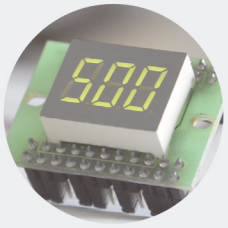
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






CONCEÇÃO DE UM TUTORIAL DE INICIAÇÃO AO CO-PROJETO DE HARDWARE/SOFTWARE

HÉBER MIGUEL DOS SANTOS
Novembro de 2015

POLITÉCNICO
DO PORTO

CONCEÇÃO DE UM TUTORIAL DE INICIAÇÃO AO CO-PROJETO DE HARDWARE/SOFTWARE

Héber Miguel dos Santos

Departamento de Engenharia Eletrotécnica

Mestrado em Engenharia Eletrotécnica e de Computadores

Área de Especialização em Planeamento Industrial

2015

Relatório elaborado para satisfação parcial dos requisitos da Unidade Curricular de Tese/Dissertação do Mestrado em Engenharia Eletrotécnica e de Computadores

Candidato: Héber Miguel dos Santos, Nº 1141313, 1141313@isep.ipp.pt
Orientação científica: Manuel Gradim de Oliveira Gericota, mgg@isep.ipp.pt
Co orientação científica: Roberto Ribeiro Neli, neli@utfpr.edu.br

Departamento de Engenharia Eletrotécnica
Mestrado em Engenharia Eletrotécnica e de Computadores
Área de Especialização em Planeamento Industrial

2015

Dedico este trabalho aos meus pais e irmão, Rute, Djalma e Jorge.

Agradecimentos

Ao meu Deus fiel e poderoso que nunca nos desampara, agradeço primeiramente a Deus e a minha família, pelo apoio e suporte em todos os momentos que precisei, desde financeiros a emocionais, pois sempre em todos os sentidos, mesmo estando longe, estiveram ao meu lado.

Agradeço a todos os meus amigos da graduação na UTFPR-CM que sempre me auxiliaram e estiveram presentes em minha vida.
Agradeço aqueles que fizeram parte dos motivos para que eu tomasse a decisão de vivenciar essa experiencia inesquecível em minha vida que foi esse intercâmbio.
Em um lugar muito especial agradeço a todos meus velhos amigos e novos que vivenciaram e compartilharam comigo esse período, aqueles que me ajudaram a realizar o sonho de viajar pela europa, aqueles que mesmo sendo de outros países se tornaram irmãos de consideração.
Agradeço em especial a uma pessoa que se tornou mais que uma amiga e me proporcionou o privilégio de poder vivenciar momentos tão incríveis em minha vida, me ajudou em de muitas formas, desde suporte de amizade, ao emocional.
Agradeço aos amigos que me ajudaram em minha dificuldade com a língua inglesa e aquele em especial que teve paciência e me ajudou grandemente na escrita dos documentos.
Agradeço aqueles que vieram juntamente comigo encarar essa aventura e se tornaram pessoas tão especiais em minha vida.
Agradeço aos meus últimos amigos que deram ajuda com moradia e bons momentos durante essa estadia.

Agradeço também as universidades UTFPR-CM e ISEP-IPP por me proporcionaram essa experiencia de vida e profissional.

Agradeço grandemente a todos os profissionais que fizeram parte do processo e tiveram paciência e sempre disponibilidade em ajudar.

Agradeço aos meus orientadores que por maior que fosse a minha pessoal dificuldade em me comunicar com eles para pedir ajuda, sempre se mostraram presente e me ajudaram em todo o processo desse trabalho.

Resumo

A crescente evolução dos dispositivos contendo circuitos integrados, em especial os FPGAs (*Field Programmable Logic Arrays*) e atualmente os *System on a chip* (SoCs) baseados em FPGAs, juntamente com a evolução das ferramentas, tem deixado um espaço entre o lançamento e a produção de materiais didáticos que auxiliem os engenheiros no Co-Projecto de *hardware/software* a partir dessas tecnologias.

Com o intuito de auxiliar na redução desse intervalo temporal, o presente trabalho apresenta o desenvolvimento de documentos (tutoriais) direcionados a duas tecnologias recentes: a ferramenta de desenvolvimento de *hardware/software* VIVADO; e o SoC Zynq-7000, Z-7010, ambos desenvolvidos pela Xilinx.

Os documentos produzidos são baseados num projeto básico totalmente implementado em lógica programável e do mesmo projeto implementado através do processador programável embarcado, para que seja possível avaliar o fluxo de projeto da ferramenta para um projeto totalmente implementado em *hardware* e o fluxo de projeto para o mesmo projeto implementado numa estrutura de *harware/software.*

Palavras-Chave

FPGA, SoC, ARM, Co-Projecto *hardware/software*, Tutoriais, VIVADO, ZYBO *board*, Zynq-7000.

Abstract

The increasing development of devices using integrated circuits, particularly FPGAs (Field Programmable Logic Arrays) and more recently System on a Chip (SoCs) based on FPGAs, along with the development of tools, has left a gap between its release and the production of instructional materials to assist engineers in the Co-Project of hardware/software based on these technologies.

Aiming to reduce this gap, this report presents the development of a set of documents (tutorials) directed to two recent technology and development tools: the VIVADO Design Suite; and the SoC Zynq-7000, Z-7010, both developed by Xilinx.

These documents are based on a basic structure that allows a simple design example to be completely implemented not only in programmable logic but also using the embedded programmable processor, in order to be possible to evaluate and compare the design flow of a project fully implemented in hardware and the design flow of the same project implemented using a hardware/software approach.

Keywords

FPGA, SoC, ARM, Co-Project *hardware/software*, Tutorials, VIVADO, ZYBO *board*, Zynq-7000.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

AMBA - *Advanced Microcontroller Bus Architecture*

AP SoC - *All Programmable System-On-Chip*

APS - *Application Programming Software*

APU - *Application Processing Unit*

ARM - *Advanced RISC Machines*

ASIC - *Application Specific Integrated Circuit*

AXI - *Advanced Extensible Interface*

CISC - *Complex Instruction Set Computer*

COTS - *Components Off-The-Shelf*

CPLD - *Complex Programmable Logic Devices*

CPU - *Central Processing Units*

DSP - *Digital Signal Processors*

EDA - *Electronic Design Automation*

EMIO - *MIO* Estendida

EPROM - *Erasable Programmable Read-Only Memory*

ESL - *Electronic System-Level*

FET - *Field-Effect Transistor*

FPGA - *Field-Programmable Gate Arrays*

HDL - *Hardware Description Language*

HS - *Horizontal Sync*

IDE - *Integrated Design Environment*

IEEE - *Institute of Electrical and Electronic Engineers*

IP - *Intellectual Property*

LCA - *Logic Cell Array*

MIO - *I/O Multiplexados*

MOSFET - *Metal–Oxide–Semiconductor Field-Effect Transistor*

MPGA - *Mask Programmable Gate Arrays*

MPSoC - *MultiProcessadores SoC*

NOC - *Network-On-Chip*

PAL - *Programmable Array Logic*

PL - *Programmable Logic*

PLA - *Programmable Logic Array*

PLI - *Programmable Language Interface*

PROM - *Programmable Read-Only Memory*

PS - *Processing System*

RISC - *Reduced Instruction Set Computer*

RTL - *Register Transfer Level*

SDF - *Standard Delay Format*

SDK - *Software Development Kit*

SoC - *System-On-Chip*

SPLD - *Simple Programmable Logic Device*

SRAM - *Static Random-Access Memory*

TCPA - *Tightly Coupled Processor Arrays*

TTL - *Transistor–Transistor Logic*

UV-EPROM - *Ultra-Violet Erasable Programmable Read Only Memory*

VGA - *Video Graphics Array*

VHDL - *VHSIC HDL*

VHSIC - *Very High Speed Integrated Circuit*

VITAL - *Initiative Toward ASIC Libraries*

VS - *Vertical Sync*

1. INTRODUÇÃO

O Co-Projeto de Hardware/Software é um conceito que apesar de definido há mais de 20 anos e da sua pertinência em termos da futura evolução dos sistemas de processamento de dados e sistemas embarcados, tem tido dificuldades em se afirmar devido à falta de apoio ao processo de aprendizagem de novas ferramentas de projeto.

O surgimento recente da ferramenta de desenvolvimento o VIVADO [1], e um novo circuito, o Zynq-7000 [2], que associa lógica programável a dois núcleos de processamento ARM Cortex-A9 [3], ambos da Xilinx, constituem mais um passo na direção da generalização do Co-Projeto de Hardware/Software.

Com a lei de Moore se cumprindo [4], de que a cada 18 meses o número de transístores dos circuitos integrados dobraria, é visível o crescimento da complexidade dos sistemas. Cada vez mais se torna mais complicado acompanhar o conhecimento de todos os avanços dessa área tecnológica.

1.1. OBJETIVOS DO TRABALHO

Com a intenção de auxiliar essa grande dificuldade do mercado de desenvolvimento de circuitos programáveis em *Hardware* e *Software* em acompanhar a elaboração de materiais didáticos que auxiliem na iniciação da aprendizagem das novas tecnologias, este trabalho

teve como objetivo a conceção de um tutorial com a descrição passo-a-passo de um exemplo que permitisse a um novo utilizador o conhecimento da ferramenta VIVADO e de um kit de desenvolvimento baseado no novo dispositivo Zynq-7000, neste caso o kit ZYBO *board* integrado com o Zynq-7000, Z-7010, produzido pela Digilent [5], de forma a que o tutorial sirva de referência ao desenvolvimento de projetos futuros.

O tutorial divide-se em duas partes:

- Implementação utilizando exclusivamente a parte da lógica programável;
- Implementação com a utilização da parte da lógica programável e da parte de processamento, cobrindo ambos os aspetos: o desenvolvimento de hardware, de software e da respetiva interface.

O tutorial abrange também algumas das interfaces periféricas do kit, com o intuito de demonstrar as funcionalidades da placa.

E para que o tutorial tenha potencial de ser utilizado como material de auxílio à iniciação da aprendizagem dessa tecnologia, podendo abranger o maior número de utilizadores, este é escrito em Inglês.

1.2. SEQUENCIAÇÃO DO DESENVOLVIMENTO DO TRABALHO

A estrutura de desenvolvimento do trabalho está apresentada no diagrama da Figura **1**.

1.3. ESTRUTURA DO RELATÓRIO

Representando todo o embasamento, conhecimento e desenvolvimento do trabalho realizado este relatório está dividido da seguinte forma:

- No primeiro Capítulo estão definidos com um breve embasamento os objetivos e a estrutura com as fases de elaboração deste trabalho.
- No segundo Capítulo para enquadramento teórico do trabalho, está apresentada uma revisão histórica dos dispositivos FPGAs, juntamente com a integração da apresentação do atual conceito da tecnologia fundamentado nos dispositivos SoCs, e por fim um breve histórico da evolução das ferramentas de desenvolvimento.

Figura 1 – Estrutura do desenvolvimento do trabalho.

- O terceiro Capítulo descreve a crescente e complexa evolução do Co-Projecto de Hardware/Software, demonstrando a grande dificuldade presente na aprendizagem das novas ferramentas e dispositivos realizando uma comparação pedagógica e industrial.

- O quarto Capítulo apresenta uma descrição da placa de desenvolvimento utilizada (ZYBO board) e do Software de desenvolvimento utilizado (VIVADO), dando um foco para o processador Zynq-7000, o protocolo de comunicação AXI e a interface VGA, presentes no kit ZYBO.

- No quinto Capítulo são apresentados os documentos (tutoriais) produzidos, abordando seus objetivos, conteúdo e objetivo pedagógico apresentado, uma explicação do conteúdo e do fluxo de projeto utilizado nos documentos.

- O sexto Capítulo contém as conclusões que foram retiradas da realização do trabalho, bem como algumas perspetivas para trabalhos futuros.

- No sétimo Capítulo estão listadas as referências documentais que serviram de base para a conceção deste trabalho.

2. ENQUADRAMENTO DO TRABALHO

Este capítulo apresenta uma revisão histórica dos dispositivos FPGAs desde a origem, tecnologias empregadas e características dos dispositivos, segue para um enquadramento atual das tecnologias, contextualiza os FPGAs e dentro dos novos desenvolvimentos os dispositivos SoCs e finaliza com uma revisão da evolução das ferramentas de desenvolvimento para esses dispositivos.

2.1. HISTÓRICO DOS *FPGAS*

A tecnologia atual denominada *Field-Programmable Gate Arrays* (*FPGAs*) ou em português Arranjo de Portas Programável em Campo, que descreve a principal tecnologia em *hardware* utilizada neste trabalho, teve seus primeiros vislumbres em 1959, quando Dawon Kahng e Martin M. (John) Atalla [6], pela empresa Bell Labs inventou o *metal–oxide–semiconductor field-effect transistor* (*MOSFET*) como uma continuação à patente já registrada do *field-effect transistor* (FET). Partindo desse invento foi então verificado os primeiros passos rumo à tecnologia de base, caracterizando a menor divisão em componente utilizado no desenvolvimento dos FPGAs.

No mesmo período da invenção do *MOSFET*, muitas outras pesquisas estavam em desenvolvimento baseadas em outra tecnologia já em aprimoramento na época, os transístores de junção bipolar, um exemplo é o primeiro Circuito Integrado (CI) baseado em transístores de junção bipolar, desenvolvido em germânio por Jack Kilby pela *Texas Instruments* [7], e o desenvolvido em silício por Robert Noyce [8] onde foram realizadas algumas correções no projeto.

Com o molde dessas ideias as tecnologias evoluíram até os primeiros circuitos integrados lógicos que primeiramente seguiram a linha dos transístores de junção bipolar, sendo desenvolvidos em 1962 os primeiros CIs *Transistor–transistor logic (TTL)*, tendo uma crescente através da série 74 [9] [10].

Após a estruturação dessas invenções, dando inicio ao desenvolvimento industrial de componentes digitais, a implementação dos dispositivos ainda se baseava em dois métodos descritos por: utilização de componentes em catálogos já pré projetados com funções lógicas genéricas, com um custo considerado baixo, os *Components Off-The-Shelf (COTS)*; ou a conceção de dispositivos dedicados, o que agregava um alto custo e tempo de desenvolvimento e acabava por ser empregado apenas em dispositivos que permitiriam uma altíssima demanda, nomeados *Application Specific Integrated Circuit (ASIC)* [11].

Então a partir dessas tecnologias em 1969 se dá o desenvolvimento das *Programmable read-only memory (PROM)*, em português, memórias programáveis uma única vez, de leitura, sendo de fato, os primeiros dispositivos nos quais era permitido ao utilizador a implementação de uma função *Booleana* especifica limitada [11] [12]. Na estrutura interna das *PROMs*, como pode ser visto na Figura 2 de [11] e como é aí descrito, as linhas de endereços eram usadas como linhas de entrada do circuito lógico, enquanto as linhas de dados funcionavam como saídas desse mesmo circuito. A estrutura das *PROMs* são baseadas em uma matriz fixa de funções lógicas ‘E’ e uma matriz programável de função lógica ‘OU’. Dessa forma, considerando um sistema lógico descrito por uma tabela de verdade com ‘n’ entradas, o descodificador adaptado pela memória possibilita todos os 2^n termos de produto à saída (no esquema representado pela Figura 2 as ‘n’ entradas e linhas de conexões entre os dispositivos lógicos são representados por uma linha para simplificar o esquema). Por conta desta característica, as PROMs foram muito usadas para implementação de conversores de código, geradores de caracteres e tabelas de armazenamento de dados, aplicações onde a funcionalidade é usualmente descrita através

de tabelas de verdade, e não por equações Booleanas. A funcionalidade descrita é realizada com a configuração das ligações programáveis da matriz ‘OU’ programável que eram realizadas por fusíveis em cada interceção.

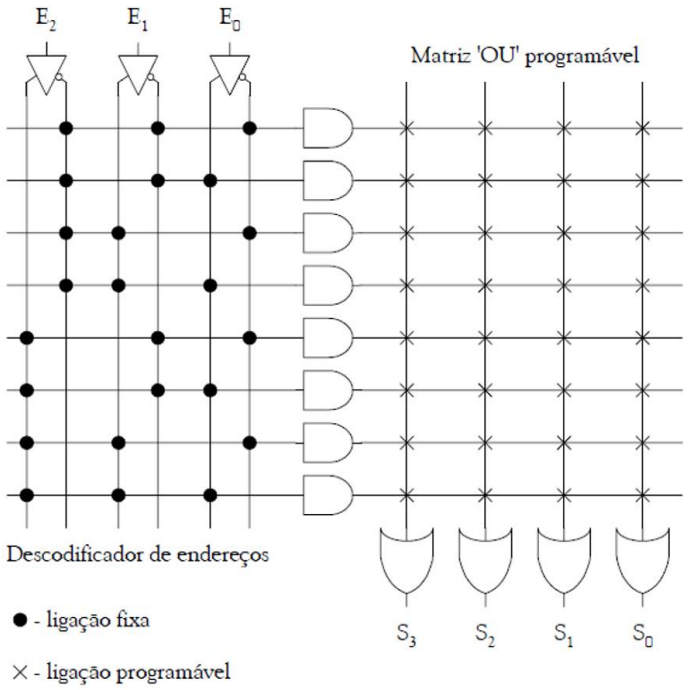


Figura 2 – Arquitetura interna de uma PROM. [11]

A estrutura das PROMs mostrada foi útil para vários sistemas como relatado. Porém, sistemas digitais normalmente trabalham com variáveis adicionais o que torna esta tecnologia ineficiente para implementação de circuitos lógicos.

Então em 1972 foi concebido o primeiro dispositivo específico para a implementação de circuitos lógicos pela Philips denominado *Programmable Logic Array (PLA)* ou do português, matriz de lógica programável. Sua estrutura é apresentada na Figura 3 [11]. A sua diferença em relação às PROMs é possuir as duas matrizes lógicas programáveis. Com o *PLA* é possível configurar todos os tipos de funções lógicas desde que não exceda o número de produtos disponíveis. Esse dispositivo não obteve um sucesso elevado, pois sua estrutura era lenta e o processo de programação era baseado no mapa dos fusíveis dificultando o projeto, pois não se assemelhava com os circuitos ou funções *Booleanas* [11] [13] [14].

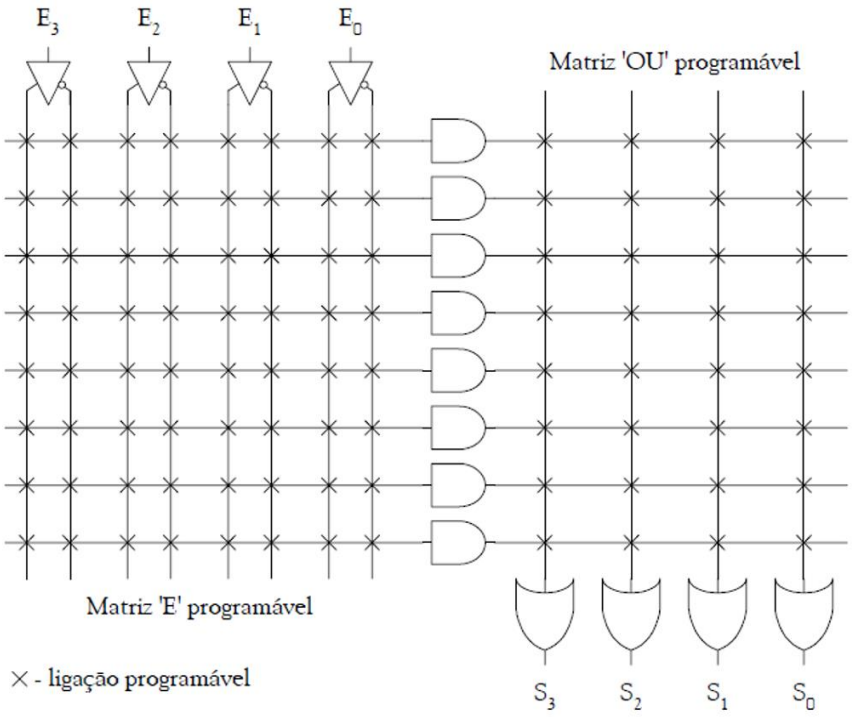


Figura 3 - Arquitectura interna de uma PLA. [11]

Para corrigir os problemas encontrados com os PLAs foi então desenvolvido uma arquitetura de grande importância e evolução do contexto, os chamados *Programmable Array Logic* (PAL), neste caso em português, lógica de matriz programável. Sua diferença, como mostra a Figura 4 [11], é que apenas a matriz de função lógica 'E' é programável sendo que a matriz 'OU' é fixa. Essa alteração traz novamente inúmeras possibilidades, porém reduziu-se a complexidade do dispositivo e ouve um aumento da velocidade. Também algumas versões das PALs foram produzidas com a integração de blocos flip-flop nas saídas da matriz 'OU', auxiliando assim o desenvolvimento de projetos sequenciais [11] [14].

Nos dispositivos descritos as ligações programadas possuíam a característica de permitir apenas uma gravação no dispositivo. Em 1979 a evolução continua com a criação de uma nova tecnologia descrita por *Ultra-Violet Erasable Programmable Read Only Memory* (UV-EPROM), do português, memória de leitura programável e apagável por ultra violeta, lançada pela AMD. A estrutura desse dispositivo chamado PAL 22V10, como mostra a [11] tem como principal diferença a arquitetura do conjunto de recursos disponíveis em cada uma das saídas, que constituem um bloco nomeado macrocélula. A macrocélula permite a programação da polaridade do sinal de saída, a colocação da saída em estado de

alta impedância (um terceiro estado) e a inclusão de um flip-flop na saída, a qual pode igualmente ser realimentada, reaparecendo como entrada da matriz ‘E’ [11].

