Tipo de Estrutura	Campos	Tipo	Descrição
pc_tap_pins	*Baud*	*int*	Baud rate da ligação ao controlador
	Com	*int*	Port da ligação ao controlador
	Action	*pc_tap_pins_action* (ou *int*)	Acção pretendida: "*P_READ*" ('*0*'); "*P_WRITE*" ('*1*').
	TDI_val	*int*	Valor para escrita na linha *TDI* ('*0*' ou '*1*')
	TMS_val	*int*	Valor para escrita na linha *TMS* ('*0*' ou '*1*')
	SRST_val	*int*	Valor para escrita na linha *SRST* ('*0*' ou '*1*')
	TCK_val	*int*	Valor para escrita na linha *TCK* ('*0*' ou '*1*')
	TDO_val	*int*	Reservado
Svf	*Baud*	*int*	*Baud rate* da ligação ao controlador
	Com	*int*	*Port* da ligação ao controlador (*1 - 16*)
	Verb	*Int*	Nível de informação devolvida (*0 - 4*)
	Action	*pc_comandos* (ou *int*)	Acção pretendida: *"SCAN"* ('*0*'); "*SVF*" ('*1*'); "*XSVF*" ('*2*'); ou "*NO_ACTION"* ('*3*').
	Hex	*Int*	Formatação de retorno: Binário ('*0*'); Hexadecimal "*LittleEndian*" ('1'); ou hexadecimal de ordem contrária ('*2*')
	File	*char **	Caminho do ficheiro SVF/XSVF
	Retorno	*int*	Reservado
	id_man	*int*	Reservado
	id_part	*int*	Reservado
	id_ver	*int*	Reservado
	leave_on	*int*	Deixar JTAG activo após conclusão da operação ('*0*' ou '*1*')
	dont_initialize	*int*	Não (re)inicializar JTAG antes da operação ('*0*' ou '*1*')
	lowspeed	*int*	Modo low-speed ('*0*' ou '*1*')
	grouping	*int*	Agrupamento de ciclos de TCK ('*0*' ou '*1*')

Vários destes parâmetros serão novamente abordados em momentos oportunos.

5.2.4. COMUNICAÇÃO SERIE COM O CONTROLADOR

A implementação da comunicação com o controlador recorreu à referida biblioteca "*rs232.h*". Esta, por si só, não complementa o *software* com nenhuma funcionalidade de especial interesse que justificasse a sua utilização, e que não pudesse ser facilmente programada, tendo sido apenas utilizada para economia de tempo. Esta confere ao programa a versatilidade de suportar diversos parâmetros de comunicação (*baud rate* e

portas *COM*), que podem assim ser recebidos como argumentos pela biblioteca de controlo, passando estes a terem de ser apenas definidos na aplicação gráfica.

As suas características fundamentais são:

- Código funcional em *Windows* (*MinGW*) e *Linux* (*GNU Compiler Collection − GCC*);
- Utilização do método de "*polling*"[24] para recepção de informação;
- *Handshaking*, controlo de fluxo, e funcionamento por interrupções não implementado;
- Configuração pré-definida para comunicações de: *8 databits*; sem paridade; e *1 stopbit*.

As alterações efectuadas nesta biblioteca visaram apenas a inclusão de suporte para outras *baud rates*, como *576000 bd*, que veio a ser o valor utilizado no sistema final. Estas alterações são feitas no ficheiro "*rs232.c*" e são apenas necessárias se se escolher, na aplicação gráfica ou como argumento da biblioteca de controlo, uma *baud rate* não suportada. Em *Windows*, esta biblioteca suporta apenas a utilização das portas *COM1* a *COM16*. A selecção de uma porta *COM* ou *baud rate* não suportada leva a que seja reportada ao utilizador uma mensagem de erro que identifica claramente a situação.

As funções criadas para directamente interagirem com o controlador são denominadas com o prefixo "*io_*" (*e.g. io_setup*; *io_tdi*; *etc.*), estão contidas no ficheiro "*main.c*", e não devem ser externamente acedidas. Estas funções, implementadas de forma similar, diferem no comando que enviam ao controlador e na necessidade de receber algum dado de retorno (*e.g. io_tdo*) ou não (*e.g. io_tdi*). Como exemplo, descreve-se a forma de implementação de uma função de controlo (assíncrono) de uma linha do conector JTAG:

[24] *Polling* refere-se à operação de verificar se existem dados em espera para ser recebidos, contraria a outros métodos que prevêm, por exemplo, a imposição automática de uma interrupção para recepção dos dados recebidos.

Algoritmo de execução da função ***io_tdi*** (*LibBS.dll*):

1: Regista número de *bytes* a enviar;

2: Se o valor de *TDI* já for o valor alvo pretendido, termina a execução. Verificação feita através da memória, sem comunicação com o controlador.

3: Envia comando de escrita de *TDI* e o seu respectivo valor, para a ligação série já previamente estabelecida;

4: Regista, em memória, o novo valor de *TDI*;

O passo '*2:*' é utilizado para optimização do desempenho do sistema, ao evitar o envio de informação desnecessária para o controlador, e os resultados da sua implementação estão descritos, juntamente com a descrição dos restantes testes efectuados, na secção 6.2.1.

No caso de se executar uma função sincronizada com o controlador, *i.e.* que aguarda algum retorno do controlador antes de permitir ao programa prosseguir com a sua operação, seguem-se posteriormente os seguintes passos:

5: É invocada a função *read_answer* que aguarda e devolve a resposta do controlador;

6: Em caso de falha (retorna '*-1*'), a função "*io_*" que origina a chama à função de leitura é invocada recursivamente, de forma a repetir todo o processo, até ser finalmente retornado o valor da leitura.

Nos testes executados, o passo '*6:*' só num caso extremo, onde foi utilizado um conversor Série-USB diferente e inadequado à velocidade configurada, levou ao congelamento da execução do programa, pelo que não se entendeu necessário adicionar qualquer outra condição de saída do ciclo recursivo (que pode, entendendo-se necessário, ser facilmente programada).

As falhas da função *read_answer* podem ser devidas a *time-out* ou recebimento de dados corrompidos. No primeiro dos casos, como não ocorreu qualquer resposta, é aguardado um tempo pré-definido, na esperança de esvaziar algum possível *buffer* (do conversor Série-USB ou MCU) que possa ter causado a perda de dados, e seguidamente enviado o comando '\r' para limpar algum possível comando corrompido em espera no controlador (se for enviado um comando e falhar o envio da terminação '\r', não só esse comando não será executado como irá causar um erro no próximo comando enviado). No segundo caso, os dados recebidos são considerados corrompidos se não forem numéricos, ou

simplesmente não estiverem de acordo com o esperado (*e.g.* resposta do valor de *TDO* deverá ser, exclusivamente, '*0*' ou '*1*'). Durante estes passos, todos os *bytes* enviados e recebidos são registados, inclusivamente os duplicados.

Os valores referentes ao tempo de espera máximo pela resposta do controlador (*time-out)*, assim como o tempo de espera em caso de falha, são definidos em:

```
#define TIME_OUT_1uS 100000 // time-out after
100ms in read_answer
#define ERROR_WAIT_TIME 100 // time to wait after
time-out in read_answer
```

Alterando-se a *baud rate* da ligação para valores muito inferiores ao escolhido (*576000 bd*) pode ser necessário a alteração destes períodos para valores mais extensos.

A função *read_answer* em conjunto com a recursividade das funções de I/O síncronas, descrita anteriormente, compõem o mecanismo de controlo de comandos corrompidos que faz com que estes, se ocorrerem, possam não ter um impacto crítico nas tarefas a serem executadas. O modo "*low-speed*", referido na secção seguinte em maior detalhe, impõe que todos os ciclos de *TCK* sejam síncronos, o que confere uma maior fiabilidade ao sistema. Ainda assim, nos testes efectuados não se vislumbra necessidade para tal.

5.2.5. INTERPRETAÇÃO E EXECUÇÃO DE SVF E XSVF

Se, à primeira vista, pode parecer faltar funcionalidade a esta biblioteca de controlo, listando-se que dispõe de apenas quatro funções de acesso externo, tal impressão é desfeita considerando todas as capacidades do código SVF e XSVF, como descritas nas secções 2.11.1 e 2.11.2. A interpretação implementada, de código SVF (fazer-se-á, daqui em diante, apenas referencia ao código SVF, de implementação análoga ao código XSVF), permite a execução de quase todas as suas funções, como se sabe dedicadas à condução de operações de BS. A funcionalidade de interpretação de SVF não se destina apenas a ser utilizada para a execução, num determinado componente, de ficheiros SVF gerados por um qualquer *software* como os listados na Tabela 4 da pág. 80 (utilização prevista na documentação da biblioteca *lib(x)svf* utilizada), mas, especialmente, visa oferecer suporte à execução de qualquer operação cujos comandos SVF possam dar resposta.

Clarificando, ao ser pretendido, por exemplo, executar uma instrução IR (considerando um registo IR com comprimento de *8 bits*) para aplicação do modo *EXTEST* num dado dispositivo, é possível satisfazer o pretendido, segundo a especificação SVF, com a

declaração "*SIR 8 TDI(0)*". Esta pode ser rapidamente registada num ficheiro cujo caminho é enviado por argumento para a função *controlo_svf*. Assim se recomenda a utilização da biblioteca de controlo, à semelhança do que efectua a aplicação gráfica, onde comandos SVF são gerados dinamicamente, de acordo com os pedidos do utilizador, e rapidamente executados pela *libBS*.

Relativamente à biblioteca *lib(x)svf* utilizada para a interpretação de SVF, a sua implementação foi um pouco dificultada por esta ser destinada a sistemas *Unix*, sendo mesmo incompatível com o *SO Windows*, e por a sua implementação só estar prevista em interfaces de comunicação síncronas, ou, excepcionalmente, assíncronas se utilizando um controlador USB de comportamento específico (*i.e. FT232H/FT2232H/FT4232H*). Esta requeria também a programação de bastantes funções cujo funcionamento obedecesse a um conjunto de procedimentos descritos na sua documentação (existe algum código sugerido para estas, mas que era, pelas razões referidas mais à frente, em parte incompatível com este projecto).

Para se efectuar o *porting* de uma aplicação *Unix* para *Windows*, existem três alternativas recomendadas[78]:

- Utilizar bibliotecas de terceiros que emulem as bibliotecas *Unix* necessárias;
- Transformação nativa do código, reescrevendo-o para sistemas *Windows*;
- Correr a aplicação utilizando o subsistema *Portable Operating System Interface* (POSIX).

Inicialmente optou-se pela utilização de bibliotecas alternativas mas este procedimento acabou por ser preterido em favor da reescrita do código problemático, de forma a o tornar compatível com o novo sistema. As funções internas da biblioteca não efectuam, por si só, chamadas a funções do sistema, apenas o código que as controlava o fazia. Este veio a ser quase totalmente reformulado, nos interesses do actual projecto, pelo que deixou de fazer sentido manter qualquer biblioteca de funções *Unix*. O código referido está presente no ficheiro "*main.c*" e foi escrito para ser acedido como uma biblioteca dinâmica, utilizando-se a principal função de controlo de BS da biblioteca de controlo: *controlo_svf*. Nesta foi implementada a comunicação série, recorrendo à referida "*rs232.h*" (compatível com SO

Windows e *Linux*); e definidos todos os parâmetros, listados na Tabela 20, que controlam as chamadas às respectivas funções da biblioteca *lib(x)svf*.

As funções de programação necessária para a execução da *lib(x)svf* foram associadas à biblioteca de controlo através da seguinte estrutura:

```
static struct libxsvf_host h = {
   .udelay = h_udelay,
   .setup = h_setup,
   .shutdown = h_shutdown,
   .getbyte = h_getbyte,
   .pulse_tck = h_pulse_tck,
   .pulse_sck = h_pulse_sck,
   .set_trst = h_set_trst,
   .set_frequency = h_set_frequency,
   .report_tapstate = h_report_tapstate,
   .report_device = h_report_device,
   .report_status = h_report_status,
   .report_error = h_report_error,
   .realloc = h_realloc,
```

Referem-se apenas sumariamente algumas particularidades da programação, ou reprogramação nos casos em que o código sugerido era útil, de cada uma destas:

- ***udelay***:

  Função de funcionamento semelhante a *pulse_tck*, mas cuja execução se deve alongar durante um determinado período de tempo, para dar resposta aos comandos de temporização do SVF. O código sugerido utilizava funções de sistema *Unix* ("*usleep*"), e o código equivalente em *Windows* ("*sleep*") funciona com menor precisão. Foi implementado um novo mecanismo de temporização (baseado nas funções *QueryPerformanceCounter* e *QueryPerformanceFrequency*) capaz da precisão indicada.

- ***setup***; ***shutdown***; e ***set_trst***:

  Funções implementadas no *firmware*, foi apenas necessário o envio do respectivo comando.

- ***getbyte*** e ***realloc***:

  Funções para ler o ficheiro SVF e alocação de memória. Foi mantido o código sugerido, por não trazer incompatibilidades nem com o SO nem com o projecto desenvolvido.

- ***set_frequency***:

  Funcionalidade não implementada. Esta função fica assim a apenas reportar uma mensagem indicando que o comando de estabelecimento da frequência foi ignorado.

- ***report_tapstate***; ***report_device***; ***report_status***; e ***report_error***:

  Todas as funções de reporte de informação foram conduzidas para ficheiros e formatadas de acordo com o mais conveniente para a aplicação gráfica. Os ficheiros editados, e respectiva formatação, é referida mais em diante. A escrita nos ficheiros é conduzida de forma a que sejam actualizados à medida que a execução se processa, mas não bloqueados, permitindo ler cada um dos eventos registados no momento em que ocorrem.

A função restante, ***pulse_tck***, é a principal função da actuação do BS, e merece uma descrição mais detalhada. Esta faz uso do comando *cycle_tck* do *firmware* e nela foram implementados os dois modos de funcionamento: normal ou de baixa velocidade (de operação síncrona com o controlador). O primeiro tem ainda a opção de se agrupar ciclos de *TCK*.

O modo de baixa velocidade, cujo nome pode induzir a ideia errada, não surge com a motivação de dar uma alternativa de funcionamento a baixa velocidade, já que o modo normal dificilmente se pode descrever como de "alta velocidade". Este modo pretende possibilitar que as operações de BS sejam conduzidas sincronamente entre o computador e o controlador, para precaver eventuais problemas de incompatibilidade com algum dispositivo BS mais peculiar (que não se verificaram nos testes efectuados), ou se, por algum motivo, a comunicação série, seja esta efectuada através do conversor Série-USB escolhido ou por qualquer outro circuito, venha a mostrar problemas de perda de tramas. Este modo, por ser executado de forma mais demorada, permite também um acompanhamento em tempo real das tarefas, através da leitura da informação que é continuamente reportada. De outra forma, a sua leitura à medida que vai sendo escrita no respectivo ficheiro de *output*, à máxima velocidade que o sistema suporta, é problemática e não foi possível na aplicação gráfica.

Esta função ("*pulse_tck*") é um factor fundamental na velocidade de execução do sistema por ser responsável pela actuação do controlador em quase todas as operações BS

efectuadas pela biblioteca. A especificação SVF permite determinar *bits* não significativos, tanto no que concerne a *TDI* (no parâmetro *SMASK* do SVF) como a *TDO* (no parâmetro *MASK*). Assim, a devolução de TDO pode ser evitada em casos em que este não tenha qualquer utilidade.

O funcionamento do modo de execução mais lento desenrola-se como se apresenta em seguida:

Algoritmo de execução da função ***pulse_tck*** no modo "*low-speed*" (*LibBS.dll*):

1: **Se** *TDI* for significativo (*SMASK = '1'*):
2: Contabiliza um *bit* de *TDI* significativo;
3: **Se** valor de *TDI* pretendido ≠ valor registado da linha:
4: Envia comando para escrita de *TDI* (***io_tdi***).
5: **Se** valor de *TMS* pretendido ≠ valor registado da linha:
6: Envia comando para escrita de *TMS* (***io_tms***).
7: Envia comando para escrita de *TCK* a '*0*' (***io_tck***);
8: Envia comando para escrita de *TCK* a '*1*' (***io_tck***);
9: Envia comando para leitura de *TDO* e guarda temporariamente o seu valor (***io_tdo***).
10: **Se** *TDO* for significativo (*MASK = '1'*):
11: Contabiliza um *bit* de *TDO* significativo;
12: **Se** *RMASK*[25] *= '1'*:
13: Regista valor de *TDO* para futuro retorno;

O funcionamento descrito é caracterizado como síncrono porque, a cada ciclo de *TCK*, o computador espera pela resposta do controlador, para a recepção de *TDO*, fazendo-os progredir em conjunto.

No funcionamento normal, descrito em seguida, porque *TDO* não é obrigatoriamente lido, é possível que a biblioteca de controlo acabe de enviar todos os comandos necessários para

[25] *RMASK* é um parâmetro adicional que não pertence à especificação SVF, e é descrito no final desta subsecção.

o controlador, mas este ainda esteja atrasado na sua execução, dando origem a filas de espera.

No modo de funcionamento normal, a actuação do controlador processa-se da seguinte forma:

Algoritmo de execução da função ***pulse_tck*** em modo "normal" (*LibBS.dll*):

1: **Se** *TDI* for significativo (*SMASK = '1'*):
2: Contabiliza um *bit* de *TDI* significativo;
3: Regista valor de *TDI* num *bit* do *byte* "*req*".
4: Regista valor de *TDI* num *bit* do *byte* "*req*";
5: **Se** (*TDO* for significativo) ou (*RMASK = '1'*) ou (*Action = "SCAN"*[26])
6: Regista pedido de *TDO* num *bit* do *byte* "*req*";
7: **Se** *TDO* for significativo (*MASK = '1'*):
8: Contabiliza um *bit* de *TDO* significativo.
9: Envia *byte* "*req*" para ser executado no controlador (***io_tck_cycle***).

A variável "*req*" referida obdece à formatação definida na função "C*ycle_TCK*" do *firmware*, descrita na secção 5.1.7.

A quantidade de informação transferida para efectuar cada ciclo de *TCK* em cada um dos modos é variável. No primeiro apresentado, esta depende do valor anterior das linhas de *TDI* e *TMS*, pelo que a execução de diferentes ficheiros SVF, mesmo que de igual tamanho, pode ter velocidades muito diferentes; já no modo regular, só é enviada mais informação quando for necessário receber valores de *TDO*. A diferença entre a quantidade de *bytes* enviados/recebidos nos dois modos de funcionamento é muito significativa, na pior das hipóteses *15* contra *3 bytes*, o que leva a uma diferença de velocidade na execução de código SVF muito considerável.

O modo de funcionamento normal, ao despender do envio de tantos comandos, e sua posterior interpretação no controlador, consegue uma mais célere execução das operações.

[26] *SCAN* é o nome que identifica, no programa, a operação de procura e listagem de dispositivos na cadeia de BS

Num caso em que as operações não requeiram a leitura dos valores de *TDO*, é esperado que o processamento no computador seja independente do controlador, podendo o computador terminar de interpretar todo o código SVF e ficar então a aguardar a conclusão das tarefas no controlador, que deverão estar em fila de espera para serem interpretadas e executadas. Neste modo não é possível saber, no computador, o que está a ser executado a cada momento. A utilização de cada um dos modos de funcionamento descritos repercute-se na operação geral da forma que se sugere na Figura 73:

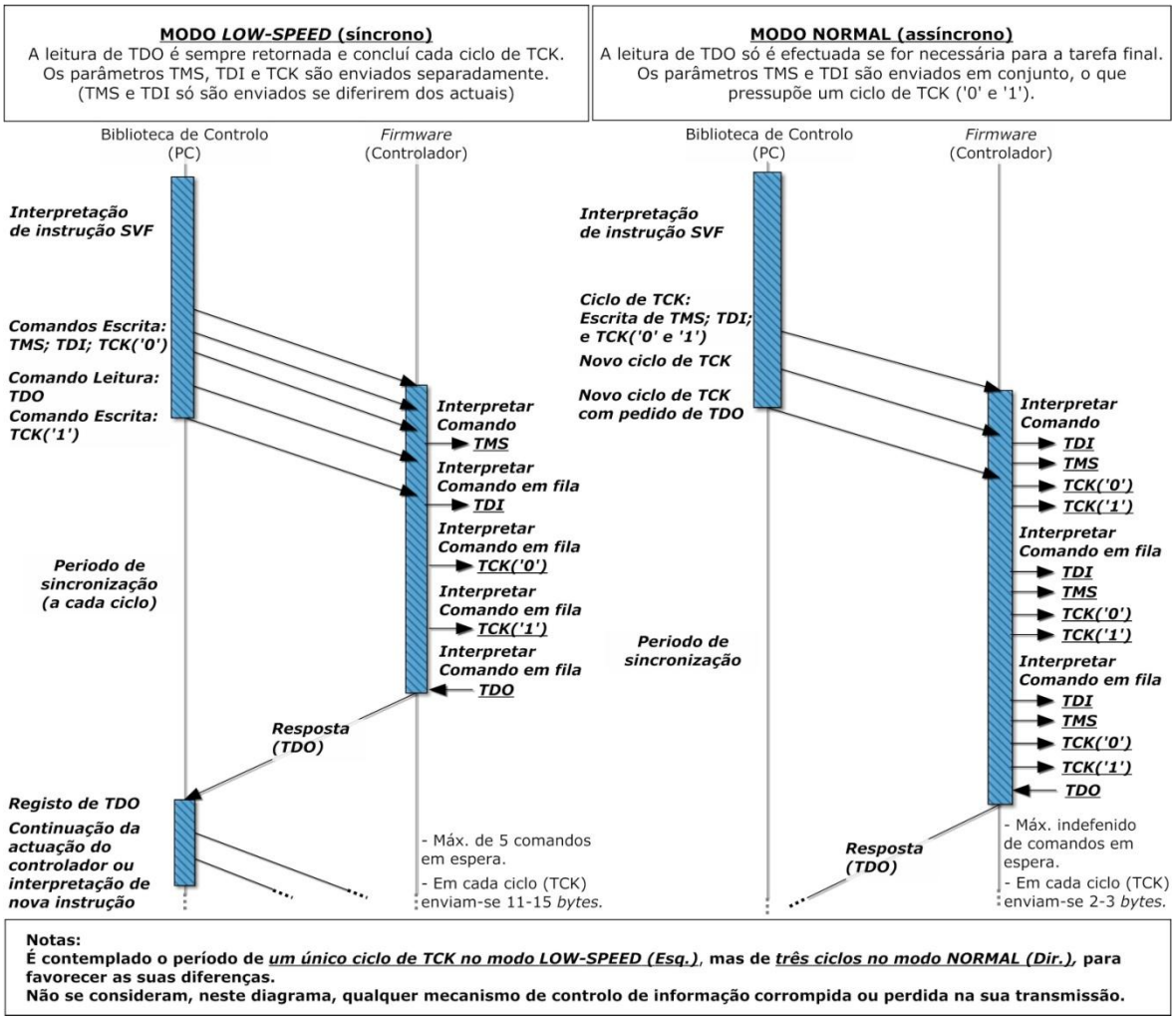


Figura 73 Execução de ciclos de *TCK* de acordo com o modo de funcionamento

A execução do modo de baixa velocidade, ao obrigar a sincronização entre o processamento do computador com o do controlador, ao final de cada ciclo de *TCK*, garante que se conheça pelo menos o ciclo de *TCK* em que o controlador está ocupado, e que não existam em fila de espera, no controlador, mais do que os comandos respeitantes a esse mesmo ciclo.

O comportamento contemplado, no modo assíncrono, ainda não prevê a execução de mais do que um ciclo por comando, como se permitiu na função respectiva do *firmware* (secção 5.1.7). Esta funcionalidade pode ser activada com o parâmetro "*grouping*" na chamada à função da biblioteca de controlo. A biblioteca irá então, se em modo de operação normal, registar os ciclos de *TCK*, com os respectivos valores de *TMS* e *TDI*, antes de os enviar. Apenas são agrupados os ciclos iguais. O envio dos ciclos registados acontece nas seguintes circunstâncias:

- Novo ciclo difere do anterior no valor de *TMS* ou *TDI*;
- Novo ciclo requer a leitura de *TDO*;
- São registados oito ciclos (limite máximo definido);
- Terminou a interpretação do código SVF e devem-se concluir todos os envios pendentes.

Para além da programação destas funções fundamentais para a execução de código SVF, fizeram-se também algumas mudanças nas próprias funções da biblioteca, com o intuito de facilitar a observabilidade do seu funcionamento e de o dinamizar, permitindo agora controlar quando as funções procedem à (re)inicialização da interface JTAG no controlador, assim como o momento em que esta é desactiva. Esta segunda mudança é de muita importância para o sistema, já que, para se utilizar a biblioteca como se indicou, para a interpretação de comandos singulares ou conjuntos de comandos, em vez de ficheiros SVF de tarefas completas, é imprescindível controlar que o programa não termine ou reinicialize o funcionamento do controlador, e consequentemente se perca o controlo dos controladores TAP alvo, quando ainda se pretende executar mais algum comando que dependa dos anteriores. Estas alterações foram feitas no ficheiro "*play.c*", e são fulcrais para grande parte das funcionalidades da aplicação gráfica.

Tendo-se até então descrito o modo de actuação do controlador construído, de especial interesse no desempenho do sistema, e evitando-se também a descrição do fluxo de processos envolvidos na interpretação de comandos SVF (ficheiros "*svf.c*" e "*xsvf.c*"), deve-se, por último, referir apenas algumas particularidades em que a implementação proposta para a interpretação dos comandos se desvia da especificação SVF. Existem limitações, nativas da *lib(x)svf*, que não foram rectificadas no desenvolvimento deste

projecto, e envolvem a execução dos seguintes comandos (a finalidade de cada comando SVF é sumariamente descrita na anterior secção 2.11.1):

- Os comandos "*PIO*" (*Parallel Input/Output*) e "*PIOMAP*" (*Parallel Input/Output Map*) não são suportados, sendo reportado um erro;

- No comando "*RUNTEST*" o parâmetro "*[MAXIMUM max_time SEC]*" é ignorado. Utilizando também o parâmetro *SCK* associado a alguma temporização (selecciona o relógio do sistema para a temporização, preterindo o relógio de teste *TCK*), a execução da temporização é feita em série, *i.e.* primeiro são executados os impulsos JTAG necessários, e só depois se temporiza a espera desejada;

Por outro lado, é disponibilizado um novo parâmetro, "*RMASK*", nos comandos "*HDR*"; "*HIR*"; "*SDR*"; "*SIR*"; "*TDR*"; e "*TIR*". De aplicação similar ao parâmetro "*MASK*", este permite seleccionar os *bits* de *TDO* que devem ser devolvidos ao utilizador, sendo reportados na formatação escolhida através do parâmetro "*hex*" da estrutura que serve como argumento da função *controlo_svf* (que inicia a interpretação do código SVF). Este parâmetro "*RMASK*" não será utilizado nos ficheiros SVF que se possa querer executar, já que não está sequer previsto no *standard*, mas é de implementação necessária para que possa ser utilizado pela aplicação gráfica permitindo que esta receba os valores de *TDO* que necessite.

5.2.6. FUNÇÃO *CONTROLO_SVF*

A função *controlo_svf* é a principal função de acesso externo da biblioteca de controlo, esta permite executar comandos SVF/XSVF e procurar dispositivos na cadeia BS alvo.

Os seus parâmetros de entrada já foram apresentados e descritos na Tabela 20. O seu *output*, para além dos comandos enviados para o controlador, é distribuído em quatro ficheiros: "*data1.dat*"; "*data2.dat*"; "*devices.dat*"; e "*BSR_R.dat*".

Em "*data1.dat*" é fornecida toda a informação, incluindo a informação constante dos outros ficheiros, servindo de registo das operações. Este ficheiro pode ser lido por outra aplicação à medida que é escrito pela *libBS*, como se faz na aplicação gráfica ao ser utilizado o modo de baixa velocidade. O nível de informação nele registada está dependente do parâmetro "*verbose*" da *svf* (argumento da função *controlo_svf*), que pode

ter um valor entre *'0'* (apenas informação crítica) e *'4'* (toda a informação). A informação que incluí é a seguinte:

- Parâmetros de execução:
  "*[<nome> <valor>]*" (*e.g.* "*[Port COM1]*");
- Tarefas a serem executadas:
  "*[<nome>]*" (*e.g.* "*[SETUP]*");
- Indicações sobre operações com atrasos:
  "*[DELAY:<μ_segundos>, TMS:<valor>, NUM_TCK:<num_ciclos>]*";
- Indicação dos impulsos JTAG:
  "*[TMS:<val_TMS>, TDI:<val_TDI>, TDO_ARG:<TDO_esperado>, TDO_LINE:<TDO_lido>, RMASK:<pedido_retorno>, RC:<val_sucesso>]*";
- Estado TAP actual do controlador TAP alvo:
  "*[LIB_TAP_<nome_estado>]*" (*e.g.* "*[LIB_TAP_DRUPDATE]*");
- Indicação da declaração SVF a ser executada:
  "*[STATUS] <comando_SVF>*", (*e.g.* "*[STATUS] SDR 15 TDI (0)*");
- Informação sobre o processo de alocação de memória:
  "*[REALLOC:<nome_da_variável>:<tamanho_em_bytes>]*");
- Valores de retorno de *TDO* indicados com o parâmetro "*RMASK*":
  "*Return Value: <valor_binário/hexadecimal>*"
- Descrição dos dispositivos encontrados na cadeia BS:
  "*FOUND: idcode=<val_hex>, revision=<val_hex>, part=<val_hex>, manufactor=<val_hex>*";
- Indicação qualitativa sobre o sucesso da operação;
- Estatísticas de desempenho (descritas na secção 5.2.8);
- Indicação de erros ocorridos.