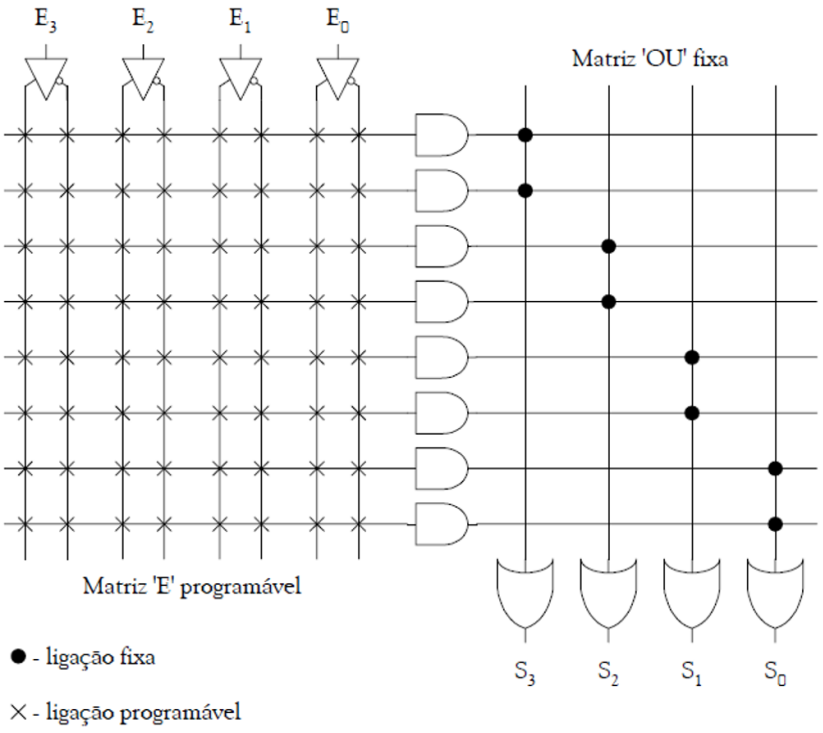


Figura 4 – Arquitectura interna de uma PAL. [11]

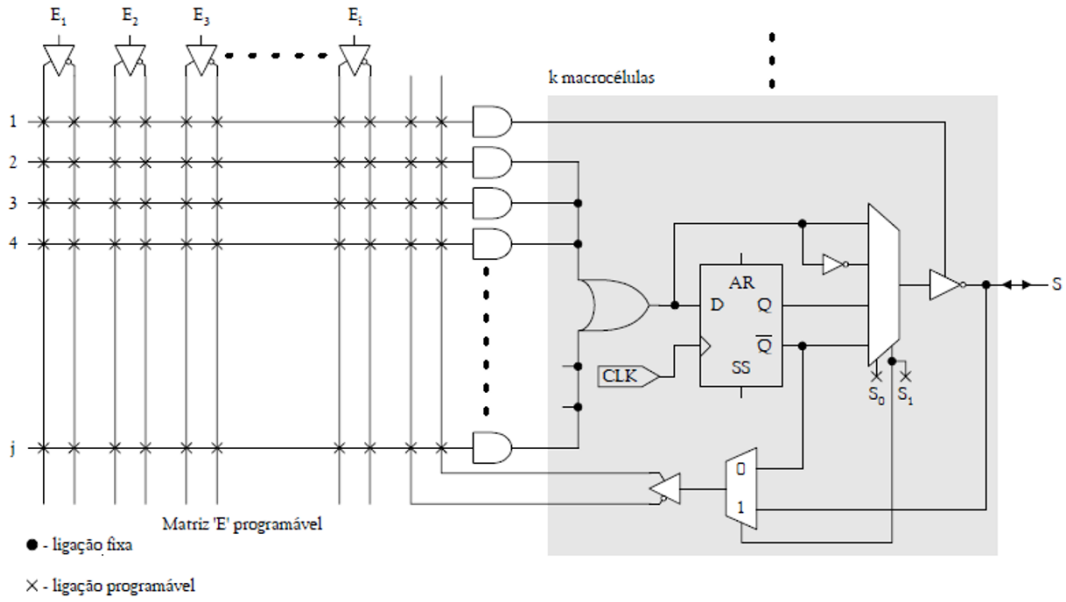


Figura 5 – Arquitectura interna de uma PAL 22V10. [11]

Após essas ultimas considerações no desenvolvimento das PLAs e PALs, suas características, elevadas velocidades e baixo custo, e sobretudo o fato de possuírem algumas centenas de portas lógicas, levaram a que no seu conjunto fossem nomeadas de

Simple Programable Logic Devices (SPLDs), em português, dispositivos de lógica programável simples. Ao tempo, eram a primeira opção na implementação de equações lógicas *Booleanas* com a estrutura de soma de produtos. Com o passar do tempo novas funcionalidades foram acrescidas como pinos bidirecionais programáveis e configuração flexível dos registros e dos sinais de relógio. Dentre outras vantagens, são destacadas o baixo consumo, desenvolvimento rápido e alta fiabilidade [11] [15].

Em paralelo com esse período surgiram outras tecnologias focadas ao desenvolvimento de memórias e que assim foram adaptadas ao conceito das estruturas SPLD. Eram esses os conceitos de memórias *Static Random-Access Memory* (SRAM), e *Erasable Programmable Read-Only Memory* (EPROM).

Considerando então a chegada ao limite de expansão alcançado pelas SPLDs devido à grande complexidade da arquitetura, onde o crescimento das matrizes programáveis impediam o aumento do número de entradas, elementos e funções, o desenvolvimento seguinte foi baseado na ideia de arquiteturas baseadas em múltiplos SPLDs. Surgiram assim os dispositivos nomeados *Complex Programable Logic Devices* (CPLDs), em português, dispositivos lógicos programáveis complexos, que foram introduzidos no mercado pela ALTERA em 1984, com as famílias EP310 [16], e em 1988 com as famílias MAX 5000, 7000 e 9000 [11] [17].

A grande demanda dos novos dispositivos fez com que os fabricantes melhorassem cada vez mais todas as funcionalidades e acrescentassem novas, porém mantendo a mesma base estrutural.

Então ainda com o objetivo de aumentar ainda mais a densidade de dispositivos lógicos, funcionalidades e flexibilidade de projeto, características que a estrutura utilizada até então já não suportava, foi em 1985 lançado pela XILINX uma nova arquitetura, totalmente diferenciada da anterior, capaz de replicar a densidade lógica, acrescentando-lhes flexibilidade, das denominadas *Mask Programmable Gate Arrays* (MPGAs), do português, matriz de portas com máscaras programáveis, e assim reduzir o tempo de projeto para dispositivos ASIC. Foi então desenvolvida a nomeada *Logic Cell Array* (LCA), ou do português, matriz de células lógicas, sendo esse o protótipo para o que definiu por *Field Programmable Gate Array* (FPGA).

Como a Figura 6 [18] nos mostra a estrutura era formada por uma matriz de blocos lógicos independentes com opções de programação e com várias estruturas de Entrada e Saída (E/S) também independentes [11] [14].

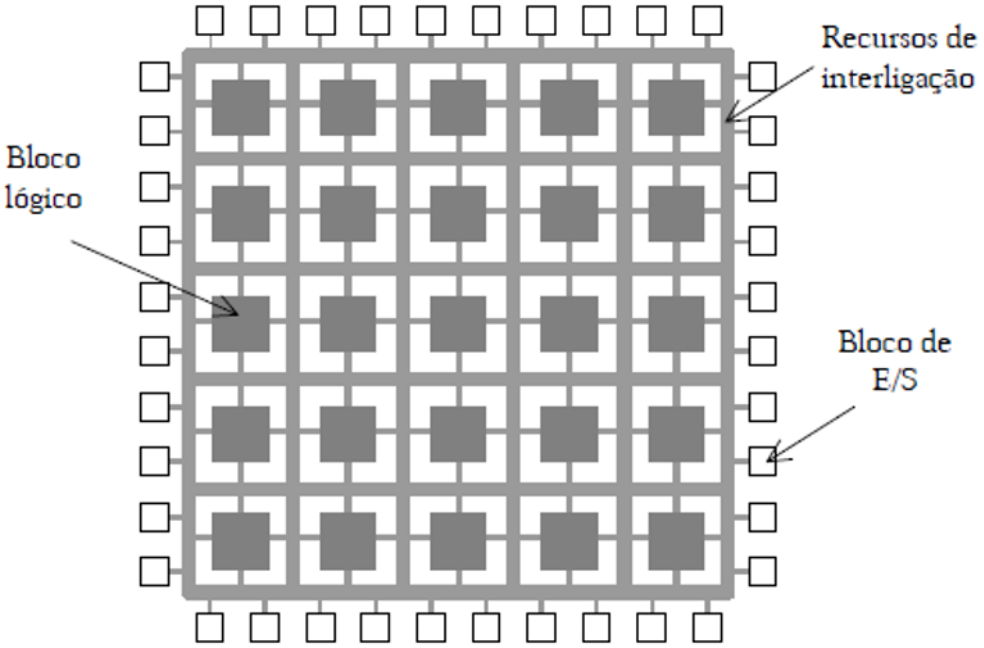


Figura 6 – Arquitetura interna de uma FPGA. [18]

Desde então inúmeras variações são desenvolvidas, como a criação de novos blocos lógicos já dedicados a tarefas específicas até à utilização de novas tecnologias de construção dos dispositivos em tamanhos cada vez mais reduzidos, chegando a escalas nanométricas. A maioria dos avanços e variações podem ser verificados com o acompanhamento dos lançamentos dos desenvolvedores dominantes do mercado de FPGAs: a ALTERA e a XILINX, podendo ser verificado em [19] e [20] respetivamente.

Atualmente os FPGAs estão em uma escala de produção onde os dispositivos de topo ultrapassam a quantidade de 5 milhões de blocos lógicos. As estruturas utilizadas nos dispositivos são caracterizadas por trabalharem em 3 dimensões nomeadamente 3D CIs ou 3D Tri-Gate Transístor. As tecnologias de produção variam a litografia dos CIs (menor escala dos processos de construção dos CIs) entre 14nm e 16nm (nanómetros) [21] [22], a mesma tecnologia utilizada nos processadores de computadores da 6ª geração da Intel [23].

Para que se possa compreender a diversificação e capacidade dos FPGAs existentes atualmente a Tabela 1 [24] apresenta um resumo com os valores médios de dispositivos e capacidades de cada umas das principais famílias de FPGAs da Xilinx.

Tabela 1 – Resumo com valores médios das famílias líderes de FPGAs da Xilinx. [24]

Features	Artix™-7	Kintex™-7	Virtex®-7	Spartan®-6	Virtex-6
Logic Cells	215,000	480,000	2,000,000	150,000	760,000
BlockRAM	13Mb	34Mb	68Mb	4.8Mb	38Mb
DSP Slices	740	1,920	3,600	180	2,016
DSP Performance (symmetric FIR)	930GMACS	2,845GMACS	5,335GMACS	140GMACS	2,419GMACS
Transceiver Count	16	32	96	8	72
Transceiver Speed	6.6Gb/s	12.5Gb/s	28.05Gb/s	3.2Gb/s	11.18Gb/s
Total Transceiver Bandwidth (full duplex)	211Gb/s	800Gb/s	2,784Gb/s	50Gb/s	536Gb/s
Memory Interface (DDR3)	1,066Mb/s	1,866Mb/s	1,866Mb/s	800Mb/s	1,066Mb/s
PCI Express® Interface	x4 Gen2	Gen2x8	Gen3x8	Gen1x1	Gen2x8
Analog Mixed Signal (AMS)/XADC	*Yes*	*Yes*	*Yes*	-	*Yes*
Configuration AES	*Yes*	*Yes*	*Yes*	*Yes*	*Yes*
I/O Pins	500	500	1,200	576	1,200
I/O Voltage	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.35V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V, 3.3V	1.2V, 1.5V, 1.8V, 2.5V
EasyPath™ Cost Reduction Solution	-	*Yes*	*Yes*	-	*Yes*

Outro conceito que atualmente está levando a tecnologia dos FPGAs a outro patamar e trazendo uma visão de projeto diferenciada são os nomeados *System-On-Chip* (SoC) do português sistemas em chip. Seus conceitos e características serão abordados na seção 2.2.

2.2. ATUALIDADE, CONCEITO E INTEGRAÇÃO *SOCS E FPGAS*

Os SoC possuem uma estrutura base que "lembra" a arquitetura dos microcontroladores atuais, pois são caracterizados por dispositivos que comportam os componentes de um sistema completo em um único chip e diferenciado por ter como característica a presença de um ou mais processadores, podendo esse ser, um computador ou apenas a junção de sistemas de comunicação com *Digital Signal Processors (DSPs)* ou coprocessadores, por exemplo [25]. Sua definição é utilizada para nomear dispositivos como o conhecido sistema Snapdragon (SoC), muito utilizado em celulares, que foi anunciado em novembro de 2006 e incluiu um processador Scorpion, bem como outros blocos pré-projetados no dispositivo para aplicações específicas [26].

A ideia dos SoC se torna clara com a imagem relatada na Figura 7.

O cruzamento dos SoCs e das FPGAs é verificado devido à grande evolução dos FPGAs como citado ao final da seção 2.1. Com a conceção de FPGAs com milhões de dispositivos lógicos, milhares de somadores, multiplicadores, funções de DSP, entre outros, as aplicações também evoluíram e cada vez mais, mais dispositivos eram agregados aos FPGAs. Várias aplicações foram desenvolvidas, através das ferramentas de desenvolvimento para FPGAs (que serão abordadas na seção 2.3), genericamente para serem implementadas através das unidades lógicas programáveis de FPGAs diversos. São esses os chamados *Intellectual properties* (IPs) [27] do português propriedade intelectual, caracterizados por dispositivos descritos por HDLs mascarados por uma interface de conexão com outros IPs integrados nos projetos de FPGAs. Devido à grande capacidade disponível nos FPGAs, passou-se a implementar também IPs de processadores utilizando os recursos disponíveis e as unidades de lógica programável dos FPGAs.

A implementação de processadores tinha como objetivo a junção de elementos programáveis em *software*, além do *hardware* programável contido nos FPGAs. Uma das alternativas encontradas com a crescente capacidade dos FPGAs foi a emulação de processadores a partir do próprio *hardware*. Como exemplo temos o *MicroBlaze*, o processador da XILINX disponível em todas as suas ferramentas de desenvolvimento para FPGAs [28]. Tal "adaptação" era muito requisitada para auxiliar na resolução de problemas que necessitavam de processamentos em *software*, em muitos casos em projetos que necessitavam de processamento e interfaces exteriores. Porém a estrutura de processador "emulada" implementada por blocos lógicos não possui a mesma eficiência encontrada nos cores de processadores atualmente desenvolvidos. Por isso, assim como os demais dispositivos alocados internamente nos FPGAs, os processadores passaram a ser acoplados nos dispositivos FPGAs, sendo inicialmente nomeados apenas como FPGAs com processadores integrados, pois as estruturas de processadores eram consideradas "básicas".

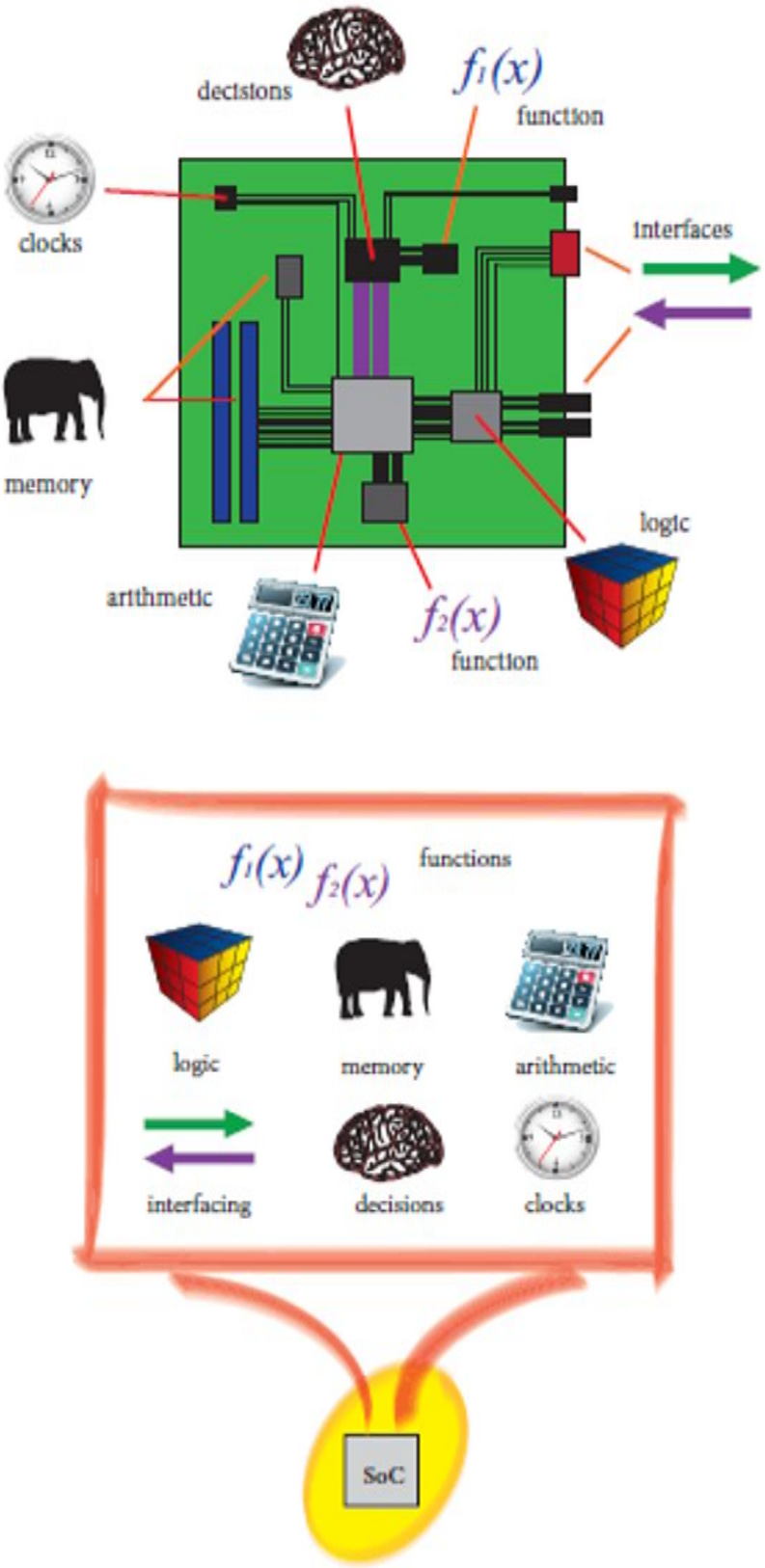


Figura 7 – Comparação entre um sistema em uma placa de circuito impresso (em cima) e um Sistema em Chip (em baixo). [29]

Atualmente a arquitetura de processadores mais utilizada nos SoCs com FPGAs é a arquitetura *Reduced Instruction Set Computer* (RISC), ou do português, computador com um conjunto reduzido de instruções. A arquitetura RISC foi desenvolvida como um resultado do projeto 801, que começou em 1975 na IBM TJWatson Centro de Pesquisa, e foi concluída no início dos anos 80 [30]. O trabalho que cada instrução da máquina RISC executa é simples e direto. Assim, o tempo necessário para executar cada instrução pode ser reduzido e o número de ciclos também reduzido. Tipicamente, o tempo de execução de instruções está dividido em cinco fases, ciclos de máquina, e assim que o processamento de uma fase está terminado, a máquina prossegue com a execução da segunda fase. No entanto, quando os recursos usados pela primeira fase ficam livres são utilizados para executar a mesma operação mas da próxima instrução. A operação das instruções é realizada de forma encadeada, semelhante à linha de montagem no processo de fábrica - *pipelining*.

Os dois maiores fabricantes de FPGAs, ALTERA e XILINX, utilizam em seus dispositivos SoC os processadores *Advanced RISC Machines* (ARM) da companhia ARM *Holdings*. A arquitetura ARM é semelhante ao RISC pois incorpora as características típicas da arquitetura RISC como:

- Uma arquitetura uniforme de arquivo e carga de registros, onde o processamento de dados é operado apenas no conteúdo dos registros e não diretamente sobre conteúdos da memória;

- Modos de endereçamento simples, com todos os endereços de carga e armazenamento determinados somente a partir do conteúdo dos registros e dos campos das instruções.

Melhorias na arquitetura básica RISC permite aos processadores ARM alcançar um bom equilíbrio de alta performance, pequeno tamanho do código, baixo consumo de energia e área de silício pequena [31].

A abordagem dos processadores ARM com a arquitetura RISC reduz significativamente os transístores se comparado com a típica arquitetura *Complex Instruction Set Computer* (CISC), ou do português, computador com conjunto de instruções complexas, utilizada nos processadores de 32 bits da maioria dos computadores pessoais. Essa abordagem reduz o custo, aquecimento e consumo de energia, o que se torna o ideal para dispositivos

alimentados por baterias como computadores portáteis, *smartphones*, *laptops*, *tablets* e outros sistemas embarcados. Com um projeto mais simples torna-se mais fácil a construção de unidades mais eficientes, com vários núcleos de processamento, *Central Processing Units* (CPUs), a um custo menor, proporcionando maior eficiência energética para servidores.

Os processadores ARM são divididos em 3 famílias de processadores consideradas clássicas, nomeadas, ARM7, ARM9 e ARM11, muito utilizadas até hoje, e outras 4 séries nomeadas Cortex-A, Cortex-R, Cortex-M e SecurCore que possuem características específicas de cada série como, respetivamente, processadores de alto desempenho para sistemas operacionais, alta performance para aplicações em tempo real, solução de baixo custo para microcontroladores embarcados e processadores para alta segurança.