O valor de TDO devolvido obedece à formatação indicada no parâmetro de entrada "*hex*", que pode ser: binária ('*0*'), onde o LSB é o primeiro a ser recebido, tal como esperado do

funcionamento do BS, e portanto ocupa a posição do *most significant bit* (MSB); hexadecimal "*little endian*" ('*1*'), onde o LSB é o último a ser recebido, e portanto mantém a mesma posição de menos significativo; e hexadecimal de ordem inversa à anterior ('*2*'). Em suma, é importante notar que, pressupondo a leitura de um registo de BS, o binário recebido terá os seus *bits* com a ordenação inversa do valor do registo lido. Este valor, convertido para hexadecimal, resultará no valor devolvido pela opção de hexadecimal de ordem inversa. Este hexadecimal não pode ser classificado como do tipo "*big endian*" já que apenas inverte a ordem dos *bits* recebidos. Recomenda-se a utilização do hexadecimal do tipo "*little endian*", que devolve o valor tal como presente no registo lido.

O ficheiro "*data2.dat*" contempla apenas os erros críticos que levam à interrupção da execução da *libBS* ou que descredibilizam a tarefa concluída. É seguro considerar como bem sucedida uma tarefa quando o ficheiro "*data2.dat*" permanece vazio. A informação "*FINISHED without errors*", presente no final do ficheiro *data1.dat*, pode sempre confirmar o referido. Se em "*data2.dat*" existir qualquer texto, este deve ser apresentado como uma mensagem de erro.

O ficheiro "*devices.dat*" (não confundir com "*devices.cfg*", pertencente à aplicação gráfica) é apenas utilizado para listar os dispositivos encontrados em operações de procura numa cadeia de BS. Os dispositivos são listados de acordo com a ordem contrária à que ocupam na cadeia BS, *i.e.* os últimos elementos da cadeia serão os primeiros a ser listados, e a formatação do seu código de identificação, já convertido para hexadecimal, obedece ao seguinte exemplo (conteúdo do ficheiro *devices.dat* depois de analisar a cadeia BS da placa de teste):

```
Idcode:0x00000000,Revision:0x0,Part:0x0000,Manufa
ctor:0x000
Idcode:0x0940303f,Revision:0x0,Part:0x9403,Manufa
ctor:0x01f
```

No exemplo apresentado, o último dispositivo da cadeia não dispõe de registo de identificação (*SN74BCT8245A*), pelo que se regista um valor de identificação nulo.

Por último, "*BSR_R.dat*" é utilizado para facilitar o acesso aos valores de *TDO* cujo retorno foi indicado no parâmetro "*RMASK*" do código SVF. Os dados devolvidos são alheios à formatação indicada no parâmetro "*hex*" da estrutura de entrada, que só abrange a informação orientada ao utilizador (ficheiro *data1.dat*), sendo sempre apresentados em

binário. Deste ficheiro consta também o valor do total de células lidas, de acordo com a seguinte formatação:

```
44
10000000000001110000000010101000000000000011
```

No valor registado, o primeiro *bit* é o menos significativo, *i.e.* é mantida a ordem com que são recebidos os valores de *TDO*, idêntica à utilizada para registar os valores binários no ficheiro "*data1.dat*".

5.2.7. FUNÇÕES *CHECK_CONNECTION* E *CONTROLO_TAP*

A função *check_connection* recebe como parâmetros a porta série virtual utilizada (*COMx*) e a *baud rate* da ligação UART, com os quais estabelece, se possível, a ligação ao controlador. Seguidamente confirma que este se trata do controlador correcto, através do seu comando de identificação ("*ID_Controlador*", secção 5.1.7). Em caso de sucesso nestas duas tarefas, a função retorna o valor '*1*'. Como não utiliza qualquer ficheiro para *output*, retornando apenas o seu sucesso ou insucesso, não é possível saber o que levou à sua eventual falha. Esta função é apenas utilizada na interface gráfica para verificações rápidas. Para se saber que problema está a ocorrer, em caso de falha, basta correr a função *controlo_svf* com o valor '*3*' no parâmetro "*action*" (equivalente a "*NO_ACTION*", segundo a lista "*pc_comandos*"). Qualquer erro ocorrido será assim convenientemente reportado.

A função *controlo_tap* invoca directamente as funções de I/O de acordo com a estrutura "*pc_tap_pins*" (Tabela 20) recebida, com o objectivo de escrever as linhas do conector JTAG (excepto *TDO*) com os valores indicados; ou de efectuar a leitura de cada uma das suas linhas. Esta efectua o seu *output* recorrendo à criação dos ficheiros "*data1.dat*" e "*data2.dat*", que mantém a sua função já descrita (secção 5.2.6).

5.2.8. RECOLHA DE INFORMAÇÃO ESTATÍSTICA

No seguimento do interesse que veio sendo referido, de colecção de informação que possa servir para efeitos de análise e comparação nos testes ao sistema, é fornecido, de acordo com os dados recolhidos no decorrer de cada execução da função *controlo_svf*, o conjunto de informação apresentado em seguida, servindo como exemplo a execução de um ficheiro SVF para apagar, programar e verificar a memória *FLASH* do MCU *ATmega16*, com *3483* linhas de declarações SVF:

```
STATS:
- Total number of clock cycles: 72743
- Number of significant TDI/TDO bits: 55394/4984
- Transferred bytes: 145510b/4986b/150496b
(Sent/Recv./Total)
- Corrupted send/received blocks: 0 (0 from controller)
- Time taken: 35.17 seconds (est. trans. time: 2.61s)
- Used CPU time: 12.25 seconds
- Avg. TCK Update frequency: 4.14 KHz
```

Destes dados, que se entendem suficientes para o seu objectivo, dois são médias calculadas: O tempo estimado de transferência e a frequência de actualização de *TCK*.

O número total de ciclos de relógio refere-se ao total de ciclos de *TCK* no controlador TAP alvo. O conceito do total de *bits* de *TDI* e *TDO* significativos deverá ser claro: Neste exemplo tanto existem valores de *TDI* relevantes, relativos aos necessários para apagar e programar a memória alvo, como de *TDO*, relevantes devido à verificação efectuada à memória *FLASH* do dispositivo alvo. Em seguida mostra-se o número *bytes* enviados e recebidos, que é um dos grandes factores no desempenho do sistema. Complementando esta informação segue-se a quantidade de blocos de dados corrompidos (*1* ou mais *bytes*, dependendo do comando), que são contabilizados na biblioteca de controlo à medida que são detectados, tal como acontece no *firmware* do controlador (no final da execução da função *controlo_svf*, o programa comunica com o controlador, através do comando "*ER*", para receber a informação de quantos erros foram detectados nos dados recebidos).

O tempo apresentado, respeitante à duração da operação, é contabilizado desde que se inicia a interpretação do código SVF. Já o tempo perdido em transferência é um valor estimado, obtido através do seguinte cálculo:

$$Tempo_{Transf} = \frac{(Bytes_{recebidos} + Bytes_{enviados}) \cdot Impulsos_{trama\_D8N1}}{Baud\ Rate}$$

Onde se consideram *10* os impulsos de cada trama (destinada ao envio de um caractere/*byte*), segundo as configurações utilizadas na secção 5.1.3. Embora a comunicação UART estabelecida seja *full-duplex*[27], não está previsto, no código desenvolvido, situações em que haja a transmissão simultânea nos dois sentidos, o que invalidaria a expressão proposta. Este tempo de transferência considera apenas o tempo

[27] *Full-duplex* caracteriza uma ligação onde ambos os dispositivos podem enviar e receber informação em simultâneo.

dispendido na transferência dos dados enviados/recebidos à velocidade definida, não contemplando quaisquer atrasos introduzidos pelo conversor Série-USB, factor que deverá ser significativo (a ser abordado nos testes da secção 6.2.1).

O tempo total de execução do CPU do computador considera o tempo dispendido em todas as suas *threads*[28]. Por fim, a frequência com que se actualiza *TCK* (metade da frequência real de *TCK*), é apenas uma estimativa da frequência média conseguida, sendo óbvio pela análise ao código que a actualização de *TCK* é marcada por momentos de grande velocidade seguidos de consideráveis esperas. Esta frequência foi calculada pela seguinte expressão:

$$Freq_{Act} = \frac{2 \cdot Ciclos_{TCK}}{Tempo_{Operação}}$$

5.2.9. INTERPRETAÇÃO DE BSDL

O estado de desenvolvimento e maturidade do código "*bsdl2jtag.c*" utilizado − na sua versão mais recente (*v.1.4*) ainda não modificada − não satisfazia totalmente as expectativas para o corrente projecto, pelo que foram efectuadas várias alterações e se mantém, mesmo assim, esta como uma das áreas que poderia beneficiar de novos desenvolvimentos. Listam-se as modificações empreendidas directamente no seu código, em grande parte com vista à sua melhor integração com a aplicação gráfica desenvolvida:

- Como este programa era inicialmente utilizado como um *software stand-alone*, corrido em linha de comandos, foi necessária a sua alteração para que se pudesse aceder a este como uma biblioteca;

- Redefiniram-se os *Inputs*/*Outputs* do código, para que fossem conduzidos para ficheiros, com o intuito de poderem ser interpretados não pela própria biblioteca de controlo, que não tem utilização para tais, mas por outro *software* como a aplicação gráfica desenvolvida;

- Implementou-se a identificação e exportação do nome do componente (*Device name*) de acordo com o ficheiro BSDL lido, informação útil para apresentação ao utilizador;

[28] *Thread* pretende designar cada uma das linhas de processamento simultâneo do CPU

- Implementou-se a distinção entre pinos de "*Output*" e pinos "*Internos*", no ficheiro BSDL, útil para facilitar o controlo dos dispositivos alvo na aplicação gráfica;

- Desactivou-se a listagem dos pinos físicos dos encapsulamentos dos componentes, por ser extensa e não utilizada;

- Alterou-se a organização da informação devolvida;

- Excluiu-se a leitura de informação sobre o comprimento dos registos de *BYPASS* e *IDCODE* por estes estarem definidos na norma BS (e portanto serem imutáveis);

- Acrescentou-se o registo de informação de identificação do componente (*Device idcode*) de acordo com o registo *IDCODE*, em hexadecimal, imprescindível para se poder verificar a adequabilidade do ficheiro BSDL para determinado dispositivo;

- Alterou-se a colecção de instruções para que, para além das instruções IR base obrigatórias, da norma *IEEE 1149.1*, sejam também lidas e devolvidas todas as instruções opcionais disponíveis para o componente (nome e codificação);

A listagem dos comprimentos dos registos e de todas as instruções IR é especialmente significativa para os objectivos do projecto, pois pretende-se disponibilizar, na interface gráfica, um controlo assistido abrangente. Com tal objectivo, implementou-se, por exemplo, o mecanismo de introdução automática de instruções IR, bastando seleccionar o nome da instrução desejada e o dispositivo alvo, o que necessita da leitura das instruções e respectiva codificação do ficheiro descritivo de cada componente. Estes automatismos serão abordados em maior detalhe quando em referência ao desenvolvimento da aplicação gráfica. Desta forma, a aplicação está preparada para contemplar automaticamente as instruções que possam surgir.

Realçam-se também algumas limitações encontradas no código utilizado para a interpretação de BSDL, que não foram corrigidas neste projecto:

- Não distingue directamente entre os *bits*, do registo BSR, de *Output* (de dois e três estados) e *bits* bidireccionais. Os *bits* de *output* de dois estados pôdem, ainda assim, ser distinguidos dos restantes através da atribuição da informação sobre o *bit* de controlo extra aos *bits* de *output* de três estados ou bidireccionais;

- Na listagem dos pinos físicos do componente não era recolhida nem fornecida a identificação do tipo de encapsulamento a que se referiam. Este procedimento foi removido do actual projecto, mas esta informação poderia vir a ser utilizada;

- Não são lidas algumas informações complementares, como, por exemplo, a frequência máxima de *TCK* suportada pelo componente.

Para além destas limitações, foi também encontrado um problema mais grave, cuja causa está para além de todo o código acrescentado. Na leitura de alguns ficheiros BSDL, pouco frequentes, que fogem de alguma forma à sintaxe esperada, o programa vê a sua execução bloqueada. A causa deste problema não foi descoberta devido à dificuldade de depurar problemas em código acedido através da invocação de uma biblioteca dinâmica do tipo *unmanaged*. Na interface gráfica foi implementada a funcionalidade que deverá resolver o problema − "Verificação de erros nos ficheiros BSDL" − a ser abordada, juntamente com o problema descrito, na secção 5.3.8. É, ainda assim, de considerar a implementação de um mecanismo que aborte a execução do programa em caso do seu tempo de execução se prolongar para lá do que seria expectável na pior das hipóteses, a fim de impedir o encravamento de todo o sistema.

A função de acesso externo às funcionalidades desta parte do programa da *libBS.dll* (*read_BSDL*) requer, como argumento, apenas o caminho para o ficheiro BSDL a ser interpretado:

```
__declspec(dllexport) void __cdecl
read_BSDL(char * file_location)
```

Esta função está implementada no ficheiro "*bsdl2jtag.c*" e pretende traduzir os ficheiros BSDL para uma descrição do componente numa formatação mais intuitiva (que pode ser reorganizada alterando-se as linhas identificadas, em comentário, como responsáveis pela impressão da informação). Esta cria, no mesmo directório de *libBS.dll*, o ficheiro "*description.dat*", com a informação obtida do ficheiro BSDL.

Expõe-se a descrição do MCU *ATmega16*, tal como devolvida pela função criada:

```
1.   Device name ATmega16
2.   Device idcode C940303F
3.   BSR length 141
4.   IR length 4
5.   instruction EXTEST 0000 BSR
6.   instruction IDCODE 0001 DIR
7.   instruction BYPASS 1111 BR
```

```
     (...)
8.   bit 140 D 1 *
9.   bit 139 D 0 *
     (...)
10.  bit 90 O 1 PB2 89 0 Z
11.  bit 89 C 0 *
     (...)
12.  bit 72 I 1 RESET
     (...)
13.  bit 0 D 0 *
```

De notar que a numeração das linhas foi acrescentada para facilitar a sua referência e não constam do ficheiro descritivo. Várias linhas estão também omissas devido à extensão do ficheiro. A ordenação das informações no ficheiro descritivo devolvido, que deverá ser interpretado por algum *software*, como a aplicação gráfica desenvolvida, é mantida. No entanto, nem todas as linhas estão sempre presentes. A primeira linha, que identifica o nome do dispositivo existirá sempre, mas, por exemplo, a segunda já surgirá apenas se o dispositivo dispuser de registo de identificação, acontecendo o mesmo relativamente à linha '*6.*'. O *software* responsável pela leitura desta informação deverá ter em conta esta mutabilidade parcial dos ficheiros devolvidos.

Descreve-se rapidamente a restante informação fornecida, relembrando que o número das linhas indicado é apenas válido no exemplo actual: As linhas '*3.*' e '*4.*' especificam os comprimentos dos registos BSR e IR, em *bits*; As linhas '*5.*' a '*7.*' identificam as várias instruções disponíveis, segundo a seguinte formatação:

- ***instruction*** *<nome_da_instrução> <codificação> <registo_alvo>*

O registo alvo de cada instrução, aquele que será seleccionado entre *TDI* e *TDO*, apresenta-se pela sua abreviatura: *Boundary Scan Register* ("*BSR*"); *Instruction Register* ("*IR*"); *Bypass Register* ("*BR*"); e *Device Identification Register* ("*DIR*"). As instruções opcionais, como a instrução *AVR_RESET* ("*1100*") que está disponível no dispositivo em análise, não têm a indicação do seu registo alvo.

As restantes linhas (linha *8.* em diante) são responsáveis pela informação relativa a cada uma das células do registo BSR. As células são apresentadas em ordem descendente e seguem a seguinte formatação (os parâmetros entre parêntesis rectos surgem apenas nas células bidireccionais):

- ***bit*** *<nº_célula> <tipo> <valor_default> <descrição>* [*<célula_de_controlo>*] [*<valor_default>*] [*<resultado>*]

O *bit* número zero indica a célula menos significativa do registo BSR, e portanto o primeiro valor a ser arrastado para *TDI* na sua escrita.

O tipo de cada célula pode ser distinguido em quatro casos:

Tabela 21 Tipos de células BSR na interpretação de BSDL

Designação:	Tipo de célula:
I	*Input* ou de observação
O	*Output* ou Bidireccional
C	Controlo
D	Interna

O campo da descrição identifica o nome do pino físico controlado pela célula, mas pode também ser utilizado para apresentar uma qualquer descrição da célula (não deverá conter espaços), opção oferecida na interface gráfica. É apresentado um '*', como consta na linguagem BSDL, se a célula não estiver directamente associada a qualquer pino físico.

Os restantes parâmetros, realçados entre parêntesis rectos, surgem apenas nas células bidireccionais, servindo para distinguir entre as células do tipo '*O*': as bidireccionais daquelas que servem unicamente para *Output*. O campo da "célula de controlo" indica o número do *bit* do registo BSR responsável pelo controlo da direccionalidade da célula em questão. Este *bit* deve ser do tipo de controlo ('*C*'), como pode ser conferido no exemplo apresentado. Os dois restantes parâmetros podem ser desprezados, também não sendo utilizados na aplicação desenvolvida. Estes indicam o estado em que a célula de controlo desactiva o *driver* da célula original (será o mesmo valor *default* apresentado na linha da própria célula de controlo, como se pode ver no exemplo fornecido) e o resultado esperado na célula bidireccional para esse valor: '*Z*' indica o estado de alta impedância, *i.e.* funciona como *input*. Outras possibilidades são "*Weak0*" e "*Weak1*", que indicam sinais fracos (normalmente ocorrem em circuitos do tipo *emitter-coupled logic* − ECL)[6].

5.3. APLICAÇÃO GRÁFICA

A aplicação gráfica, que tira partido de todas as funcionalidades implementadas na biblioteca de controlo descrita, foi desenvolvida com a pretensão de ser utilizada por qualquer utilizador mesmo que alheio às especificidades do *IEEE 1149.1*. No entanto, embora se considere que o resultado obtido seja bastante intuitivo, não é possível simplificar a aplicação da forma pretendida e, ao mesmo tempo, disponibilizar o controlo

total sobre as funcionalidades à disposição. Um exemplo imediato desta dicotomia de intentos é, por exemplo, quando se fornece ao utilizador a possibilidade de se editar o registo BSR (que se apresenta acompanhado de conveniente identificação de cada célula), mas o resultado da sua escrita e leitura está dependente da instrução que se escolhe atribuir ao controlador TAP alvo: *SAMPLE; PRELOAD*; *EXTEST*; *etc*. Era possível obrigar as leituras a serem feitas em modo *SAMPLE* e as escritas em *EXTEST*, tornando o processo transparente, mas tal seria uma simplificação excessiva, fosse o utilizador querer proceder de outra maneira. Ao longo do desenvolvimento desta aplicação, como se fará conta nesta descrição, existiram situações em que foram realmente programadas simplificações, como a introdução automática de instruções (*e.g.* a aplicação de instruções de *Bypass* quando se pretende isolar um dispositivo alvo), mas sempre com a preocupação de tentar não limitar a controlabilidade das operações.

A aplicação foi dotada de diversas funcionalidades que, mesmo podendo, em alguns casos, não estar totalmente relacionadas entre si (*e.g.* a programação de MCUs e a verificação de ficheiros BSDL), conseguissem aumentar o número de situações onde a esta será útil. Como esta aplicação viria a servir de teste a todo o restante sistema, para além das funcionalidades pretendidas foram sendo acrescentadas algumas opções para teste e depuração de problemas, das quais não se antevê proveito para o seu utilizador final. Ainda assim, estas foram mantidas disponíveis nos menus de opções.

Os ficheiros listados na Figura 74 compõem o projecto da programação da aplicação:

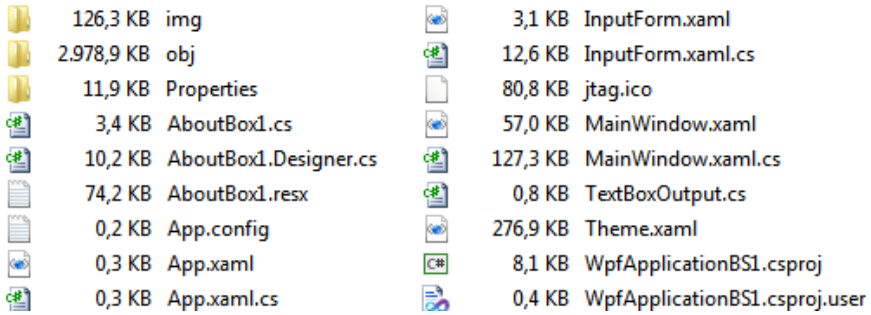


Figura 74 Listagem dos ficheiros da programação da aplicação gráfica

O código foi escrito em linguagem *C#* (ficheiros com extensão "*.cs*") e a sua compilação foi efectuada através do "*MS Visual Studio 2012*", tendo-se definido uma plataforma *x86* como alvo, para garantir compatibilidade com o código da biblioteca de controlo. O sistema gráfico da interface é baseado em WPF, cujo *design* é descrito em linguagem XAML (ficheiros com extensão "*.xaml*"). Embora o IDE permita um controlo satisfatório

sobre o *design*, à medida que este se torna mais complexo é proveitosa a edição manual do próprio código XAML, utilizando-se o "*designer*" da aplicação apenas para verificação.

O nome do projecto é "*WpfApplicationBS1*", tendo sido dado um nome mais convencional à aplicação: "*JTAG Toolbox*". Foi-lhe associada o seguinte ícone:

Figura 75 Ícone da aplicação gráfica

A execução do programa desenvolvido carece da instalação do "*.NET Framework 4.5*". À pasta resultante da compilação da aplicação devem também ser acrescentados os seguintes ficheiros:

- *libBS.dll* – ficheiro da biblioteca de controlo;
- *avrsvf.exe* e *devices.cfg* – ferramenta executável da *Atmel* para criação de ficheiros SVF;
- "*manid.dat*" − ficheiro que reúne as informações necessárias para a identificação de dispositivos;
- Pasta "*Bsdl_checker*" – programa para verificação de ficheiros BSDL.

O propósito de cada um destes ficheiros que, à excepção de "*libBS.dll*", surgem aqui mencionados pela primeira vez, é enunciado no decorrer desta descrição. Na execução da aplicação, dependendo das tarefas executadas, poderão ser criados vários outros ficheiros.

Como complemento, foi reunido um largo conjunto de ficheiros BSDL destinados a serem carregados na aplicação quando necessários. Estes encontram-se na pasta "*BSDLFiles*", a qual inclui também dois ficheiros descritivos relativos aos componentes da placa de teste, criados pela própria aplicação (com a extensão "*.dev*"), que não respeitam a linguagem BSDL e se destinam a serem utilizados apenas com esta aplicação.

5.3.1. ACESSO À BIBLIOTECA DE CONTROLO

A informação descrita nesta subsecção é de especial interesse por servir de exemplo no caso de se pretender desenvolver um novo *software* que tire igualmente partido das

funções da biblioteca de controlo, como se havia já incentivado, tentando-se guiar e facilitar a sua célere implementação no novo desenvolvimento.

O acesso às funções da biblioteca de controlo foi feito através do método de importação de código de uma biblioteca dinâmica ("*DllImport*"). Este método permite a um programa escrito em linguagem *C#* comunicar com a API de outros componentes nativos, geralmente um *dll* escrito em *C* ou *C++*. Porque a aplicação vai utilizar componentes nativos (linguagem *C*), foi necessário adicionar uma "*flag*" ao compilador identificando o destino como uma plataforma *x86*. Esta, ao especificar que a aplicação tem como alvo uma plataforma nativa (*x86*), permitirá que a aplicação invoque funções nativas. Não definir este parâmetro no compilador deverá resultar num erro sempre que sejam chamadas as funções da biblioteca de controlo.

Para a chamada de funções de uma biblioteca não gerida ("*unmanaged*"), o *.NET Framework*, utilizado pela aplicação gráfica, requer a representação do protótipo da função em código gerido ("*managed*"), para que seja possível a sua correcta invocação e *marshalling* de dados. O processo de *marshalling* é também mandatório, em alguns casos, para se conseguir comunicar. Este processo é similar à serialização dos dados, e é um mecanismo usado para transportar informação entre processos ou *threads*. Foi utilizado, no caso actual, apenas para o envio das variáveis do tipo *string* das estruturas que servem como argumento das funções da biblioteca dinâmica.

As funções da biblioteca destinadas a ser acedidas externamente foram então declaradas, no código da aplicação, com o seguinte protótipo:

```
[DllImport("libBS.dll", EntryPoint = "controlo_svf",
CallingConvention = CallingConvention.Cdecl)]
extern static void controlo_svf(SVF pc);
```

A primeira linha do excerto de código transcrito informa sobre o *dll* que se pretende aceder, no caso a biblioteca de controlo *libBS.dll*; da posição a partir da qual surge a função pretendida ("*Entry Point*"); e da convenção de chamada. Por omissão, o método "*dllimport*" utiliza a convenção *StdCall*, que é diferente da utilizada pelo código da biblioteca de controlo, devendo por isso ser escolhida a *Cdecl*, tal como já se havia feito referência anteriormente.

As estruturas que servem como argumentos às funções, e são declaradas no programa da biblioteca, devem ser novamente declaradas na aplicação. No entanto, estas requerem,

pelos mesmos motivos de compatibilidade entre a aplicação e a biblioteca, de cuidados específicos na sua declaração. Serve como exemplo a declaração da estrutura *SVF* que é utilizada como argumento da função *controlo_svf*:

```
[StructLayout(LayoutKind.Sequential, CharSet =
CharSet.Ansi)]
public struct SVF
   {
  public int baud;
  public int com;
  [MarshalAsAttribute(UnmanagedType.LPStr)]
  public string file;
   (...) //Restantes declarações
}
SVF dados = new SVF();
```

A classe *StructLayoutAttribute* apresentada permite escolher o *layout* sequencial da estrutura e o conjunto de caracteres para o qual serão convertidas as *strings* "*managed*". Os compiladores de *C#*; *C++*; entre outros; já utilizam este tipo de *layout* e de conjunto de caracteres por definição, pelo que a sua referência poderia ser dispensada. Nas variáveis do tipo *string*, foi necessário fazer *marshalling* ao seu conteúdo, através da classe *MarshalAsAttribute*, escolhendo-se o tipo "*UnmanagedType.LPStr*", que corresponde ao envio de um apontador para um vector de caracteres *American National Standards Institute* (ANSI) terminado em '*\0*', compatível com o código da biblioteca de controlo.

As funções podem ser agora utilizadas da forma convencional ao longo do código, como no seguinte exemplo, onde se mostra a atribuição dos parâmetros da ligação à estrutura enviada como argumento e, posteriormente, a chamada da função externa:

```
dados.baud = int.Parse( textboxbaud.Text );
dados.com = int.Parse( Regex.Match(
  comboboxcom.SelectedItem.ToString(), @"\d+").Value );
(...) // Preenchimento da restante estrutura SVF
controlo_svf(dados);
```

No entanto, é de referir que o fluxo de execução do programa passará para a biblioteca dinâmica, o que significa que a aplicação gráfica deixará de responder durante os momentos em que a função externa estiver a ser executada. Tal facto só acontece, mais uma vez, por se tratar de código *unmanaged*, e leva a que a interface gráfica congele, não dando qualquer resposta. Para o evitar, é necessário criar uma nova *thread* de processamento e encarrega-la de executar as funções da biblioteca, de forma que a aplicação não cesse a sua própria execução, permitindo que a interface do utilizador permaneça funcional. Com esta abordagem é necessário implementar mecanismos de

sincronização das *threads* em execução, para que a aplicação reconheça a conclusão das tarefas, interprete os seus resultados, e actualize a interface gráfica em concordância.

As funções invocadas retornam convenientemente os valores devidos. Já as linhas de *standard error* e *standard output* são perdidas. Por exemplo, na ocorrência de um erro na biblioteca de controlo, encontraram-se apenas duas formas eficazes de o comunicar: escrevendo-o num ficheiro de registo ou retornando-o como resultado da execução da função.

5.3.2. UTILIZAÇÃO DE PROGRAMAS EXTERNOS

Salvo algumas pequenas parcelas de código, para satisfazer necessidades pontuais na programação, não foi utilizado código externo no programa desenvolvido, como havia acontecido na biblioteca de controlo, tendo todas as funcionalidades, apresentadas nesta descrição, sido integralmente desenvolvidas no interesse do corrente projecto.

No entanto, foram utilizadas duas ferramentas de terceiros: "*Goepel BSDL SyntaxChecker*"; e "*Atmel AVRsvf v.1.7*". Estas são operadas automaticamente pela aplicação gráfica, que lhes fornece os parâmetros de entrada necessários e interpreta o seu *output*. Descrevem-se sucintamente estas ferramentas:

- *Goepel BSDL SyntaxChecker v4.1*

  Software freeware para a análise da sintaxe de ficheiros BSDL de qualquer fabricante. Para além da indicação de cada erro, esta devolve igualmente uma pequena descrição sobre estes, permitindo ao utilizador corrigir os eventuais problemas.

- *Atmel AVRsvf v.1.7*

  Ferramenta oficial da *Atmel*, também gratuita, para a criação de ficheiros SVF capazes de programar os seus MCUs com suporte para BS. As suas funções incluem: Apagar, programar ou verificar as memórias *FLASH* e *EEPROM*; programar e verificar *fuses* e *lockbytes*; e verificar a assinatura dos componentes.

Estes *softwares* externos são utilizados através da criação de um processo para a sua execução, redireccionando as suas linhas de *standard output* e *standard error* para a aplicação. A sua execução não precisa da criação de múltiplas *threads*, se o objectivo for correr sincronamente o *software* externo, pois a interface gráfica não deixará de ser actualizada. O excerto seguinte resume a implementação do referido:

```
System.Diagnostics.ProcessStartInfo procStartInfo =
new System.Diagnostics.ProcessStartInfo("cmd", "/c " +
command);
procStartInfo.RedirectStandardOutput = true;
procStartInfo.RedirectStandardError = true;
 (...)
System.Diagnostics.Process proc = new
System.Diagnostics.Process();
proc.StartInfo = procStartInfo;
proc.Start();
```

A variável "*command*", destacada a negrito, indica o programa que se pretende executar, que deverá estar localizado no directório da aplicação gráfica. O seu *output* é recebido em "*proc.StandardOutput*" ou "*proc.StandardError*".

5.3.3. FUNCIONAMENTO GERAL

A interface gráfica inicia-se na janela "*Main Window*", cujo *design* se encontra no ficheiro "*MainWindow.xaml*" acompanhado da sua programação no ficheiro homónimo de extensão "*.cs*", principais ficheiros do programa. Apresenta-se na Figura 76 o seu aspecto visual e organização das suas funcionalidades:

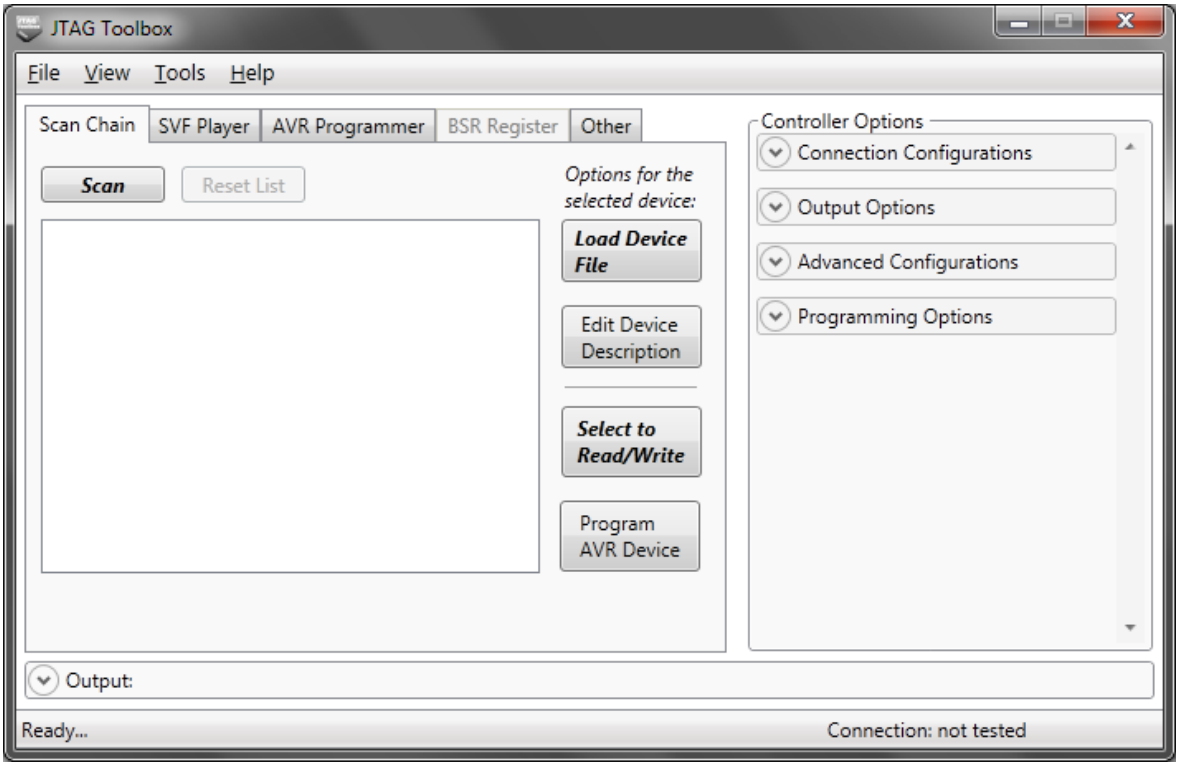


Figura 76 Aspecto inicial da aplicação gráfica (janela *MainWindow*)

O menu superior e toda a área central predominante (à esquerda) permitem efectuar as várias tarefas disponibilizadas na aplicação, que irão sendo descritas no decorrer das

próximas secções. À direita, empilha-se um conjunto de menus expansíveis com as opções e parâmetros de funcionamento de intervenção nas funcionalidades oferecidas. Em baixo, a expansão da barra de *output*, inicialmente fechada, vem mostrar uma consola onde são apresentadas todas as informações (de carácter não essencial) relativas ao decorrer das tarefas e seus resultados, como exemplificado na Figura 77:

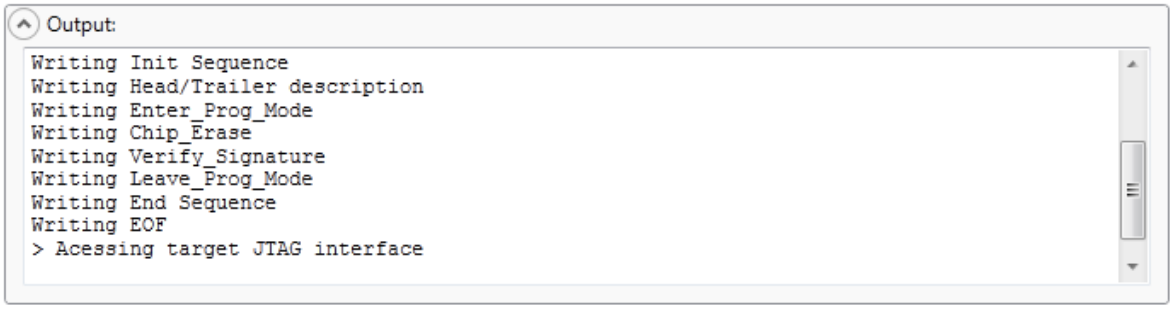


Figura 77 Exemplo do aspecto da caixa de *output* expandida

O texto apresentado é limpo de cada vez que é executada uma nova operação que recorre ao controlador. Toda a informação do *standard output* e *standard error* da aplicação é redireccionada para esta consola. Alguns erros poderão, após serem capturados na execução do programa (utilizando o mecanismo de *try-catch*), ser enviados para o *standard error*, pelo que surgirão na janela de *output* acompanhados de informação técnica, mas na sua generalidade, esse tipo de erros internos, se ocorrerem, não são comunicados ao utilizador se não forem críticos. A barra de *scroll* da janela de *output* acompanha automaticamente o surgimento de novo texto.

No menu sobre o *output* ("*Output options*", na Figura 78) dispõem-se as seguintes opções:

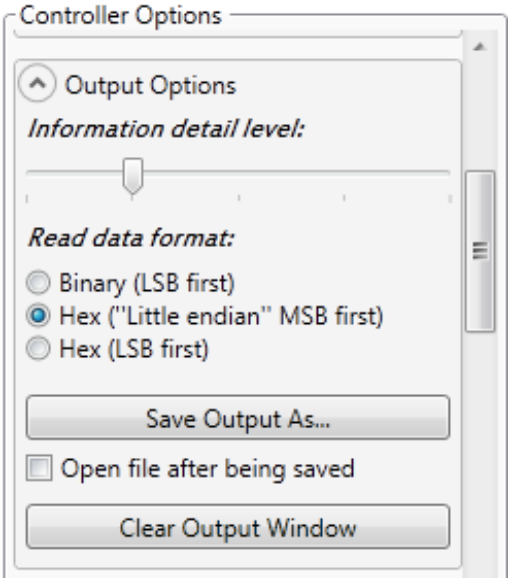


Figura 78 Opções de *output* da aplicação gráfica

O nível de detalhe de informação (*0 - 4*) serve para comunicar à biblioteca de controlo, no seu parâmetro de entrada "*verb*", o nível de retorno de informação pretendido no ficheiro "*data1.dat*" (descrito na anterior secção 5.2.6), que é integralmente disposta na caixa de *output* da interface. Recomenda-se seleccionar, no máximo, o nível '*2*' de detalhe, ao pretender-se utilizar o modo de baixa velocidade para se obter as instruções em tempo real, ou manter o nível '*1*' pré-definido, ao utilizar-se o modo de alta velocidade. Um nível de detalhe superior tem apenas utilidade na depuração do sistema e pode causar instabilidade na aplicação por originar um conjunto demasiado extenso de informações. O formato dos dados de retorno serve apenas para preencher o argumento "*hex*" da biblioteca, que devolverá os valores pedidos nesse formato, não tendo mais qualquer intervenção nos restantes dados presentes na aplicação. O botão para guardar o texto da consola num ficheiro de texto surge de seguida, juntamente com a opção de abrir o ficheiro guardado. Esta função foi prevista para comparar operações efectuadas, já que a janela de *output* é reinicializada com cada nova tarefa. Em último lugar aparece o botão para limpar o conteúdo da caixa de *output*.

Ainda relativamente à organização da interface gráfica, a barra de estado ocupa a posição mais inferior da janela e indica a situação da aplicação, que pode estar pronta a ser executada ("*Ready*") ou em estado ocupado ("*Busy*"), caso onde é também descrita a operação a ser executada. Só é possível executar uma operação de cada vez (contando que recorra ao controlador), sendo necessário esperar até que o programa a conclua para se poder voltar a utilizar o controlador de BS. No decorrer da execução de uma operação, a interface gráfica mantém-se funcional, podendo ir-se preparando a operação seguinte, mas esta apenas pode ser executada ao sinal "*Ready*" na barra de estados. Excepto na execução de ficheiros SVF ou na programação de MCUs, esta transição deverá ser rápida, se quase instantânea.

Antes de se iniciar a descrição das possibilidades da aplicação, quer-se ainda deixar uma referência à opção "*About*", que abre uma nova janela, apresentada na figura Figura 79:

Figura 79 Menu "*Help*" e janela de informação sobre a aplicação (janela "*AboutBox1"*)

Querendo-se alterar esta informação, como esta é obtida dos atributos *assembly* do projecto, é necessário editar o ficheiro "*Properties\AssemblyInfo.cs*" em:

```
[assembly: AssemblyTitle("JTAG Toolbox")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("LABORIS ISEP-IPP")]
[assembly: AssemblyProduct("JTAG Toolbox")]
(...)
```

5.3.4. LIGAÇÃO AO CONTROLADOR

Os parâmetros da ligação ao controlador são definidos na interface e enviados como argumento para a biblioteca de controlo. Estes podem ser escolhidos no conjunto de opções "*Connection Configurations*", apresentado na Figura 80:

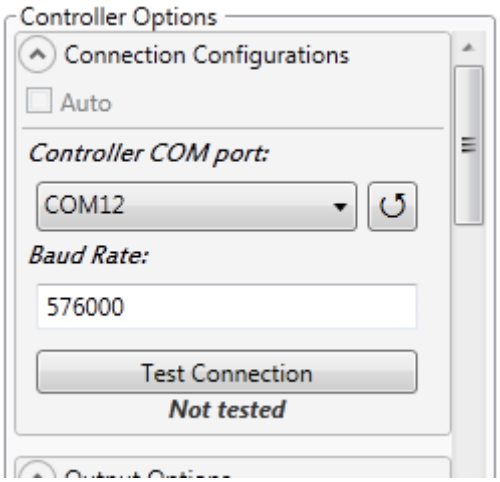


Figura 80 Opções sobre a ligação controlador

Para o estabelecimento da comunicação com o controlador é necessário definir a porta *COMx* e a *baud rate* da ligação. A lista das portas *COM* é devolvida pelo próprio sistema, pelo que se o controlador só for ligado após a aplicação já estar em funcionamento, é conveniente utilizar o respectivo botão para actualização da lista a fim de reverificar as portas *COM* disponíveis. É sempre automaticamente seleccionada a última entrada a ser devolvida pelo sistema. Porque o sistema vai registando as portas *COM* existentes pela ordem que surgem, independentemente da sua numeração, normalmente a última entrada devolvida corresponde sempre ao controlador, já que não é muito usual actualmente a utilização de outros periféricos baseados em portas série, mas entende-se que tal possa não se verificar, razão pela qual se pretendeu implementar um modo automático. Este visava a escolha e teste automático dos parâmetros da ligação, mas acabou por ver a sua implementação preterida em função de outros desenvolvimentos que se mostravam mais importantes.

A função de verificação da ligação recorre à função "*check_connection*" da biblioteca de controlo e retorna o resultado do teste na pequena legenda apresentada ("*Not Tested*", na Figura 80), actualizando também o campo respectivo da barra de estados da aplicação.

Os passos de alteração e verificação dos parâmetros da ligação só são necessários se ocorrer algum erro relacionado, ao efectuar qualquer uma das funcionalidades da aplicação, dispensando-se a atenção do utilizador a estas opções de cada vez que inicie a aplicação. O valor da *baud rate*, não sendo alterado no *firmware*, estará sempre correcto, e a porta *COM*, pelo mecanismo referido, deverá igualmente ser correctamente escolhida sem intervenção do utilizador.

5.3.5. EXECUÇÃO DE FICHEIROS SVF (SEPARADOR "*SVF PLAYER*")

Não sendo o separador principal, inicia-se a descrição das funcionalidades da aplicação pelo separador que permite a execução de ficheiros SVF. A capacidade de executar código SVF é de importância central no funcionamento do programa ao ser necessária para todas as suas funções. A Figura 81 mostra este separador:

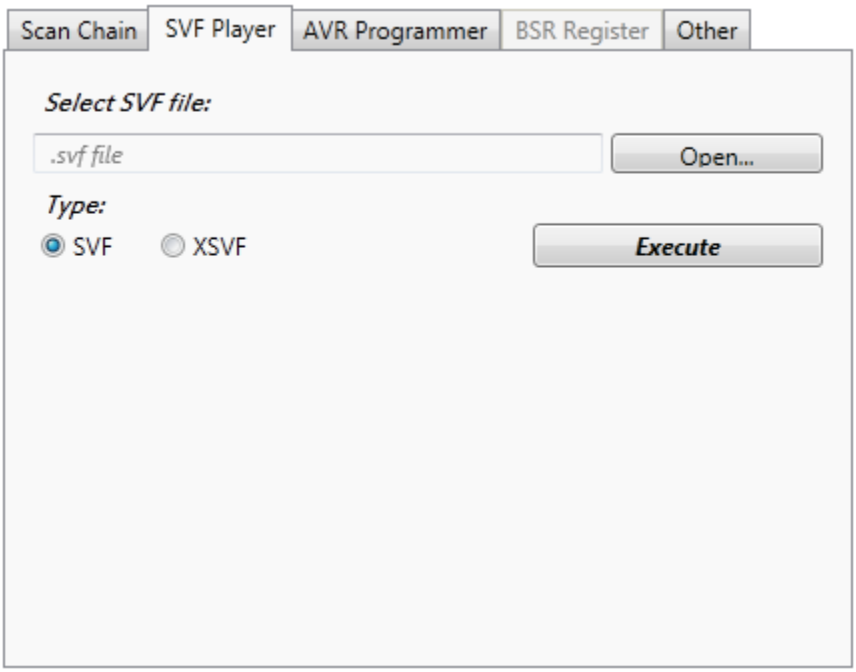


Figura 81 Separador "*SVF Player*"

Esta ferramenta de execução muito simples, requer apenas a selecção do ficheiro SVF a executar, abrindo uma janela para explorar os ficheiros do sistema, e da selecção do tipo de código nele contido (SVF ou XSVF).

A informação recebida na consola, sobre a execução do procedimento, depende especialmente de dois factores: o nível de detalhe escolhido nas opções de *output*; e o modo de operação do controlador de BS. Este último é definido em "*Advanced Configurations*", onde surgem também outras opções de interesse, apresentada na Figura 82:

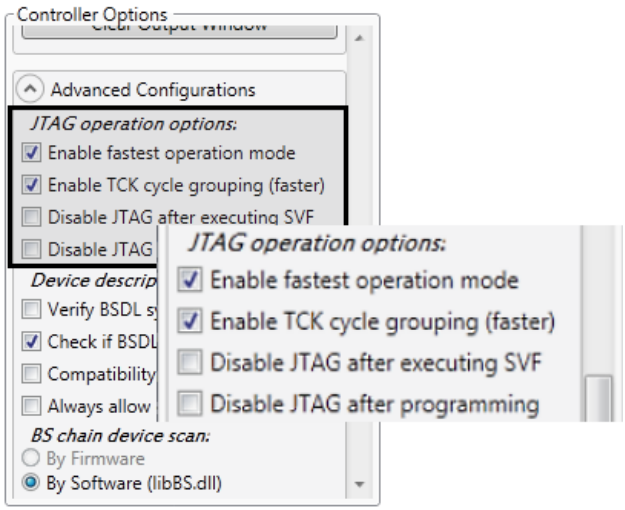


Figura 82 Secção "*JTAG operation options*" do menu "*Advanced Configurations*"

A aplicação, por definição, utiliza o modo de funcionamento normal com agrupamento, no que concerne à forma de actuação do controlador pelo programa da biblioteca de controlo. A vantagem real de utilizar o modo de baixa velocidade comparativamente à predefinição é, na prática, apenas o facto de se conseguir actualizar o *output* da consola à medida que se vão executando as instruções, ficando-se com uma ideia clara das tarefas JTAG a serem executadas a cada momento ("*Status*") e a sua velocidade. O modo mais célere é accionado na primeira opção deste conjunto e, à excepção de não possibilitar a referida actualização da informação no decorrer da operação de BS, não se encontrou qualquer problema na sua execução. A opção seguinte – "*Disable JTAG after executing SVF*" – declara o intuito de, após executar um ficheiro SVF, colocar o conector JTAG em modo de alta impedância e cessar o controlo da cadeia BS alvo. Deve manter-se esta opção desactivada se o ficheiro SVF for incompleto e pretender-se continuar o controlo BS em seguimento das operações já executadas. Em seguida apresenta-se um excerto do *output* da consola no modo de baixa velocidade (pelo que esta informação foi sendo devolvido à medida que se processavam as tarefas), no nível '*2*' de detalhe:

```
> Acessing target JTAG interface
(...) //Parâmetros Verbose, Com port, e Baudrate
[Connection estabilished]
[Correct BS Controller Found!]
[Executing SVF file: 'prog.svf']
[SETUP]
(...) //Instruções SVF iniciais
[STATUS] RUNTEST 7E-3 SEC
[STATUS] SDR 15 TDI (3600) TDO (0067) MASK (00FF)
[STATUS] SDR 15 TDI (3700) TDO (0000) MASK (00FF)
(...) //Restantes instruções SVF e Info estatística
FINISHED without errors.
```

Este tipo de *output* ocorre em todas as tarefas que recorrem à função *controlo_svf* da biblioteca de controlo, ou seja, todas as que carecem da interpretação de comandos SVF.

Para além da última mensagem do *output*, que indica se ocorreram erros, mostra-se também uma janela a indicar o sucesso da operação ou, caso contrário, o problema ocorrido. No caso de, existindo verificações de *TDO* definidas nas instruções SVF, for verificado algum valor não esperado, a execução é imediatamente interrompida e apresenta-se o erro "*TDO mismatch*". Assim, ao executar ficheiros SVF em que se quer, por exemplo, verificar previamente alguma condição antes de se efectuar o procedimento desejado, a biblioteca de controlo cessará o funcionamento logo que falhe a verificação planeada.

Durante a execução do ficheiro SVF surge uma barra de *loading* indeterminada, que indica que o programa está a executar o procedimento pretendido. A seu lado, um botão para abortar a operação interrompe abruptamente a tarefa, mediante confirmação do utilizador. Estes elementos aparecem também convenientemente enquadrados em outros separadores, e são independentes entre si (*i.e.* se estiver a ser executada uma tarefa em determinado separador, apenas nele aparecerão estes elementos). A Figura 83 mostra a sua disposição na parte inferior dos separadores:

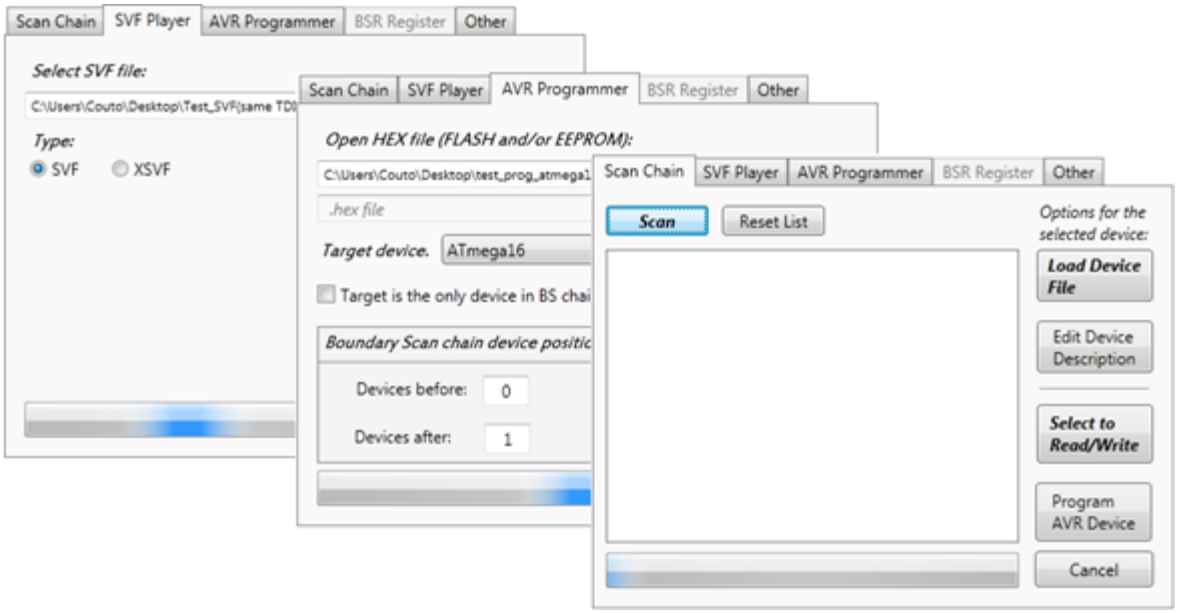


Figura 83 Barra de *Loading* e botão "*Cancel*" nos vários separadores

5.3.6. PROGRAMAÇÃO DE MCUS AVR (SEPARADOR "*AVR PROGRAMMING*")

A lista dos MCUs *AVR* suportados é carregada, ao iniciar a aplicação, directamente do ficheiro "*devices.cfg*", pertencente à ferramenta *AVRsvf*, que deve estar presente no directório da aplicação. A compatibilidade da aplicação com os diversos MCUs é garantida por esse mesmo ficheiro, onde estão guardadas as características técnicas necessárias de cada MCU. Como os dispositivos são assim carregados, é expectável que, caso surja uma nova actualização da ferramenta da *Atmel*, que aumente a lista de dispositivos compatíveis, baste substituir a versão actualmente utilizada para que a aplicação gráfica considere os novos componentes e executável, contando que o seu funcionamento base não sofra alterações.

A programação de MCUs *AVR* com suporte de BS é feita através do separador "*AVR Programming*", apresentado na Figura 84:

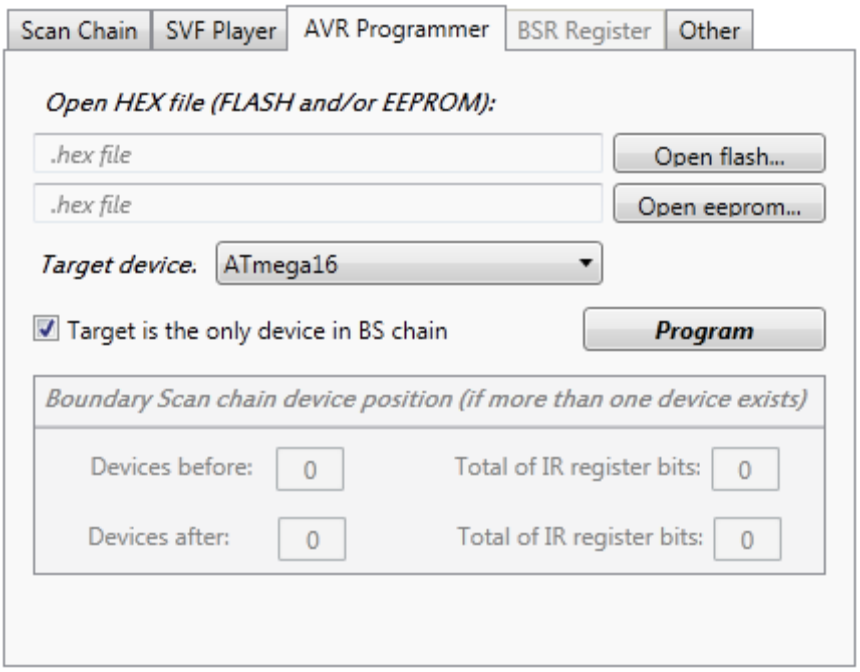


Figura 84 Separador "*AVR Programmer*"

O espaço superior é destinado ao carregamento dos ficheiros, de extensão "*.hex*", para programação na memória *FLASH* e *EEPROM* do MCU. Estes só têm de ser seleccionados se forem necessários para a operação que se pretende.

Se o MCU que se pretende programar não for o único elemento com interface BS na placa alvo, é necessário que a programação seja para ele direccionada, pelo que é preciso desseleccionar a opção "*Target is the only device in BS chain*", activando-se a caixa onde se pode indicar o seu posicionamento. Esta deve conter o número de dispositivos BS antes e depois do MCU *AVR*, assim como o comprimento total dos seus registos IR. Esta informação é necessária para colocar todos os outros dispositivos em modo de *BYPASS* e calcular os *headers* e *trailers* do código SVF.

O objectivo do botão "*Program*" tem de ser escolhido com recurso às opções que surgem no menu "*Programming Options*", como listadas na Figura 85:

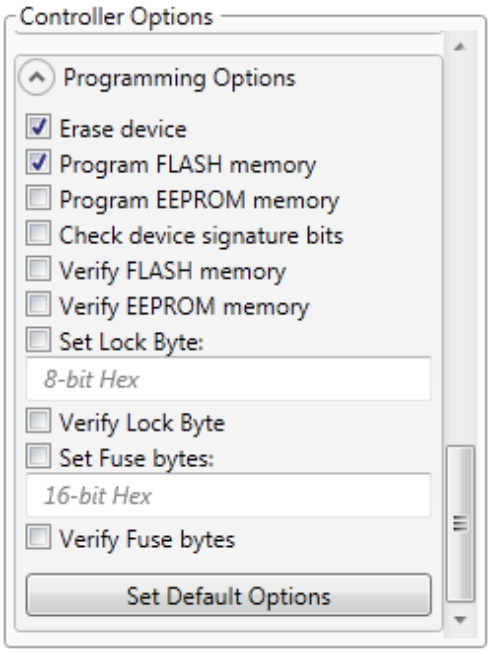


Figura 85 Opções sobre a programação de MCUs *AVR*

As opções recomendadas para a programação implicam apagar previamente a memória do dispositivo e depois programar a sua memória *FLASH* com o ficheiro "*.hex*" que deve ser seleccionado para o efeito (opções escolhidas pelo botão de repor a configuração inicial). As restantes opções disponíveis devem ter uma designação suficientemente clara para não suscitar dúvidas.

O sistema efectua a programação desejada em três etapas fundamentais: Primeiramente, este recolhe as opções do utilizador e o caminho dos ficheiros necessários, e cria os parâmetros de entrada para o *software AVRsvf*. Este é executado, criando um ficheiro SVF para o objectivo definido nos seus parâmetros de entrada. Finalmente, o ficheiro SVF é executado, numa operação em tudo similar à da execução de um ficheiro SVF no respectivo separador. Estas fases são referidas na consola (nível '*2*' de detalhe, modo de funcionamento mais célere), no decorrer da tarefa, como se transcreve:

```
> Reading programming parameters:
(...)
> Executing AVRsvf module:
AVRsvf v1.7 (C) 2007 Atmel Corp.
(...)
Writing Head/Trailer description
Writing Verify_Signature
Reading HEX input file.. OK
(...)
> Acessing target JTAG interface
Real-time information output is not possible in
Faster speed mode.
Please wait for operation conclusion...
```

A velocidade de execução desta tarefa depende do MCU alvo e, também, dos restantes componentes da cadeia BS. Na posterior secção 6.2.2 indica-se o desempenho obtido na programação do *ATmega16* utilizado na placa alvo.