O atual mercado de SoCs com FPGAs continua sobre domínio das duas principais empresas de FPGAs, a XILINX e a ALTERA, este última recentemente adquirida pela Intel. O mercado conta com outros fabricantes, detentores porém de uma percentagem reduzida do mercado. Para uma breve comparação das tecnologias desenvolvidas atualmente a Tabela 2 mostra a abrangência tecnológica dos principais dispositivos SoCs das duas marcas.

Tabela 2 – Principais características dos SoCs da ALTERA e XILINX [32]**.**

	Altera SoC FPGAs	**Xilinx Zynq-7000 EPP**
Processor	*ARM Cortex-A9*	*ARM Cortex-A9*
Processor Class	*Application processor*	*Application processor*
Single or Dual Core	*Single or Dual*	*Dual*
Processor Max. Frequency	*1,05 GHz*	*1,0 GHz*
L1 Cache	*Data: 32 KB* *Instruction: 32 KB*	*Data: 32 KB* *Instruction: 32 KB*
L2 Cache	*Unified: 512 KB, with error correction code (ECC)*	*Unified: 512 KB*
Memory Management Unit (MMU)	*Yes*	*Yes*
Floating-Point Unit/NEON™ Multimedia Engine	*Yes*	*Yes*
Acceleration Coherency Port (ACP)	*Yes*	*Yes*
Interrupt Controller	*Generic (GIC)*	*Generic (GIC)*
On-Chip Processor RAM	*64 KB, with ECC*	*256 KB, no ECC*
Direct Memory Access Controller	*8-channel ARM DMA330* *32 peripheral requests (FPGA + hard processor system)*	*8-channel ARM DMA330* *4 peripheral requests (FPGA only)*
External Memory Controller	*Yes*	*Yes*

Memory Types Supported	*LPDDR2, DDR2, DDR3L, DDR3*	*LPDDR2, DDR2, DDR3L, DDR3*
External Memory ECC	*16 bit, 32 bit*	*16 bit*
External Memory Bus Max. Frequency	*400 MHz (Cyclone® V SoC), 533 MHz (Arria® V SoC)*	*533 MHz*
Processor Peripherals	*1x quad SPI controller with 4 chip selects* *1x NAND controller (single- and multilevel cell - MLC or SLC)* *2x 10/100/1G Ethernet controller* *2x USB 2.0 On-the-Go (OTG) controller* *1x SD/MMC/SDIO controller* *2x UART* *4x I^2C controller* *2x CAN controller* *2x SPI master, 2x SPI slave controller* *4x 32 bit general-purpose timers* *2x 32 bit watchdog timers*	*1x quad SPI or dual quad SPI controller with 2 chip selects* *1x static memory controller (NAND-SLC, NOR, or SSRAM)* *2x 10/100/1G Ethernet controller* *2x USB 2.0 OTG controller* *2x SD/SDIO controller* *2x UART* *2x I2C controller* *2x CAN controller* *2x SPI controllers (master or slave)* *2x 16 bit triple mode timer/counters* *1x 24 bit watchdog timer*
FPGA Fabric	*Cyclone V, Arria V*	*Artix-7, Kintex-7*
FPGA Logic Density Range	*25 K to 462 K LE*	*28 K to 444 K LC*
Hardened Memory Controllers in FPGA	*Up to 3, with ECC*	*Not available*
High-speed Transceivers	*Available at all densities*	*Higher-density devices only*
Analog Mixed Signal (AMS)	*Not available*	*2 x 12-bit, 1 MSPS analog-to- digital converters (ADCs)*
Boot Sequence	*Processor first, FPGA first, or both simultaneous*	*Processor first*

2.3. EVOLUÇÃO DAS FERRAMENTAS DE DESENVOLVIMENTO DE HARDWARE

Seguindo a linha evolutiva de pensamento já apresentada nas seções anteriores deste capítulo, que abrange a pesquisa e desenvolvimento dos dispositivos lógicos programáveis até à integração com a tecnologia de *software* programável (processadores integrados) presente nos dispositivos SoCs, nesta seção será abordado a evolução das ferramentas de desenvolvimento dos projetos/*designs* para tais dispositivos.

Os estudos evoluíram a partir de organizações e conferências pelo mundo. Nas décadas de 70 e 80, alguns anos antes do estudo ser reconhecido e levado como tema de pesquisa, o desenvolvimento dos primeiros dispositivos *Application-Specific Integrated Circuit* (ASIC), do português, circuitos integrados de aplicação específica, os primeiros ASICs

foram desenvolvidos ao mais baixo nível de projeto possível, a partir dos chamados diagramas esquemáticos de transístores, período esse que iniciou o aparecimento das pesquisas por aplicações denominadas *Electronic Design Automation* (EDA), do português, automação de projetos eletrônicos ao nível estrutural.

A crescente quantidade de transístores e complexidade dos sistemas que eram desenvolvidos dificultava cada vez mais o desenvolvimento dos projetos, elevando a quantidade de erros e consequentemente a necessidade de mais análises durante o seu desenvolvimento. Dessa forma, as aplicações EDA passaram a tomar um nível evolutivo constante, visto que a capacidade de integração de dispositivos (transístores, entre outros circuitos), cada vez mais em escalas menores, crescia exponencialmente como pode ser verificado pelo aumento da densidade lógica dos CIs citados na seção anterior. Na Figura 8 pode-se verificar a sequência evolutiva das aplicações EDA que se iniciou com os diagramas esquemáticos de transístores, seguidos pelo *design* em nível de portas lógicas (*gates*) e, posteriormente, pelo *design* em nível de registradores *Register Transfer Level* (RTL), onde foram introduzidas as *Hardware Description Language* (HDL), do português, linguagem de descrição de *hardware*, chegando à atualidade e sendo cada vez mais utilizadas na elaboração de projetos em nível de sistema *Electronic System-Level* (ESL).

As ferramentas tiveram uma grande mudança na forma de descrição e fluxo de projeto, resultado da crescente complexidade dos sistemas, sendo que várias camadas de abstração foram acrescentadas de forma a automatizar a sintetização dos sistemas partindo das linguagens de alto nível para a configuração abstrata do *hardware.* Após a integração da síntese dos sistemas a partir do nível RTL, várias camadas foram acrescentadas ao fluxo de projetos, como, testes, análises, simulações e integração de ferramentas de descrição de alto nível, como mostra a Figura 9.

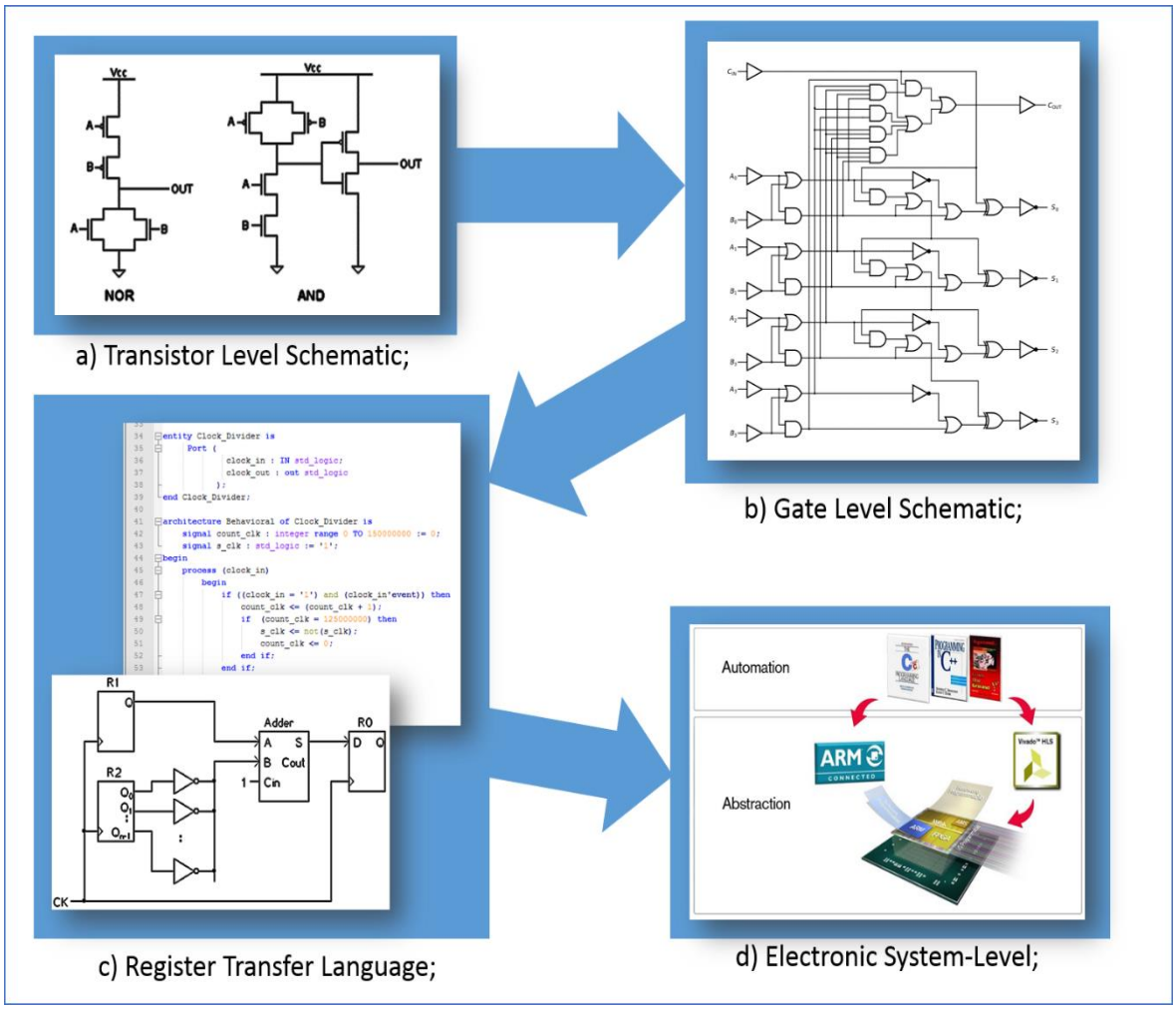


Figura 8 - Diagramas em nível de: a) Transístor; b) Portas lógicas; c) RTL; d) ESL

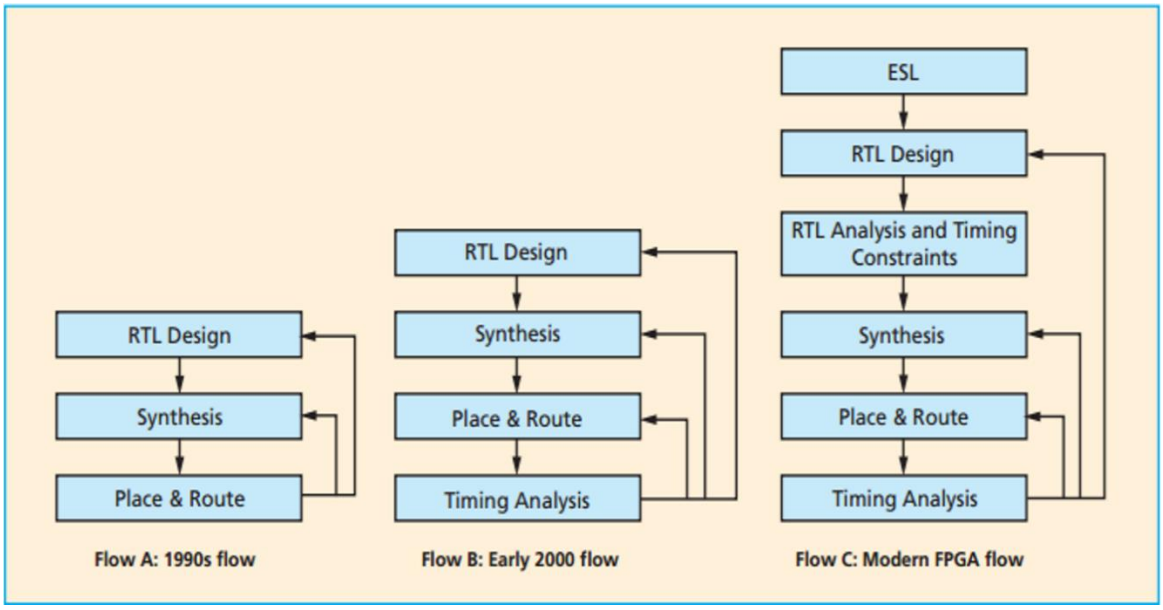


Figura 9 – FPGA, evolução do fluxo de projeto [33]**.**

2.3.1. LINGUAGENS DE DESCRIÇÃO

Com o aumento do tamanho e da complexidade dos projetos, quando desenvolvido ao nível de portas lógicas a probabilidade da existência de erros de projeto é grande, além de demandar muito tempo de desenvolvimento. Por estas razões, alguns vendedores de ferramentas EDA desenvolveram ferramentas de projeto com base na utilização de linguagens de descrição de *hardware*, HDL.

A ideia das linguagens de descrição de *hardware* é o fato de descreverem o funcionamento e as conexões de *hardware*. Algumas características das linguagens mais utilizadas são:

- VERILOG:

  Na década de 80 foi criada uma nova HDL chamada de Verilog por Phil Moorby. Em 1985, a *Gateway Design Automation*, introduziu esta linguagem no mercado, juntamente com um simulador lógico chamado de Verilog-XL.

  Uma das características do Verilog e do Verilog-XL era o *Programmable Language Interface* (PLI), do português, interface de linguagem programável, ponto principal do Verilog. Outro nome utilizado para esta interface é *Application Programming Software* (APS), que é uma livraria de funções que permitem que *software* externo introduza dados numa aplicação e aceda a dados dessa mesma aplicação. Por isso, o PLI Verilog é uma API que permite estender as funcionalidades do Verilog e do simulador.

  Outra característica de grande relevância associada ao Verilog e Verilog-XL é a capacidade de ter a informação de tempos de atraso especificada num ficheiro de texto externo, denominado de *Standard Delay Format* (SDF). Isto permitia que ferramentas como *post-place-and route timming analysis packages* gerassem o ficheiro SDF que poderia ser usado pelo simulador para fornecer respostas mais precisas.

  O Verilog se manteve uma HDL proprietária até 1990, quando a *Cadence Design Systems*, então a proprietária dos direitos, por conta da popularização da linguagem concorrente, arriscou colocando a linguagem em domínio publico.

  A popularidade do Verilog a tornou uma das principais linguagens de HDL. Em 1993, o *Institute of Electrical and Electronic Engineers* (IEEE) criou um comité para fazer com que o Verilog fosse estabelecido como uma norma IEEE.

Em 1995 foi lançada oficialmente a norma IEEE do Verilog, que foi oficialmente chamada de IEEE 1364-1995.

Nos próximos anos a linguagem teve alguns aprimoramentos. Em 2001 e em 2005 foram efetuadas algumas pequenas alterações a esta norma, que passarm a ser chamadas de Verilog 2001 e Verilog 2005. Em 2009, o Verilog 2005 e um subconjunto, o SystemVerilog se fundiram no que passou a ser chamado SystemVerilog 2009 (IEEE Standard 1800-2009).

O SystemVerilog é a versão mais alto nível da linguagem, pois possui grande suporte para desenvolvimento a nível de sistema. Sua última versão é a SystemVerilog 2012 (IEEE Standard 1800-2012). [34] [35] [36]

- *Very High Speed Integrated Circuit* HDL (VHDL):

O desenvolvimento do VHDL iniciou-se em 1990, quando o Departamento de Defesa dos Estados Unidos da América criou o programa *Very High Speed Integrated Circuit* (VHSIC), cujo principal objetivo era impulsionar as pesquisas ligadas à tecnologia de circuitos integrados digitais.

O programa objetivava resolver, entre outras coisas, a dificuldade de reproduzir circuitos integrados. De forma a resolver este problema, foi desenvolvido um novo tipo de HDL chamado de VHSIC HDL (VHDL), lançado em 1981. Um ponto forte deste processo é que a indústria se envolveu desde o início do projeto. Em 1983 uma equipa composta pela *Intermetrics*, IBM e Texas *Instruments* foi contratada para desenvolver o VHDL, sendo lançada oficialmente em 1985.

Em 1986 o Departamento de Defesa dos Estados Unidos da América doou todos os direitos do VHDL ao IEEE, para auxiliar na divulgação e aceitação da linguagem pela indústria. Após algumas modificações efetuadas para corrigir alguns problemas conhecidos da linguagem, o VHDL foi lançado como uma norma oficial, o IEEE 1076, em 1987. Esta linguagem foi estendida em 1993 e novamente lançada em 1996.

O VHDL é muito abrangente, cobrindo desde o nível de abstração funcional (equações booleanas e RTL), até o nível de abstração comportamental (algoritmos), suportando também alguns *designs* a nível de sistema. Porém, o VHDL é bastante fraco no que diz respeito ao nível de abstração estrutural (a nível de transístor e de portas lógicas), com pouco versatilidade ao criar a modelação de tempos de atraso. Verificou-se que o VHDL não tinha precisão suficiente em relação à temporização, para ser usado como um simulador

ao nível do transístor e da porta lógica. Por esta razão foi lançada em 1992 na *Design Automation* Conference a VHDL *Initiative Toward* ASIC *Libraries* (VITAL). VITAL foi uma forma encontrada de melhorar as funcionalidades do VHDL com foco nas modelações de tempos em ambientes de projeto de ASIC e FPGA. O resultado final engloba uma livraria de funções primitivas e métodos associados para efetuar o *back-annotating* de informações de atrasos, modelada em livrarias, onde o mecanismo de anotação do atraso é baseado no mesmo formato usado pelo Verilog. [35] [36]

3. DIFICULDADE DE APRENDIZAGEM DAS FERRAMENTAS

Para ser compreendida a dificuldade durante o processo de aprendizagem de ferramentas *Electronic Design Automation* (EDA), neste capítulo será apresentada uma introdução da evolução da complexidade dos sistemas baseados em *Hardware/Software* e uma breve comparação entre a demanda versus a dificuldade de ensino e aprendizagem tanto educacional quanto industrial das ferramentas e dispositivos.

O sistema estabelecido e utilizado no fluxo de projeto do Co-Projeto ou *Co-Design* de *Hardware/Software* teve seu início durante a década de 90, quando ficou claro que os microprocessadores não estariam disponíveis apenas como componentes discretos para sistemas a nível de placas, mas como componentes programáveis em *software* para qualquer projeto de Circuitos Integrados (CIs).

Embora o Co-Projeto de hardware e software já tenha sido exercido desde a introdução do primeiro microprocessador pela engenharia, a definição do Co-Projeto de *Hardware/Software* começou a ser considerado como o processo de criação simultânea e

coordenada de um sistema de *hardware* eletrónico e componentes de *software*, com base na descrição da funcionalidade do sistema e com o auxílio da automatização do fluxo de projeto [37].

O primeiro trabalho foi desenvolvido por Shiv Prakash [38], onde um problema de co-síntese foi formulado como um programa linear inteiro misto que determinava simultaneamente uma topologia de multiprocessador e uma programação com atribuição de tarefas para a arquitetura. Desde então, o problema de particionar automaticamente tarefas ou processos em hardware e software foi logo reconhecido como um tema de pesquisa cada vez mais importante ao ponto de se formar uma comunidade de investigação [37].

Inicialmente, o maior foco das pesquisas esteve direcionado no particionamento dos sistemas entre o *hardware* e o *software*. Em um esquema simples é possível verificar na Figura 10 a etapa de maior concentração de trabalhos no início dos estudos.

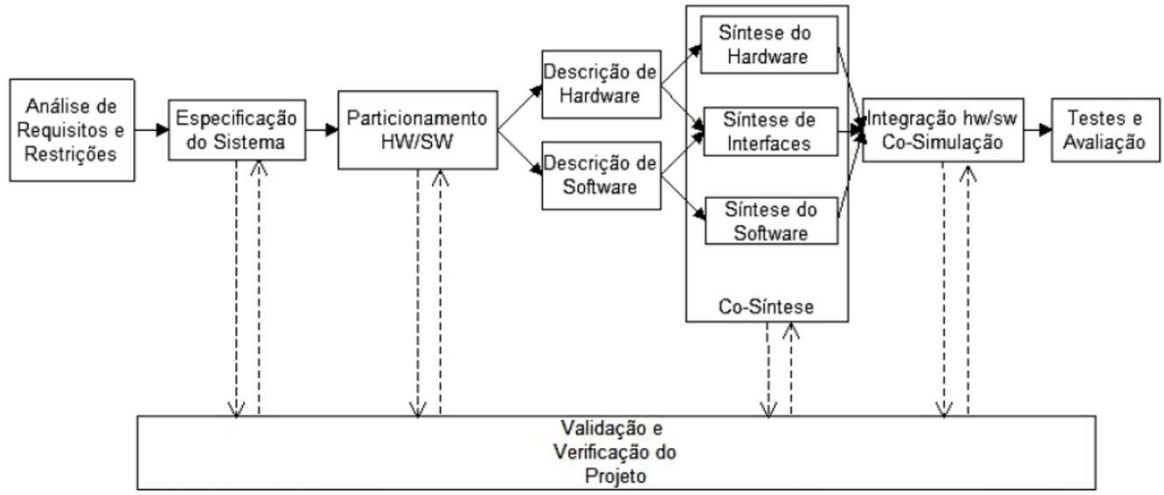


Figura 10 – Fluxo simplificado de um Co-projecto de Hardware/Software. [39]

Nesse período, uma especificação funcional de comunicação dos módulos ou tarefas especificadas em uma linguagem baseada em C era mapeado sobre uma determinada plataforma de *hardware*, a qual consistia de uma unidade de processamento central *Central Processing Unit* (CPU) e um bloco de *hardware* ou circuito integrado ASIC, com ambas as partes se comunicando através de um barramento e utilizando uma memória compartilhada, ou registos para a implementação de um *buffer*. Dessa forma, a plataforma principal já estava definida, dependendo apenas do *Hardware* que seria implementado. [37]

Nos anos seguintes, e até ao início dos anos 2000, não só o problema de particionamento de *hardware/software* foi estudado, mas também, e com grande consideração para tipos mais complexos de arquiteturas, incluindo mais de uma CPU, a evolução da execução de programas *single-threaded* foi estendido à multiprogramação e multiprocessamento. E também a co-simulação começou a se tornar uma importante área de pesquisa para a validação inicial de decisões de projeto [37] [40].