5.3.7. PESQUISA DE DISPOSITIVOS NAS CADEIAS DE BS (SEPARADOR "*SCAN CHAIN*")

A procura de dispositivos na placa alvo é iniciada através do botão "*Scan*", em destaque no separador "*Scan Chain*" (apresentado na Figura 86). O sistema alvo utilizado foi a placa de testes anteriormente descrita (secção 4.2), constituída por uma cadeia de dois dispositivos (um dos quais sem registo de identificação).

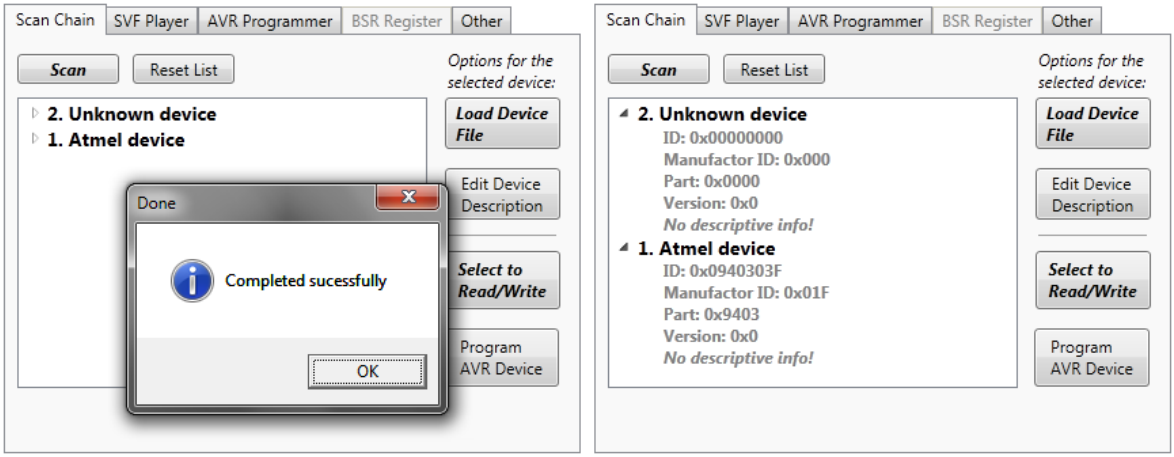


Figura 86 Pesquisa da cadeia de BS (esq.) e listagem dos detalhes dos dispositivos (dir.)

A operação de procura e listagem deverá ser quase instantânea para o utilizador.

Listam-se algumas informações fornecidas na consola para apoiar a descrição dos passos executados:

```
> Acessing target JTAG interface
(...)
FOUND: idcode=0x00000000, revision=0x0,
part=0x0000, manufactor=0x000
FOUND: idcode=0x0940303f, revision=0x0,
part=0x9403, manufactor=0x01f
(...)
> Searching JEDEC manufactor ID codes for: 000...
Device manufactor not found.
> Searching JEDEC manufactor ID codes for: 01F...
Found device manufactor: Atmel
```

Para procurar os dispositivos, a aplicação acede à biblioteca de controlo, recorrendo à função *controlo_svf* (com a opção "*SCAN*"), que pesquisa e retorna os vários dispositivos encontrados. A aplicação pesquisa então a identificação do fabricante de cada um dos

dispositivos no ficheiro "*manid.dat*", obtendo o nome de cada fabricante. O ficheiro "*manid.dat*" foi criado partindo da informação constante do *standard* JEDEC para identificação de fabricantes[79] (última actualização em Agosto de 2012), e tem a seguinte organização:

```
001,AMD
002,AMI
083,Fairchild
004,Fujitsu
085,GTE
(...)
3DD,Welink Solution Inc.
```

Os fabricantes estão distribuídos no *standard* JEDEC em oito grupos ("*banks*"), de acordo com o seu valor hexadecimal de identificação. Para uma pesquisa mais eficiente, a aplicação procura pelo nome do fabricante correspondente à identificação obtida apenas no seu grupo.

A informação dos vários dispositivos encontrados é registada numa lista hierárquica de itens seleccionáveis (classe *System.Window.Controls.TreeView* do *.NET Framework*), apresentada na anterior Figura 86 (à direita). Esta informação permanece a cinzento enquanto não for associado um ficheiro descritivo ao dispositivo.

5.3.8. GESTÃO DE FICHEIROS DESCRITIVOS

Os ficheiros BSDL são necessários para desbloquear o separador *BSR Register* (para edição manual do registo BSR de um dispositivo alvo) e para actuar sobre dispositivos específicos numa cadeia de BS (quer para operações BS como para programação de MCUs *AVR*). Qualquer uma destas funções necessita que sejam carregados os ficheiros BSDL de todos os dispositivos encontrados, mesmo que não se actue sobre eles, pois é necessário conhecer o comprimento dos seus registos para cálculo dos *headers* e *trailers* e aplicação de instruções de *BYPASS* quando necessário. Este facto está bem explícito na aplicação, que devolve mensagens de erro esclarecedoras se não for cumprido algum requisito. Não são necessários ficheiros BSDL para se efectuar operações de BS sobre toda a cadeia, através do separador "*Other*" (secção 5.3.10) ou executando ficheiros SVF; nem ao programar MCUs, contando que, se não for o único elemento da cadeia, serão necessárias informações adicionais.

O carregamento de ficheiros BSDL é feito seleccionando um dispositivo e carregando em "*Load Device File*", que abre uma janela de exploração do sistema para selecção do

respectivo ficheiro descritivo. O seu carregamento vem mostrar um conjunto ligeiramente diferente de características do dispositivo, como apresentado na Figura 87:

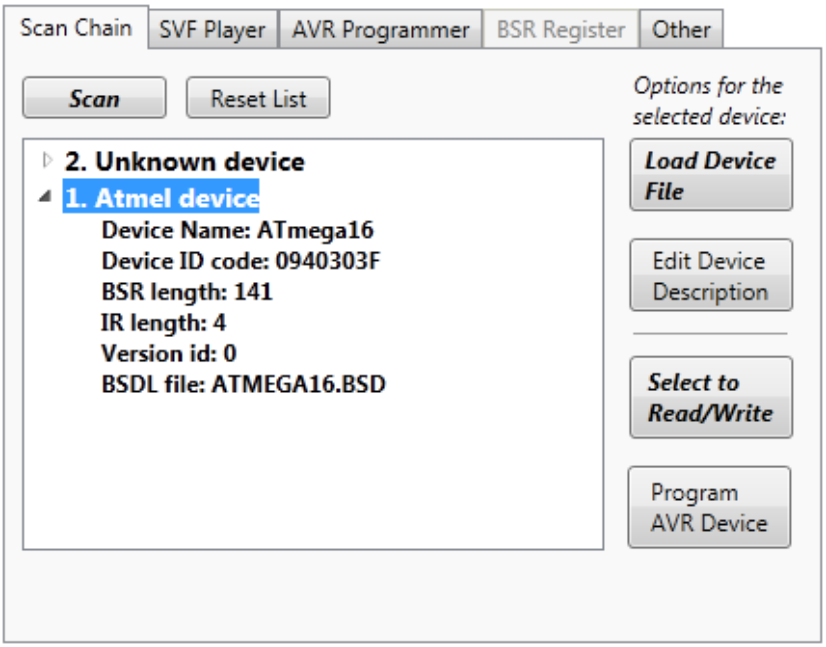


Figura 87 Ficheiro BSDL carregado para dispositivo *Atmel ATmega16*

O carregamento dos ficheiros BSDL para ambos os dispositivos da placa de teste retorna a seguinte informação na consola:

```
> Trying to load BSDL file...
File correctly matched the selected device
Descriptive file loaded
> Trying to load BSDL file...
Unable to verify if the file correctly match the
selected device. (No ID code?)
Descriptive file loaded
```

Como se verifica, a aplicação, antes de concluir o processo de importação do ficheiro descritivo, verifica se o código de identificação do fabricante e do componente (o código de identificação relativo à versão do componente é ignorada) corresponde ao dispositivo seleccionado. No segundo componente do caso apresentado, esta verificação é ignorada porque o dispositivo seleccionado não contém registo de identificação. Outra forma de ultrapassar esta verificação, se por algum motivo se pretender importar um ficheiro BSDL que não corresponda ao dispositivo alvo, é através da desactivação da opção "*Check if BSDL file match the device*", do menu "*Advanced Configurations*", nas opções sob a legenda "*Device description options*", como se mostra na Figura 88:

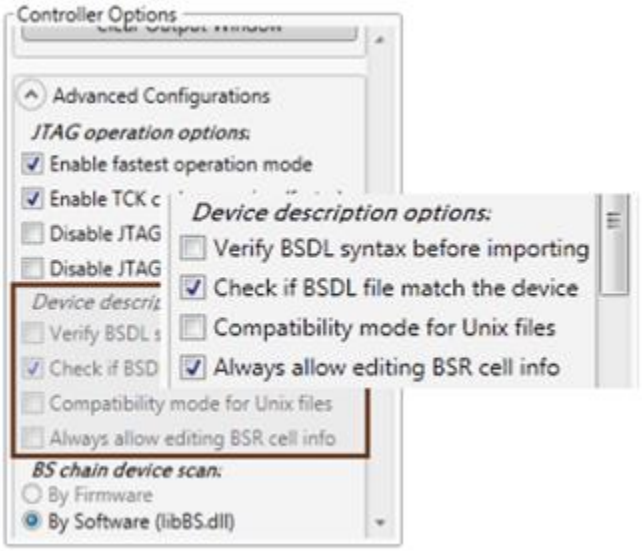


Figura 88 Secção "*Device description options*" do menu "*Advance Configurations*"

A primeira opção do referido conjunto activa a verificação da sintaxe dos ficheiros BSDL no decorrer da sua importação. Esta verificação recorre ao já referido *software* "*Goepel BSDL Syntax Checker*".

Esta ferramenta dispõe de uma aplicação gráfica, não utilizada, e de uma aplicação em linha de comandos, que devolve o seu *output* em dois ficheiros que são criados no directório da aplicação gráfica: "*BsdChGoe.log*" e "*BsdErrGo.log*". O primeiro regista a execução do programa e o seu resultado, enquanto o segundo guarda apenas os erros encontrados no ficheiro BSDL. Quando a opção de verificação da sintaxe está activa, os ficheiros BSDL importados são fornecidos à aplicação referida, cujos dois ficheiros devolvidos são depois lidos pela aplicação gráfica e as suas informações pertinentes listadas na consola (erros, "*warnings",* e resultado). Após carregamento de um ficheiro BSDL problemático, a ocorrência é notificada através de uma janela de erro e de uma mensagem na consola relativa aos erros encontrados:

```
> Checking BSDL file syntax... (Using GOEPEL
SyntaxChecker)
Compile Temp_data\ATMEGA16_c_erros.BSD: Fail
"ATMEGA16_c_erros.BSD" (113:5) : error : E :
expected: colon, read: 'DO' (identifier); Device
package pin mappings.
(...)
3 error(s), 0 warning(s).
```

Esta verificação pode também ser conduzida num ficheiro BSDL sem se necessitar de o importar e o associar a qualquer dispositivo na aplicação, utilizando a opção do menu superior "*Tools*", indicada na Figura 89:

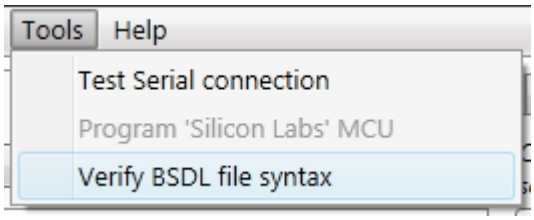


Figura 89 Menu "*Tools*"

A possibilidade de verificar os erros dos ficheiros BSDL é importante porque, dependendo do erro que possa existir, a sua importação poderá fazer o programa da biblioteca de controlo encravar, como se havia feito referência (secção 5.2.9). Este problema ocorreu com alguns ficheiros BSDL disponíveis *on-line*, que continham um pequeno erro na sua sintaxe, suficiente para desrespeitar o algoritmo da *libBS.dll*, mas nem todos os erros surtem este efeito.

A terceira opção, do grupo de opções sobre os ficheiros descritivos (Figura 88), diz respeito à activação de um modo de compatibilidade para ficheiros *Unix*. Esta foi implementada porque alguns ficheiros BSDL encontrados, que encravavam o funcionamento da *libBS.dll*, tinham sido criados em SO *Linux*, pois as suas linhas tinham uma terminação diferente ('\r' é utilizado em *Linux* e "\n\r" em *Windows*). Este modo converte as terminações das linhas do ficheiro para as utilizadas no *Windows* antes da importação do ficheiro. A última opção − "*Always allow editing BSR cell info*" – visa permitir que os dados dos ficheiros BSDL sejam alterados, processo que será referido mais tarde (secção 5.3.9).

Além da utilização de ficheiros BSDL para a obtenção das informações necessárias sobre os dispositivos, dois outros métodos são possíveis: Introdução manual pelo utilizador ou carregamento de ficheiros descritivos de extensão "*.dev*" criados pela própria aplicação gráfica.

Para introduzir manualmente a informação necessária pela aplicação (ou editar a informação já importada, quer através de um ficheiro "*.bsdl*" como "*.dev*"), utiliza-se o botão "*Edit device description*", que abre uma nova janela (*InputForm*), exemplificada na Figura 90:

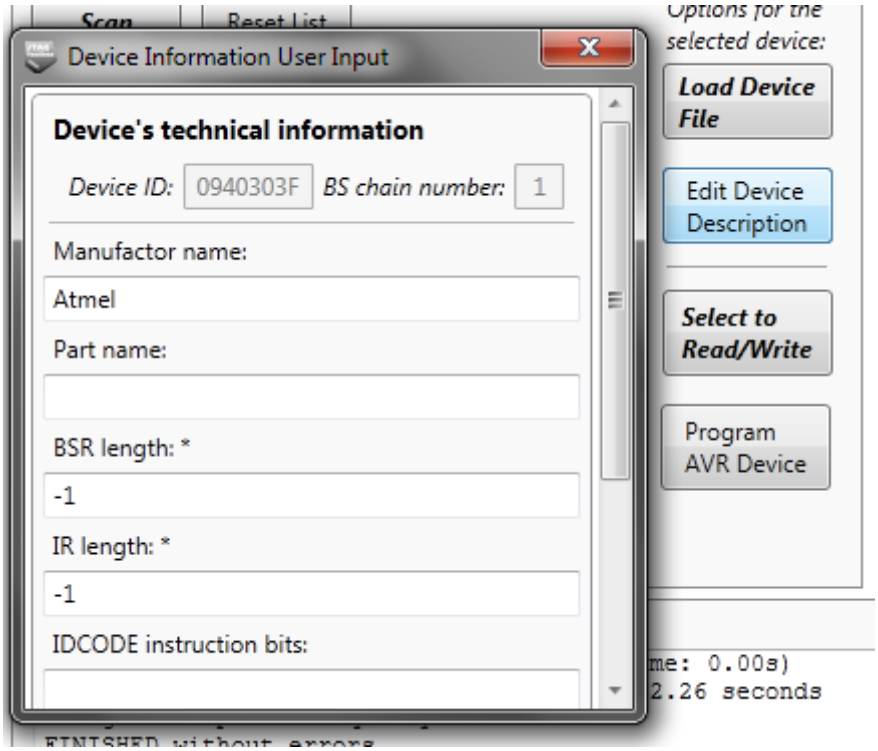


Figura 90 Janela de introdução/alteração manual dos dados do dispositivo

No caso apresentado, não havia sido carregado qualquer ficheiro para o dispositivo seleccionado, pelo que as caixas de texto surgem vazias ou com o valor '*-1*'. Os campos disponíveis não cobrem toda a informação que seria carregada do ficheiro BSDL, apenas a essencial:

- Nome do fabricante, já preenchido de acordo com o código de identificação;
- Nome do componente;
- Dimensão do registo BSR e IR (necessário para cálculo do tamanho dos vectores a enviar para cada registo e cálculo de *headers* e *trailers*);
- Codificação das instruções dos modos *SAMPLE* (ou *SAMPLE/PRELOAD* em componentes que não utilizem o *IEEE 1149.1-2001* ou posterior)*; PRELOAD; EXTEST* e *IDCODE* (necessário para introdução automática de instruções).

Se a codificação das instruções *SAMPLE* e *PRELOAD* forem iguais, a aplicação adoptará, nos menus acessíveis ao utilizador, a instrução aglutinada *SAMPLE/PRELOAD*, que torna mais claro o facto de ambas se tratarem, nesse caso, de uma única instrução. No fundo do formulário de introdução de informação, apresentado na seguinte Figura 91, oferece-se a opção de apagar as descrições dos *bits* BSR, e das instruções opcionais, que possam existir

para o dispositivo seleccionado. Esta opção só faz sentido quando se altera a informação relativa a um dispositivo sobre o qual já foi carregado um ficheiro descritivo. Também é possível editar manualmente a descrição dos *bits* do BSR através do separador "*BSR Register*".

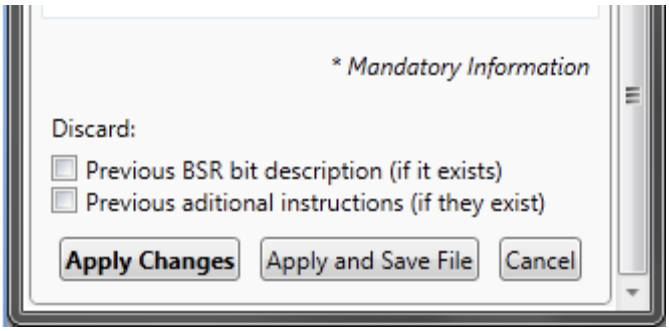


Figura 91 Opções da janela de introdução/alteração manual dos dados do dispositivo

As alterações efectuadas podem ser apenas aplicadas ou também guardadas num ficheiro "*.dev*" para posterior carregamento, evitando-se proceder novamente às mesmas alterações.

A introdução da informação necessária, através de qualquer um dos métodos referidos, em todos os dispositivo da cadeia alvo, vem possibilitar a utilização dos botões "*Select to read/write*" e "*Program AVR device*" num dispositivo previamente seleccionado.

5.3.9. EDIÇÃO DO REGISTO DE BOUNDARY SCAN (SEPARADOR "*BSR REGISTER*")

A opção "*Select to read/write*", tendo-se previamente seleccionado um dispositivo, faz a aplicação saltar para o separador "*BSR Register*", onde cria uma tabela (classe *System.Windows.Forms.DataGridView*) que descreve cada um dos *bits* do seu registo BSR, associada apenas ao dispositivo seleccionado, como convenientemente indicado na zona superior da janela.

A seguinte Figura 92 mostra este separador no estado em que se encontra após o componente *ATmega16* ser seleccionado:

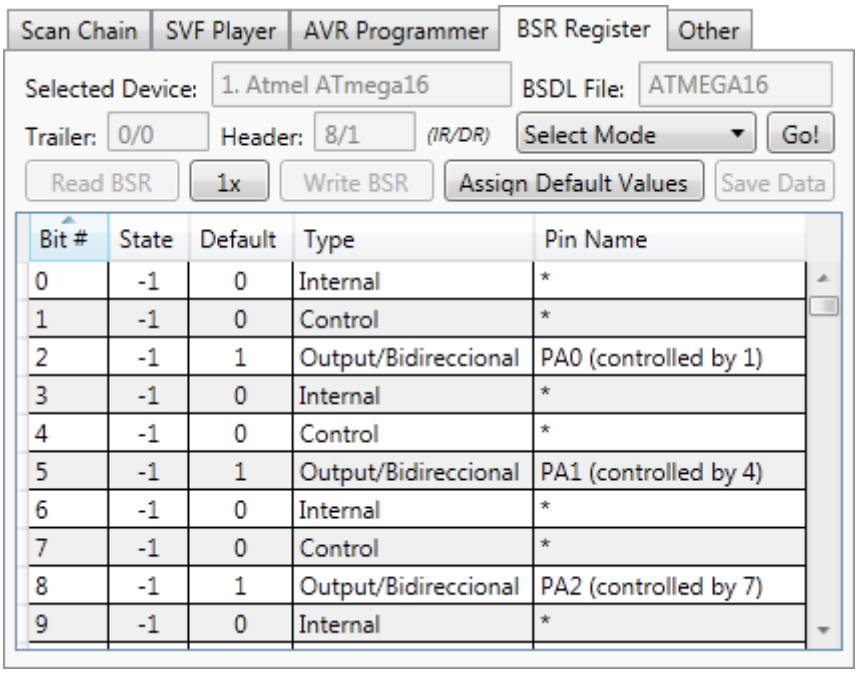


Figura 92 Separador "*BSR Register*"

Este separador afectará apenas o dispositivo previamente seleccionado até que seja, novamente no separador "*Scan Chain*", seleccionado um outro dispositivo e accionada a mesma opção para a sua escrita/leitura.

No cabeçalho superior da janela surgem diversas informações:

- Posição do dispositivo na cadeia BS;
- Nome do dispositivo;
- Ficheiro descritivo carregado (ou indicação de introdução manual dos dados);
- *Trailer* e *trailer* calculados, apresentados no seguinte formato:

  <quantidade_total_de_células_IR>/<quantidade_de_dispositivos>

Na tabela criada, cada linha é referente a uma das células do registo BSR do dispositivo. A disposição das colunas é reordenável pelo utilizador. A ordem das linhas pode ser definida de acordo com qualquer uma das colunas (ordenadas, no exemplo da anterior Figura 92, por ordem crescente da numeração dos *bits*). Descrevem-se as suas várias colunas:

- "*Bit #*":

  Indica célula do registo BSR. A célula número '*0*' é a menos significativa, cujo valor é o primeiro a entrar em *TDI* e o primeiro a sair em *TDO*. Não é possível editar este campo.

- "*State*":

  Valor lógico que é lido, ou que se pretende escrito, na célula BSR respectiva. Este campo é formatado condicionalmente, de forma a destacar, com preenchimento verde, as células BSR com o valor lógico '*1*'. O seu valor é sempre editável pelo utilizador.

- "*Default*":

  Valor lógico padrão, predefinido como seguro segundo o ficheiro descritivo do componente, para a célula em questão. Campo de edição condicionada.

- "*Type*":

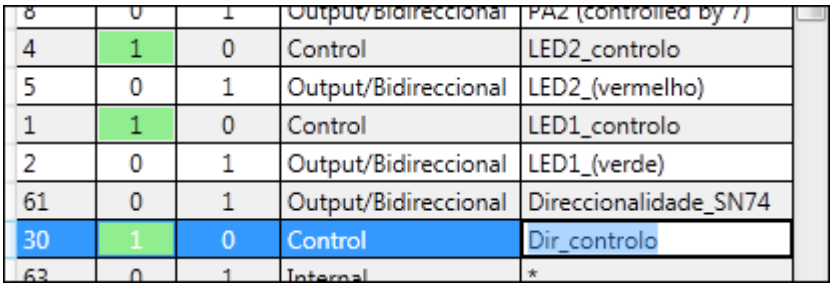
  Indica o tipo de célula BSR, de acordo com a informação do ficheiro descritivo. Campo de edição condicionada.

- "*Pin name*":

  Descrição da célula BSR, cujo conteúdo corresponde normalmente ao nome do pino físico que lhe está directamente associado. Tratando-se de uma célula bidireccional, é adicionalmente incluída, entre parêntesis, a indicação do número da célula que comanda a sua direccionalidade. Campo de edição condicionada.

Todos estes campos, à excepção do número do *bit*, são editáveis pelo utilizador. A opção "*Always allow editing BSR cell info*" em "*Advanced Configurations/Device description options*" (Figura 88, pág. 179), selecionada por omissão, deve estar activa para se editar informação importada de ficheiros BSDL.

Assim, é possível complementar as descrições obtidas dos ficheiros BSDL (muito limitadas por só incidirem sobre as células que comunicam directamente com um pino físico), para, por exemplo, facilitar o controlo das operações de escrita/leitura no registo BSR numa determinada montagem, como se exemplifica na Figura 93:



8	0	1	Output/Bidireccional	PA2 (controlled by 7)
4	1	0	Control	LED2_controlo
5	0	1	Output/Bidireccional	LED2_(vermelho)
1	1	0	Control	LED1_controlo
2	0	1	Output/Bidireccional	LED1_(verde)
61	0	1	Output/Bidireccional	Direccionalidade_SN74
30	1	0	Control	Dir_controlo
63	0	1	Internal	*

Figura 93 Exemplo da customização das células BSR (*ATmega16* na placa de teste)

A informação acrescentada ou alterada pelo utilizador pode ser gravada, num ficheiro "*.dev*" para carregamento futuro, no botão "*Save Data*" (ainda inactivo na anterior Figura 92). O ficheiro guardado pode depois ser carregado da mesma forma que um ficheiro BSDL.

Quanto ao processo de escrita e leitura do registo BSR, este requer alguns cuidados. Considerando um componente recém seleccionado, os botões de leitura e escrita apenas são activos após ser definido algum modo para o controlador TAP alvo – *SAMPLE*, *PRELOAD*, ou *EXTEST* − que deve ser escolhido na pequena lista "*Select Mode*" e accionado no botão "*Go!*". Os elementos citados, juntamente com as restantes opções disponíveis, são apresentados na seguinte Figura 94:

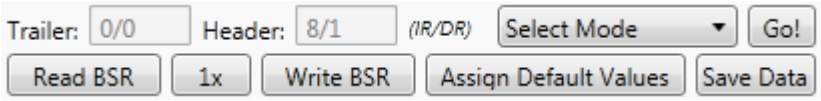


Figura 94 Controlos das operações no registo BSR

O valor lógico de cada *bit* ("*State*") surgirá na tabela ainda com o valor '*-1*', após selecção do modo de escrita/leitura, por não ter sido efectuada qualquer leitura nem introduzido quaisquer valores para escrita. Utilizando o botão "*Assign Default Values*", são automaticamente atribuídos os valores seguros. É de ter atenção que esta função apenas transcreve os valores seguros para a coluna referente aos estados, não os aplicando no dispositivo. Após este passo, já podem ser alterados os *bits* pretendidos e escritos no dispositivo utilizando o botão "*Write BSR*" já activo. Em alternativa, é possível ler os valores de todos os *bits*, com o botão "*Read BSR*", o que fará a coluna "*State*" passar a listar os valores reais de cada *bit*, não sendo necessário atribuir-lhes os seus valores seguros.

A utilização desta ferramenta requer algumas bases sobre o funcionamento do *IEEE 1149.1*. Dependendo do modo utilizado, o resultado da leitura ou escrita no registo será

diferente. Por exemplo, no modo *PRELOAD* as escritas no BSR destinam-se ao pré carregamento dos valores pretendidos, e não surte qualquer efeito fisicamente verificável no *hardware* da placa alvo. Para que tal aconteça, deve ser (preferencialmente depois dos valores pretendidos terem sido carregados no modo *PRELOAD*) alterado o modo do controlador TAP alvo para *EXTEST*, onde os valores introduzidos se tornarão efectivos.

Quando são executadas leituras no modo *EXTEST* (que se recomendam que sejam efectuadas no modo *SAMPLE*, onde não existe este problema), é necessário, devido ao funcionamento do BS, deslocar novo vector de *bits* para o registo BSR, à medida que se recebe os *bits* nele guardados. Este novo vector comandará os pinos do *hardware* (se em modo *EXTEST*), embora a operação tenha sido apenas de leitura, o que poderá confundir algum utilizador desconhecedor. Os *bits* carregados nas operações de leitura correspondem aos *bits* assumidos como seguros, na informação disponível sobre o componente, ou aos *bits* que possam ter sido definidos pelo utilizador na coluna "*State*".

Ambas as operações de leitura e escrita deverão ser concluídas rapidamente, dependendo do tamanho dos registos dos componentes, pois não serão procedimentos de BS complexos. As operações de leitura podem ser efectuadas singularmente, lendo o registo BSR uma única vez, ou de forma contínua, onde o programa permite monitorizar todo o registo, servindo de ferramenta de depuração de funcionamentos simples. O utilizador pode optar entre ambos os modos de leitura recorrendo ao botão "*1x*/*Cont.*" disponibilizado do lado direito do botão de leitura (Figura 94). Quando neste botão se lê "*1x*", a aplicação apenas lerá uma vez o registo BSR, actualizando consequentemente a informação apresentada na tabela. Actuando neste botão, passará a ler-se "*Cont.*" e estará seleccionado o modo contínuo. Uma vez iniciada a leitura no modo contínuo, a tabela será continuamente actualizada, devendo-se voltar a premir o botão do modo de leitura para que esta seja terminada. Esta informação é convenientemente descrita no *output* consola no decorrer da monitorização:

```
> Continuously reading the BSR register.
  Press "Cont." button to stop.
  Don't perform any other JTAG operation before
ending this operation.
```

Para além disso, é também indicado, na barra de estado da aplicação, o decorrer da operação e a frequência à qual a tabela do registo BSR está a ser actualizada. A velocidade de actualização não é constante, e depende essencialmente do comprimento do registo BSR do dispositivo alvo e do comprimento da cadeia BS alvo.

Nas operações de uma leitura única, para além da actualização da tabela do registo BSR, é devolvido pela consola o valor do registo de acordo com a formatação determinada no menu "*Output Options*" (Figura 78, pág. 167). Por omissão, os dados são devolvidos em hexadecimal:

```
Return Value:
0x0808022011adb6db6c09b6db6d2002ca4922bd
```

No formato binário, ao valor lido junta-se também uma indicação de quantos *bits* foram devolvidos, *i.e.* o comprimento do registo BSR.

Na prática, através deste separador é possível, por exemplo, controlar o *output* dos pinos de qualquer componente BS de uma placa alvo sem a intervenção da sua lógica interna. É também facilmente recolhido o valor de cada um desses pinos, em qualquer momento do funcionamento normal do componente (se no modo *SAMPLE*, que não interfere no funcionamento dos componentes), permitindo a depuração de eventuais problemas na placa alvo sem recorrer a um multímetro ou outro equipamento externo. Tendo-se preenchido a descrição das células com a sua função na placa de teste, optado por uma organização que agrupe os pinos de interesse para a monitorização, e activando-se o modo de leitura contínua, considera-se bastante intuitivo seguir as mudanças dos estados de cada um dos pinos da placa (realçados a cor verde quando activos) e a sua influência no resto do sistema. Mais detalhes relativos às possibilidades que o BS confere, ou mais especificamente o seu registo BSR, foram já abordados anteriormente (2.5 e seguintes).

5.3.10. CONTROLO JTAG MANUAL ASSISTIDO (SEPARADOR "*OTHER*")

A selecção do dispositivo alvo (no separador "*Scan Chain*", através da opção "*Select to Read/Write*") não serve apenas para editar o seu registo BSR. Este fica também seleccionado para outros controlos disponibilizados no separador "*Other*", apresentado na Figura 95:

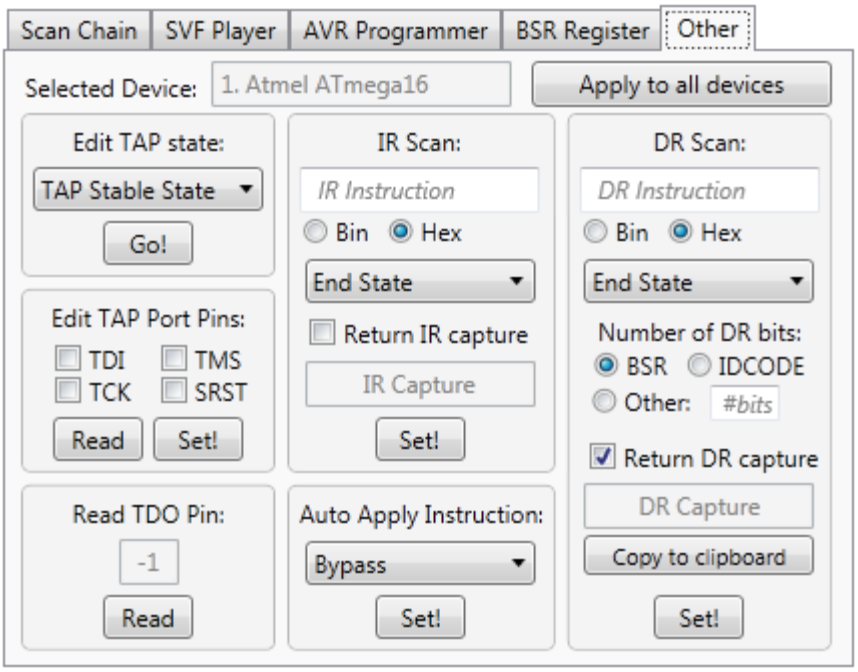


Figura 95 Separador "*Other*" aplicado a um dispositivo alvo

Este separador está sempre activo, mesmo sem ter sido seleccionado qualquer dispositivo ou sequer listados os componentes da placa alvo. No entanto, nessas situações, os controlos disponibilizados destinam-se a toda a cadeia BS alvo, não se diferenciando entre os vários componentes. A indicação do dispositivo seleccionado surge novamente no topo da janela, acompanhada do botão "*Apply to all devices*", que remove a selecção actual para que as operações efectuadas nos registos IR e TDR sejam aplicadas a todos os dispositivos que possam existir, e não apenas a esse único.

Ao ser desseleccionado o dispositivo alvo, ou se nunca for seleccionado qualquer dispositivo, o separador "*Other*" aparece com um aspecto ligeiramente diferente: É desactivada a funcionalidade de introdução automática de instruções, já que esta não abrange múltiplos dispositivos, e a aplicação deixa de conhecer o tamanho dos vectores a enviar (pois já não respeitam as dimensões dos registos IR e TDR guardadas), pelo que surge novo campo para a sua introdução em "*IR Scan*" e são desactivados os comprimentos "*BSR*" e "*IDCODE*" em "*DR Scan*", como na seguinte Figura 96:

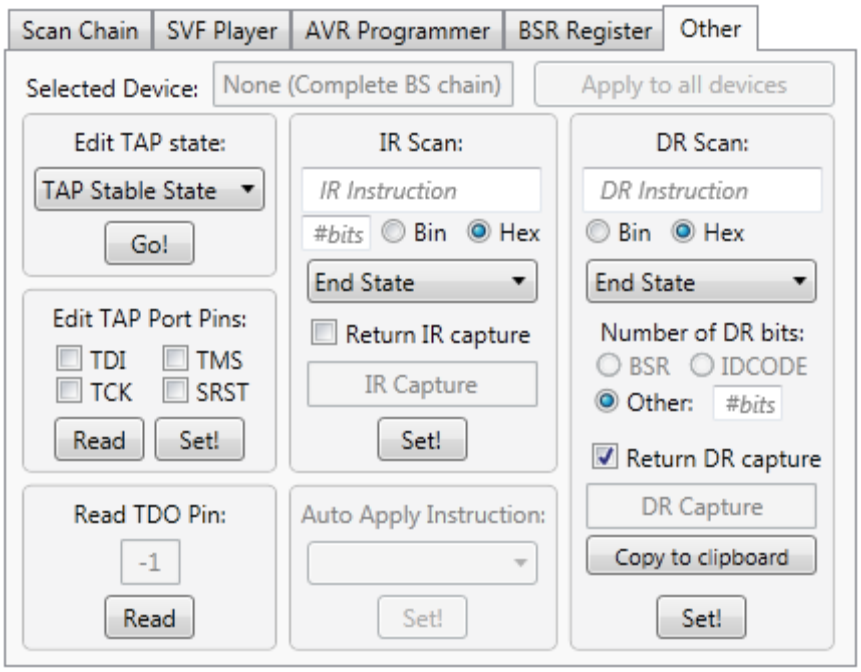


Figura 96 Separador "*Other*" aplicado a toda a cadeia, na aplicação gráfica

A primeira coluna de operações ("*Edit TAP state*"; "*Edit TAP port pins*"; e "*Read TDO pin*") afecta sempre todos os componentes da cadeia BS, independentemente da selecção de um dispositivo, o que não poderia ser de outro modo tendo em conta que as linhas de controlo (*TCK* e *TMS*) têm de ser partilhadas paralelamente por todos os controladores TAP da placa alvo. O objectivo destas três ferramentas é o controlo directo do conector JTAG, que devem ser suficientemente claras. Como a aplicação não toma conhecimento das operações a serem executadas pelo utilizador através deste controlo directo do conector JTAG, a sua utilização pode inviabilizar a selecção actual do dispositivo alvo (*e.g.* removendo as instruções de *BYPASS*).

A caixa que permite a aplicação automática de instruções num componente alvo (na parte inferior da coluna central) possibilita a escolha entre as instruções cuja codificação foi encontrada no carregamento do ficheiro descritivo do componente. Ao ser aplicada alguma instrução num dispositivo seleccionado, todos os outros são colocados em modo de *BYPASS* (para aplicar instruções diferentes nos diversos elementos da placa de teste, não deve ser seleccionado qualquer componente alvo, e utiliza-se o campo "*IR Scan*").

As opções de "*IR Scan*" e "*DR Scan*" permitem, respectivamente, a introdução de vectores nos registos IR e TDR do componente alvo, ou de todos os componentes da cadeia BS, de acordo com a selecção prévia. Estas aceitam valores hexadecimais (no seu formato comum, denominado de "*little endian*" segundo os tipos de formatação definidos na

aplicação) ou binários. Os valores recolhidos dos respectivos registos respeitam a formatação definida nas opções de *output* e são retornados ao ser activada a devida opção. Em ambas estas ferramentas existe uma caixa destinada à apresentação deste valor, para além de ser retornado na consola. O valor lido pode ser prontamente copiado para a área de transferência através do botão "*Copy to clipboard*".

Estas funções visam permitir a livre utilização das possibilidades que o *IEEE 1149.1* oferece.

5.3.11. CONSIDERAÇÕES SOBRE A PROGRAMAÇÃO

No descrito nesta secção, poucas vezes se fez referência quanto à forma de programação das funcionalidades destacadas. Evitou-se tal abordagem para simplificação da exposição, já que rapidamente se sobrecarregaria esta secção de descrições de algoritmos e excertos de código. Pretende-se, ainda assim, deixar uma breve indicação sobre a base de funcionamento da aplicação, referindo-se apenas algumas informações que possam ser úteis para uma futura evolução das funcionalidades oferecidas pela aplicação ou, especialmente, para a programação de uma nova aplicação que recorra também à linguagem *C#*, sabendo-se que estão muito aquém de explicar integralmente o funcionamento de qualquer uma das funcionalidades já implementadas, algo que não é pretendido.

A listagem de dispositivos alvo, obtida através da operação de "*SCAN*", está armazenada na variável *scan_list*, em qualquer momento da execução do programa:

```
public devices[] scan_list;
```

Esta variável é um vector de dispositivos encontrados, cuja informação de cada um é armazenado na estrutura "*devices*":

```
public struct devices
{
   public string idcode_value;
   public string manid;
   public string part;
   public string version;
   public string man_name;
   public string bsdl_file;
   public string part_name;
   public int BSR_length;
   public int IR_length;
   public string idcode; //instructions stored as
string to preserve the amount of binary digits
```

```
  public string extest;
    (...) Restantes instruções obrigatórias
  public int num_opt_inst; //Amount of optional
instructions (optional_inst)
  public instructions[] optional_inst;
  public BSR[] BSR_register;
  public int trailer_IR;
  public int trailer_DR_bypass;
  public int header_IR;
  public int header_DR_bypass;
}
```

As instruções IR opcionais são guardadas no campo "*optional_inst*", em estruturas do tipo *instructions*, que contêm apenas dois campos: o seu nome (*"name"*) e codificação ("*code*").

As informações guardadas na estrutura dos componentes (variável "*scan_list*") são inicialmente obtidas através da leitura do código de identificação de cada componente encontrado e depois completadas através do carregamento do seu ficheiro descritivo, ou por actualização do utilizador. A descrição dos componentes presentes na placa alvo fica disponível, após processo de "*SCAN*", no ficheiro "*devices.dat*", de onde se principia a definição dos dispositivos encontrados:

```
var lineCount =
File.ReadLines("devices.dat").Count();
scan_list = new devices[(int)lineCount];
(...) //colecta dos códigos de identificação
```

Para cada dispositivo encontrado é adicionado um item principal à listagem apresentada ao utilizador que indica o fabricante do dispositivo, obtido procurando o nome correspondente à sua codificação JEDEC, do seu código de identificação, no ficheiro "*manid.dat*" (através da função *find_id*).

Cada um destes dispositivos terá vários subitens não seleccionáveis, apenas indicadores de algumas informações que os distingam dos restantes. O seu preenchimento é hierárquico e segue o seguinte excerto:

```
Dispatcher.Invoke(new Action(() =>
{
  deviceItem = new TreeViewItem();
  deviceItem.Header = (i + 1).ToString() + ". " +
scan_list[i].man_name + " device";

//IDCODE INFORMATION:
  TreeViewItem subitem = new TreeViewItem();
  subitem.Header = "ID: 0x" +
scan_list[i].idcode_value;
```

```
   subitem.Focusable = false;
   (...) //Outras formatações
   deviceItem.Items.Add(subitem);

//MANUFACTOR INFORMATION:
   subitem = new TreeViewItem();
   (...) //Preenchimento dos restantes subitens
```

Esta informação é actualizada quando forem carregados os ficheiros descritivos. A descrição do registo BSR, obtida na leitura destes ficheiros, é guardada num vector, no campo "*BSR_register*", onde cada um dos seus elementos representa um dos *bits* do registo BSR, e segue a estrutura *BSR*:

```
public struct BSR
{
   public string type;
   public int safe_value;
   public string description;
   public int control_cell;
   public int dis; //unused
   public char rslt; //unused
}
```

Após ser seleccionado o dispositivo pretendido na aplicação, para a apresentação e leitura dos valores na tabela do separador do registo de BSR, é feita uma lista dos *bits* BSR do dispositivo para associação à tabela. Esta lista é independente da estrutura anterior. Cada elemento desta lista contempla a informação de cada uma das cinco colunas apresentadas na tabela, informação que deve poder ser lida e alterada, como definido na sua classe própria ("*bitBSR*"). Esta classe é necessária para efectuar o *binding* do conteúdo das variáveis com o seu campo na tabela apresentada, e implementa um método de notificação de eventos ("*INotifyPropertyChanged*") aplicado na variável do valor lógico da célula BSR ("*bvalue*"), que cria um evento de actualização na interface gráfica sempre que lhe for definido novo valor (campo "*set*"):

```
public class bitBSR : INotifyPropertyChanged
{
   public event PropertyChangedEventHandler
PropertyChanged;
   private int _bvalue;
   public int bit { get; set; }
   public int bvalue {
         get{ return _bvalue; }
         set{
               _bvalue = value;
               RaisePropertyChanged("bvalue");
         }
   }
   public int defaultvalue { get; set; }
   public string type { get; set; }
```

```
  public string description { get; set; }
  (...) //Função RaisePropertyChanged
}
List<bitBSR> data_bsr;
```

Foram realçadas as variáveis referentes ao conteúdo de cada campo de uma linha da tabela. Apenas "*bvalue*" é actualizada automaticamente na interface gráfica logo que ocorre qualquer mudança, devendo as restantes serem actualizadas manualmente. A lista do registo BSR do dispositivo alvo ("*data_bsr*"), é então preenchida adicionando-se as suas entradas linha a linha, de acordo com a informação guardada do dispositivo seleccionado:

```
data_bsr = new List<bitBSR>();
for (int j = scan_list[i].BSR_length - 1; j >= 0;
j--)
{
  var data_bit = new bitBSR
  {
  (...) //preenchimento dos campos BSR
  };
data_bsr.Add(data_bit);
}
```

É importante manter esta listagem porque após associação dos seus campos à tabela apresentada ao utilizador, este poderá ler e alterar directamente o valor em memória de cada um desses dados no programa. Quando o utilizador introduz qualquer valor, o conteúdo da lista é alterado sem necessidade de qualquer processo adicional de actualização. Para a associação dos campos da tabela às suas variáveis na programação é necessário definir o "*Binding*" de cada campo da tabela através da edição do código XAML da interface gráfica, tal como se exemplifica:

```
<DataGrid (...) ItemsSource="{Binding data_bsr,
UpdateSourceTrigger= PropertyChanged}" (...)>
  <DataGrid.Columns>
    <DataGridTextColumn x:Name="datagrid_pin"
    Header="Bit #" (...) Binding="{Binding
    Path=bit}"/>
    <DataGridTextColumn x:Name="datagrid_state"
    Header="State" (...) Binding="{Binding
    Path=bvalue,
    UpdateSourceTrigger=PropertyChanged}">
    (...) //restantes colunas
```

A qualquer momento em que se pretenda efectuar alguma operação que necessita de algum valor definido na tabela, não irá ser necessário proceder à sua leitura pois a lista, anteriormente definida como "*data_bsr*", tem a sua informação sempre actualizada. A actualização do estado lógico de cada célula do registo BSR na interface gráfica acontece de forma transparente, bastando preencher o seu novo valor na lista. Pretendendo-se

actualizar programaticamente a restante informação do utilizador, é apenas necessário a actualização dos itens da tabela, tal como em:

```
BSRGrid.Items.Refresh();
```

Tendo, da forma sugerida, guardada e organizada toda a informação respeitante a cada dispositivo da cadeia BS, qualquer tarefa BS que se pretenda executar carece apenas da construção das declarações SVF que lhe concedam resposta, como, por exemplo, para a introdução de uma instrução, recebida em código hexadecimal, num dispositivo específico, e aplicação de *BYPASS* aos restantes:

```
   BSR_svf.Write("TRST ABSENT;\n");
   BSR_svf.Write("ENDIR IDLE;\n");
   string tdi = (Convert.ToInt32(Math.Pow(2,
scan_list[i].header_IR) - 1)).ToString("X");
   BSR_svf.Write("HIR " +
scan_list[i].header_IR.ToString() + " TDI(" + tdi
+ ");\n");
   tdi = (Convert.ToInt32(Math.Pow(2,
scan_list[i].trailer_IR) - 1)).ToString("X");
   BSR_svf.Write("TIR " +
scan_list[i].trailer_IR.ToString() + " TDI(" +
tdi + ");\n");
   BSR_svf.Write("STATE RESET;\n");
   svf_instance = "SIR " +
scan_list[i].IR_length.ToString() + " TDI(";
   hex = Convert.ToInt32(hex, 2).ToString("X");
   svf_instance += hex + ");\n";
   BSR_svf.Write(svf_instance);
```

Que resulta num código SVF semelhante ao que se apresenta:

```
TRST ABSENT;        // TRST não implementado
ENDIR IDLE;         // Termina em Run-Test/Idle
HIR 8 TDI(FF);      // Header: FF = BYPASS
TIR 0 TDI(0);       // Trailer: não existe
STATE RESET;        // Opcional
SIR 4 TDI(2);       // IR: 2 = SAMPLE/PRELOAD
```

Recomenda-se, para funções da aplicação gráfica que escrevam código SVF, a utilização do estado "*Exit2*" do controlador TAP para efectuar um deslocamento faseado dos dados para a cadeia de BS alvo, especialmente tratando-se do acesso aos registos TDR. Esta medida evita que se possa gerar algum possível *overflow* de memória devido aos vectores demasiado extensos (o registo BSR de um único componente pode ter milhares de células e o sistema permite detectar e utilizar cadeias de BS com até *255* elementos). A biblioteca de controlo aloca dinamicamente o espaço em memória, pelo que não deverá ter problema a lidar com vectores mais compridos, mas a própria aplicação gráfica, para operações de tratamento de dados, como conversões entre binário e hexadecimal, beneficia deste

particionamento dos dados, que podem assim ser facilmente passados também para a biblioteca. A seguinte Figura 97 indica a evolução dos estados do controlador que se sugeriu para o deslocamento de dados (aplicável aos registos IR e TDR):

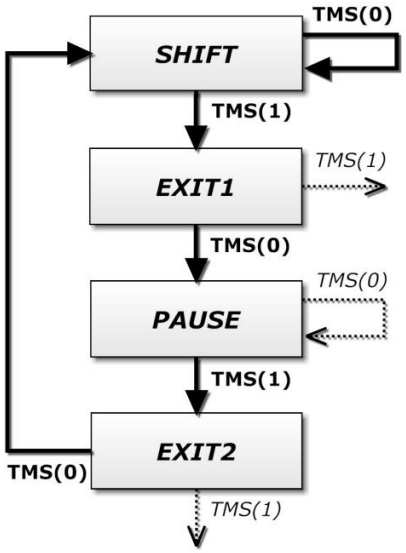


Figura 97 Parcela do diagrama de estados de um controlador TAP

O procedimento indicado repercute-se na escrita do código SVF da forma que se exemplifica de seguida. Serve de exemplo a escrita de *54 bits*, de valor '*1*', no registo TDR de um elemento que está colocado na cadeia de BS de forma a ter um *trailer* e *header* de *5* células (*e.g.* está na posição central de uma cadeia de *11* elementos onde todos os outros estão em modo de *Bypass*).

```
ENDDR IDLE;
HDR 5;
TDR 5;
SDR 54 TDI(3FFFFFFFFFFFFF);
```

```
ENDDR DRPAUSE;
HDR 5;
TDR 0;
SDR 16 TDI(FFFF);
HDR 0;
SDR 16 TDI(FFFF);
SDR 16 TDI(FFFF);
TDR 5;
SDR 6 TDI(3F);
STATE IDLE;
```

No excerto da esquerda são enviados os *54 bits*, excluindo os enviados no *header* e *trailer* (*HDR* e *TDR*), e terminada a execução no estado *Run-Test/Idle* (*ENDDR*). À direita surge o código com a mesma função mas enviando o mesmo vector em parcelas de *2 bytes*. De notar que os *headers*/*trailers* têm de ser definidos e anulados nos momentos oportunos, como realçado, pois não se aplicam enquanto se enviam as parcelas intermédias. Por fim, para os dados serem actualizados no controlador TAP, porque os varrimentos terminam agora no estado de *Pause-DR*, tem de se encaminhar o controlador para o estado *Run-*

Test/Idle (*STATE*), que implica a passagem no estado *Update-DR*. A aplicação gráfica escreve os códigos SVF que encaminha para a biblioteca de controlo já de acordo com este parcelamento de dados em conjuntos de *2 bytes*.

O código SVF deve depois ser encaminhado para a função "*controlo_svf*" da biblioteca de controlo, de acordo com a formatação de argumentos e considerações anteriormente referidas na secção 5.3.1.

Porque foram acrescentadas várias opções e variantes ao funcionamento de cada funcionalidade, e pela extensão considerável, compreende-se que possa ser complicada a tarefa de análise do código desenvolvido, pelo que se espera que esta pequena descrição possa ser uma ajuda se se pretender efectuar alguma alteração ao funcionamento da aplicação. Para tal, realçam-se algumas considerações importantes: as variáveis e estruturas onde são armazenadas as informações necessárias sobre cada componente; a forma como é possível associar valores em memória com campos da interface gráfica através do seu "*binding*"; a possibilidade de escrever dinamicamente código SVF que realize as operações necessárias; e por último, a formatação dos argumentos necessários para a utilização da biblioteca de controlo. Estes conceitos poderão ser reutilizados na programação de uma nova aplicação que recorra também à biblioteca de controlo.

6. TESTE E OPTIMIZAÇÃO

Descrevem-se de seguida os resultados da implementação das diversas funcionalidades, fazendo-se menção às limitações técnicas conhecidas, assim como aos problemas encontrados e que não puderam ser corrigidos até à data de conclusão deste projecto. Adicionalmente, juntam-se também considerações sobre as abordagens de optimização de processos conduzidas e uma apreciação qualitativa acerca do desempenho final do sistema.

6.1. FUNCIONALIDADE DO SISTEMA

O sistema final conseguiu efectuar da forma prevista, nos testes conduzidos, todas as funcionalidades que oferece, dentro do definido na descrição do *software*. No entanto, a execução de código XSVF foi testada de forma limitada, já que a linguagem não intervém no funcionamento da aplicação, e não é directamente compatível com a programação de MCUs *ATmega* por não suportar temporizações absolutas (*i.e.* expressas em unidades de tempo). Ainda assim, os testes efectuados, embora mais simples, envolvendo as funções XSVF mais comuns de varrimento de vectores de dados, permitem presumir o seu correcto funcionamento.

Destaca-se o correcto funcionamento das seguintes operações fundamentais:

- Imposição de valores lógicos em todos os pinos I/O do MCU *ATmega16* ou *SN74BCT8245A*, verificados fisicamente nos portos da placa alvo;

- Captura dos valores de entrada fisicamente forçados a ambos os componentes, sendo esta perceptivelmente quase instantânea após accionamento da opção de leitura na aplicação gráfica;

- Acesso às funcionalidades de cada um dos controladores TAP alvo em simultâneo, através da definição manual dos vectores de instruções e de dados a enviar (tendo em conta a ordenação dos dispositivos; respectivos comprimentos dos registos; codificação das instruções e organização dos vectores de dados);

- Programação de MCUs *AVR* e restantes opções (identificação de assinatura, escrita de *fuse*/*lock bytes*, *etc.*), verificadas auferindo-se o correcto funcionamento do MCU programado.

A aplicação gráfica do sistema desenvolvido foi apenas testada em SO *Windows 7 64 bits*. Como a aplicação foi programada e compilada com vista à sua utilização nesse SO; recorre à versão *4.5* do *.Net Framework* (as versões dos *softwares* utilizados encontram-se listadas na secção 3.5.3); e utiliza algumas funções do sistema; recomenda-se, no caso de se verificarem problemas a correr a aplicação em determinado SO, que esta seja recompilada utilizando-se o respectivo projecto.

Como a biblioteca de controlo é programada em código nativo, embora utilize algumas funções de sistema, não se crê que venha a apresentar problemas de compatibilidade com as várias versões do *Windows*. No entanto, porque o acesso a este tipo de código tem vindo a ser condicionado por razões de segurança, nas versões mais recentes do SO, não se pode garantir que o método de acesso à biblioteca, efectuado pela aplicação gráfica, não possa vir a ser incompatível com futuras versões do *Windows*.

6.1.1. COMPILAÇÃO DAS LIMITAÇÕES TÉCNICAS

As limitações técnicas relevantes para as funcionalidades oferecidas pelo corrente sistema, que foram sendo conhecidas e registadas ao longo do seu desenvolvimento, e os problemas encontrados nos procedimentos de teste ao protótipo funcional que ficaram por solucionar, são compilados na seguinte Tabela 22:

Tabela 22 Listagem das limitações significativas e problemas do sistema final

Limitação:	Programa envolvido:
• *Comunicação Serie:*	
Apenas são aceites valores de *baud rate* pré-definidos;	*LibBS.dll* (*rs232.c*)
Apenas são aceites portas Serie no intervalo *COM1 - COM16*.	*LibBS.dll* (*rs232.c*)
• *Interpretação de SVF/XSVF:*	
Comandos "*PIO*" (*Parallel Input/Output*) e "*PIOMAP*" (*Parallel Input/Output Map*) não suportados;	*LibBS.dll* (*svf.c*)
O parâmetro "*[MAXIMUM max_time SEC]*", no comando "*RUNTEST*", é ignorado;	*LibBS.dll* (*svf.c*)
As temporizações segundo *SCK* (*System Clock*), no comando "*RUNTEST*", são conduzidas em série com as operações JTAG (não em simultâneo como seria expectável).	*LibBS.dll* (*svf.c*)
• *Programação de MCU AVR:*	
Não é possível ler ou escrever os valores da calibração do oscilador.	*n.a.* (*AVRsvf.exe*)
• *Interpretação de BSDL:*	
Não se distingue totalmente os vários tipos de pinos (*e.g.* pinos de output de três estados e pinos direccionais);	*LibBS.dll* (*bsdl2jtag.c*)
Alguns erros de sintaxe no código BSDL dos ficheiros importados podem causar o congelamento da biblioteca de controlo.	*LibBS.dll* (*bsdl2jtag.c*)
• *Funções automáticas acessíveis através de terminal:*	
Algumas funções inadequadas, ou pouco convenientes, em cadeias de múltiplos dispositivos (*e.g.* "*Teste01*");	*Firmware* (*GccApplication1.c*)
Informação de ajuda ("*Help*") incompleta por limitações de memória.	*Firmware* (*GccApplication1.c*)
• *Problemas na utilização da interface gráfica:*	
Utilizar as opções de desactivar o controlo JTAG após a programação e execução de SVF pode impedir a posterior execução de operações manuais;	*Aplicação Gráfica*
A selecção de ficheiros ignora as suas extensões (*.svf*; *.hex*; *.xsvf* ; *.dev*; e *.bsd*/*.bsdl*);	*Aplicação Gráfica*
É possível que haja a acumulação, por tempo indeterminado, de ficheiros temporários na pasta "*Temp_Data*" da aplicação;	*Aplicação Gráfica*
Alguns erros raros, de origens diversas, podem levar à necessidade de reiniciar a aplicação.	*Aplicação Gráfica*

Realça-se que as limitações das funções implementadas no *firmware* apenas não foram resolvidas pois o seu funcionamento foi preterido pelo das funções implementadas na biblioteca de controlo (*e.g.* leitura dos registos de identificação) ou aplicação gráfica (*e.g.* introdução de instruções IR). Os problemas da aplicação gráfica podem ser resolvidos com o seu posterior desenvolvimento, não devendo ter uma resolução complicada. Já as limitações da biblioteca de controlo poderão ser bem mais complexas de colmatar.

6.2. DESEMPENHO DO SISTEMA

As comparações entre os resultados dos diversos testes efectuados e respectivas conclusões obtidas no conjunto de testes levados a cabo, parcialmente descritos nesta secção, vieram justificar várias alterações ao sistema desenvolvido, que foram já consideradas nas secções relativas ao seu desenvolvimento. Assim, as optimizações aqui propostas e aceites têm já a sua implementação contemplada nas respectivas descrições de *hardware/software* expostas anteriormente. Realça-se, ainda assim, a serventia da sua descrição por mostrarem os factores limitadores do desempenho do sistema, apresentando-se as formas encontradas para melhorar os resultados obtidos e concluindo-se quanto à sua significância. Deixa-se assim uma análise ao sistema que poderá contribuir para apreciação global do seu desempenho e, também, como ponto de partida para futuras melhorias.

6.2.1. ABORDAGENS TÉCNICAS PARA OPTIMIZAÇÃO

A única tarefa que importa realmente analisar em termos de desempenho é a de execução de código SVF, já que é a base de funcionamento de todo o sistema. Em cada teste foram geralmente efectuadas algumas alterações a nível de *hardware* e *software*, registando-se o resultado da execução de um mesmo ficheiro SVF antes e depois de cada modificação, assim auferindo-se a significância de cada optimização ou alternativa testada.