Suprimindo agora algumas das próximas descobertas presentes nesta linha do tempo, embora não menos importantes na evolução até tecnologias atuais, porém por se iniciar nessa época será apresentado o momento onde as tecnologias de Co-projecto iniciam a era da modernidade, onde todos os sistemas, ferramentas em estudo ou desenvolvidos, se agrupam em um complexo fluxo de projeto e dispositivos.

O aprimoramento das tecnologias hoje presente nos SoCfez com que as etapas de simulação e verificação automática dos projetos se tornassem mais complexas. Os atrasos temporais causados por transporte de dados das tarefas de comunicação através de *links* de comunicação são considerados. O mapeado através da introdução dos chamados de tarefas de comunicação entre as funções dependentes de dados, mapeando-os em recursos de comunicação como parte integrante do problema de particionamento. Também as redes de comunicações complexas precisam ser considerados para o desempenho e análise de custos, incluindo não só os barramentos do processador e conexões ponto-a-ponto, mas também da *network-on-chip* (NOC) (tipo de interconexão criada para realizar a comunicação interna entre os dispositivos de um SoC). Um exemplo de uma complexa arquitetura de multiprocessadores SoCs (MPSoC) pode ser verificado na Figura 11. A comunicação de tarefas, nesse caso, pode envolver vários saltos e exigir encaminhamentos a serem determinados em cima do mapeamento de tarefas.

Considerando toda essa crescente complexidade entre sistemas em *hardware* e também em *software,* assim como [37] refere, a aplicação e conhecimento de técnicas de Co-projecto de *Hardware/Software* é uma obrigação para todos aqueles que querem manter-se com os desafios de mais e mais complexos projetos de sistemas eletrônicos no futuro. Isto não só se sustenta para projetistas de SoCs, mas também para engenheiros de software e hardware envolvidos no desenvolvimento de sistemas distribuídos, como complexos sistemas automotivos em rede, aviónicos, sistemas de manufatura e sistemas embarcados em geral.

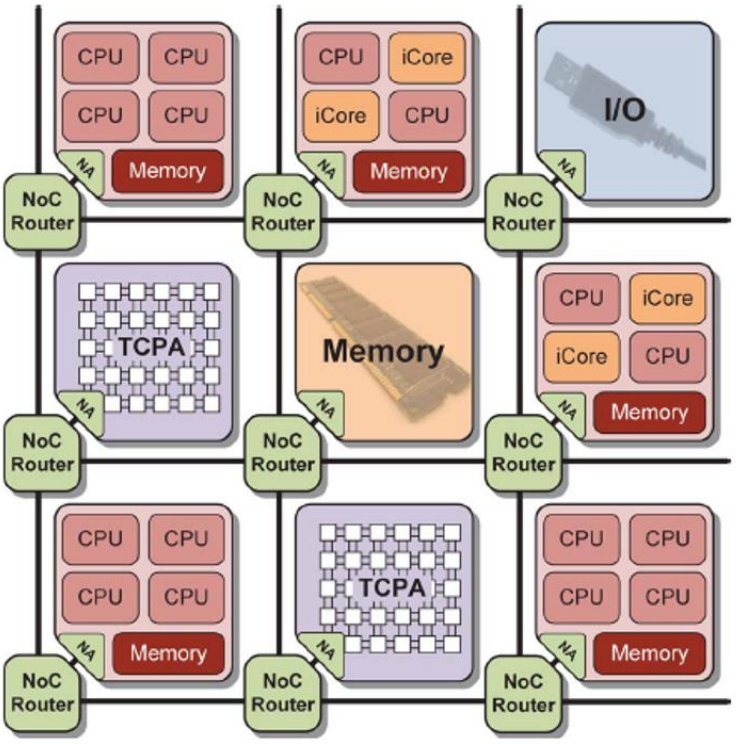


Figura 11 - Exemplo de uma arquitetura MPSoC complexa, incluindo vários quadros de diferentes CPUs processadores, Tightly Coupled Processor Arrays (TCPAs), memória e blocos de entrada/saída (I/O) com quadros interligados por uma NoC. [41]

3.1. PONTO DE VISTA PEDAGÓGICO E PROFISSIONAL

Todas as tecnologias citadas nos capítulos e secções anteriores, desde os dispositivos físicos, até as ferramentas de desenvolvimento dos projetos, apresentam grandes e constantes evoluções, sempre mostrando cada vez mais elevadas complexidades. Pode ser verificado que para todos os dispositivos estão disponíveis ferramentas que auxiliam no desenvolvimento dos projetos. Todos os fabricantes garantem formas para a exploração de seus dispositivos.

Quando volta-se a pensar nos profissionais que se encontram em nível da escala de desenvolvimento como utilizador dessas tecnologias, deparamos com uma vasta gama de conhecimentos e informações que são geradas de forma que se torna complicada a absorção de tudo a pensar no curto prazo dado pela evolução tecnológica.

Olhando ao profissional em formação ingressado em uma universidade, se torna visível a quantidade de informações que precisam ser absorvidas em um curto prazo de tempo, onde na maioria das vezes o processo ensino aprendizagem acaba sendo parado, alcançando

apenas uma época evolutiva da tecnologia em questão que corresponde a vários anos antes do atual nível da tecnologia atual.

Quando é voltada essa visão ao profissional da indústria verificasse que o tempo disponível para o desenvolvimento requerido pela indústria não engloba, em sua maioria, o tempo de aprendizagem das novas tecnologias, e pela grande demanda exige do profissional uma velocidade de desenvolvimento que só é alcançada com o domínio das ferramentas.

Atualmente o mercado desses dispositivos não consegue acompanhar a produção de materiais na mesma velocidade em que as novas tecnologias são lançadas, os materiais de iniciação a essas ferramentas acabam não sendo totalmente produzidos ou acompanham um atraso significativo ao lançamento dos produtos. Dessa forma, a produção dos materiais de auxílio a iniciação das ferramentas, se torna a cargo daqueles que necessitam desvendar suas tecnologias.

4. DESCRIÇÃO DA PLACA DE DESENVOLVIMENTO E *SOFTWARE* UTILIZADO

Os tutoriais desenvolvidos neste trabalho foram baseados em duas novas tecnologias da empresa XILINX que se encontram na posição das novas tecnologias em que os materiais para um básico desenvolvimento inicial não foi desenvolvido. São esses o SoC Zynq-7000, Z-7010 [2] e o *Software* de desenvolvimento VIVADO [1]. Para trabalhar com o SoC Z-7010 foi utilizado o kit de desenvolvimento desenvolvido pela DIGILENT, o ZYBO Zynq™-7000 *Development Board* [5]. Neste capítulo será mostrado uma breve descrição dos dispositivos.

4.1. PLACA DE DESENVOLVIMENTO ZYBO *BOARD*

A placa de desenvolvimento ZYBO *Board* é um kit desenvolvido pela Digilent, com um nível de recursos necessários para o desenvolvimento de sistemas embarcados e uma plataforma de desenvolvimento de circuitos digitais, baseado no menor modelo de SoC da família Xilinx [22] Zynq-7000, o Z-7010 [2]. O Z-7010 é baseado na arquitetura do Xilinx *All Programmable System-on-Chip* (AP SoC) [43], que integra um processador dual-core

ARM Cortex-A9 [31] com a série logica 7 de FPGAs da Xilinx. A placa onde o Z-7010 está acoplado possui um rico conjunto de multimídia e conectividade de periféricos disponíveis [44].

O Zynq Z-7010 tem a capacidade de hospedar um projeto de sistema completo, integrando memórias, vídeo e áudio I/O, dupla função USB, *Ethernet* e *slot* SD, permitindo o desenvolvimento de projetos sem nenhum hardware adicional necessário. A placa também possui seis conectores PMod disponíveis para o desenvolvimento de qualquer projeto com varias interfaces.

As características dos principais componentes presentes na placa podem ser verificadas na lista a seguir:

- 650Mhz dual-core Cortex-A9 processor
- DDR3 memory controller with 8 DMA channels
- High-bandwidth peripheral controllers: 1G Ethernet, USB 2.0, SDIO
- Low-bandwidth peripheral controller: SPI, UART, CAN, I2C
- Reprogrammable logic equivalent to Artix-7 FPGA
- 4,400 logic slices, each with four 6-input LUTs and 8 flip-flops
- 240 KB of fast block RAM
- Two clock management tiles, each with a phase-locked loop (PLL) and mixed-mode clock manager (MMCM)
- 80 DSP slices
- Internal clock speeds exceeding 450MHz
- On-chip analog-to-digital converter (XADC)
- ZYNQ XC7Z010-1CLG400C
- 512MB x32 DDR3 w/ 1050Mbps bandwidth

- Dual-role (Source/Sink) HDMI port
- 16-bits per pixel VGA source port
- Trimode (1Gbit/100Mbit/10Mbit) Ethernet PHY
- MicroSD slot (supports Linux file system)
- OTG USB 2.0 PHY (supports host and device)
- External EEPROM (programmed with 48-bit globally unique EUI-48/64™ compatible identifier)
- Audio codec with headphone out, microphone and line in jacks
- 128Mb Serial Flash with QSPI interface
- On-board JTAG programming and UART to USB converter
- GPIO: 6 pushbuttons, 4 slide switches, 5 LEDs
- Six Pmod connectors (1 processor-dedicated, 1 dual analog/digital, 3 high-speed differential, 1 logic-dedicated)

A placa ZYBO é compatível com a nova ferramenta de alto desempenho da Xilinx, o Vivado *Design* Suite, bem como também o conjunto de ferramentas ISE / EDK. Estes conjuntos de ferramentas são utilizados para fundir o projeto de *hardware* do FPGA com o desenvolvimento do *software*. Eles podem ser usados para projetar sistemas de qualquer complexidade, desda programas simples como um programa que controla alguns LEDs, até um sistema operacional completo executando vários aplicativos de servidor em conjunto.

As interfaces presentes na placa ZYBO podem ser verificadas na Figura 12, juntamente com a descrição da Tabela 3.

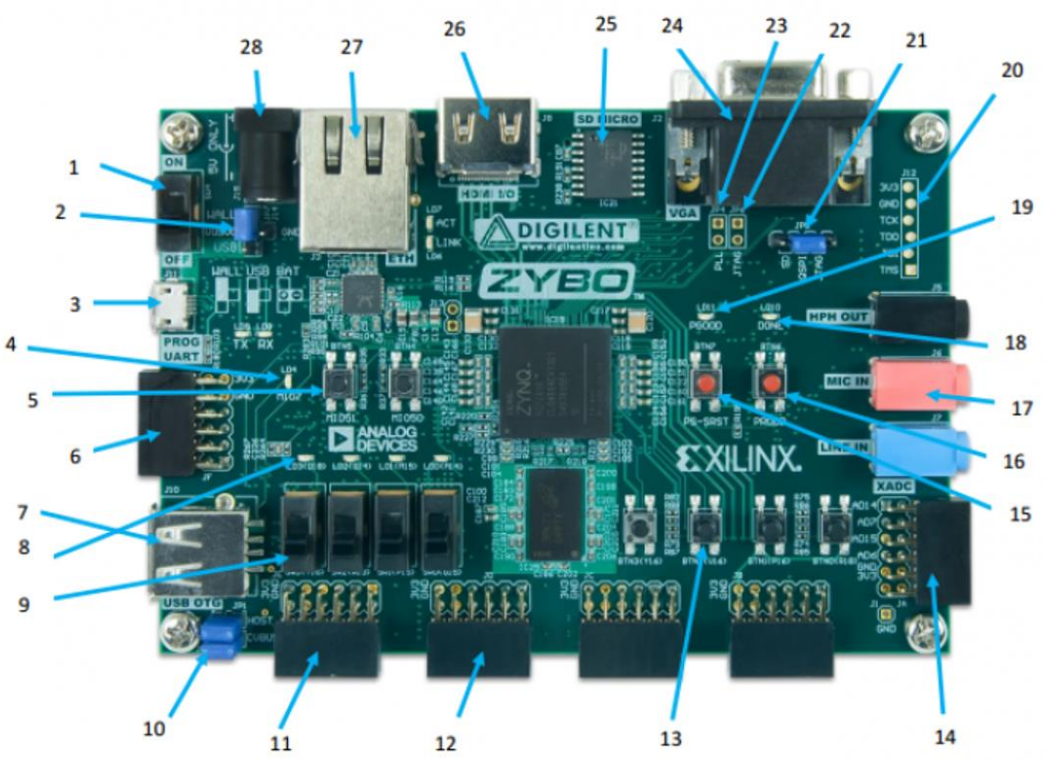


Figura 12 – Interfaces Placa ZYBO. [44]

Tabela 3 – Descrição das Interfaces da Placa ZYBO. [44]

Ref.	Interface Descrição	Ref.	Interface Descrição
1	Power Switch	15	Processor Reset Pushbutton
2	Power Select Jumper and battery header	16	Logic configuration reset Pushbutton
3	Shared UART/JTAG USB port	17	Audio Codec Connectors
4	MIO LED	18	Logic Configuration Done LED
5	MIO Pushbuttons (2)	19	Board Power Good LED
6	MIO Pmod	20	JTAG Port for optional external cable
7	USB OTG Connectors	21	Programming Mode Jumper
8	Logic LEDs (4)	22	Independent JTAG Mode Enable Jumper
9	Logic Slide switches (4)	23	PLL Bypass Jumper
10	USB OTG Host/Device Select Jumpers	24	VGA connector
11	Standard Pmod	25	microSD connector (Reverse side)
12	High-speed Pmods (3)	26	HDMI Sink/Source Connector
13	Logic Pushbuttons (4)	27	Ethernet RJ45 Connector
14	XADC Pmod	28	Power Jack

Mais informações dos dispositivos podem ser visualizadas em [44].

4.1.1. SoC Zynq-7000, Z-7010

A característica que define o Zynq é que ele combina um processador dual-core ARM Cortex-A9 [3], com um tradicional *Field Programmable Gate Array* (FPGA) como estrutura lógica. Embora os processadores dedicados já terem sido agrupados a FPGAs antes, nunca foi feito algo na mesma proporção. No Zynq, o ARM Cortex-A9 é um processador capaz de rodar sistemas operacionais completos, como Linux, enquanto a lógica programável é baseado em uma arquitetura de FPGA da série 7 da Xilinx [45] [46]. A arquitetura é complementada por interfaces *Advanced eXtensible Interface* (AXI) padrão da indústria, que oferecem alta largura de banda e conexões de baixa latência entre as duas partes do dispositivo – processador e lógica programável [47]. Isto significa que o processador e lógica podem ser usados para executarem o que fazem melhor, sem a sobrecarga de interface entre dois dispositivos separados fisicamente. Acrescenta-se também os benefícios resultantes de simplificar o sistema a um único chip como reduções no tamanho físico e custo global.

Como pode ser verificado na Figura 13 a arquitetura do Zynq está dividida entre o *Processing System* (PS) e o *Programmable Logic* (PL), conectados através dos barramentos AXI. Na região PS estão disponíveis todas as interfaces de comunicação que podem ser visualizadas, e na região PL encontra-se toda a gama dos dispositivos lógicos conectados a todas as interfaces da placa ZYBO.

O PL é praticamente idêntico a um Xilinx FPGA Artix série 7, exceto que ele contém várias portas dedicadas e o barramento AXI que realiza a comunicação com o PS. O PL pode ser configurado tanto pelo processador como através da porta JTAG.

O PS consiste de muitos componentes, incluindo a *Application Processing Unit* (APU, que inclui 2 processadores Cortex-A9), ou do português, Unidade de Aplicação de Processamento, arquitetura de barramento de comunicação *Advanced Microcontroller Bus Architecture* (AMBA), controlador de memória DDR3, e vários controladores de periféricos com suas entradas e saídas multiplexadas dedicadas a 54 pinos (chamados I/O multiplexados, ou MIO pinos). Possui também controladores de periféricos que não têm suas entradas e saídas ligadas ao MIO, onde a rota do seu I/O é direcionada por meio do PL, através do (EMIO) interface MIO Estendida. Os controladores de periféricos são

conectados aos processadores como escravos através da interligação AMBA, e contêm registros de controlo de leitura e gravação que são endereçáveis no espaço de memória dos processadores. A PL também está ligado à interconexão como um escravo, e os projetos podem implementar múltiplos núcleos na arquitetura FPGA onde cada um também contém registros de controlo endereçáveis. Além disso, os núcleos implementados no PL podem ativar interrupções para os processadores (ligações não mostradas na Figura 13) e também podem executar DMA de acesso à memória DDR3.

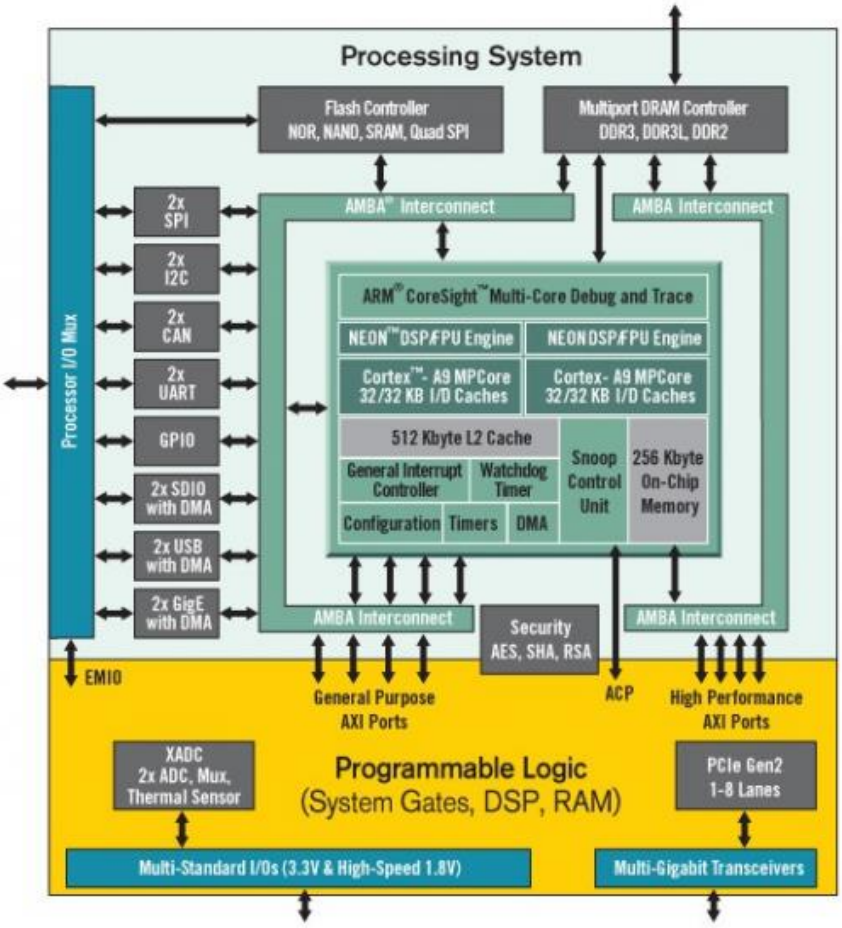


Figura 13 – Arquitetura do SoC Zynq. [44]

4.1.2. BARRAMENTO DE COMUNICAÇÃO AXI (INFORMAÇÃO RETIRADA DE [48])

"A Xilinx adotou o protocolo AXI (Advanced eXtensible Interface) para IPs a partir dos FPGAs Spartan-6 e Virtex-6 e o continua a utilizar para as séries 7 e Zynq 7000. AXI faz parte da família de barramentos para os processadores ARM, sendo a última versão AXI4."

“O limite de transferência no AXI4 é de uma rajada de até 256 dados por transação. O AXI4-Lite permite somente 1 dado transferido por transação e o AXI4-Stream suporta uma rajada com tamanho ilimitado de dados.”

“O *Read Address Channel* é onde o IP envia um sinal informando que deseja iniciar uma transação de leitura. Este canal traz informações de endereçamento e alguns sinais de *handshaking*. O *Read Data Channel* contém os dados que são transferidos durante uma operação, juntamente com os sinais de *handshaking* associados.” A Figura 14 mostra a arquitetura do canal.