Os testes efectuados visam a identificação das áreas onde o sistema despende a maior parte do tempo de execução das tarefas, e implementação de modificações quando favoráveis. O resumo dos resultados obtidos encontra-se no Anexo D. Compreendem-se, de seguida, os procedimentos e conclusões retiradas das modificações testadas mais relevantes:

- Influência da frequência de processamento do controlador:

Foram registados os resultados obtidos após a execução de um mesmo ficheiro SVF para três frequências de relógio distintas: *1 MHz*; *8 MHz*; e *18,432 MHz*. A diferença entre as

duas primeiras implementações, onde se aumenta a frequência em *800%*, resultou numa diminuição do tempo não utilizado em transferência de *~28%*. O posterior aumento, menos significativo (*230%*), surtiu um efeito menos pronunciado, não atingindo *5%* de redução quando se esperariam, de acordo com o incremento anterior, valores a rondar os *8%*. Tendo o aumento da frequência sido pouco proporcional à redução do tempo das tarefas (excluindo transferências), estima-se que o factor mais significativo no dispêndio de tempo não será o processamento dos comandos no MCU. Utilizou-se a frequência de *18,432 MHz*.

- Influência do processamento do computador na interpretação de SVF:

Para se verificar se o processamento dos ficheiros SVF é um factor limitador na sua frequência de execução, foi temporariamente implementado um modo de simulação na biblioteca de controlo, que permitiu que esta funcionasse normalmente mas nunca sincronizando com o controlador (não esperando receber qualquer informação, pelo que não foi atrasada pelo MCU). Correndo o ficheiro SVF de teste, pôde auferir-se a frequência máxima que o processamento do computador utilizado permitia, de *370 KHz* (poderia ser superior aumentando-se a prioridade do processo). O que significa, de acordo com o tempo gasto na tarefa, que o processamento do computador toma menos de *1%* do tempo dispendido na execução dum ficheiro SVF (já descontando o tempo das transferências), pelo que, tal como esperado, a optimização do fluxo de execução da biblioteca de controlo não é um factor significativo.

- Influência da velocidade de transferência:

Foi executado o mesmo ficheiro SVF, que originava a transferência de *170 KB* entre o computador e controlador, para vários valores de *baud rate*: *4800*; *19200*; *76800*; *230400*; e *576000*. Os resultados, como esperado, mostram uma grande melhoria no desempenho com o aumento da velocidade de transmissão. No entanto, verificou-se que o percentual da redução do tempo de execução das operações não era constante, indiciando que o conversor Série-USB introduz um atraso na condução das transferências que se torna mais notório a velocidades de transmissão superiores. Este atraso também é notório ao verificar frequências diferentes utilizando conversores diferentes com a mesma *baud rate*.

- Diferença entre a velocidade de execução de ficheiros SVF cujos valores de *TDI* são constantes ou variáveis:

Ambos os ficheiros SVF utilizados (com e sem valores de *TDI* constantes) eram executados à mesma velocidade, embora na prática fossem evitáveis os comandos para re-escrita de *TDI* quando este se mantinha constante. Foi implementado que apenas sejam escritos os valores de *TDI* e *TMS* se o novo valor diferir do anterior, que passa a ser guardado na memória do computador, evitando o envio de informação desnecessária. Na execução dos ficheiros com um valor de *TDI* constante, a informação enviada passou de *270 kB* para *170 kB*, o que aumentou de forma aproximadamente proporcional a frequência de actualização de *TCK* (*~44 %*).

- Resultado da redução do fluxo de dados entre o controlador e computador através do modo assíncrono:

Inicialmente a actuação do controlador tinha sido implementada apenas com o seu funcionamento síncrono, onde devolvia o valor de *TDO* em todos os ciclos de *TCK*. Alterou-se essa situação fazendo-se com que apenas fosse devolvido o valor de *TDO* em situações em que este fosse preciso. As melhorias foram significativas: aumento de *~35%* na frequência de *TCK* de um ficheiro SVF sem pedidos de *TDO*. Seguindo esta iniciativa, implementou-se o comando para efectuar o ciclo de *TCK*, escrever as linhas de *TDI* e *TMS*, e requisitar a leitura de *TDO* se necessária, através do envio de apenas *2 bytes*, resultando em frequências de actualização de *TCK* mais de cinco vezes superiores às obtidas quando o funcionamento era síncrono.

- Resultado do agrupamento de comandos para efectuar ciclos de *TCK*:

Nas situações em que se efectuavam vários ciclos de *TCK* com o mesmo valor de *TDI* e *TMS*, tinham de ser enviado repetidamente o mesmo comando. Implementou-se um agrupamento de comandos iguais, até um conjunto de *8* pedidos, que são depois executados com um único comando (*2 bytes*), que incluí a informação de quantas vezes deverá ser repetido. O resultado desta implementação oscilou entre não suscitar qualquer diferença (em operações que necessitam de receber os valores de *TDO*), até melhorias muito significativos, mais do que duplicando a frequência de actualização de *TCK*.

- Comparação de diferentes conversores Série-USB:

O desenvolvimento e teste do sistema foi sempre baseado no conversor Série-USB *CP2102*, no entanto houve oportunidade de comparar o seu resultado com outras

alternativas, os componentes: *Prolific PL2303* e *FTDI FT232R*. O primeiro destes componentes não se mostrou adequado ao fluxo de informação do sistema, não sendo fiável a qualquer velocidade e levando o *software* a alertar para a recepção de informação corrompida no computador, algo que não ocorreu em nenhuma situação quando utilizando o conversor inicial. Estes erros não críticos surgiam ainda com *baud rates* tão baixas quanto *19200 bd*, sendo que com velocidades superiores a operação era interrompida por serem detectados valores de *TDO* não esperados, de acordo com o ficheiro SVF executado que foi escrito de forma a verificar os valores de *TDI* enviados. Já a alternativa do fabricante FTDI não comprometia em termos de fiabilidade, no entanto, o seu desempenho mostrou-se inferior à opção utilizada, garantindo frequências de funcionamento inferiores com a mesma *baud rate* do *CP2102* utilizado. A opção inicialmente escolhida é facilmente preferida.

Algumas destas optimizações, como o modo de funcionamento assíncrono ou o mecanismo de agrupamento de comandos, são de utilização opcional e podem ser desactivadas na aplicação gráfica.

6.2.2. RESULTADOS DO PROTÓTIPO FUNCIONAL

Os resultados obtidos, para apreciação do desempenho do controlador, consistem no conjunto de dados recolhidos pela biblioteca de controlo (referidos na secção 5.2.8) ao executar as várias funcionalidades implementadas. Destes valores, apenas se listam o tempo necessário para finalização das tarefas e a frequência média estimada de actualização de *TCK*. Os restantes dados recolhidos são também interessantes para se verificar a repercussão dos diferentes modos de funcionamento implementados, e sugerir a razão de alguns testes conseguirem frequências notoriamente distintas de outros. Estes dados foram utilizados para apoiar as optimizações previamente referidas, e estão registados no Anexo D.

O sistema, ao executar várias vezes a mesma tarefa, com as mesmas condições de entrada, pode mostrar alguma variação no período de tempo necessário para a sua conclusão. Os valores indicados espelham o resultado mais frequente após várias execuções da mesma tarefa (moda). Os testes foram conduzidos em cada um dos modos implementados para a actuação do controlador: Modo *Low-Speed* (síncrono); Modo normal (assíncrono); e Modo

normal com agrupamento parcial de ciclos de *TCK*. Na Tabela 23 mostram-se os resultados da execução de ficheiros SVF:

Tabela 23 Execução de ficheiro SVF [Tempo decorrido (frequência act. *TCK*)]

Ficheiro SVF com:	C/Agrupamento:	Normal:	Low-Speed:
Valores de TDI constantes:	*6,76s (14,12 KHz)*	*23,65s (4,03 KHz)*	*1m12s (1,33 KHz)*
Valores de TDI variáveis:[29]	*19,38s (4,92 KHz)*	*23,58s (4,05 KHz)*	*1m28s (1,08 KHz)*
Nota: Ficheiros de 2279 instruções, sem verificações de *TDO*.			

Os resultados apresentados mostram o intervalo de velocidade a que é expectável que a biblioteca de controlo consiga interpretar e executar o código SVF quando não forem lidos valores de *TDO*. *i.e.* no caso do modo mais célere, por exemplo, esperam-se velocidades aproximadamente entre os *5* e os *14 KHz*, no pior e melhor dos casos, respectivamente. Naturalmente, a frequência conseguida depende das instruções SVF executadas. Os ficheiros SVF utilizados continham maioritariamente instruções para escrita do registo TDR do componente alvo. Não foi utilizada nenhuma instrução que introduza qualquer temporização.

A programação do MCU *ATmega16* deveria gerar resultados similares aos anteriores, a nível da frequência de actualização de *TCK*, já que basicamente consistem também na execução de um ficheiro SVF. Este ficheiro será, no entanto, mais variado e representativo de um caso real. Na Tabela 24 são registados os resultados de apagar e programar a memória *FLASH* do MCU, e, adicionalmente, efectuar também a sua verificação, que acarreta a necessidade de ler valores de *TDO*. Para comparação, juntaram-se os dados relativos às mesmas operações quando efectuada pelo programador ISP dedicado *USBasp* (Anexo A) e *software Khazama AVR Programmer*.

Tabela 24 Programação MCU *ATmega16* [Tempo decorrido (frequência act. *TCK*)]

Tarefas:	C/Agrupamento:	Normal:	*Low-Speed*:	*USBasp*:
Apagar + Programar *FLASH*	*19,94s (8,72 KHz)*	*38,67s (4,50 KHz)*	*2m36s (1,26 KHz)*	*11s*
Apagar + Programar *FLASH* + Verificação *FLASH*	*40,48s (7,81 KHz)*	*1m16s (4,15 KHz)*	*4m34s (1,15 KHz)*	*19s*
Notas: Ficheiro HEX de 3,93 *Kb;* O MCU *ATmega16* é o único elemento da cadeia BS; Tempo de execução do módulo *AVRsvf* não contabilizado (pouco significativo: *<1s*).				

[29] Ficheiro SVF onde os vectores enviados obrigam à alteração do valor da linha de TDI a cada ciclo. *e.g.* {*010101…*}

Cada modo representa frequências bastante distintas. No modo mais optimizado, a velocidade de programação começa a aproximar-se da conseguida através do sistema ISP dedicado.

Os resultados da pesquisa de componentes nas cadeias de BS alvo dizem respeito à análise da placa de teste desenvolvida, que apenas dispõe de uma modesta cadeia de dois elementos. Estes listam-se na Tabela 25:

Tabela 25 Pesquisa da cadeia de BS [Tempo decorrido (frequência act. *TCK*)]

Cadeia BS:	C/Agrupamento:	Normal:	*Low-Speed:*
ATmega16 + SN75BCT8245A	*0,04s (3,68 KHz)*	*0,04s (3,49 KHz)*	*0,16s (925 Hz)*

Porque esta operação requer o recebimento dos valores de *TDO*, a activação do agrupamento de ciclos não traz qualquer benefício (a diferença verificada é fruto da referida variação de resultados). Crê-se que esta operação seja, de qualquer forma, suficientemente rápida mesmo em placas alvo com cadeias de BS bem mais preenchidas.

A edição do registo BSR, nas suas operações de leitura e escrita, toma os tempos registados na Tabela 26:

Tabela 26 Leitura/Escrita Registo BSR [Tempo decorrido (frequência act. *TCK*)]

Componente:	Normal c/ agrupamento de ciclos:		Normal		*Low-Speed*	
	Leitura:	Escrita:	Leitura:	Escrita:	Leitura:	Escrita:
ATmega16 (141 bits)	*0,09s (3,71 KHz)*	*0,03s (10,21 KHz)*	*0,09s (3,69 KHz)*	*0,07s (5,08 KHz)*	*0,51s (673 Hz)*	*0,36s (956 Hz)*
SN74BCT8245A (18 bits)	*0,02s (3,58 KHz)*	*<0,01s (10,23 KHz)*	*0,02s (3,40 KHz)*	*0,01s (4,94 KHz)*	*0,09s (668 Hz)*	*0,05s (1,08 KHz)*

O processo de leitura é mais lento que o de escrita por necessitar de receber valores de *TDO*, o que directamente aumenta o fluxo de dados; e, para além disso, por inibir a possibilidade de se agrupar ciclos de *TCK*.

Como a leitura do registo BSR completo toma *0,09* e *0,02* segundos (modo normal em ambos os componentes), pode supor-se que, no caso da sua leitura contínua, as frequências conseguidas serão:

$$F_{BSR\_ATMEGA} = \frac{1}{T} = \frac{1}{0{,}09\ s} \cong 11{,}11\ Hz$$

$$F_{BSR\_SN74BCT8245} = \frac{1}{0{,}02\ s} = \ 50\ Hz$$

Embora estes valores não estejam longe dos conseguidos, neste caso a gestão da interface gráfica na actualização dos valores apresentados já interfere no desempenho conseguido. Os intervalos nos quais oscilam as frequências de actualização do registo BSR completo estão indicados na seguinte Tabela 27:

Tabela 27 Leitura contínua do registo BSR [frequência leitura do BSR]

Componente:	C/Agrupamento:	Normal:	*Low-Speed:*
ATmega16 (141 bits)	*8 - 13 Hz*	*8 - 13 Hz*	*2 - 3 Hz*
SN74BCT8245A (18 bits)	*22 - 48 Hz*	*22 - 48 Hz*	*9 - 13 Hz*

Notou-se que os resultados obtidos variavam de acordo com a prioridade atribuída à linha de processamento (*thread*), do computador, que efectuava a actualização dos dados do registo, pelo que se estima que as frequências de leitura contínua possam estar dependentes do desempenho do computador. A funcionalidade de monitorização trata-se de uma função da própria aplicação gráfica, não da biblioteca de controlo, pelo que os valores recolhidos continuamente não se destinam a ser interpretados por qualquer outro *software*. Nesse sentido, embora os valores obtidos estejam inadequados a testes funcionais criteriosos, são satisfatórios pois os dados destinam-se apenas a serem acompanhados pelo utilizador.

6.2.3. CONSIDERAÇÕES SOBRE OS FACTORES LIMITADORES E POSTERIORES OPTIMIZAÇÕES

Os testes conduzidos para apreciação do desempenho do sistema, e, especialmente, todos os outros que interviram na tentativa de optimização dos processos, e se resumiram no anexo Anexo *D*, tentaram encontrar o *bottleneck* do sistema, ou pelo menos o factor mais limitador do desempenho. Como a aplicação gráfica não tem qualquer influência na frequência de funcionamento (os dados utilizados nos testes vêm directamente de *libBS.dll*), apenas é necessário considerar o *software* da biblioteca de controlo e *firmware*. É de realçar que propositadamente se utilizou como dado quantitativo a frequência média de actualização de *TCK*, fugindo ao conceito de frequência de *TCK*. Esta última, especialmente no modo mais célere com agrupamento de ciclos, poderá atingir picos comparativamente muito elevados (que se estimam entre os *2,3* a *3 MHz*, de acordo com as temporizações indicadas pelo simulador do *firmware*).

O tempo dispendido na execução de código SVF pode ser grosseiramente atribuído a apenas quatro componentes, que poderão decorrer em simultâneo:

- Processamento da biblioteca de controlo no computador;
- Transferência de informação entre o computador e o controlador;
- Processamento da interface Série-USB (atrasos introduzidos na ligação);
- Processamento do *firmware* do controlador (MCU).

Conclui-se que o tempo gasto no processamento do computador é pouco significativo, tendo-se verificado que suportaria frequências de actuação JTAG muito superiores.

O processamento do MCU, cuja influência também se mostrou pouco significativa, de acordo com o simulador do IDE de desenvolvimento, toma *11,40 μs* desde a recepção do comando até concluir a aplicação do ciclo de *TCK*. A frequência de *TCK* que permite será aproximadamente:

$$F_{TCK} = \frac{1}{11{,}40\ \mu s} \cdot 10^3 \cong 88\ KHz$$

Mas durante o tempo em que aplica o comando está também a receber dados consequentes (excepto ao tratar-se de um período de sincronização com o computador), o que é feito mais lentamente e torna este desempenho ainda menos significativo, já que terá depois de aguardar os novos dados.

Apenas o conversor Série-USB, e a velocidade de transmissão que atinge, podem ser os maiores responsáveis pela limitação da velocidade de execução, como se previa desde o início da implementação. No entanto, para além do tempo gasto em transferências, entende-se que este possa também introduzir outros atrasos temporais, como se havia referido, limitando a velocidade do sistema mais do que se previa. Considerando a velocidade de transferência de *56,25 kB/s*, determinada na secção 5.1.3, e o modo normal de funcionamento, que implica o envio de dados a serem directamente escritos no conector JTAG, em comandos de dois *bytes* por ciclo de *TCK*, tem-se:

$$Vel_{Transf\_UART} = 56{,}25\ kB/s \cdot 1024 = 57600\ B/s$$

$$F_{TCK} = \frac{57600\ B/s}{2\ B} = 28800\ ciclos_{TCK}/s = 28{,}8\ KHz$$

Teoricamente, a frequência de *TCK* no modo normal de funcionamento nunca poderia ser superior a *28,8 KHz*, valor ideal que ignora os atrasos do conversor e *driver Virtual COM Port* (VCP) na gestão das comunicações USB e UART, que se crêm com um impacto muito significativo. Pelo que se considera a velocidade de transferência como o *bottleneck* do sistema. Não se conseguindo atingir velocidades superiores de transmissão, a redução do *overhead* dos comandos enviados foi o foco principal das optimizações efectuadas.

Os benefícios das optimizações foram claros. Os primeiros resultados obtidos, ainda no decorrer do desenvolvimento, apresentavam médias de frequências de actualização de *TCK* que não superavam os *80 Hz*, o que provocava enormes tempos de espera na execução das tarefas, vindo assim a justificar as tentativas de optimização conduzidas. Actualmente, o protótipo funcional desenvolvido atinge os *14 KHz*.

Destacam-se três abordagens para posterior optimização do desempenho:

- Optimização dos dados transmitidos ao controlador;

  Continuação da abordagem conduzida de redução do fluxo de informação entre computador e controlador. Pode ser conseguido, por exemplo, através de um agrupamento mais completo dos valores de cada uma das linhas JTAG para cada ciclo, efectuando-se o seu envio conjunto para o controlador.

- Implementação mais abrangente de funcionalidades de nível superior no controlador;

  Implementar funções tais como as descritas na secção 5.1.8 (“Operações Automáticas no Controlador TAP Alvo”) no *firmware* do controlador, de uma forma eficaz, poderá constituir uma forma de acelerar muito substancialmente a velocidade de execução do sistema, já que evitaria utilizar as funções elementares de manipulação das linhas JTAG para conduzir tarefas mais complexas, mas ainda assim genéricas, como a leitura/escrita do registo TDR, restringindo-se as transferências de dados. Embora prevista, esta estratégia não foi convenientemente testada, tendo sido deixado para a biblioteca de controlo, recorrendo às ditas funções elementares do *firmware*, o cargo de

efectuar todas as tarefas de nível superior. Um caso flagrante é o facto de ter sido implementada, tanto no *firmware* como na biblioteca de controlo, a função de pesquisa da cadeia de BS alvo e retorno da identificação dos seus componentes, sendo que, no entanto, apenas a função da biblioteca de controlo é utilizada pela aplicação gráfica. Esta funcionalidade de pesquisa da placa alvo não é relevante no desempenho do sistema, mas outras funções, como para as referidas leituras/escritas dos registos, implementadas directamente no *firmware*, poderiam ser uma mais-valia.

- Melhoria do *hardware* do controlador

  Sendo necessário um aumento considerável na frequência de actuação, a substituição do *hardware* do controlador seria a solução responsável por uma melhoria mais radical. Mas esta implicaria as desvantagens já referidas no planeamento do projecto (*e.g.* custos superiores; menor acessibilidade; montagem mais dificultada; *etc.*) e a programação do seu suporte na biblioteca dinâmica. Pelo expectável bom desempenho, recomendaria-se a solução abordada na secção 3.2.6 (*FTDI Multi-Protocol Synchronous Serial Engine*).

Adicionalmente, seria possível optimizar especificamente o desempenho da monitorização do registo BSR, cujos resultados se listaram na Tabela 27, se fosse dada ao utilizador a possibilidade de escolher quais células pretende que sejam lidas continuamente, pelo que a biblioteca de controlo apenas teria de receber os valores de TDO das células indicadas, permitindo que as frequências de actualização de TCK se aproximassem mais das obtidas nas tarefas de escrita do registo BSR (mais de duas vezes superior). Esta optimização incidiria apenas sobre o programa da aplicação gráfica.

7. CONCLUSÕES

A aceitação actual da tecnologia de testabilidade integrada que a arquitectura de *Boundary Scan* garante, como ferramenta industrial para teste e depuração de circuitos impressos, é evidente e nunca foi posta em dúvida nas motivações para a pesquisa conduzida no interesse do projecto. O estudo efectuado pretende apenas justificá-la e dar a conhecer as vantagens da sua utilização em relação a outros métodos. Actualmente, o *Boundary Scan* continua a mostrar uma acentuada evolução em várias direcções, justificando a origem de novas normas e liderando a tendência de desenvolvimento da testabilidade e instrumentação integrada. A adaptação do *Boundary Scan* para áreas a que este, apenas mais recentemente, se vem alastrando, como a sua aplicação em circuitos de sinais alternados (*IEEE 1149.6*) ou outras normas referidas na secção 2.4 (“Sumário das Normas *Boundary Scan*”), pode ainda causar alguma incerteza quanto ao futuro sucesso da sua adopção, muito por causa do suporte limitado por parte dos fabricantes. No entanto, não existe qualquer dúvida quanto à mais-valia da implementação do já firmado *IEEE 1149.1*, cuja proeminência no teste industrial é alicerçada em fortes argumentos, há vários anos conhecidos: celeridade de aplicação; baixo custo; flexibilidade; entre os demais já vastamente referidos.

A descrição conduzida à norma *IEEE 1149.1* visou descrever o seu bem conhecido funcionamento, explicando as bases necessárias para uma aplicação prática generalizada.

Ambas as exposições, sobre o *Boundary Scan* como ferramenta industrial e a sua descrição técnica, serviram para o estabelecimento do conhecimento base para a construção do controlador de *Boundary Scan* proposto, e deverão ser ainda úteis para a sua correcta utilização e valorização.

Relativamente ao desenvolvimento do controlador de *Boundary Scan*, considerando os objectivos inicialmente estabelecidos, crê-se que seja evidente que estes tenham sido assegurados na sua totalidade pelas funcionalidades implementadas no controlador proposto, sob consideração subjectiva no que concerne à adequabilidade do controlador a um meio didáctico. Para além de uma utilização didáctica, vê-se potencial para o sistema, tal como se oferece, ser disponibilizado com vista a servir também como uma alternativa de baixo custo para a programação de MCUs *ATmega*, substituindo os programadores ISP, que têm uma utilização muito mais limitada; e como um *SVF player*, rivalizando favoravelmente com as opções disponíveis actualmente por facultar muitas outras funcionalidades adicionais ou comportar um custo inferior.

Uma apreciação técnica mais conclusiva quanto à aptidão do controlador é apresentada em seguida.

7.1. APRECIAÇÃO FINAL

O sistema proposto disponibiliza um conjunto considerável de ferramentas que visam a disponibilização das capacidades da norma *IEEE 1149.1*. A aplicação gráfica desenvolvida permite facilmente tirar partido da observabilidade e controlabilidade que a norma possibilita, tal como se objectivou inicialmente. Uma acrescida utilidade do sistema é garantida pela funcionalidade de executar código SVF, que o torna compatível com um substancial conjunto de pacotes de *software* de desenvolvimento, podendo ficar encarregue da execução das suas tarefas de *Boundary Scan*. Esta funcionalidade permite também o teste e depuração automáticos de PCBs, segundo uma especificação normalizada amplamente conhecida e de escrita simplificada; e a programação de componentes de qualquer fabricante, desde que forneçam ferramentas adequadas (*e.g. Atmel*; *Si-Labs*; *Xillinx*; entre outros). Ficheiros SVF que conduzam testes de interligações numa dada placa, efectuando aquela que é a maior aplicação industrial do *Boundary Scan*, podem ser criados com *software* especializado, que requerem apenas os ficheiros descritivos dos componentes e esquemático do circuito alvo, e executados pelo sistema desenvolvido. Por

outro lado, as ferramentas manuais também implementadas, destacando-se a de monitorização e escrita assistida dos registos BSR, permitem que os testes possam ser feitos de forma mais intuitiva e facilitada, mas menos criteriosa. Assim, concilia-se uma ferramenta mais eficaz e eficiente, com outras ferramentas de controlo manual que serão mais vocacionadas para a apresentação da tecnologia e introdução à utilização da norma *IEEE 1149.1*. Num caso prático elementar, acredita-se que mesmo utilizadores inexperientes na electrónica digital e *IEEE 1149.1* terão facilidade em, por exemplo, preterir a usual utilização de voltímetros para a verificação de sinais digitais nas suas placas, recorrendo ao *software* desenvolvido.

O problema técnico mais pertinente que se encontra, após consideração do desenvolvimento e teste do protótipo funcional, é o limitado controlo da frequência de actuação dos controladores TAP das placas alvo. Porque alguns componentes são, reconhecidamente, mais exigentes em relação à frequência com que são actuados, deve supor-se que alguns possam não tolerar os picos de frequência de actuação (por volta dos *2* a *3 MHz*) que se estima que o sistema possa atingir. Tal não se verificou no grupo de dispositivos alvo testado, mas este foi notoriamente modesto. Nesse sentido, foram mantidos modos de actuação diferentes, que proporcionam frequências instantâneas inferiores, tentando manter o controlador compatível com qualquer componente, algo que foi uma das principais preocupações em todo o desenvolvimento e se considera que tenha sido assegurada. Esta limitação no controlo da frequência de actuação não compromete assim qualquer funcionalidade ou aplicação do controlador, ainda que se entenda que a frequência instantânea máxima, utilizada na actuação de uma placa alvo, é um factor importante e, da forma como o controlador foi implementado, o seu conveniente controlo não é possível. De resto, outras limitações técnicas mais determinísticas, que incidem sobre aspectos particulares da implementação e foram descritas na secção 6.1.1 (“Compilação das Limitações Técnicas”), não exercem também qualquer restrição ao funcionamento planeado e compatibilidade do controlador. Quanto ao desempenho atingido, este crê-se muito satisfatório para uma aplicação em meio didáctico, como pretendido.

7.2. ABORDAGEM NA APLICAÇÃO DIDÁCTICA

Alguns projectos de controladores de BS para utilização didáctica implementam métodos de acesso remoto, por rede local ou *internet*, recorrendo, por exemplo, a vídeo capturado através de cameras *Web*, ou outras formas que permitam ao utilizador verificar as

mudanças que executa na placa alvo[80][81]. Tentou-se contrariar essa abordagem optando-se por um equipamento de complexidade muito mais reduzida, que pudesse evitar placas de circuito impresso, para que, em lugar dos futuros utilizadores terem de recorrer a métodos de acesso remoto, possam vir a possuir o seu próprio controlador e placa alvo, privilegiando-se uma aprendizagem autodidacta. Pensa-se que esta alternativa possa mostrar vantagens em relação às referidas por dar ao utilizador um maior controlo sobre o controlador, que o detém fisicamente, em detrimento de partilhar remotamente um dispositivo com outros utentes. Poderá também servir-se facilmente do controlador como apoio à depuração no desenvolvimento dos seus próprios projectos. O controlador projectado pode assim ter uma área de aplicação mais vasta do que uma unidade de controlo, fixa num laboratório, para acesso remoto, sem comprometer o seu objectivo inicial.

No entanto, entende-se que esta abordagem didáctica possa, considerando um largo número de utilizadores, ser mais dispendiosa, no valor do total de controladores, assim como trazer maior dificuldade para as actividades serem supervisionadas por algum docente, que teria mais facilidade em o fazer remotamente do que presencialmente, como pressupõe o sistema actual. A sua implementação em conjunto com uma comunicação mais versátil, por rede local ou *internet*, é deixada em aberto, tendo-se descriminado convenientemente as formas de controlo do controlador através da biblioteca dinâmica, que poderá, mediante investigação futura, ser utilizada em parceria com novo *software* que permita o seu acesso remoto.

Caso não estejam disponíveis os componentes mínimos necessários para a construção do controlador, indicou-se, durante o corrente documento, a possibilidade de se optar pela placa de prototipagem *Arduino Uno*, ou compatível, para a sua implementação, que se volta a realçar por se considerar, ao estar disponível por valores acessíveis (que podem ficar abaixo dos *10€* em *websites* de venda livre), uma óptima alternativa para a contenção dos custos de aquisição do controlador (a sua interface USB não foi testada neste projecto, mas não se antevê qualquer incompatibilidade). Esta placa permitiria a obtenção facilitada do controlador, por um custo inferior ao de reunir, individualmente, todos os componentes necessários para a sua construção, num encapsulamento que evitaria possíveis erros dos estudantes nas montagens das suas próprias placas. Na eventualidade de se considerar pertinente um acesso remoto através da *internet*, como se sugeriu anteriormente, poderia

então equacionar-se o seu emparelhamento com uma interface compatível para o efeito (*Ethernet Shield Network Expansion W5100* ou compatível).

7.3. ADAPTAÇÃO À NORMA IEEE 1149.4

Referido sucintamente no restante documento, a norma *IEEE 1149.4* prevê a utilização do BS em sistemas de sinais mistos para o teste de componentes passivos e activos. Basicamente, a sua implementação física difere do *IEEE 1149.1* ao incluir duas novas entradas analógicas na sua interface (*Analog Test Access Port* – ATAP)[21].

Considerando-se apenas o controlo da interface digital de um componente compatível com a norma *IEEE 1149.4*, é possível a utilização do actual controlador sem ser necessária qualquer alteração aos procedimentos enunciados. No entanto, o controlo dos sinais analógicos também previstos na norma, não pode ser facilmente assegurado.

Em termos de *hardware*, crê-se que a adaptação do controlador para suportar as novas funcionalidades implementadas nesta norma seria exequível, sobrando suficientes recursos no MCU do controlador para funcionar possivelmente em conjunto com um circuito de sinal analógico, como um conversor digital-analógico (*Digital-to-Analog Converter* − DAC), para operar a porta ATAP (pinos *AT1* e *AT2*). A leitura do seu valor seria feita com o conversor analógico-digital (*Analog-to-Digital Converter* − ADC) já integrado no controlador. Crê-se que a implementação das novas funções de actuação das entradas analógicas, no *firmware* do controlador, não traria problemas.

No entanto, o restante *software* de controlo desenvolvido para o corrente projecto visou apenas a norma *IEEE 1149.1* e é inadequado, pois considere-se: A descrição dos testes, forma pela qual a aplicação gráfica comanda quase todas as operações por BS, só pode ser feita eficientemente segundo SVF ou XSVF, e nenhum destes formatos de teste suporta as funções exclusivas do *IEEE 1149.4* com vista à atribuição e leitura de valores analógicos. De forma análoga, o algoritmo de interpretação da linguagem de descrição da implementação da arquitectura de BS de cada componente, onde se alicerçam todas as funções automáticas da aplicação gráfica, não suporta o formato recomendado dos ficheiros descritivos da norma *IEEE 1149.1*: uma extensão ao formato BSDL, também referida como *Analog Boundary Scan Description Language* (ABSDL). Embora as regras de sintaxe e semântica dos ficheiros BSDL anteriores sejam mantidas, o código de *parsing* de BSDL da biblioteca dinâmica teria de ser modificado para contemplar as novas

descrições necessárias[82]. Assim, considera-se que o controlo previsto, orientado para infra-estruturas baseadas no *IEEE 1149.1*, não poderá oferecer suporte total a sistemas de sinais mistos sem que fossem consideradas mudanças extensivas à actual base de funcionamento do sistema.

7.4. PERSPECTIVAS PARA DESENVOLVIMENTOS FUTUROS

A implementação faseada, com a qual se tentou garantir a modularidade do sistema, resultou nas três componentes de programação, que são referidas como: *Firmware* do controlador; Biblioteca de controlo; e Aplicação gráfica. Estas foram desenvolvidas para aplicação generalizada, e são acompanhadas, neste documento, da descrição dos seus dados de entrada e saída, pretendendo-se dar liberdade a uma futura evolução. Ao entender-se que a aplicação gráfica, tal como está desenvolvida, possa não dar a conveniente resposta numa determinada aplicação particular, conta-se que seja possível utilizar o *firmware* e biblioteca de controlo desenvolvidos associados a um novo *software* com o objectivo pretendido. As formas de integração deste *software* com as componentes desenvolvidas foram explicadas na sua descrição (secção 5). É também possível a substituição da própria biblioteca de controlo, e utilizar directamente o controlador, mas não se antevê benefícios nessa abordagem. Por outro lado, também deverá ser possível adoptar outro controlador físico, devendo-se apenas programar o seu suporte na biblioteca de controlo.

Relativamente à evolução do sistema actual, tal como se oferece, *i.e.* mantendo-se a mesma abordagem genérica, podem perspectivar-se apenas alguns desenvolvimentos à sua funcionalidade e desempenho oferecido:

No que respeita à evolução da funcionalidade do controlador, relembra-se a correcção pendente de alguns aspectos determinados do seu funcionamento actual, que foram listados na secção 6.1.1 (“Compilação das Limitações Técnicas”). À parte destes problemas menores, idealiza-se também a possibilidade de ser integrado na aplicação gráfica novo separador que vise simplificar o processo de controlo de vários componentes em simultâneo. Pretendendo-se investir um futuro desenvolvimento na melhoria do desempenho do controlador, aconselha-se que sejam consideradas as alternativas apresentadas na secção 6.2.3 (“Considerações Sobre os Factores Limitadores e Posteriores Optimizações”).

Crê-se que o sistema, tal como se oferece, esteja num estado muito satisfatório para os objectivos a que se propõe, recomendando-se apenas novos desenvolvimentos no sentido de o adaptar a outros projectos; ou na mudança de abordagem na sua aplicação didáctica (*e.g.* implementação de acesso remoto).

Referências Documentais

[1] "IEEE Std 1149.1 (JTAG) Testability Primer", Texas Instruments, 1996

[2] Yakyma, A.; "Design for Testability: A Vital Aspect of the System Architect Role"; Scaled Agile Framework − Leffingwell LLC;

Disponível, à data de 02/04/2014, em:

http://scaledagileframework.com/guidance/design-for-testability-a-vital-aspect-of-the-system-architect-role-in-safe/

[3] Barnai, A.; "Boundary Scan: The Pratical Approach"; Integrated Technology Corporation; Test & Inspection – SMT Magazine; Março 2012

[4] Riessen, R. et al.; "Designing and Implementing an Architecture With Boundary Scan"; University of Twente; Netherlands; Fevereiro 1990

[5] Coenen, J.; "Development of a Boundary Scan Test Controller Creation Tool"; Eindhoven University of Technology; Netherlands; Julho 1993

[6] Bleeker, H.; Eijnden, P.; Jong, F.;"Boundary-Scan Test: A Practical Approach"; Kluwer Academic Publishers, Netherlands, 1993

[7] Mitchell, Jeff; Lockwood, John; "Tools For In-Circuit Testing Of On-Line Content Processing Hardware", Washington University, St. Louis, 2005

[8] Bennetts, R.; "What Problems Can Boundary Scan Solve?"; EE: Evaluation Engineering 45, no. 5: pp.18-24; Maio 2006

[9] Frenzel, L.; "The Embedded Plan For JTAG Boundary Scan"; Electronic Design; Technology Report Ed. Online 19626 – pp. 38-45; Novembro de 2008

[10] Terry, M.; "New Test Strategy for Tomorrow's Manufacturing", Circuits Assembly, CMP Media Inc, New York, 2000

[11] Malian, J.; Eklow, B.; "Embedded Testing in an In-Circuit Test Environment"; Cisco Systems Inc., 2008

[12] "Boundary-Scan Tutorial", ASSET InterTech Inc., 2000

[13] "Why should you care about JTAG / Boundary Scan / IEEE 1149.x", Goepel Electronics, 2007

[14] Hallmen, K.; "In-Circuit Manufacturing Defects Analyzer (MDA) Systems For High-Density Assemblies"; CheckSum, Inc.

Disponível, à data de 14/04/2014, em:

http://www.checksum.com/pdfs/checksum_etronix.pdf

[15] "Flying Probe Guidelines"; Test Coach Corporation;

Disponível, à data de 14/04/2014, em:

http://www.testcoachcorp.com/finnProducts/downloads/DFT/DFT.pdf

[16] Leinbach, G.; Oresjo, S.; “The Why, Where, What, How, and When of Automated X-ray Inspection”; Agilent Technologies; Colorado

Disponível, à data de 14/04/2014, em:
http://www.home.agilent.com/upload/cmc_upload/All/Why_Where_What_How_When.pdf

[17] Meissner, S.; "The Basics of Boundary Scan." Circuits Assembly 16, no. 7: pp. 28; Julho 2005

[18] Goepel, H.; "Reducing the Cost of Test With Boundary Scan."; EE: Evaluation Engineering 43, no. 1: pp.28-33; Janeiro 2004

[19] Poole, Ian; “JTAG Specification”; Electronics Test And Measurement; Adrio Communications Ltd.

Disponível, à data de 16/04/2014, em:

http://www.radio-electronics.com/info/t_and_m/boundaryscan/jtag-specifications.php

[20] Felgueiras; M.; “Apoio à depuração e teste de circuitos mistos compatíveis com a norma IEEE1149.4”; Faculdade de Engenharia da Universiade do Porto; Julho 2008

[21] “Working Group Areas”; IEEE Standards Association

Disponível, à data de 20/04/2014, em:

http://grouper.ieee.org/groups/

[22] “JTAG Tutorial”; Corelis Inc.

Disponível, à data de 16/04/2014, em:

http://www.corelis.com/education/JTAG_Tutorial.htm

[23] Nelson, R.; "How software enhances boundary scan"; Test & Measurement World 23, no. 3: pp. 27; 2003

[24] Stollon, N.; “Embedded Instrumentation Integration Using IEEE Nexus 5001 and 1149.7”; HDL Dynamics; 2009

Disponível, à data de 28/04/2014, em:

http://www.nexus5001.org/sites/default/files/EI%20Using%20IEEE%205001%20and%201141_full_061509.pdf

[25] Johnson, B.; “DSP56300 JTAG Examples”; Freescale Semiconductor Inc.; Rev. 1; Agosto de 2005

[26] “IEEE Standard 1149.1 (JTAG) in the SX/RTSX/SX-A/eX/RT54SX-S Families”; Actel Corporation; Outubro 2002

[27] “IEEE Standard for Test Access Port and Boundary-Scan Architecture (IEEE Std 1149.1-2013) - Introduction”; IEEE Computer Society; Maio 2013

[28] Horwitz; H.; “IEEE 1149.1™-2013 Enables Integrated Circuit Counterfeit Protection”; IEEE-USA Today’s Engineer; Junho 2013

Disponível, à data de 10/06/2014, em:

http://www.todaysengineer.org/2013/Jun/ic-counterfeiting-IEEE-1149-1-2013.asp

[29] “IEEE Standard Test Access Port and Boundary-Scan Architecture (IEEE Std 1149.1-2001)”; IEEE Computer Society; Julho 2001

[30] Bennets, B.; “Boundary Scan Tutorial”; ASSET InterTech Inc.; Ver. 2.1; Setembro 2002

[31] “Serial Vector Format Specification”, ASSET InterTech, Inc.; Revision E; Março 1999

[32] Bridgford, B.; Cammon, J.; “SVF and XSVF File Formats for Xilinx Devices”; Xilinx, Inc; Application Note XAPP503 ver.2.1; Agosto 2009

Disponível, à data de 16/04/2014, em:

http://www.xilinx.com/support/documentation/application_notes/xapp503.pdf

[33] Waschk, K.; “Universal JTAG library, server and tools”; UrJTAG; 2008

Disponível, à data de 24/09/2013, em:

http://sourceforge.net/p/urjtag/svn/HEAD/tree/trunk/urjtag/doc/UrJTAG.txt

[34] “SVF Files”; XJTAG;

Disponível, à data de 05/11/2013, em:

http://www.xjtag.com/support-jtag/jtag-svf-files.php

[35] “EIA/JEDEC Standard: Standard Test and Programming Language (STAPL)”; Electronic Industries Alliance; JEDEC Solid State Technology Association; JESD71; Agosto 1999

[36] “Actel BSDL Files Format Description”; Actel Corporation; Application Note AC278; Julho 2006

Disponível, à data de 17/04/2014, em:

http://www.actel.com/documents/BSDLformat_AN.pdf

[37] Sheppard, A.; Giridhar, V.; "Hierarchical Scan Description Language Syntax Specification"; ASSET InterTech, Inc.; Texas Instruments Inc.; Rev. A; Agosto 1992

[38] Shuang; Y.; “IEEE 1149.1-2013 DESIGNED TO SLASH ELECTRONICS INDUSTRY COSTS”; IEEE Standards Association; Junho 2013

Disponível, à data de 10/06/2014, em:

http://standards.ieee.org/news/2013/ieee_1149.1.html

[39] Zhan, Rongkai “Jtager User Manual”, v0.2.0, 2004

Disponível, à data de 24/09/2013, em:

http://jtager.sourceforge.net/doc/jtager.html

[40] Ferreira; J.; “JTAGer: A Windows BS test controller application”; Faculdade de Engenharia da Universidade do Porto

Disponível, à data de 07/02/2014, em:

http://paginas.fe.up.pt/~jmf/hibu2k2/contents/jtager/jtager-intro.htm

[41] Ferreira, L.; “Tapper”; Faculdade de Engenharia da Universidade do Porto; cap.6 pp.6.1-6.5; 1993/1994

Disponível, à data de 07/02/2014, em:
http://paginas.fe.up.pt/~jmf/hibu2k2/tapper/tapper.pdf

[42] "JTAGTest User Manual"; SeCons Ltd.; Ver. 1.04; 2011
Disponível, à data de 07/02/2014, em:
http://www.jtagtest.com/downloads/jtagtest.pdf

[43] "ViaTAP Hardware Manual"; SeCons Ltd.; Ver. 1.5; 2011
Disponível, à data de 07/02/2014, em:
http://www.jtagtest.com/downloads/viatap.pdf

[44] "JTAG Tools", *Openwince*
Disponível, à data de 24/09/2013, em:
http://openwince.sourceforge.net/jtag/

[45] "JTAG Scan Educator Application Report", Ver.2, Texas Instruments Incorporated, 2003

[46] "Boundary Scan Coach"; GOEPEL electronics Ltd.; Ver. 1.2; Janeiro 2005
Disponível, à data de 07/02/2014, em:
http://www.goepel.com/en/jtag-boundary-scan/education/boundary-scan-coach.html

[47] "Wiggler"; Macraigor Systems LLC
Disponível, à data de 07/02/2014, em:
http://www.macraigor.com/downloads/wiggler_ds.pdf

[48] "Buffered Cable: Wiggler"; OpenWrt Wireless Freedom
Disponível, à data de 10/05/2014, em:
http://wiki.openwrt.org/doc/hardware/port.jtag.cable.buffered

[49] Oliver, S.; et al.; " Open On-Chip Debugger: OpenOCD User's Guide"; The OpenOCD Project; Ver. 0.9.0; Maio 2014
Disponível, à data de 10/05/2014, em:
http://openocd.sourceforge.net/doc/pdf/openocd.pdf

[50] "AVR JTAG ICE User Guide"; Atmel; Rev. 2475A; Setembro 2001
Disponível, à data de 10/02/2014, em:
http://www.atmel.com/images/doc2475.pdf

[51] Jones, A.; Freeman; A.; "Visual C# 2010 Recipes"; Apress.; pp 789-904, 2010

[52] Ákos, V.; "ISO JTAG ISP Programmer"; Dezembro 2008
Disponível, à data de 10/02/2014, em:
http://sem.sch.bme.hu/projektek/avr_programozo_es_debugger

[53] "PicoTAP – JTAG/Boundary Scan Controller"; GOEPEL electronics Ltd.
Disponível, à data de 10/05/2014, em:
http://www.goepel.com/en/jtag-boundary-scan/boundary-scan-instruments/hardware/controller/picotap.html

[54] "GoJTAG Downloads: PicoTAP - Driver and Schematic"; Tallinn University of Technology; TU Ilmenau; Company Testonica Lab
Disponível, à data de 10/05/2014, em:
http://www.gojtag.com/downloads

[55] "USB-Blaster Download Cable: User Guide"; Altera; Rev. 2.5; Abril 2009
Disponível, à data de 10/02/2014, em:
http://www.altera.com/literature/ug/ug_usb_blstr.pdf

[56] Waschk, K.; "ixo.de USB JTAG Pod Project: usb_jtag"; ixo.de; Janeiro 2007
Disponível, à data de 10/05/2014, em:
http://sourceforge.net/projects/ixo-jtag/

[57] "SN54LVT8980A SN74LV8980A Embedded Test-Bus Controllers"; Texas Instruments Incorporated; Rev. SCBS755B; Março 2004
Disponível, à data de 10/02/2014, em:
http://www.ti.com/lit/ds/scbs755b/scbs755b.pdf

[58] "FT2232C Dual USB UART / FIFO I.C."; Future Technology Devices International Ltd.; Ver. 1.4; 2005
Disponível, à data de 10/02/2014, em:
http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232C.pdf

[59] "USB JTAGScan Project"; Future Technology Devices International Ltd.; 2004
Disponível, à data de 10/02/2014, em:
http://www.ftdichip.com/Support/SoftwareExamples/MPSSE/FT2232C-Proj03_v11.pdf

[60] "PLD Tools: creating SVF, JAM, STAPL and other formats"; Corelis;
Disponível, à data de 13/11/2013, em:
http://corelis.com/blog/pld-tools-creating-svf-jam-stapl-and-other-formats/#.UoOnK_nJQu6

[61] "AVR095: Migrating between ATmega48, ATmega88 and ATmega168"; ATMEL Corporation, Rev. 2554A-AVR-02/04; Fevereiro 2004
Disponível, à data de 18/11/2013, em:
http://www.atmel.com/Images/doc2554.pdf

[62] "AVR512: Migration from ATmega48/88/168 to ATmega48P/88P/168P"; ATMEL Corporation, Rev. 8035A-AVR-07/06; Junho 2006
Disponível, à data de 18/11/2013, em:
http://www.atmel.com/Images/doc8035.pdf

[63] Zabolotny; W.; "Low cost USB – local bus interface for FPGA based systems"; Institute of Electronic Systems, Warsaw University of Technology

[64] "Virtual USB port for AVR microcontrollers"; Objective Development; 2013
Disponível, à data de 17/12/2013, em:

http://www.obdev.at/products/vusb/index.html

[65] "JTAG Interface: Common Pinouts"; Amontec, Inc; Ver. 1.1; Agosto 2006
Disponível, à data de 18/11/2013, em:
http://www.amontec.com/pub/amt_ann003.pdf

[66] Fischl, T.; "*USBasp - USB programmer for Atmel AVR controllers*"; Maio 2011
Disponível, à data de 03/03/2014, em:
http://www.fischl.de/usbasp/

[67] "*Windows Rundll and Rundll32 Interface*"; ID 164787; Microsoft
Disponível, à data de 10/05/2014, em:
http://support.microsoft.com/kb/164787/en-us

[68] "Introduction to WPF"; .Net Framework 4.5; MSDN
Disponível, à data de 16/12/2013, em:
http://msdn.microsoft.com/en-us/library/aa970268.aspx

[69] "ATmega48PA/88PA/168PA/328P Datasheet", ATMEL Corporation, Rev. 8161D–AVR–10/09; Outubro 2009

[70] "CP2102/9 – Single chip USB to UART bridge"; Silicon Laboratories; Rev. 1.6; Dezembro 2013
Disponível, à data de 28/01/2014, em:
http://www.silabs.com/Support%20Documents/TechnicalDocs/CP2102-9.pdf

[71] "Connecting to a target board with thr AVR JTAGICE mkII"; ATMEL Corporation, Rev. 2562C-AVR-07/06; Julho 2006
Disponível, à data de 18/11/2013, em:
http://www.atmel.com/Images/doc2562.pdf

[72] "ATmega16 Datasheet", ATMEL Corporation, Rev. 2466T-07/10; Julho 2010

[73] "SN74BCT8245A Scan Test Device - Datasheet"; Texas Instruments Incorporated Julho 1996

[74] Matan, Z.; "CodeForge: bsdl2jtag.c"; ETC s.r.o.; 2003
Disponível, à data de 14/01/2014, em:
http://www.codeforge.com/read/94570/bsdl2jtag.c__html

[75] Wolf, C.; "Lib(X)SVF"; RIEGL Research ForschungsGmbH; 2009
Disponível, à data de 14/01/2014, em:
http://www.clifford.at/libxsvf/

[76] Beelen, T.; "RS-232 for Linux and Windows"; 2013
Disponível, à data de 14/01/2014, em:
http://www.teuniz.net/RS-232/

[77] "Unmanaged Code"; .Net Framework 4.5; MSDN
Disponível, à data de 06/03/2014, em:

http://msdn.microsoft.com/en-us/library/0e91td57.aspx

[78] "Porting from UNIX to Win32"; Microsoft Development Network - MSDN
Disponível, à data de 16/01/2014, em:
http://msdn.microsoft.com/en-us/library/y23kc048.aspx

[79] "JEDEC Standard - Standard Manufacturer's Identification Code"; JEDEC Solid State Technology Association; rev. JEP106AJ, Setembro 2012

[80] Sousa, E.; "Controlador Boundary-Scan Baseado num Micro Servidor Web"; Faculdade de Engenharia da Universiade do Porto; Julho 2009

[81] Vilhena; J.; "Experimentação Remota De Infraestruturas IEEE 1149.1/.4"; Instituto Superior de Engenharia do Porto; Novembro 2013

[82] "IEEE 1149.4 Standard for a Mixed-Signal Test Bus"; IEEE 1149.4 Mixed-Signal Test Bus Working Group; 2010
Disponível, à data de 11/04/2014, em:
http://grouper.ieee.org/groups/1149/4/dot4_itc2010_final.pdf

Anexo A. Programador ISP USBasp

A programação da placa do controlador de *Boundary Scan* desenvolvido, cujo núcleo é composto pelo MCU *ATmega328P*, foi efectuada com um programador *ISP*. Havendo várias alternativas disponíveis para o efeito, a que se utilizou é denominada de "*USBasp"* e foi construída de acordo com a Figura 98:

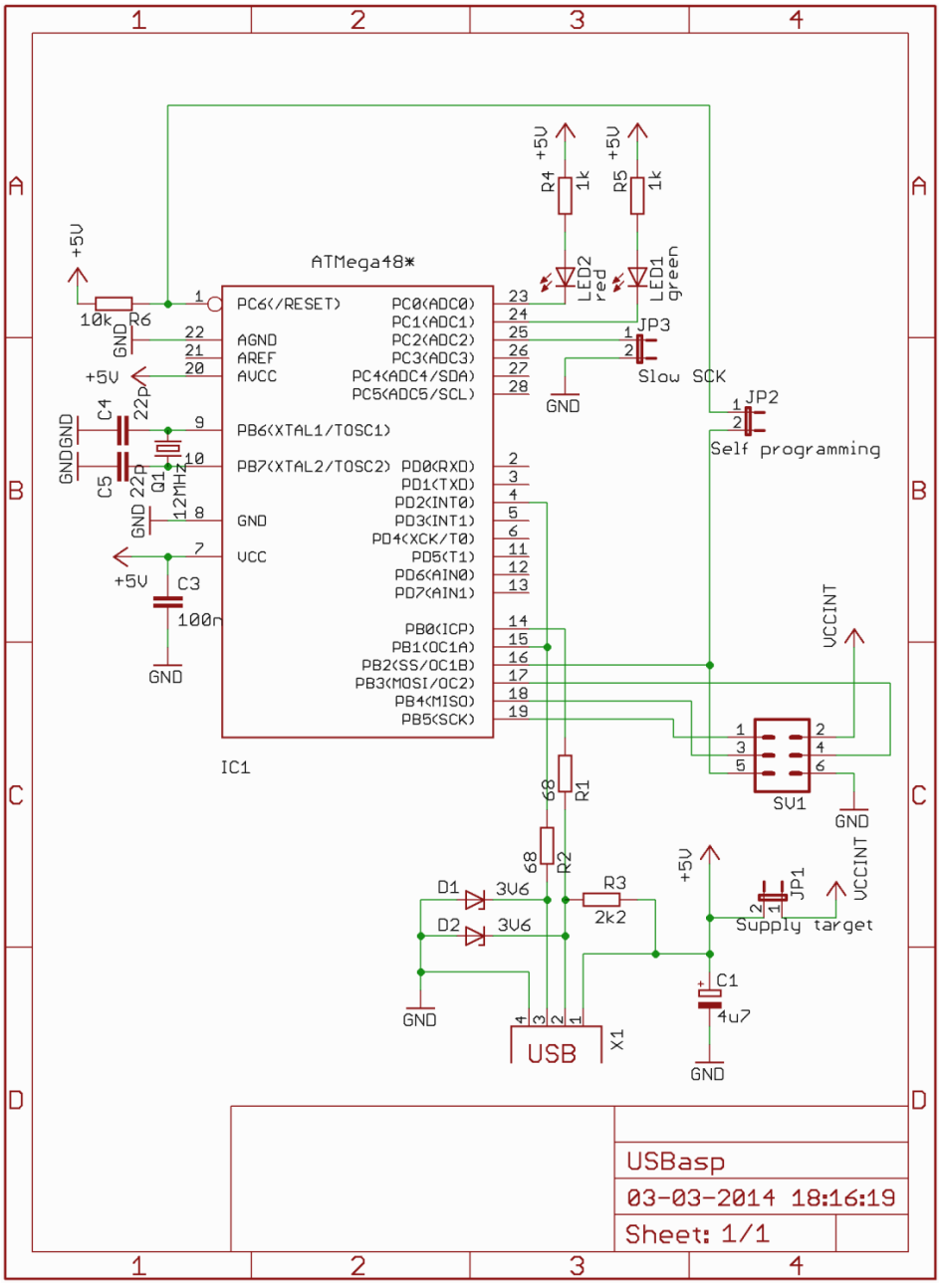


Figura 98 Esquema eléctrico do programador ISP *USBasp* (adaptado de [66])

O esquema eléctrico apresentado sofreu algumas ligeiras modificações em relação ao circuito original, ao ser preterido o conector de programação *ISP10PIN* em favor do mais pequeno *ISP6PIN*, e removidas as linhas de comunicação UART (em *PD0* e *PD1)*, originalmente presentes e cujo intuito não tem utilidade no corrente projecto.

Este programador comunica com o computador através de ligação *USB*. É composto por um *MCU* dedicado (*ATmega48* ou *ATmega8*), um cristal de *12 MHz*, e alguns componentes discretos. Como não necessita de qualquer componente *SMD*, é prática a sua montagem numa placa perfurada. O seu *firmware* e *drivers*, configuração dos *fuses* do MCU, assim como informações adicionais, estão acessíveis no *website* oficial, que também inclui desenhos para placas de circuito impresso[66]. A instalação de *drivers* no computador é necessária. Não sendo suportado pelo *AVR Studio*, foi utilizado em conjunto com o *software* "*Khazama AVR Programmer*".

Na seguinte Tabela 28 listam-se os componentes utilizados na sua montagem:

Tabela 28 Listagem do material utilizado no programador *USBasp*

Nome de Ref.:	Componente:	Unid.:	Valor p/unid.[30]:	Valor Total:
C1	*4.7uF ELEC 50V*	*1*	*0,06€*	*0,06€*
C3	*100nF MLCC 50V*	*1*	*0,035€*	*0,035€*
C4,C5	*22pF MLCC 50V*	*2*	*0,025€*	*0,05€*
LED1, LED2	*LED 3mm*	*2*	*0,098€*	*0,196€*
IC1	*ATmega48-20PU*	*1*	*3,20€*	*3,20€*
R6	*10 KΩ 1/4 W*	*1*	*0,008€*	*0,008€*
R1, R2	*68 Ω 1/4 W*	*2*	*0,008€*	*0,016€*
R3	*2.2 KΩ 1/4 W*	*1*	*0,008€*	*0,008€*
D1, D2	*Zener 3.6V BZX79*	*2*	*0,105€*	*0,21€*
Q1	*Cristal Std. 12MHz*	*1*	*0,27€*	*0,27€*
SV1 (3x2), JP1 (3x1), JP2 (3x1), JP3 (3x1)	*Male Header 2.54mm (12pin)*	*2*	*0,55€*	*1,10€*
X1	*Conector USB-A THT*	*1*	*0,41€*	*0,41€*
			Total:	*5,56€*

[30] Preços do distribuidor *Farnell*, em Portugal, à data de 28 de Fevereiro de 2014, sem contabilização de portes de envio.

Os componentes listados constam do circuito da Figura 98 e não contemplam a placa perfurada, onde foram montados, e os cabos USB e *flat cable* utilizados no seu funcionamento. No sistema desenvolvido, este programador foi montado na mesma placa do controlador de BS, para facilitar a re-programação do *firmware*. As alimentações de ambos os circuitos foram separadas pelo *jumper JP1*, pelo que o programador ISP pode ser mantido desligado quando não estiver a ser utilizado.

Anexo B. Montagem do Protótipo do Controlador e Placa de Teste

O sistema funcional desenvolvido consiste na placa do controlador, que incluí o circuito do controlador de BS e do programador *USBasp*; e na placa de teste. O programador *USBasp* foi mantido ligado à porta de ISP do MCU que efectua o controlo do BS, para programação do *firmware*. O controlador foi conectado à interface JTAG da placa de teste, que, recorde-se, tem alimentação própria por USB. Ambas as placas foram encaixadas na mesma estrutura.