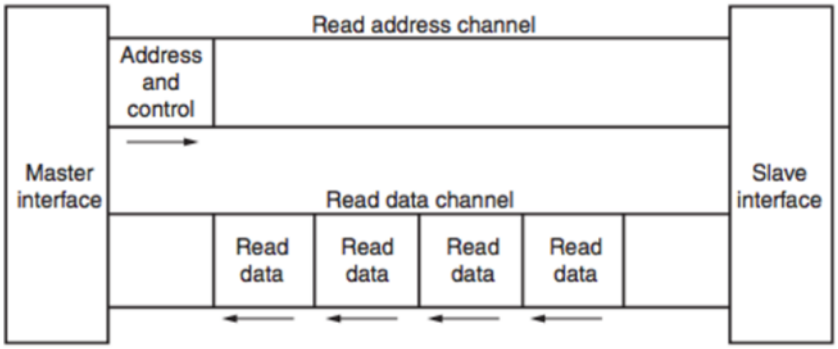


Figura 14 – Arquitetura do canal de leitura do AXI-lite. [47]

“O *Write Address Chanel* é a via utilizada pelo IP para sinalizar que deseja iniciar uma transação de escrita. Ele é idêntico ao *Read Address Channel*. O *Write Data Channel* executa uma função equivalente ao *Read Data Channel*, exceto que os dados são oriundos do master. O *Write Response Channel* é utilizado pelo *slave* para avisar a receção dos dados, ou para indicar que ocorreu um erro.” A Figura 15 mostra a arquitetura do canal.

“Os sinais de *handshaking* presentes em todas as transações de *read* e *write* são importantes para o controle dos dados que trafegam nos canais disponíveis. Os sinais são baseados em um princípio de *ready* e *valid* ou seja “pronto” e “válido”. O sinal *ready* é usado pelo *slave* para indicar que está pronto para aceitar a transferência de dados ou endereçamento, e *valid* é usado para informar que o dado emitido naquele canal é válido, para que então possa ser utilizado.”

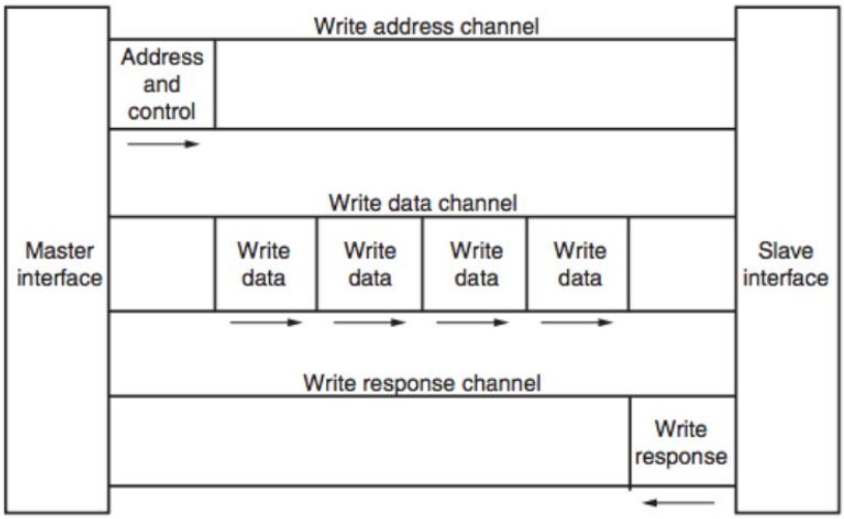


Figura 15 – Arquitetura do canal de escrita AXI-lite. [47]

4.1.3. INTERFACE DE VÍDEO VGA

O VGA é uma interface padrão para o controle de monitores analógicos. O lado da interface fornece ao monitor sinais horizontais e verticais de sincronização e magnitudes de cores.

Os sinais de sincronismo horizontal e vertical são de 0V/5V em formas de onda digitais que sincronizam o tempo do sinal com o monitor. Sendo um sistema digital, eles são fornecidos diretamente pelo FPGA (3,3V encontra o limiar mínimo para um alto lógico, de modo que o 3,3V pode ser usado em vez de 5V).

As magnitudes de cores são sinais analógicos 0V-0,7V enviados pelos fios R, G e B (em alternativa, o fio verde pode utilizar os sinais de 0,3V-1V que incorporam ambos os sinais de sincronismo horizontal e vertical, eliminando a necessidade dessa linha) [50].

O kit ZYBO *board* usa 18 pinos lógicos programáveis para criar uma porta de saída VGA analógica. Isto traduz a profundidade de cor de 16 bits e dois sinais de sincronização padrão (HS - Sincronização Horizontal, e VS - Sincronização Vertical). A conversão digital-analógica é feita usando uma simples estrutura *ladder* de resistor R-2R. A estrutura funciona em conjunto com a resistência de terminação de 75 ohms do display VGA para criar 32 e 64 níveis de sinais analógicos, sinais VGA vermelho, azul e verde. O circuito, mostrado na Figura 16, produz sinais de cor de vídeo que procedem em incrementos iguais entre 0V (totalmente desligado) e 0,7V (totalmente ligado). Com 5 bits para cada cor,

vermelha e azul e 6 bits para a cor verde, 65.536 (32 × 32 × 64) cores diferentes podem ser exibidos, um para cada padrão de 16 bits [44].

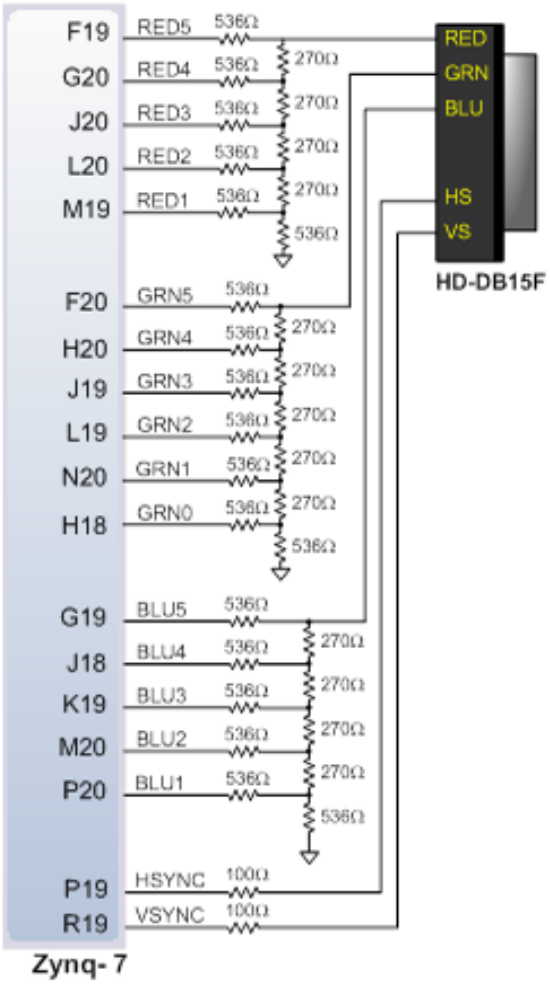


Figura 16 – Interface R-2R VGA disponível na ZYBO board. [44]

4.2. *SOFTWARE* DE PROJETO VIVADO (INFORMAÇÃO ADAPTADA DE [48])

"O VIVADO é o conjunto mais recente de ferramentas disponibilizadas pela Xilinx. A plataforma Vivado possibilita o desenvolvimento de aplicações em nível de *Hardware/Software*. O fluxo de projeto no VIVADO pode ser realizado de duas formas, primeiramente baseando em um fluxo de projeto de *Hardware* tradicional, como pode ser visto na Figura 17, ou seguindo um fluxo de projeto básico de Hardware/Software do VIVADO. Neste caso o projeto deve ser iniciado pelo VIVADO *Integrated Design Environment* (IDE), e então o Hardware criado é exportado para o Xilinx Software Development Kit (SDK). Esta ferramenta vem para substituir e integrar as funcionalidades do PlanAhead [51] e do Chipscope [52] (utilizado para testes de sinais no próprio hardware

que foi elaborado). Na Figura 18 é possível verificar o fluxo de projeto para implementação de um projeto no VIVADO IDE e as informações contidas no arquivo que são exportadas para o Xilinx SDK. A plataforma possui uma gama de opções de *Intellectual Property* (IP) que são blocos desenvolvidos com padrões usuais onde são consideradas a propriedade de criação, e também permite a implementação dos próprios IPs."

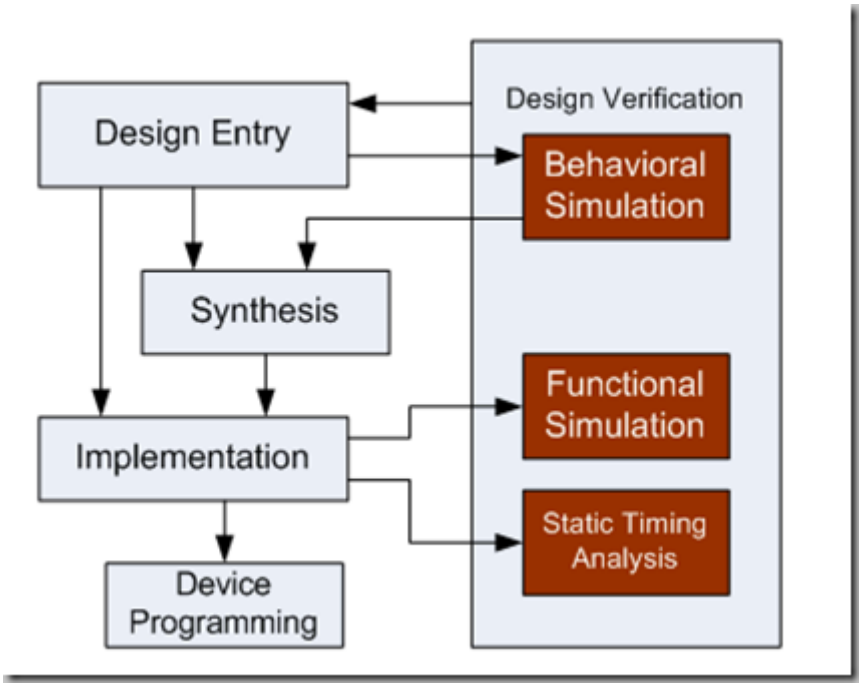


Figura 17 – Fluxo de Projeto para FPGAs. [53]

Ainda segundo [48], "Seguindo um fluxo de projeto de um sistema Hardware/Software o primeiro passo do fluxo é a configuração do PS no VIVADO, onde são configuradas as opções de memória, a frequência que o processador vai operar, os periféricos disponíveis no *hardware* que vão ser instanciados, as interfaces de comunicação com o barramento AXI e as interrupções. Na etapa de inclusão de IPs, podem ser adicionados IPs disponíveis no catálogo do Vivado ou adicionado um IP desenvolvido especificamente para o projeto. Após validado o projeto é gerado o bitstream e então exportado para o SDK como mostrado na Figura 18. O SDK possui especificações do hardware exportado, como periféricos disponíveis e seus respetivos endereçamentos. Com base neste projeto é desenvolvido um *software* que vai se comunicar com o processador. Após concluído e gerado um arquivo com a extensão ".elf" para ser enviado para placa juntamente com o arquivo ".bit" referente à etapa de hardware."

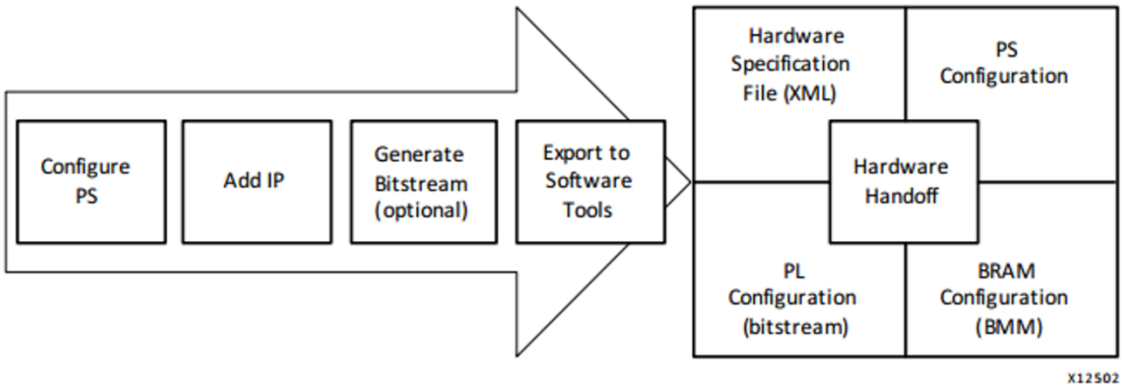


Figura 18 – Fluxo do projeto de Hardware e informações contidas no arquivo exportado para desenvolver o software. [42]

O Vivado *Design Suite* é um software EDA projetado para auxiliar no desenvolvimento de projetos. Esta ferramenta é projetada para aumentar a produtividade na concepção, integração e implementação de sistemas utilizando a série Xilinx 7, Zynq-7000 (AP) SoC, e dispositivos UltraScale.

O Vivado *Design Suite* fornece métodos de análise de projeto em cada fase do projeto. São disponíveis desde o início dos projetos podendo ser utilizado pelas ferramentas de configuração e modificações, dessa forma as alterações necessárias têm menos impacto global na programação, reduzindo assim, as iterações de projeto e acelerando a produtividade [1].

O VIVADO juntamente com todo o *Software* disponível em sua gama de projeto possui um fluxo de projeto que abrange todos os níveis de abstração, tendo ferramentas como o *Vivado High-Level Synthesis* [54], que possibilita o projeto a partir de linguagens de alto nível. Na Figura 19 é possível verificar o fluxo de projeto completo da ferramenta VIVADO.

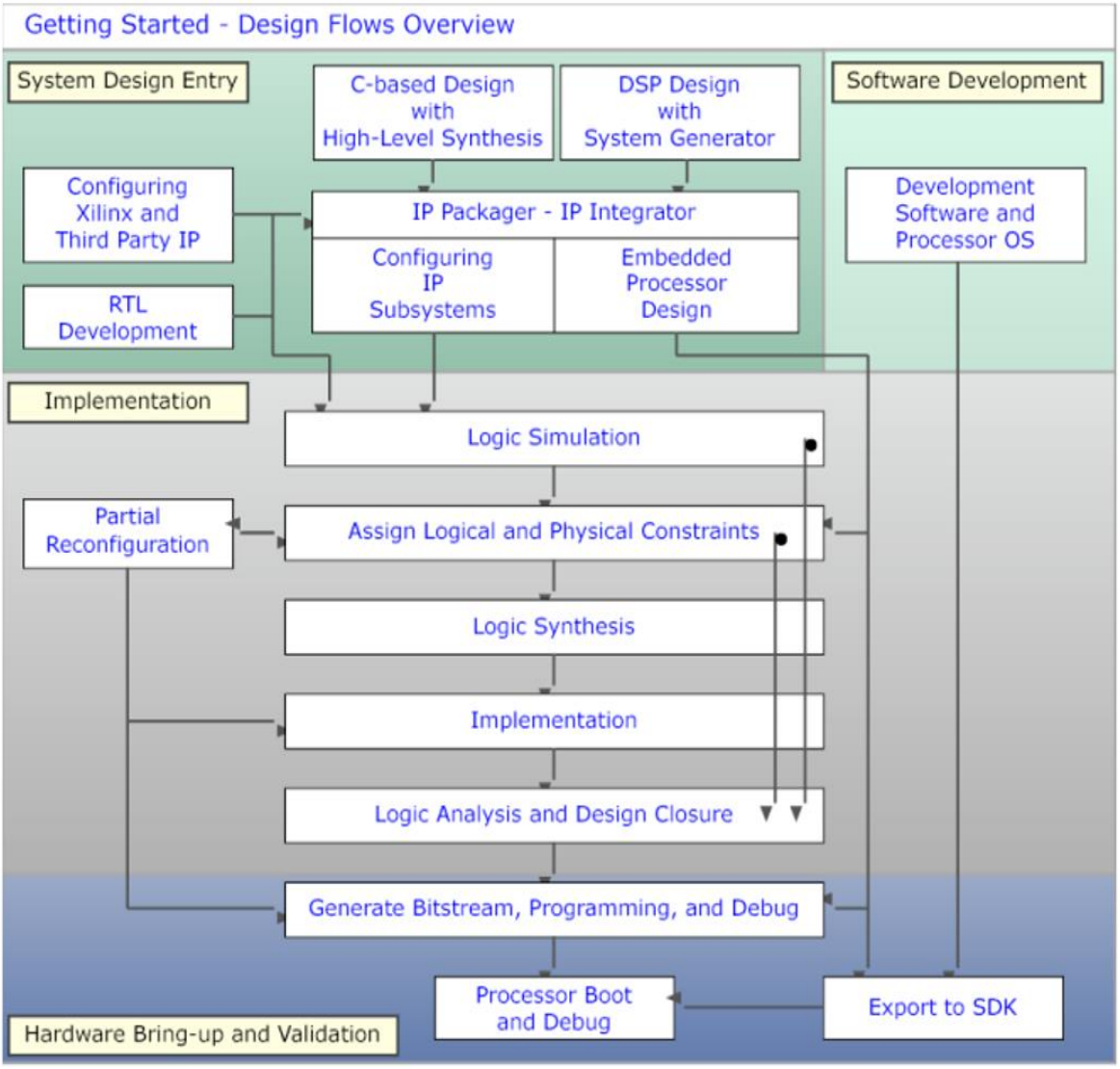


Figura 19 – Fluxo de Projeto da Plataforma Vivado. [55]

5. DOCUMENTOS PRODUZIDOS

Com o objetivo de desenvolver tutoriais com a descrição passo-a-passo de um exemplo que permita a um utilizador novo, o conhecimento da ferramenta VIVADO e do kit de desenvolvimento ZYBO *board* e que sirva de referência ao desenvolvimento de projetos futuros. Este capítulo descreve o objetivo de cada documento produzido juntamente com a apresentação dos conteúdos e informações neles contidas.

5.1. OBJETIVO DE CADA DOCUMENTO

Com o intuito de auxiliar a grande dificuldade no início da aprendizagem de novas ferramentas *Electronic Design Automation* (EDA) em foco a ferramenta VIVADO, e também dos novos dispositivos disponíveis no mercado, neste caso o processador SoC Zynq-7000, Z-7010, base do kit de desenvolvimento ZYBO *board*, o trabalho desenvolvido teve como resultado o desenvolvimento de três tutoriais baseados nas novas tecnologias citadas acima. Os tutoriais têm como objetivo apresentar passo-a-passo como desenvolver um simples projeto para a introdução da ferramenta e do dispositivo SoC em questão. O idioma para escrita dos tutoriais foi definido o Inglês para que possam alcançar um maior número de utilizadores e possam ser apresentados em todos os locais de ensino.

Os três tutoriais desenvolvidos estão disponíveis nos anexos deste documento, Anexo A, Anexo B e Anexo C.

O primeiro tutorial teve como foco abstrair o funcionamento de todos os módulos de análise e revisão de projeto presentes na ferramenta VIVADO, assim como o seu fluxo de projeto utilizado para projetos usando puramente os recursos lógicos programáveis, sendo isso obtido através do exemplo do desenvolvimento de um projeto puramente lógico (não necessariamente complexo) que utiliza as interfaces simples disponíveis na placa de desenvolvimento.

O segundo tutorial teve como foco analisar o fluxo de projeto do VIVADO para um projeto a nível de *Hardware/Software*, e como realizar o particionamento e a integração entre as duas partes. Para isso foi desenvolvido um projeto com a mesma funcionalidade do tutorial anterior, porém com a aplicação sendo executada em uma estrutura em *Hardware* desenvolvida no VIVADO e por um *Software* desenvolvido na ferramenta SDK do VIVADO sendo executada pelo processador presente na estrutura de *Hardware* criada.

O terceiro tutorial tem como foco uma avaliação do conteúdo dos tutoriais anteriores, desta forma apresentando um tutorial baseado no funcionamento de uma das interfaces externas disponíveis na placa ZYBO e apresentando como o projeto deve ser desenvolvido em conjunto com os conhecimentos já adquiridos nos tutoriais anteriores.

5.1.1. CONTEÚDO E OBJETIVO PEDAGÓGICO E O QUE SE APRENDE EM CADA DOCUMENTO.

Os três tutoriais desenvolvidos tiveram objetivos particulares de aprendizagem como citado acima. Em cada um deles o ensinamento se baseou em ferramentas específicas do fluxo de projeto abordado, o desenvolvimento dos tutoriais não teve como base a apresentação das etapas “isoladas” presentes em um fluxo de projeto, porém o intuito foi demonstrar a necessidade e funcionamento de cada uma delas de acordo com a sequência natural presente no fluxo de projeto da ferramenta VIVADO. Na Tabela 4 encontrasse um resumo dos conhecimentos adquiridos em cada um dos tutoriais.

Tabela 4 – Conhecimentos do fluxo de projeto adquiridos nos tutoriais.

Tutorial I	Tutorial II	Tutorial III
- Criar um novo Projeto no VIVADO; - Criar um Projeto em Blocos; - Adicionar um IP integrador ao projeto em blocos; - Criar um invólucro HDL para o projeto; - Simular o Projeto, em processo manual; - Análise RTL; - Associação dos Pinos; - Sintetizar o Projeto; - Associação dos Pinos por arquivos de restrições; - Implementar o Projeto; - Criar um arquivo fonte em VHDL; - Simular o Projeto com um Arquivo de Teste; - Criar um novo IP integrador; - Adicionar um novo IP ao Projeto; - Gerar o fluxo de Bits do projeto; - Programar a Placa ZYBO Board;	- Criar um novo Projeto no VIVADO; - Criar o projeto de *hardware* em Blocos de base para o projeto em *software;* - Exportar a estrutura para o SDK; - Criar um projeto no SDK; - Programar o Processador com o SDK;	- Estrutura Básica para um Projeto gerador de sinal VGA; - Exemplo de projeto de um controlador VGA para a ZYBO Board; - Compreensão da Temporização do sinal de controle e sincronismo do VGA; - Dicas para o desenvolvimento do projeto com o auxílio dos tutoriais anteriores;

5.2. EXPLICAÇÃO DO CONTEÚDO

O conhecimento adquirido nos tutoriais é baseado na apresentação do fluxo completo de projeto de dispositivos simples a serem implementados no SoC.

Os dois primeiros tutoriais são baseados na implementação de um simples contador binário de 4 bits com as funções de contagem crescente, decrescente e reiniciar. O contador binário foi escolhido como a base dos tutoriais, pois possibilita que o projeto a ser desenvolvido trabalhe com variáveis de entrada e saída, exista a possibilidade de ser implementado apenas em *hardware* e também em *hardware/software* e utiliza apenas interface de componentes já presentes na placa ZYBO *board*. O contador utiliza de três interfaces de I/O presentes na placa ZYBO os 4 Leds, 1 botão e 1 chave, como é possível ver no esquemático da Figura 20, onde está representada a conexão das interfaces da placa com o SoC Z-7010. Os dispositivos utilizados foram os 4 Leds (LD0, LD1, LD2, LD3), o botão (BTN0), e a chave (SW0), que também podem ser verificados na Figura 21.

Por padrão dos projetos o funcionamento do contador foi convencionado na contagem visual através dos 4 Leds, a chave (SW0) seleciona o sentido da contagem entre crescente/decrescente, e ao botão (BTN0) foi atribuída a função de reiniciar a contagem.

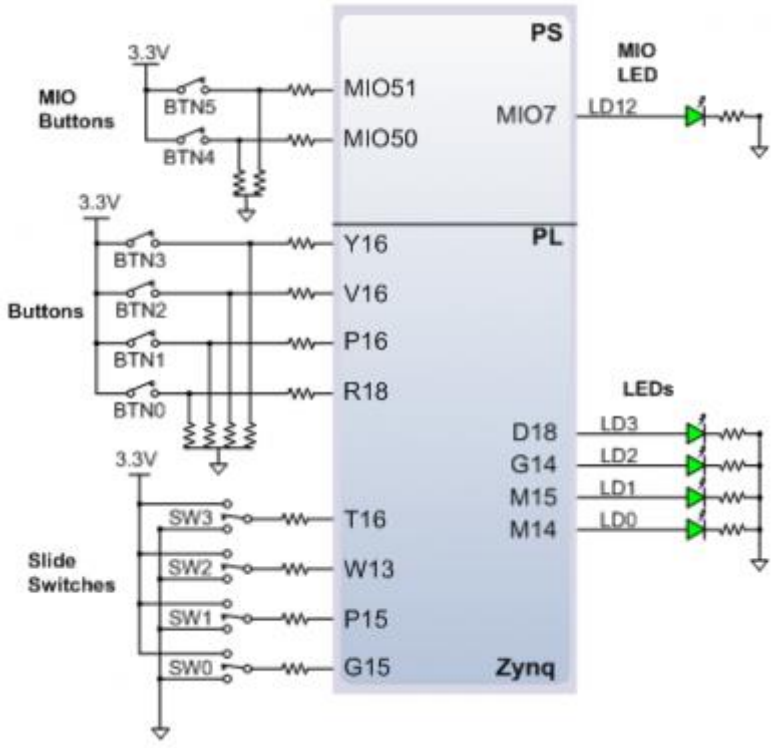


Figura 20 – Esquemático das Interfaces de I/O Básicas do kit ZYBO. [44]

Figura 21 – Componentes das Interfaces de I/O Básicas do kit ZYBO. [44]

No primeiro Tutorial foi desenvolvido todo o projeto em uma estrutura de *hardware* baseado no diagrama mostrado na Figura 22. O projeto contém dois blocos que são divididos em *Clock_Divider* (Divisor de Relógio) e *Binary Counter* (Contador Binário).

O primeiro bloco foi desenvolvido para ter a função de dividir o *clock* disponível para a arquitetura lógica do SoC da placa ZYBO, pois sua frequência nominal é de 125MHz, muito alta para ser visualizado em um simples contador. Dessa forma, esse bloco estruturado em uma entrada (clock) e uma saída (clock dividido), divide essa frequência para 1Hz para que assim possa ser possível visualizar a contagem dos Leds. Esse bloco foi desenvolvido através de um arquivo fonte em VHDL e adicionado ao projeto final através das ferramentas de criação de novos IPs. Seu código pode ser visualizado na página 49 do Tutorial I.

O segundo bloco foi desenvolvido para realizar a contagem binária com os Leds a partir da entrada do clock e das configurações setadas pelo botão e pela chave. Esse bloco foi desenvolvido com base em um IP integrador disponível na biblioteca do VIVADO. A sua configuração é interativa, realizada através de uma interface pré-definida no VIVADO. Segundo o valor do clock de entrada foi configurada a quantidade de bits necessária para a divisão do relógio para a saída. Foi acrescentado também as funções de contagem crescente/decrescente pela porta UP e a função de reiniciar, adaptando a função de limpar da porta acrescentada SCLR.

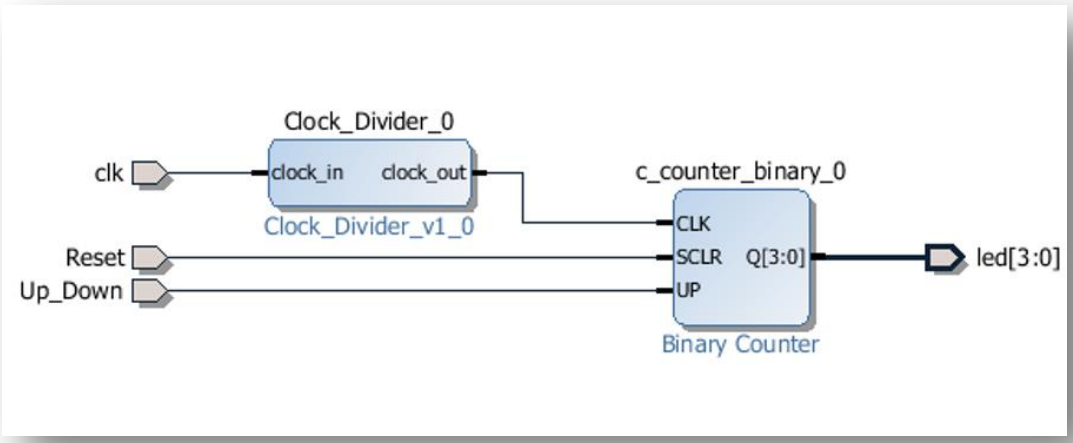


Figura 22 – Diagrama em Blocos do contador de 4 Bits desenvolvido puramente em lógica.

No segundo Tutorial o fluxo de projeto foi mudado para abranger a utilização do processador presente no SoC Z-7010. A estrutura deste projeto foi dividida em realizar o desenvolvimento de todas funções do contador implementadas em software utilizando a PL do SoC apenas para fornecer a estrutura de conexão entre o processador e as interfaces necessárias (Leds, Botão e Chave) como já foi citado acima. O diagrama em blocos da estrutura em *hardware* desenvolvida nesse tutorial, como pode ser visualizada na Figura 23, tem como foco realizar a interface estrutural de conexão e comunicação das interfaces de I/O já citadas com o *software* que será embarcado no Processador ARM Cortex A9 do SoC.

Por ser uma estrutura de alto nível, os blocos não foram desenvolvidos a partir de descrições HDLs básicas. Todos fazem parte da biblioteca de IPs presentes no VIVADO, por possuírem um padrão que é explicado no tutorial. Suas interconexões também são realizadas automaticamente.

Como já foi explicado essa estrutura tem como função criar uma interface entre o processador e a interface I/O disponível em *hardware*. Para isso a interface é criada através do barramento de comunicação AXI, representado pelo bloco AXI *Interconnect*. Esse protocolo de comunicação, como já foi explicado no capítulo anterior, possui um protocolo automático de comunicação para várias interfaces desenvolvidos pela Xilinx em que a interface I/O é uma delas. Por isso, sua conexão é automática se seguir os passos presentes no tutorial.

O bloco ZYNQ7 *Processing System* é o IP com a estrutura do processador onde o *software* após compilado será implementado. O bloco *Processor System Reset* tem como função sincronizar o processador com barramento de comunicação AXI. O bloco AXI *Interconnect* realiza a implementação e o gerenciamento do barramento de comunicação entre o processador e as interfaces I/O. Por sua vez, os blocos AXI GPIO realizam as ações requisitadas pelo processador por meio do barramento, sendo essas acender os leds ou realizar a leitura do botão e da chave.

Neste fluxo de projeto, após a estrutura ser implementada como mostrado, ela é exportada para a ferramenta SDK onde o *software* é desenvolvido utilizando as bibliotecas preparadas para trabalhar com a comunicação do processador com o barramento AXI e assim realizar seu processamento para executar o contador binário. O código desenvolvido pode ser visualizado no Tutorial II.

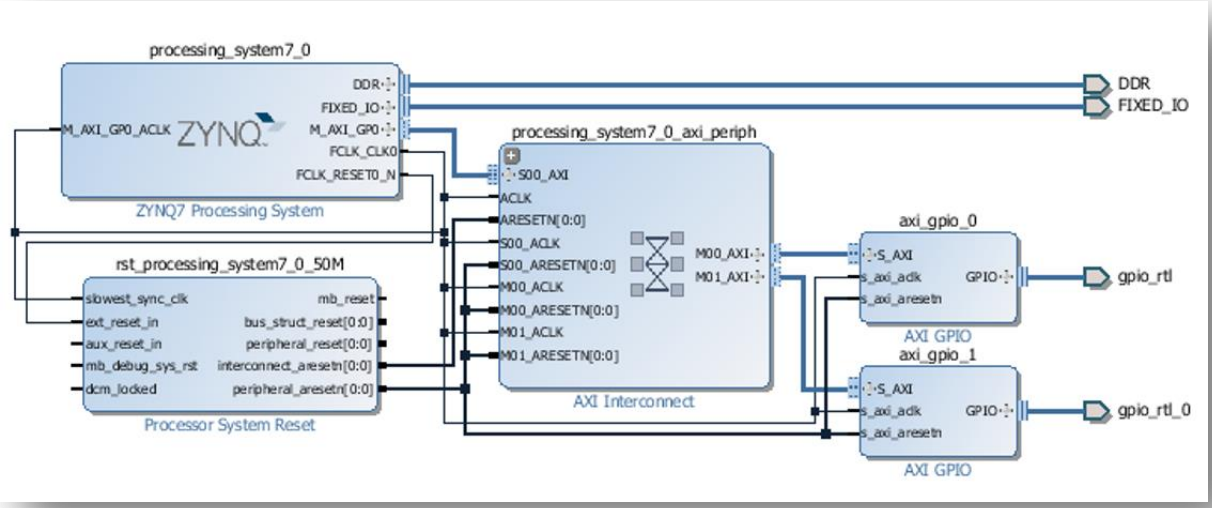


Figura 23 – Diagrama em Blocos da estrutura em hardware do contador de 4 Bits desenvolvido para a interface do software.

O terceiro tutorial tem como objetivo utilizar os conhecimentos adquiridos. Dessa forma, não foi realizado o passo-a-passo completo de seu desenvolvimento. Porém, são fornecidas todas as informações necessárias para o seu desenvolvimento.

O objetivo deste tutorial é desenvolver um *driver* em *hardware* para a interface VGA presente na placa ZYBO. O VGA totalmente em *hardware* foi escolhido para dar início a

outras séries de tutoriais que sejam focados em passar o conhecimento para a implementação dos projetos.

O projeto desenvolvido no terceiro tutorial tem como objetivo funcional o controle através dos botões da placa de alguma imagem gerada em um monitor a partir da saída VGA. A definição final do projeto foi o controlo de um círculo na tela do monitor através dos botões.

Utilizando a mesma interface de botões mostrada no esquemático da Figura 20 e na representação real da Figura 21, foi utilizado como entrada de comandos de movimento da imagem os botões (BTN0, BTN1, BTN2, BTN3) e a chave (SW0) para o controle do reiniciar.

Para a interface VGA, como pode ser visualizado na Figura 16 foram utilizadas todas as conexões disponíveis para a geração do sinal de sincronismo e da imagem.

Como já dito, o projeto desse tutorial foi totalmente em *hardware*, obtendo o diagrama mostrado na Figura 24.

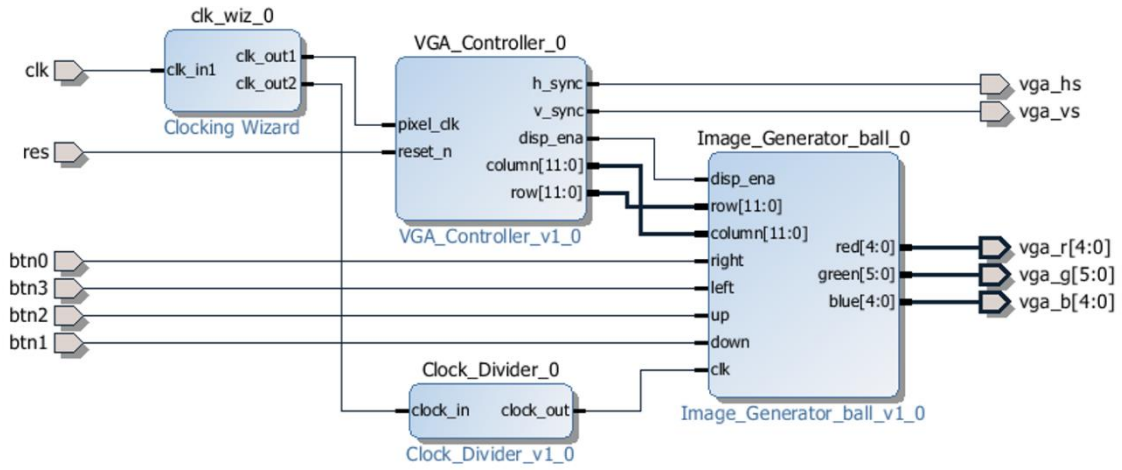


Figura 24 – Diagrama em Blocos do projeto VGA.

O projeto foi dividido em quatro blocos como mostrado. No primeiro bloco *Cloking Wizard* o *clock* é dividido em dois outros *clocks*: um deles é dividido para a frequência necessária para o sincronismo da comunicação; e o outro será enviado a outro bloco para ser dividido ainda mais e será utilizado para a leitura dos botões de comando. Esse bloco foi criado através de um IP disponível na biblioteca do VIVADO. O segundo bloco

VGA_Controller possui a entrada do reiniciar e o *clock* de sincronismo. Esse bloco gera todos os sinais de sincronismo necessário para a comunicação, o sincronismo horizontal/vertical e os sinais de varredura de tela para a implementação das imagens. O terceiro bloco *Clock_divider* realiza a divisão do *clock* do bloco anterior para que o mesmo seja utilizado na leitura do controle pelos botões. O último bloco, por sua vez, através dos sinais de varredura da tela gerados no bloco de controlo, realiza através das restrições em seu código a seleção do nível de tensão que deverá sair pelos pinos da interface VGA para que seja transmitida assim a imagem desejada programada, neste caso, a imagem que pode ser vista na Figura 25.

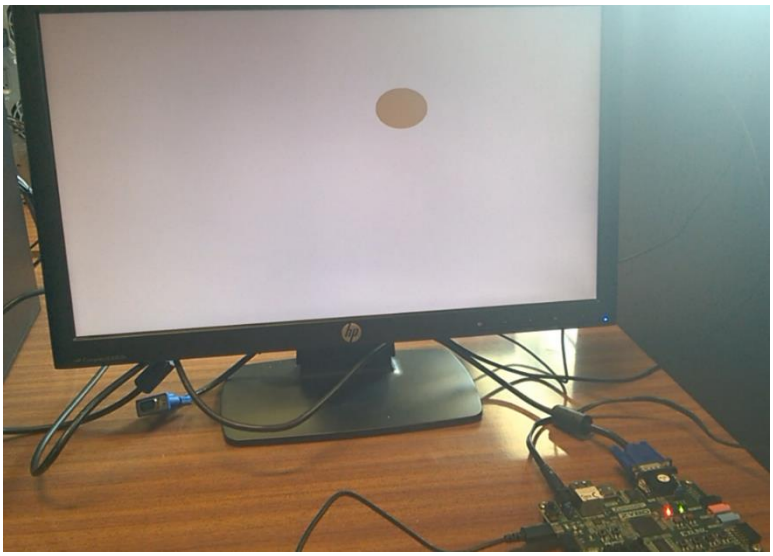


Figura 25 – Imagem Gerada pelo Projeto do Tutorial III.

5.3. FLUXO DE PROJETO APRESENTADO

No desenvolvimento dos tutoriais foram seguidos dois fluxos de projetos, um voltado para o desenvolvimento dos projetos baseados unicamente em estruturas PLs ou de *hardware* e outro voltado para o desenvolvimento de projetos em *hardware/software*.

O primeiro fluxo de projeto, como podemos ver no diagrama da Figura 26, é descrito caracterizando apenas um projeto de *Programmable Logic* (PL). Sua estrutura é baseada na conceção de um projeto desenvolvido através das HDLs e IPs, que são integrados através de um diagrama em blocos interligados. O diagrama então é envolvido em uma máscara HDL que engloba todos os blocos criados. Neste ponto já é possível realizar a simulação do comportamento esperado do projeto e realizar algumas análises na estrutura RTL. São então introduzidas as restrições que associam os *ports* da máscara HDL com os pinos do

SoC. O projeto é então sintetizado transformando-o em um nível de lógica. Mais análises podem ser feitas agora e a simulação possuirá informações temporais disponíveis. Gerando então a implementação do projeto, este será convertido para a ligação entre os dispositivos disponíveis no dispositivo SoC utilizado. Informações de recursos utilizados, entre outros, podem ser analisadas agora. O próximo passo é a geração do *bitstream* ou a conversão do projeto em um fluxo de dados que serão enviados ao SoC durante a programação. Após todas as etapas concluídas a ZYBO *board* pode ser conectada e programada.

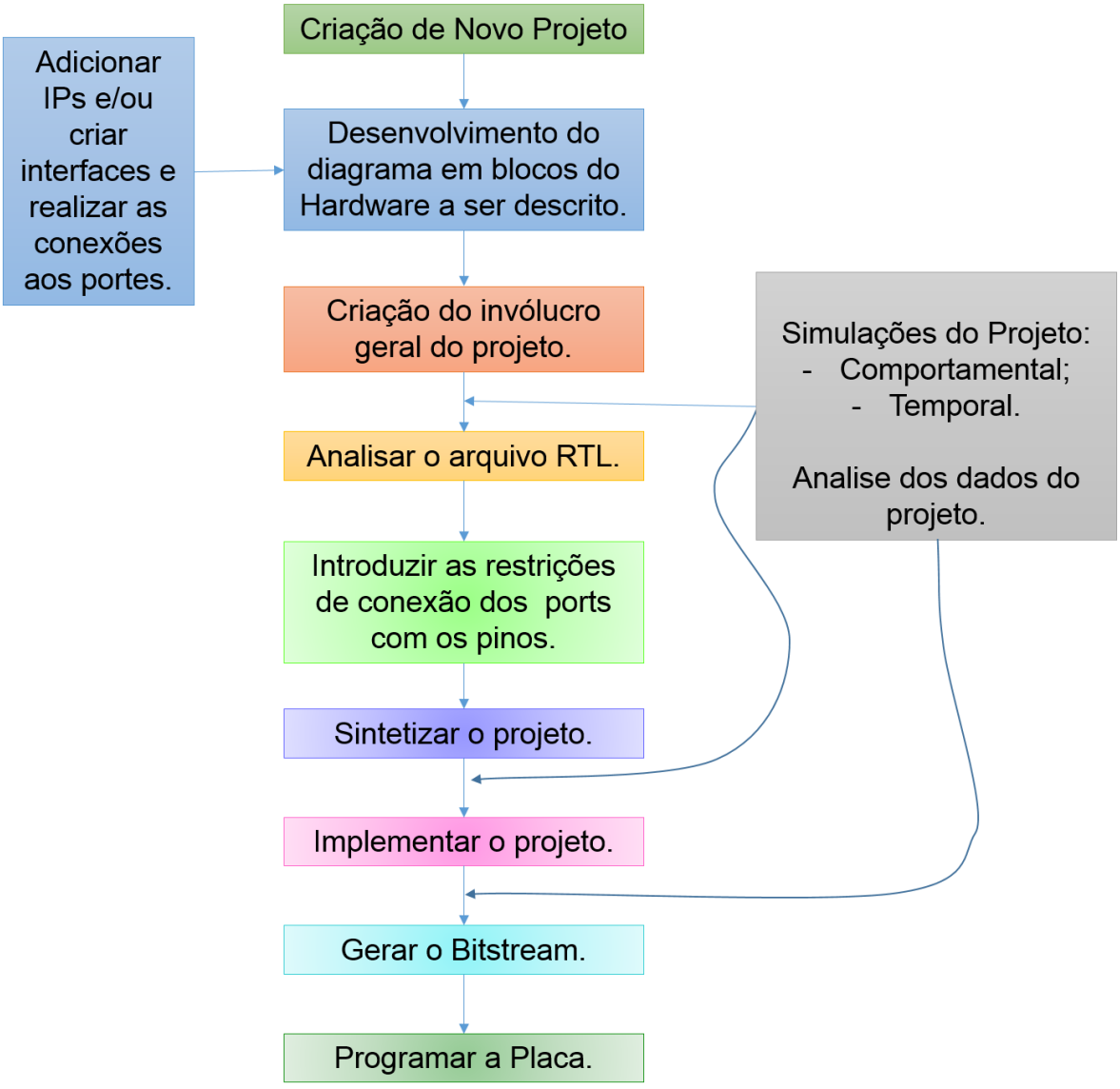


Figura 26 – Diagrama em blocos do fluxo de projeto somente PL.