As seguintes figuras mostram o sistema construído e testado:

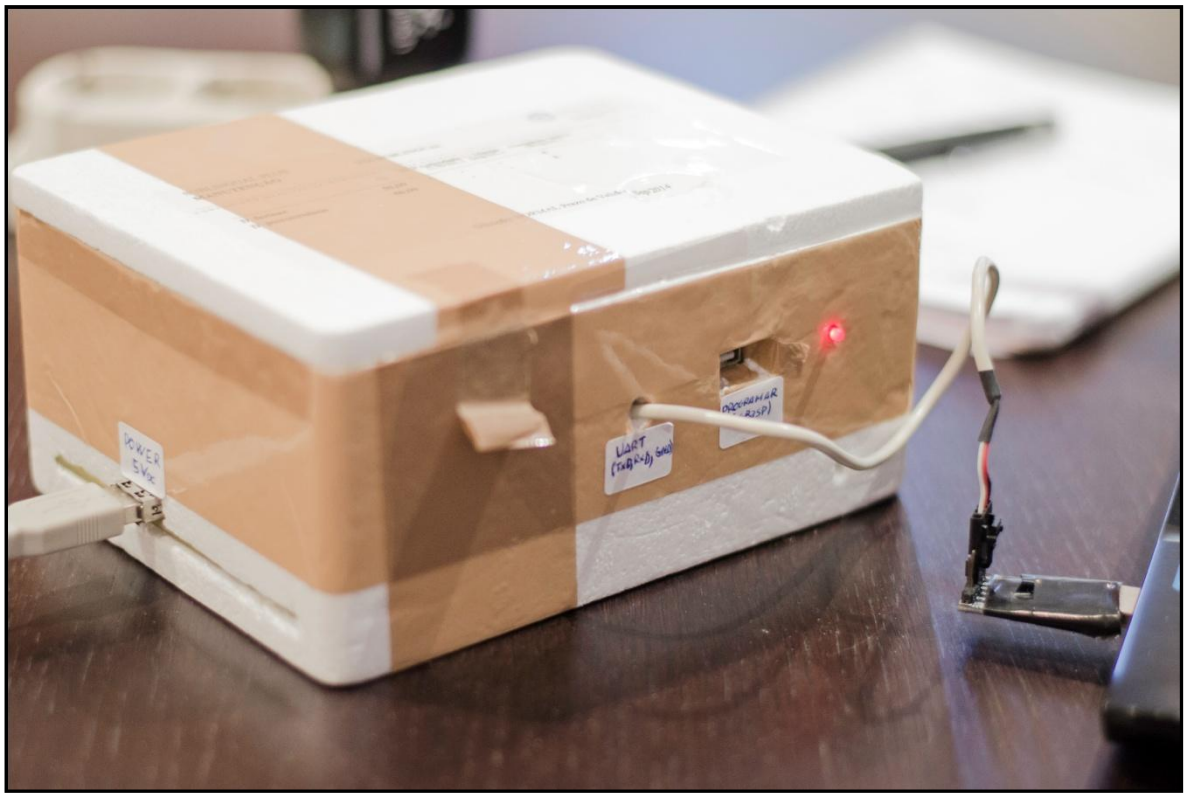


Figura 99 Protótipo funcional em operação

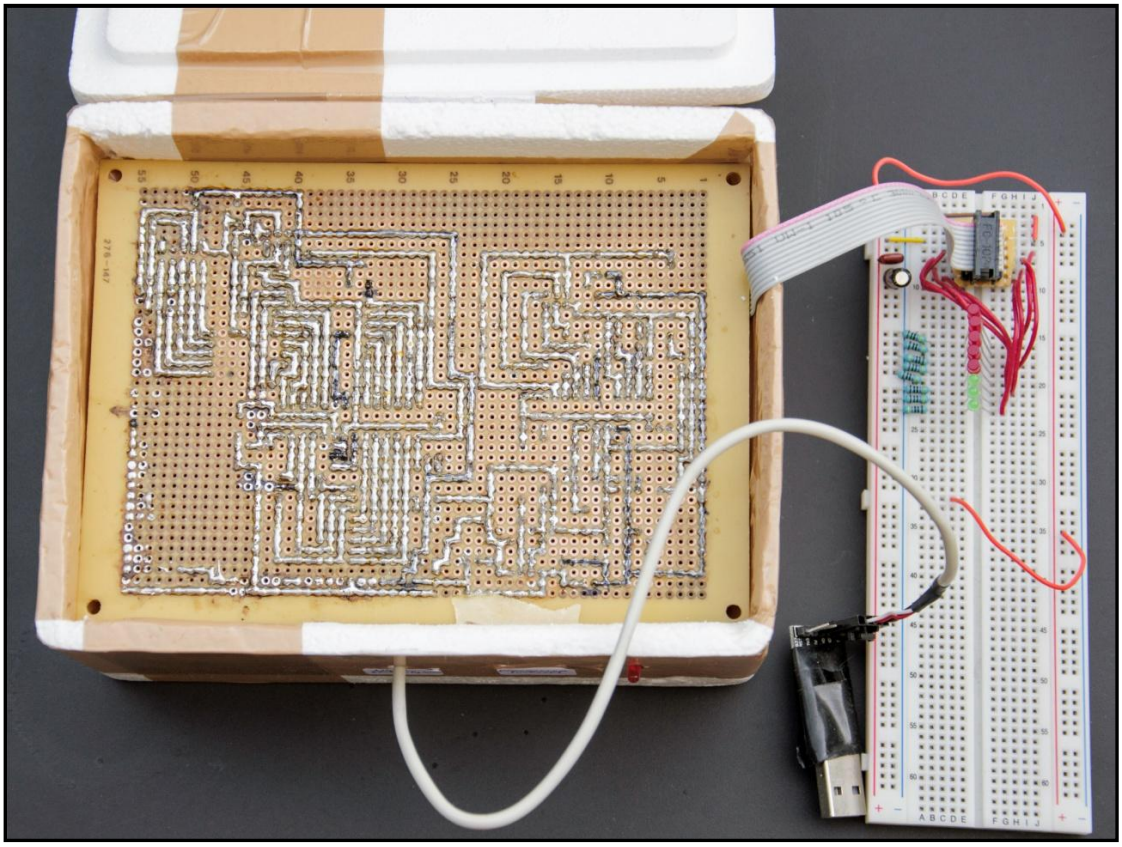


Figura 100 Protótipo funcional e *breadboard* com LEDs para teste

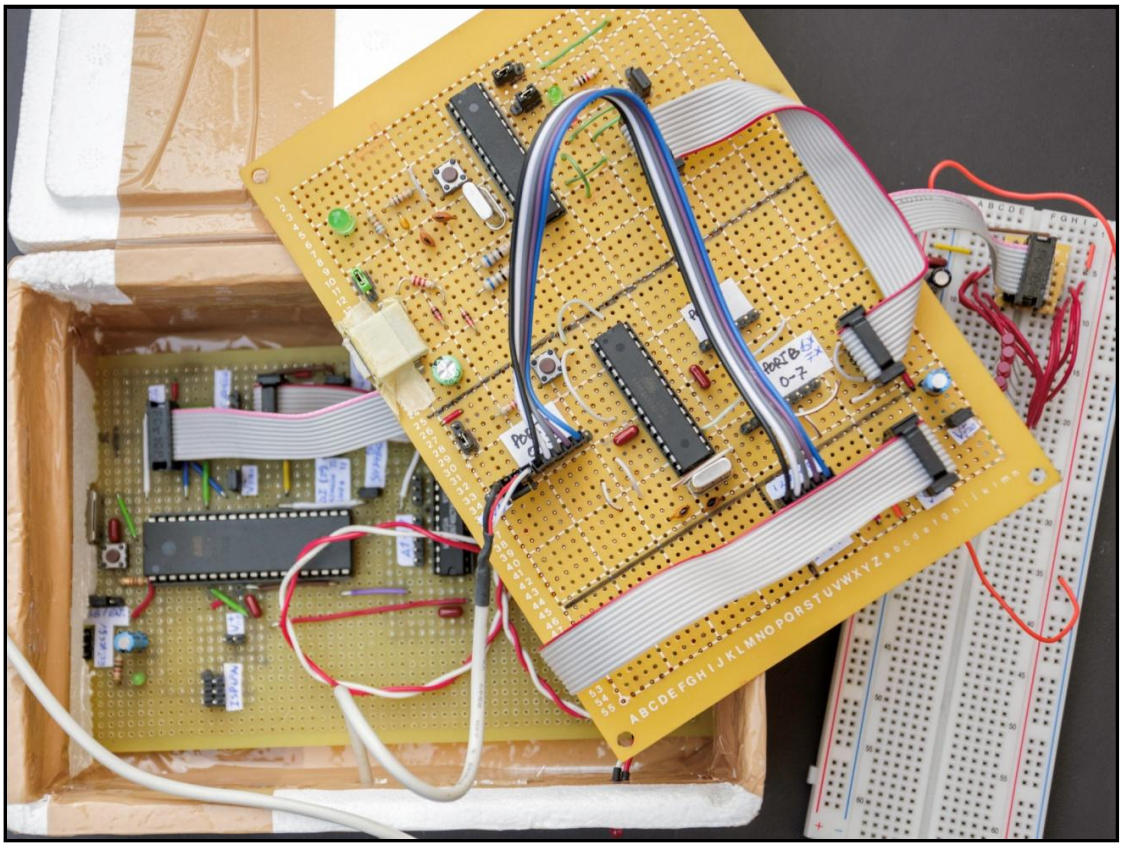


Figura 101 Placas do protótipo funcional

Anexo C. Projecto do PCB do controlador de BS

O sistema desenvolvido foi montado, no protótipo funcional, numa placa perfurada utilizando componentes THT. O circuito que serve de interface entre a conecçao USB do computador e a ligação UART da placa foi utilizado num adaptador separado. Esta é a montagem que mais se enquadra com os objectivos do projecto ao facilitar a contrução do controlador num contexto didático. Ainda assim, foi efectuado o projecto de um PCB que prevê um sistema físico mais compacto e organizado, ao incluir o circuito do conversor série-USB, dispensando a utilização do adaptador separado. Não foi incluído o programador ISP. A seguinte Figura 102 mostra o circuito utilizado no PCB:

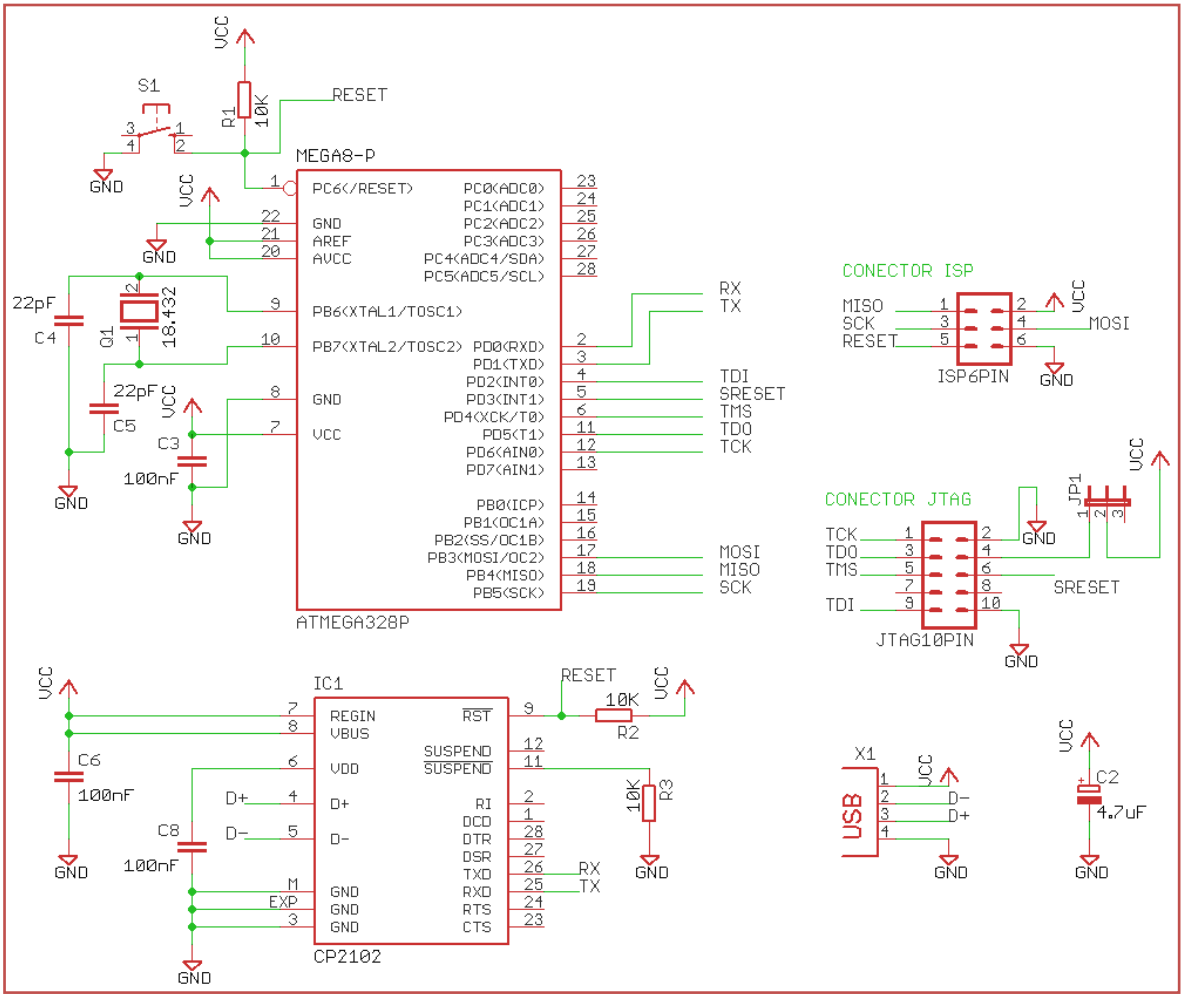


Figura 102 Esquema eléctrico utilizado no projecto do PCB

Este PCB pode tornar mais difícil a soldagem manual do sistema porque o componente *CP2102* apenas está disponível em encapsulamento *SOIC24*. Os restantes componentes escolhidos utilizam encapsulamento THT, por se crer que possam ser mais fáceis de obter e por a placa estar assim já suficientemente compacta.

Foi projectada uma placa de duas camadas com recurso ao *software* "*EAGLE 5.6.0*". Na Figura 103 apresentam-se a camada inferior e superior da placa, e a sua *silk screen*:

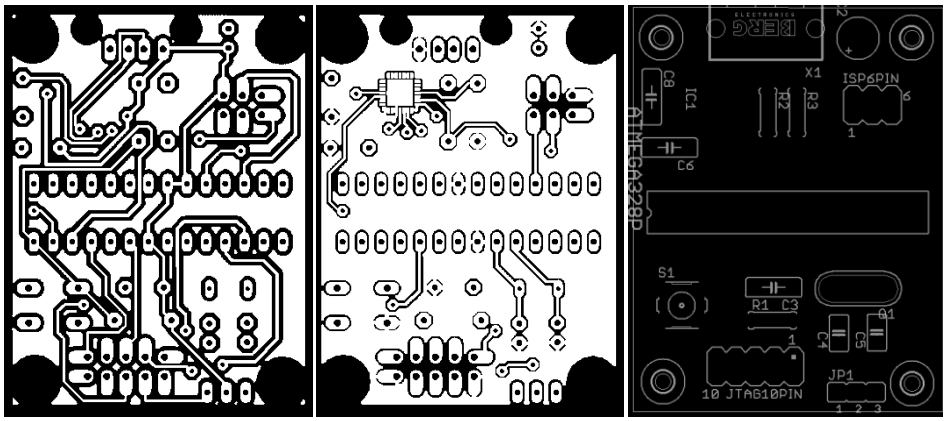


Figura 103 Camada inferior, superior, e *silk screen* do PCB do controlador de BS

Utilizando a ferramenta "*POV-Ray*" (versão *3.6.2*) e os modelos gráficos do pacote "*Eagle3d*" (versão *20110101*), reproduziu-se uma aproximação a três dimensões do PCB resultante, que se expõe na Figura 104 e Figura 105:

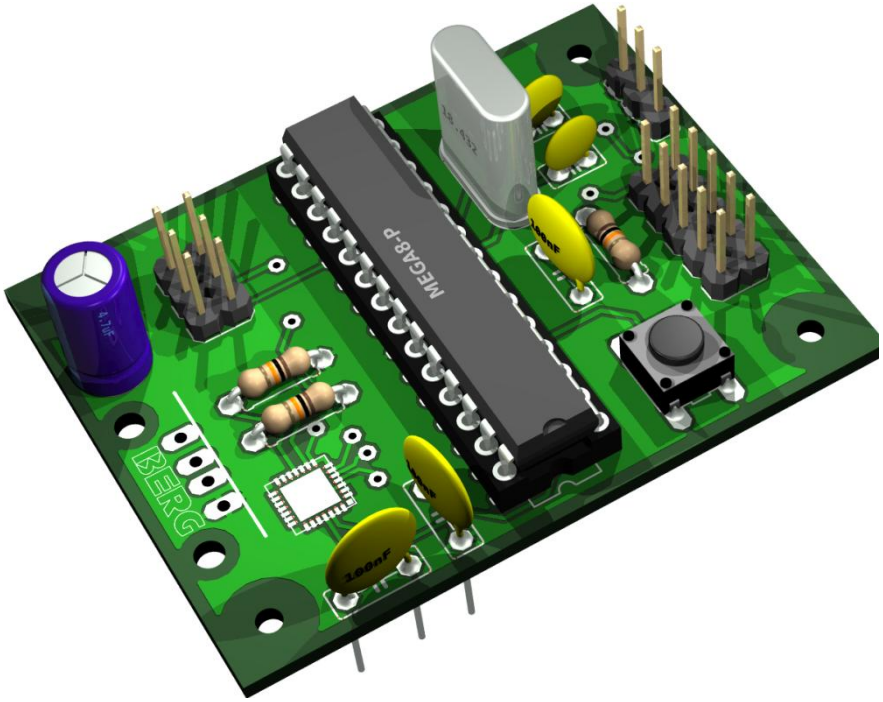


Figura 104 Aproximação gráfica da face superior do PCB

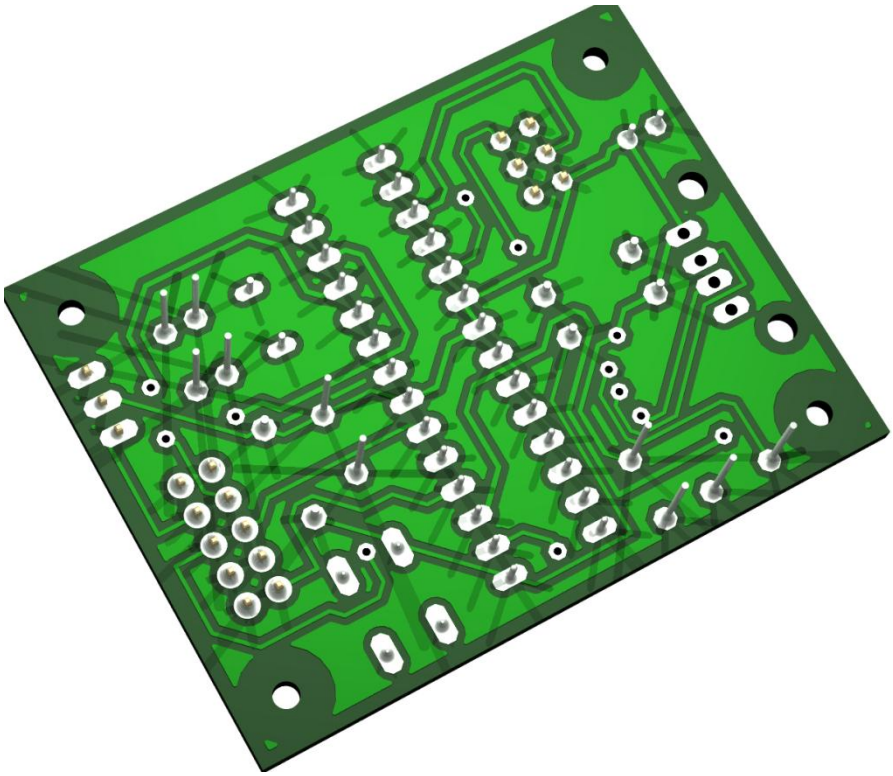


Figura 105 Aproximação gráfica da face inferior do PCB

Por falta dos modelos necessários, não estão montados o componente *CP2102* e o conector USB.

A placa tem a dimensão de *5,31cm* por *4,06cm*. Na sua face superior foi aplicado um plano de massa; os componentes SMD (*CP2102*); e as ligações do oscilador externo. Este plano serve também como alternativa no traçado das restantes interligações da placa. A camada inferior utiliza também um plano a preencher as áreas não utilizadas da placa, mas este não conduz qualquer sinal (a sua aplicação pode ser útil para economia de recursos, ao existir menos material para remover na criação das linhas do PCB).

Os componentes necessários são listados na seguinte Tabela 29:

Tabela 29 Listagem do material utilizado no PCB do controlador de BS

Nome de Ref.:	Componente:	Unid.:	Valor p/unid.[31]:	Valor Total:
ATMEGA328P	*ATMEGA328P-20PU*	*1*	*3,15€*	*3,15€*
IC1	*CP2102*	*1*	*3,53€*	*3,54€*
C3, C6, C8	*100nF MLCC 50V*	*3*	*0,035€*	*0,105€*
C2	*4.7uF ELEC 50V*	*1*	*0,06€*	*0,06€*
C4, C5	*22pF MLCC 50V*	*2*	*0,025€*	*0,05€*
ISP6PIN (3x2), JP1 (3x1), JTAG10PIN (5x2)	*Male Header 2.54mm (12pin)*	*2*	*0,55€*	*1,10€*
Q1	*Cristal Std. 18.432MHz HC-49*	*1*	*0,28€*	*0,28€*
R1, R2, R3	*10 KΩ 1/4 W*	*3*	*0,008€*	*0,024€*
S1	*Tactile SPST MC32828*	*1*	*0,105€*	*0,105€*
X1	*Conector USB-A THT*	*1*	*0,41€*	*0,41€*
			Total:	*8,82€*

[31] Preços do distribuidor *Farnell*, em Portugal, à data de 28 de Fevereiro de 2014, sem contabilização de portes de envio.

Anexo D. Resultados do Teste e Optimização do Protótipo Funcional

Os vários testes efectuados podem ter sido conduzidos em momentos diferentes do desenvolvimento do projecto, pelo que os primeiros testes ainda não contemplam as optimizações que se vieram a seguir. Por essa razão, pode não ser viável comparar o resultado entre testes distintos, mesmo considerando as configurações com que foram efectuados. É também de realçar que a execução de um mesmo ficheiro SVF sob as mesmas condições pode originar resultados ligeiramente diferentes (apenas nos dados referentes aos tempos decorridos e frequência de actuação).

Apresentam-se de seguida alguns resultados que foram registados no decorrer das optimizações, e são referidos na secção 6.2.1. Confirma-se a correcta execução da tarefa de cada um dos testes apresentados, sem qualquer erro, excepto se existir indicação do contrário.

Tabela 30 Diminuição de fluxo de dados - I

Resultados originais:		
Configuração:	*Baud rate: 19200; F_{MCU}= 18.435MHz*	
Descrição do Ficheiro SVF:	***Mantém o valor de TDI***	***Valores de TDI diferentes***
Total de ciclos de TCK:	*15886*	
Bytes transferidos: (Env/Rec/Total)	***270812b / 15888b / 286700b***	***270812b / 15888b / 286700b***
Tempo decorrido (tempo transf.):	*2m49s (2m29s)*	*2m50s (2m29s)*
Frequência média de actualização de TCK:	*188 Hz*	*186 Hz*
Evitando repetir a atribuição de TDI e TMS quando desnecessária:		
Configuração:	*Baud rate: 19200; F_{MCU}= 18.435MHz*	
Descrição do Ficheiro SVF:	***Mantém o valor de TDI***	***Valores de TDI diferentes***
Total de ciclos de TCK:	*15886*	

Bytes transferidos: (Env/Rec/Total)	***170996b / 15888b / 186884b***	***213332b / 15888b / 229220b***
Tempo decorrido (tempo transf.):	*1m57s (1m37s)*	*2m18s (1m59s)*
Frequência média de actualização de TCK:	*271 Hz*	*230 Hz*

Tabela 31 Frequência de relógio de MCU

Variação da frequência do MCU:			
Frequência MCU:	***1MHz***	***8MHz***	***18.432MHz***
Baud rate:		*4800*	
Aumento de freq. do MCU:	*--*	*8,00x*	*2,30x*
Total de ciclos de TCK:		*15886*	
Tempo decorrido (s/ tempo transf.):	***29s***	***21s***	***20s***
Evolução do tempo decorrido:	*n.a.*	***-27,59%***	***-4,76%***
Frequência média de actualização de *TCK*:	*76 Hz*	*77 Hz*	*77 Hz*

Tabela 32 Velocidade da ligação UART entre o *CP2102* e o MCU

Variação da Baud rate:			
***Baud rate*:**	***4800***	***19200***	***76800***
Total de ciclos de TCK:	*15886*	*15886*	*15886*
Bytes transferidos: (Env/Rec/Total)	*170996b / 15888b / 186884b*	*170996b / 15888b / 186884b*	*170996b / 15888b / 186884b*
Tempo decorrido: (Tempo transf.)	***6m49s (6m29s)***	***1m56s (1m37s)***	***39s (24s)***
Tempo Estimado Transferência:	*389s*	*97s*	*24s*
Evolução do tempo de Transf.:	*--*	*-75,06%*	*-75,26%*
Evolução do tempo total:	*--*	*-71,64%*	*-66,38%*
Evolução da *baud rate*:	*--*	*400,00%*	*400,00%*

Tempo decorrido (s/ tempo transf.):	*20s*	*19s*	*15s*
Frequência média de actualização de TCK:	*77 Hz*	*273 Hz*	*814 Hz*
Baud rate:	***230400***	***570600***	***1152000***
Total de ciclos de TCK:	*15886*	*15886*	*Falhou*
Bytes transferidos: (Sent/Rec/Total):	*170996b / 15888b / 186884b*	*170996b / 15888b / 186884b*	
Tempo decorrido (Tempo transf.):	***31s (8s)***	***29s (3s)***	
Tempo Estimado Transferência:	*8s*	*3s*	
Evolução do tempo de Transf.:	*-66,67%*	*-62,50%*	
Evolução do tempo total:	*-20,51%*	*-6,45%*	
Evolução da *baud rate*:	*300,00%*	*247,66%*	
Tempo decorrido (s/ tempo transf.):	*23s*	*26s*	
Frequência média de actualização de TCK:	*1.02 KHz*	*1,09 KHz*	

Tabela 33 Modo de funcionamento sem leitura de *TDO* obrigatória

Implementação de funcionamento assíncrono:		
Configuração:	*Baud rate: 230400; F_{MCU}: 18.432MHz*	
Modo de funcionamento:	***Síncrono***	***Assíncrono***
Ficheiro SVF:	*Mesmo valor de TDI*	
Total de ciclos de TCK:	*15886*	
Bytes transferidos: (Env/Rec/Total)	***170996b / 15888b / 186884b***	***139224b / 2b / 139226b***
Tempo decorrido (s/ tempo transf.):	***23s***	***17s***
Frequência média de actualização de TCK:	*1,02 KHz*	*1,38 KHz*
Evolução da frequência:	--	*35,29%*

Tabela 34 Diminuição de fluxo de dados - II

Optimizações dos dados transferidos:			
Configuração:	*Baud rate: 230400; F_{MCU}: 18,432 MHz*		
Estado:	*Original*	*+ Redução comandos TDI e TMS*	*+ Implementação função Cycle_TCK*
Total de ciclos de *TCK*:	*15886*		
Bytes transferidos: (Env/Rec/Total)	***139224b / 2b / 139226b***	***133168b / 2b / 133170b***	***37852b / 2b / 37854b***
Tempo decorrido (tempo transf.):	*23s (6s)*	*21s (5s)*	*6s (1s)*
Frequência média de actualização de TCK:	***1,38 KHz***	***1,51 KHz***	***5,30 KHz***
Evolução da frequência:	*--*	*9,42%*	*250,99%*

Tabela 35 Funcionamento no computador sem sincronização

Sem comunicação com controlador (simulação)	
Configuração:	*Baud: n.a; F_{MCU}: n.a.*
Total de ciclos de *TCK*:	*15886*
Tempo decorrido (tempo transf.):	*0.09s (0.00s)*
Frequência média de actualização de TCK:	***369.18 KHz***

Tabela 36 Comparação de interfaces UART-USB I – Programar FLASH

Tarefa de programação de memória (*SiLabs CP2102 x FTDI FT232R x Prolific PL2303xx*)	
Descrição do Ficheiro SVF:	*Programação da memória FLASH do MCU ATmega16*
Total de ciclos de *TCK*:	*91313*
Bytes transferidos: (Env/Rec/Total)	*85097b / 2b / 85099b*
Baud rate:	*576000*
Tempo estimado de Transferência:	*1,48s*

Componente:	***CP2102***	***FT232R***	***PL2303xx***
Tempo decorrido:	*21,18 s*	*19,91 s*	***Falhou***[32]
Frequência média de actualização de TCK:	***8.62 KHz***	***9.17 KHz***	

Tabela 37 Comparação de interfaces UART-USB II – Verificar FLASH

Tarefa de verificação de memória (*SiLabs CP2102 x FTDI FT232R x Prolific PL2303xx*)			
Descrição do Ficheiro SVF:	*Verificação da memória FLASH do MCU ATmega16*		
Total de ciclos de *TCK*:	*74999*		
Bytes transferidos: (Env/Rec/Total)	*80301b / 11394b / 91695b*		
Baud rate:	*576000*		
Tempo estimado de Transferência:	*1,59s*		
Componente:	***CP2102***	***FT232R***	***PL2303xx***
Tempo decorrido:	*21,20s*	*3m4,52s*	***Falhou***[32]
Frequência média de actualização de TCK:	***7.07 KHz***	***812.90 Hz***	

[32] Com *baud rates* inferiores, o *PL2303* utilizado permitia efectuar as tarefas mas eram frequentemente encontrados erros nas verificações dos dados enviados.