O segundo fluxo de projeto utilizado pode ser compreendido através do diagrama da Figura 27. A estrutura inicia com a concessão de um novo projeto. No nível de desenvolvimento do diagrama em blocos, o diagrama é resumido ao desenvolvimento da

estrutura que irá dar suporte ao *software*. A estrutura criada é envolvida em uma máscara HDL. As restrições para a associação dos portos com os pinos são adicionadas. Neste ponto, o projeto é validado para verificar qualquer possível erro na estrutura, e os próximos passos são contornados, indo direto para a geração do *bitstream*, o qual por sua vez realiza previamente a síntese e implementação do projeto. A estrutura do projeto criado é então exportada para a ferramenta SDK onde é desenvolvido o *Software* e então passado para a ZYBO *board.*

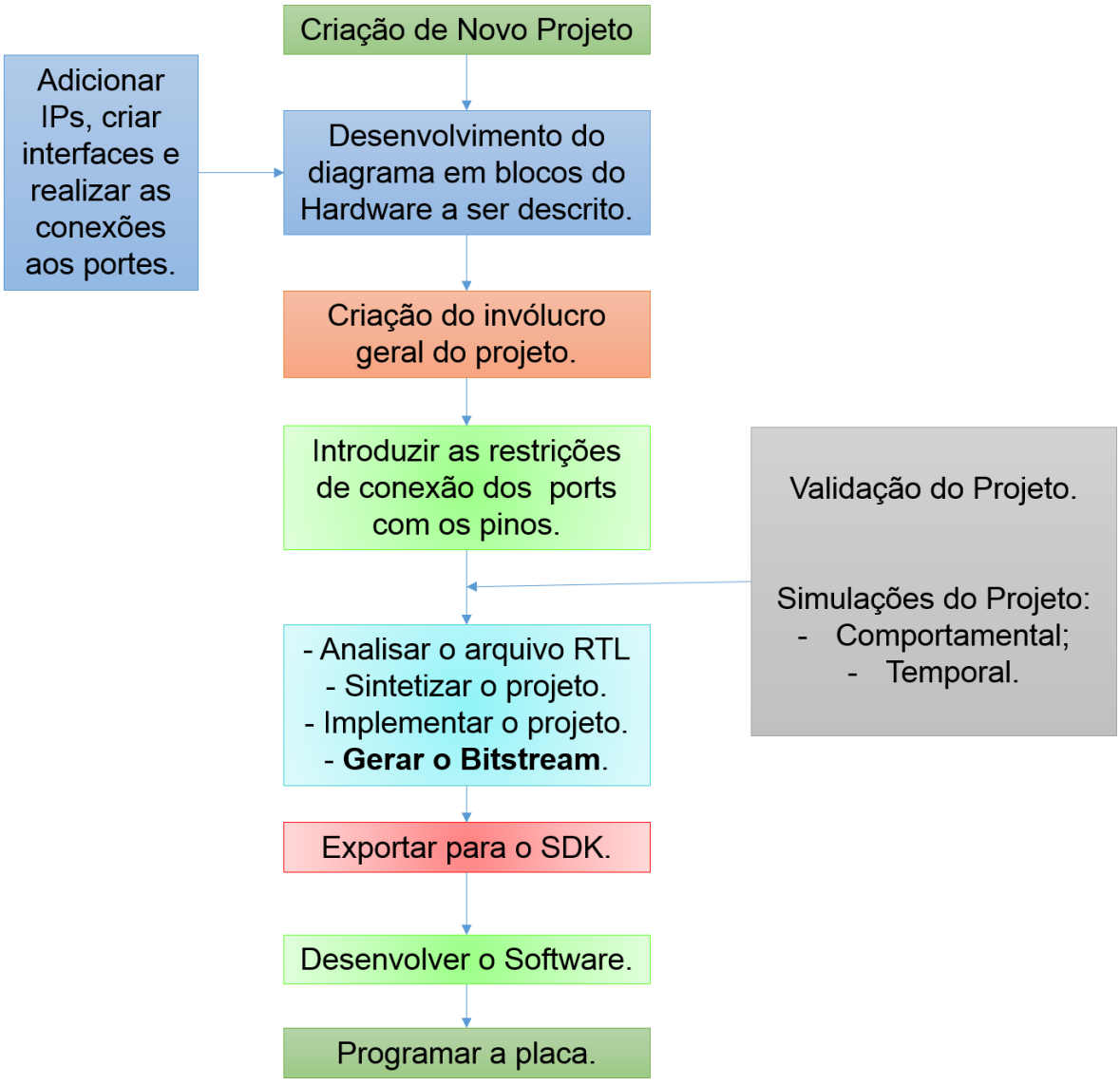


Figura 27 - Diagrama em blocos do fluxo de projeto Hardware/Software.

6. CONCLUSÕES

Durante o período de desenvolvimento deste trabalho foi possível constatar na prática a grande evolução dos dispositivos e ferramentas perante a disponibilidade de materiais que simplifiquem a iniciação de conhecimento para o desenvolvimento das novas tecnologias. Após o período de um ano letivo em busca de informações para a criação dos documentos, próximo as datas finais somente foi possível encontrar algumas informações, de forma que a tecnologia de topo já foi trocada, sendo abordado as novas ferramentas de desenvolvimento de sistemas em alto nível.

Com o desenvolvimento dos documentos foi possível criar um material para a iniciação nessa tecnologia de forma clara e prática. Os documentos permitiram seguir uma linha de projeto constante e ao final, sem dificuldades, obter a conclusão do projeto proposto pelo tutorial em questão.

O desenvolvimento do terceiro tutorial confirmou o pressuposto do material ser utilizado como base para projetos futuros, uma vez que para a sua realização é necessário a utilização das etapas de projeto descritas no primeiro tutorial.

A divisão do tutorial contribui de forma que o usuário mesmo não possuindo grandes conhecimentos da tecnologia, possa seguir de forma passo-a-passo e obter sucesso na criação do projeto. Por seu lado, os usuários que possuem conhecimentos mais avançados

podem compreender a dinâmica de cada uma das etapas necessárias presente no fluxo de projeto.

A decisão da escolha do idioma de escrita dos tutoriais em inglês se mostrou correta, pois a grande maioria dos materiais encontrados nessa área estão disponíveis em inglês e a falta de material disponível não foi relacionado a uma baixa influência da língua nessa área, pois mesmo em inglês não existiam documentos produzidos sobre alguns dos aspetos abordados.

Para trabalhos futuros verificam-se dois grandes potenciais; primeiramente, no desenvolvimento de novos documentos (tutoriais) com o intuito de aprofundar na tecnologia já abordada, explorando os demais periféricos do kit ZYBO e outras vertentes de projeto da ferramenta VIVADO; em segundo, no desenvolvimento de documentos (tutoriais), visando a atualização constante de materiais para as novas tecnologias, como as novas ferramentas de desenvolvimento de *Software* em alto nível para dispositivos SoC e MPSoC.

Referências Documentais

[1] XILINX, "Vivado Design Suite," XILINX, [Online]. Available: http://www.xilinx.com/products/design-tools/vivado.html. [Accessed 14 Set 2015].

[2] Xilinx, "Zynq-7000 Silicon Devices," [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/silicon-devices.html. [Acedido em 16 Nov 2015].

[3] ARM, "The ARM Cortex-A9 Processors," 2009. [Online]. Available: http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf. [Acedido em 16 Nov 2015].

[4] R. Schaller, "Moore's law: past, present and future," *Spectrum IEEE,,* 1997.

[5] DIGILENT, "ZYBO Zynq™-7000 Development Board," [Online]. Available: http://digilentinc.com/Products/Detail.cfm?NavPath=2,400,1198&Prod=ZYBO. [Acedido em 16 Nov 2015].

[6] Computer History Museum, "The silicon Engine: A Timeline of Semiconductors in Computers," 1833-1979. [Online]. Available: http://www.computerhistory.org/semiconductor/timeline.html. [Acedido em 15 Set 2015].

[7] J. Kilby, "The Chip that Jack Built," Texas Instruments, 12 Set 1958. [Online]. Available: http://www.ti.com/corp/docs/kilbyctr/jackbuilt.shtml. [Acedido em 15 Set 2015].

[8] R. Noyce, "Solid-State Micrologic Elements," *Solid-State Circuits Conference. Digest of Technical Papers. 1960 IEEE International* , pp. 82-83, 10-12 Fev 1960.

[9] J. Serrano, "Introduction to FPGA design," *CAS - CERN Accelerator School: Course on Digital Signal Processing,* pp. 231-247, 09 Jun 2007.

[10] K. FRANZ, "History of the FPGA," Digilent, 16 Jan 2015. [Online]. Available: https://blog.digilentinc.com/index.php/history-of-the-fpga/. [Acedido em 15 Set 2015].

[11] M. G. d. O. Gericota, *Metodologias de teste para FPGAs (Field Programmable Gate Arrays) integradas em sistemas reconfiguráveis,* Porto, 2003.

[12] M. Sachdev, Defect Oriented Testing for CMOS Analog and Digital Circuits, Kluwer, 1998.

[13] W. S. Carter, "The Evolution of Programmable Logic. Symposium on VLSI Circuits Digest of," *IEEE,* 01 Jun 1991.

[14] J. R. Stephen Brown, *Architecture of FPGAs and CPLDs: A Tutorial,* Department of Electrical and Computer Engineering University of Toronto, 1996.

[15] A. K. Sharma, Programmable Logic Handbook: Plds, Cplds, and Fpgas, New York: McGraw-Hill, 1998.

[16] ALTERA, *8 Macrocell EPLD,* 1984.

[17] ALTERA, "MAX 5000," 1988. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ds/m5000.pdf. [Acedido em 17 Set 2015].

[18] J. Rose, A. El Gamal e A. Sangiovanni-Vincentelli, "Architecture of field-

programmable gate arrays," *IEEE*, Jul 1993.

[19] J. Lazzaro, "Altera Parts History," [Online]. Available: http://www-inst.eecs.berkeley.edu/~cs294-59/fa10/resources/Altera-history/Altera-history.html. [Acedido em 17 Set 2015].

[20] J. Lazzaro, "Xilinx Part Family History," [Online]. Available: http://www-inst.eecs.berkeley.edu/~cs294-59/fa10/resources/Xilinx-history/Xilinx-history.html. [Acedido em 17 Set 2015].

[21] ALTERA, [Online]. Available: https://www.altera.com/. [Acedido em 17 Set 2015].

[22] XILINX, [Online]. Available: http://www.xilinx.com/. [Acesso em 17 Set 2015].

[23] Intel, "Product Brief: 6th Gen Intel® Core™ Processor Platform," 2015. [Online]. Available: http://www.intel.com.br/content/www/br/pt/processors/core/6th-gen-core-family-mobile-brief.html. [Acedido em 17 Set 2015].

[24] XILINX, "What is a FPGA? Field Programmable Gate Array (FPGA)," [Online]. Available: http://www.xilinx.com/fpga/. [Acedido em 18 Set 2015].

[25] M. Maxfield, "ASIC, ASSP, SoC, FPGA – What's the Difference?," 23 Jun 2014. [Online]. Available: http://www.eetimes.com/author.asp?section_id=36&doc_id=1322856. [Acedido em 18 Set 2015].

[26] BDTI, "Analysis: Qualcomm's 1 GHz ARM "Snapdragon"," 05 Dez 2007. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1275485&. [Acesso em 18 Set 2015].

[27] K. FRANZ, "What Is an IP and How Do You Create One?, Digilent," 23 Jan 2015. [Online]. Available: https://blog.digilentinc.com/index.php/what-is-an-ip-and-how-do-you-create-one/. [Acedido em 18 Set 2015].

[28] XILINX, "MicroBlaze Soft Processor Core," [Online]. Available: http://www.xilinx.com/tools/microblaze.htm. [Acesso em 18 Set 2015].

[29] R. A. E. M. A. E. a. R. W. S. L. H. Crockett, The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC, Glasgow, Scotland: University of Strathclyde, 2014.

[30] G. Radin, "THE 801 MINICOMPUTER," *IBM Thomas J. Watson Research Center*, 11 Nov 1981.

[31] ARM, "ARM Processor Architecture," ARM, [Online]. Available: http://www.arm.com/products/processors/instruction-set-architectures/index.php. [Acedido em 05 Out 2015].

[32] ALTERA, "Architecture Matters: Choosing the Right SoC FPGA for Your Application," Nov 2013. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01202-embedded-system-soc-design-considerations.pdf. [Acedido em 10 Nov 2015].

[33] S. Jeeawoody, "New Tools Take the Pain Out of FPGA Synthesis," *Xcell Journal* , 2012.

[34] IEEE STANDARD ASSOCIATION, "1800-2012 - IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language," [Online]. Available: http://standards.ieee.org/findstds/standard/1800-2012.html. [Acesso em 13 Nov 2015].

[35] C. Maxfield, FPGA World Class Design, Burlington: Elsevier, 2009.

[36] G. R. Smith, FPGA 101: Everything you need to know to get started, Elsevier, 2010.

[37] Jürgen Teich, "Hardware/Software Codesign: The Past, the Present, and Predicting the Future," *Proceedings of the IEEE,* 2012.

[38] A. C. P. Shiv Prakash, "Synthesis of Application-Specifc Heterogeneous Multiprocessor Systems," *Jounal of Parallel and Distributed Computing,* 1992.

[39] M. A. Dias, "Co-projeto de hardware/software para correlação de imagens," São Carlos, 2011.

[40] N. F. ,. V. C. Liem C., "System-on-a-chip cosimulation and compilation," *IEEE Des. Test Comput.,* 1997.

[41] T. J. Roloff S. Hannig F., "Approximate time functional simulation of resource-aware programming concepts for heterogeneous MPSoCs," em *Design Automation Conference (ASP-DAC)*, Sydney, NSW, 2012.

[42] Xilinx, "Vivado Design Suite User Guide: Embedded Processor Hardware Design," [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug898-vivado-embedded-design.pdf. [Acedido em 16 Nov 2015].

[43] Xilinx, "Expanding the All Programmable SoC Portfolio," [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc.html. [Acedido em 16 Nov 2015].

[44] Digilent, "Ref Manual," 2015. [Online]. Available: https://reference.digilentinc.com/zybo:refmanual. [Acedido em 16 Nov 2015].

[45] S. M., "Xilinx Redefines State of the Art With New 7 Series FPGAs," *Xcell Journal,* 2010.

[46] Xilinx, "7 Series FPGAs Overview," 2014. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. [Acedido em 16 Nov 2015].

[47] Xilinx, "AXI Reference Guide," 2012. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf. [Acedido em 16 Nov 2015].

[48] É. S. d. Cunha, "Desenvolvimento de Sistemas Embarcados utilizando Plataformas FPGA com Dispositivos ARM," Porto Alegre, 2014.

[49] Xilinx, "AXI Reference Guide," 2011. [Online]. Available: http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf. [Acedido em 16 Nov 2015].

[50] S. Larson, "VGA Controller (VHDL)," 01 Aug 2013. [Online]. Available: https://eewiki.net/pages/viewpage.action?pageId=15925278. [Acedido em 17 Nov 2015].

[51] Xilinx, "PlanAhead Design and Analysis Tool," [Online]. Available: http://www.xilinx.com/tools/planahead.htm. [Acedido em 16 Nov 2015].

[52] Xilinx, "ChipScope Pro 11.4 Software and Cores," [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/chipscope_pro_sw_cores_ug029.pdf. [Acedido em 16 Nov 2015].

[53] M. K. Chaitanya, "FPGA Design Flow," 2012. [Online]. Available: https://digitaltagebuch.wordpress.com/2012/11/26/fpga-design-flow/. [Acedido em

16 Nov 2015].

[54] Xilinx, “Vivado High-Level Synthesis,” [Online]. Available: http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html. [Acedido em 16 Nov 2016].

[55] Xilinx, “Vivado Design Suite User Guide, Design Flows Overview,” 30 Set 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_3/ug892-vivado-design-flows-overview.pdf. [Acedido em 17 Nov 2015].

[56] G. Estrin, “Organization of Computer Systems-the Fixed Plus Variable Structure Computer,” *IEEE*, p. pp: 33, 15 Dez 1960.

[57] J. Woldstad, "ZYBO Manual," 06 Mar 2015. [Online]. Available: https://reference.digilentinc.com/zybo:refmanual. [Accessed 11 Set 2015].

[58] XILINX, “Vivado Design Suite, User Guide: System-Level Design Entry,” 01 Jul 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug895-vivado-system-level-design-entry.pdf. [Acedido em 22 Oct 2015].

[59] XILINX, "Vivado Design Suite Synthesis," 24 June 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug901-vivado-synthesis.pdf. [Accessed 11 Set 2015].

[60] XILINX, "Vivado Design Suite Implementation," 24 June 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug904-vivado-implementation.pdf. [Accessed 11 Set 2015].

Anexo A. Tutorial I - Vivado 2014.x/2015.x Quick Start Tutorial to ZYBO Board

VIVADO 2014.X/2015.X QUICK START TUTORIAL TO ZYBO BOARD

Héber Miguel dos Santos

Instituto Superior de Engenharia do Porto

Department of Electrical Engineering

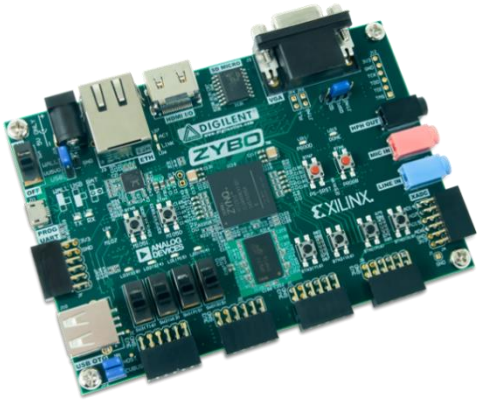
Master in Electrical and Computer Engineering

NOV/2015

Brief Introduction:

The Vivado Design Suite that is explored in this tutorial delivers a SoC-strength, IP-centric and system-centric, it's the next generation development environment that has been built from the ground up to address the productivity bottlenecks in system-level integration and implementation. The Vivado Design suite is a Generation Ahead in overall productivity, ease-of-use, and system level integration capabilities [1].

The tutorial base board used is the ZYBO Board (Zynq Board) that is a feature-rich, ready-to-use, entry-level embedded software and digital circuit development platform built around the smallest member of the Xilinx Zynq-7000 family, the Z-7010. The Z-7010 is based on the Xilinx All Programmable System-on-Chip (AP SoC) architecture, which tightly integrates a dual-core ARM Cortex-A9 processor with Xilinx 7-series Field Programmable Gate Array (FPGA) logic. The Zynq Z-7010 coupled with the rich set of multimedia and connectivity peripherals available on the ZYBO, it can host a whole system design. The on-board memories, video and audio I/O, dual-role USB, Ethernet and SD slot will have your design up-and-ready with no additional hardware needed. Additionally, six Pmod connectors are available to put any design on an easy growth path [2].



Summary:

Figure List:

General software, hardware, files and Information needed:

- Xilinx Vivado 2014.x/2015.x;
- ZYBO Zynq-7000 Development Board;
- ZYBO Board reference Manual;
- ZYBO Master XDC;
- Micro USB cable.

Software and hardware needed in specifics projects:

- SDK Webpack;
- Power supply 5V;
- Monitor with the VGA and HDMI interfaces;
- Micro SD-card minimal 4Gb;

Important Reference Links:

https://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,1198&Prod=ZYBO

http://www.zynqbook.com/

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug910-vivado-getting-started.pdf

Tutorial's Knowhow:

This tutorial explains, step-by-step, how to implement a Binary Counter in a Xilinx ZYBO Zynq-7000 Board from Digilent, using the new Vivado 2014.x/2015.x Suite.

Starting the Vivado Software:

1. To start Vivado, double-click the desktop icon (Figure 1):

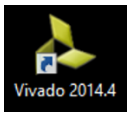


Figure 1 – Vivado Icon.

Or start the Vivado from the Start menu by selecting: Menu Start → All Programs → Xilinx Design Tools → Vivado 201x.x. The main page looks like Figure 2:

**Note: Your start-up path is set during the installation process and may differ from the one above.*

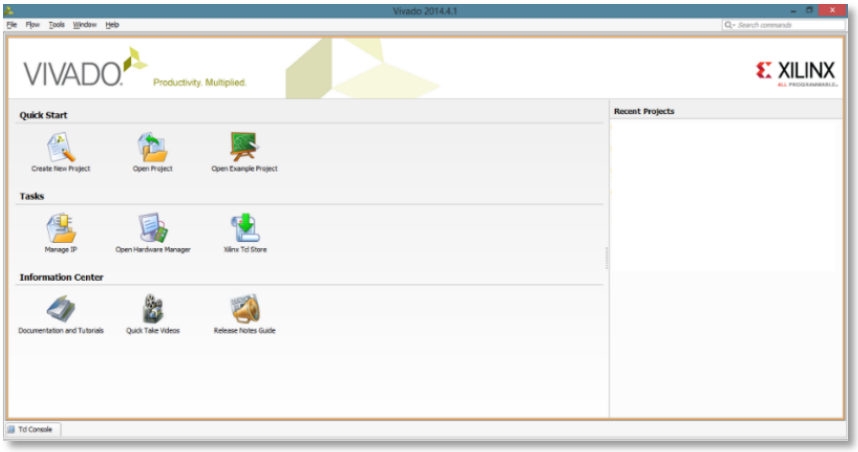


Figure 2 – Vivado Started.

Accessing Help:

At any time during the tutorial, you can access the online help for additional information about the Vivado software and related tools:

1. To open Help, follow Figure 3 instructions
2. Launch the Vivado Help Contents from the Vivado Menu → Help. It contains information about creating and maintaining your complete design flow in Vivado

Figure 3 – Vivado "Help".

Create a New Project:

Create a new Vivado project that will target the FPGA device on the ZYBO Board from Digilent.

1. Open Vivado and click on the icon "Create New Project" (Figure 4)

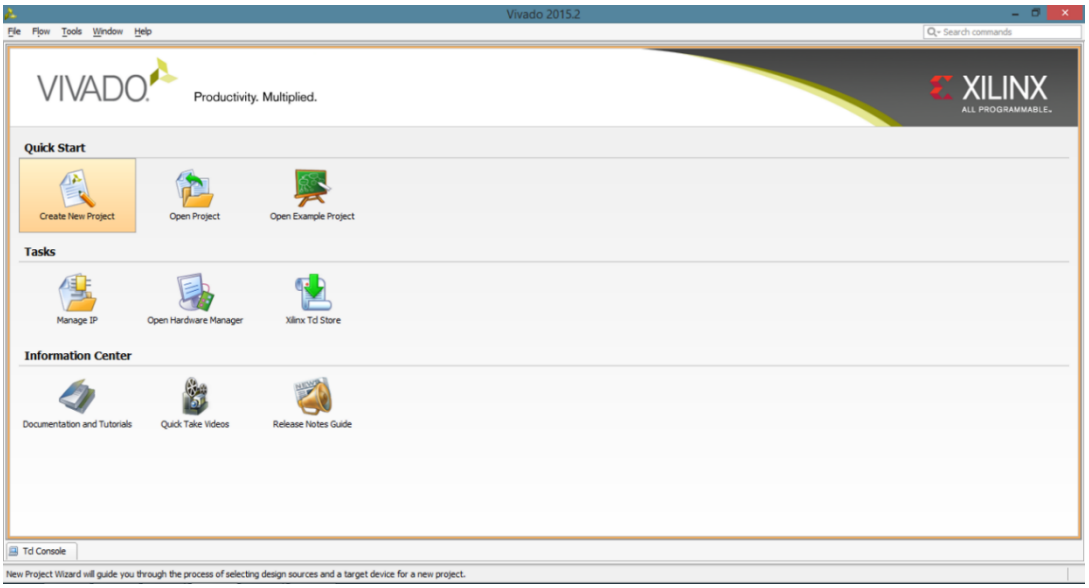


Figure 4 – Vivado Starting a New Project.

2. Click “Next” on the first page (Figure 5)

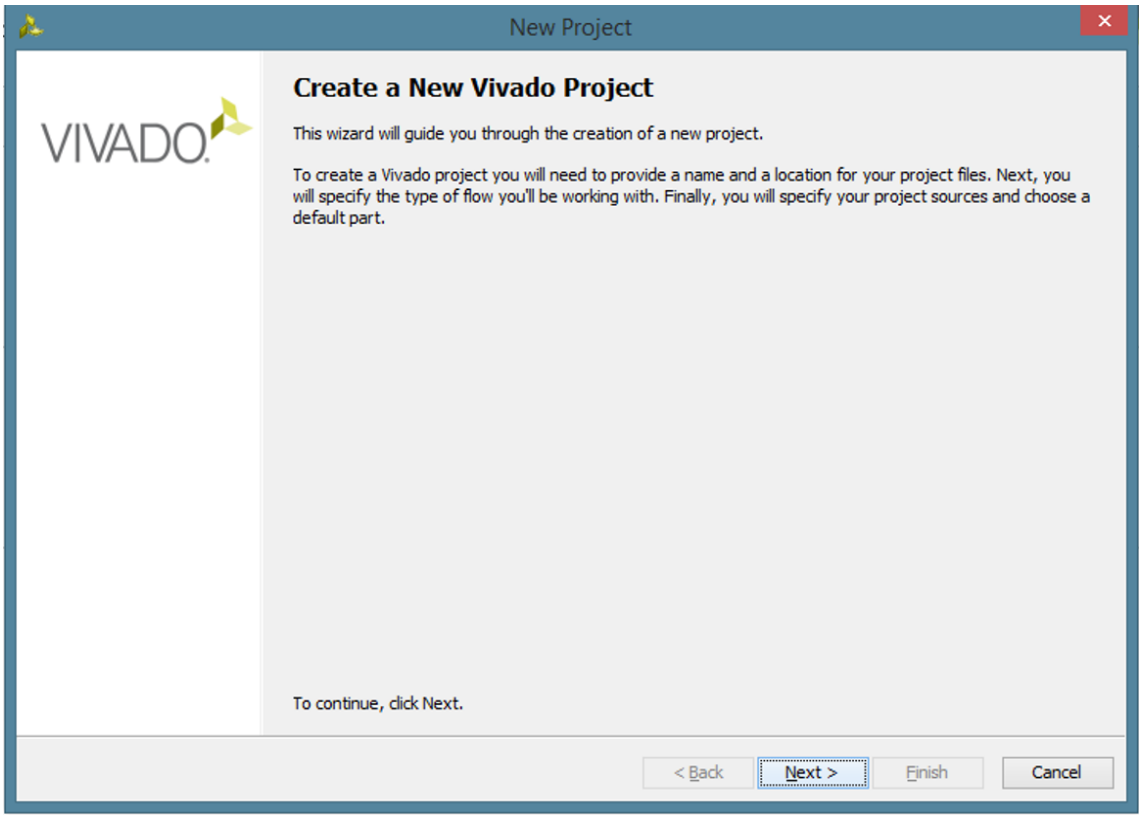


Figure 5 – Creating a New Project.

3. Enter a project name along with its location and click “Next” (Figure 6)

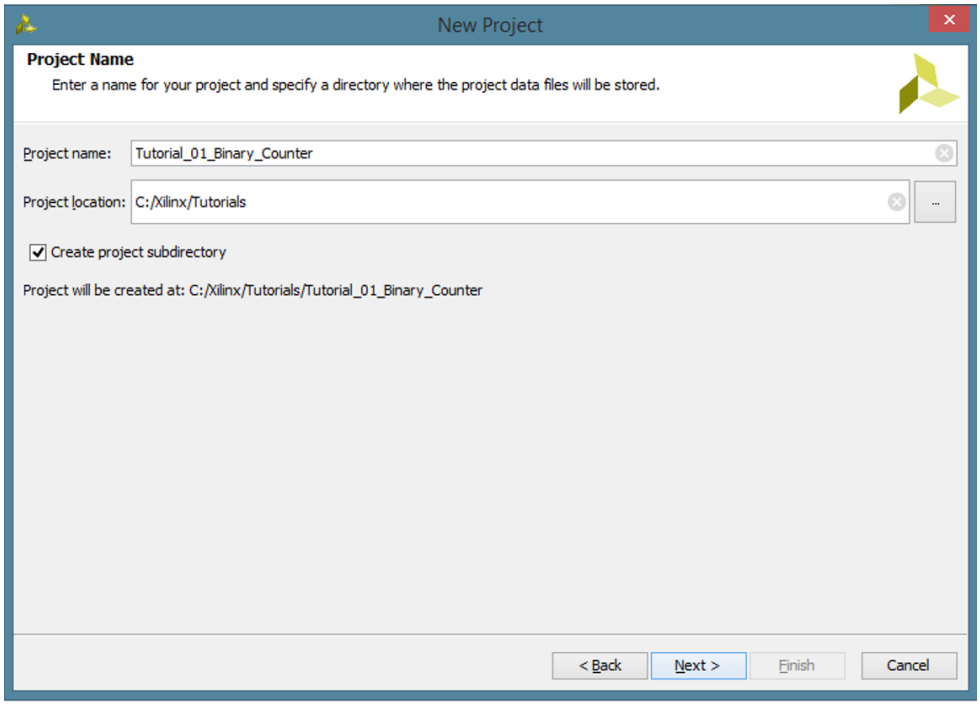


Figure 6 – Vivado name of new project.

4. Choose the correct project type option: "RTL Project" because in this design flow sometimes you need to use the information from the register transfer level (RTL) to do analyses or editions. Verify that the "Do not specify sources at this time" box is checked (Figure 7)

**Note: The "Register-Transfer-Level" create high-level representations of a circuit from the lower-level representations generated by the hardware description languages (HDLs) like Verilog and VHDL. It is a typical practice in modern digital design.*

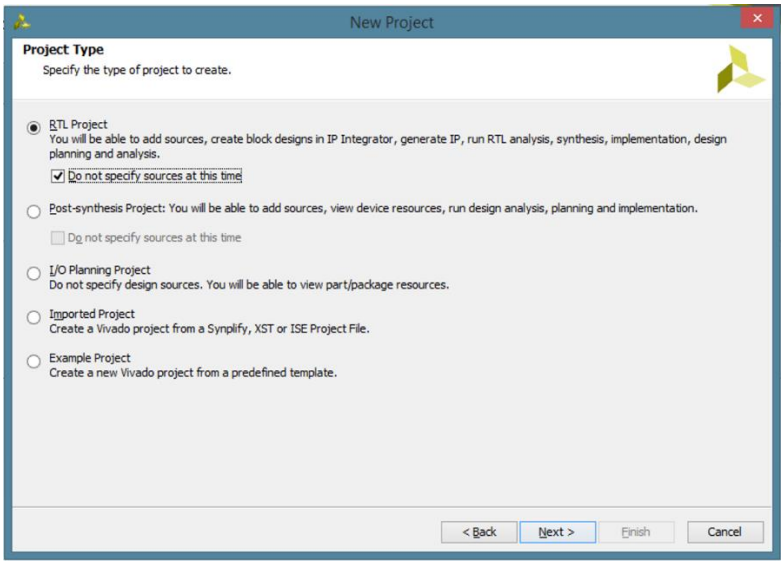


Figure 7 – Vivado project type of new project.

5. Select the correct part. For this tutorial, we are using the Zynq-7000 with the specifications shown below and in Figure 8. Click "Next" (Figure 8) and then click "Finish" (Figure 9):

- Product category: All
- Family: Zynq-7000
- Sub-Family: Zynq-7000
- Package: clg400
- Speed Grade: -1
- Temp Grade: C

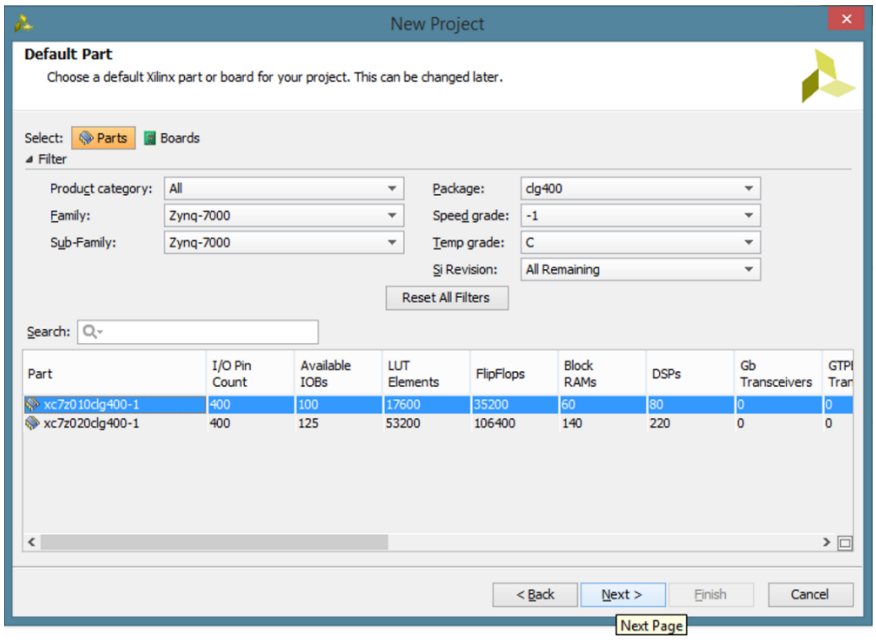


Figure 8 – Vivado default part.

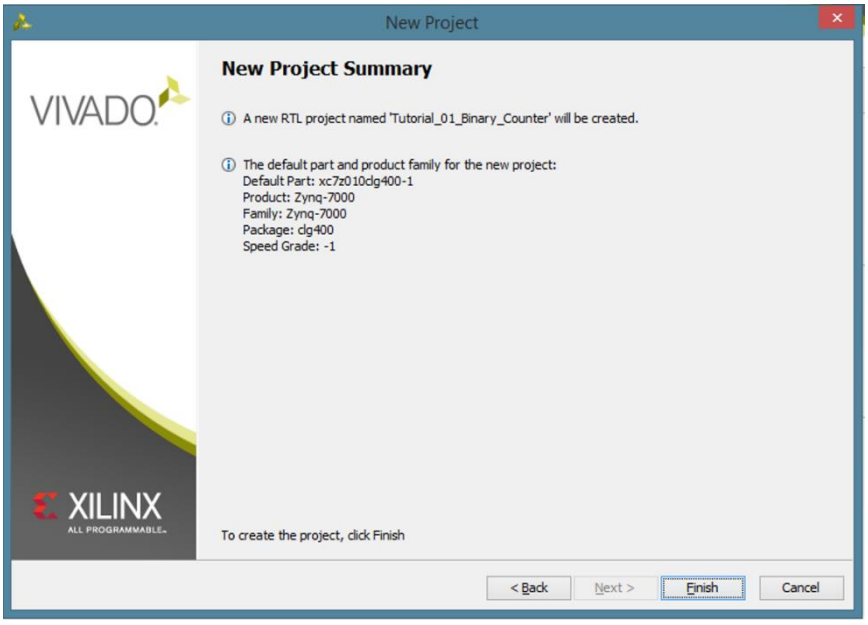


Figure 9 – New project details.

6. After a few seconds, your Project should be created and there will appear a window similar to the one shown on Figure 10

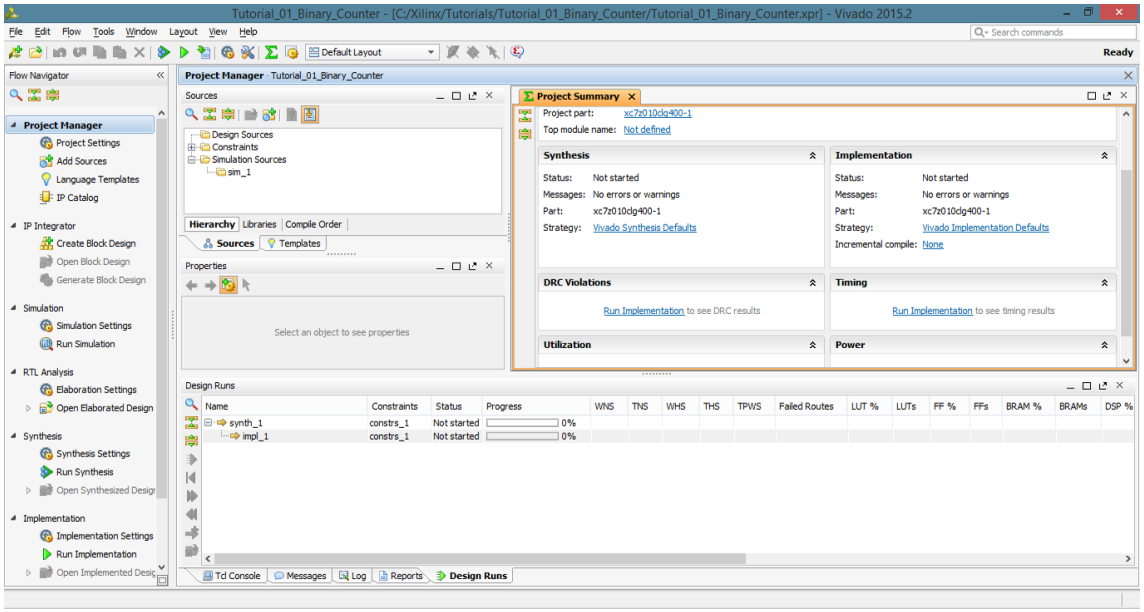


Figure 10 – Vivado first page of a new project.

Create a Block Design:

**Note: From now on all the steps will be based in a design flow to develop a simple 4 bits Binary Counter for easy comprehension of some tools and methods process.*

Create a Block Design to continue the project and then follow the steps.

1. Click on the “Create Block Design” button within the Flow IP Integrator Manager (Figure 11)

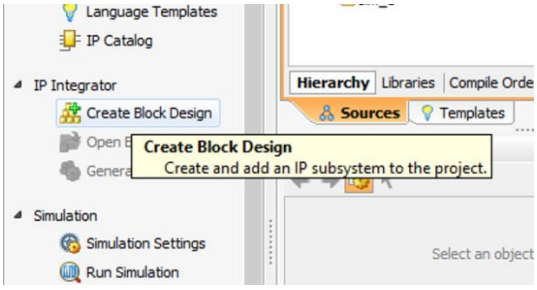


Figure 11 – Create Block Design.

2. Then specify the design name and click “OK” (Figure 12)

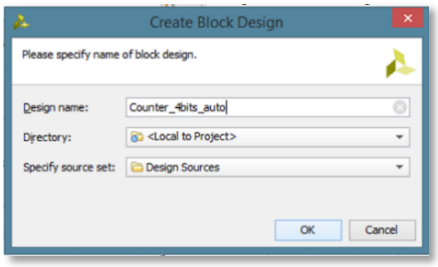


Figure 12 – Design Block Name.

3. Now a blank block diagram will appear (Figure 13)

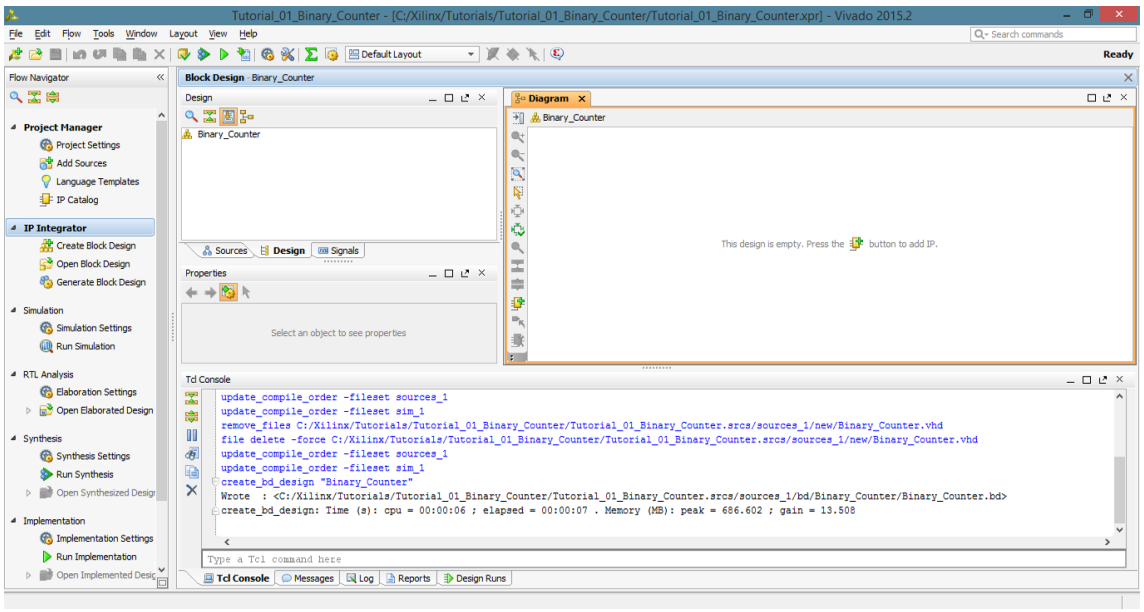


Figure 13 – Block Diagram Created.

Adding an IP integrator in a block design project:

1. Click on the icon selected in Figure 14 “Add IP”

Figure 14 – Add IP.

2. Then the IP Catalog will appear (Figure 15)

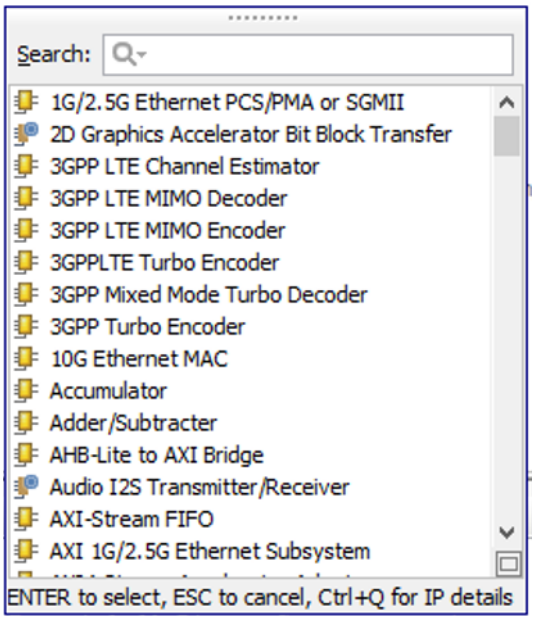


Figure 15 – IP Catalog.

3. On the “Search” zone write ‘Binary’ and double click on the select “Binary Counter” (Figure 16)

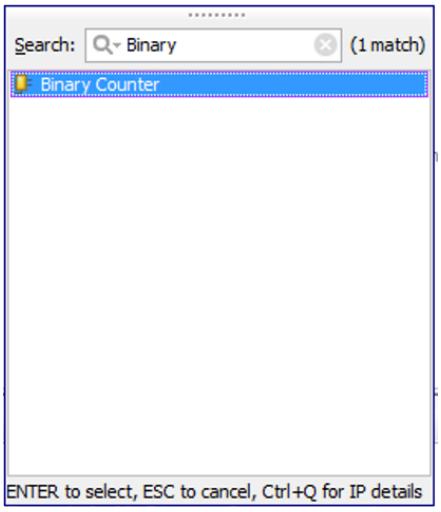


Figure 16 – IP Catalog filtered by Binary Word.

4. Then the block binary counter will appear (Figure 17)

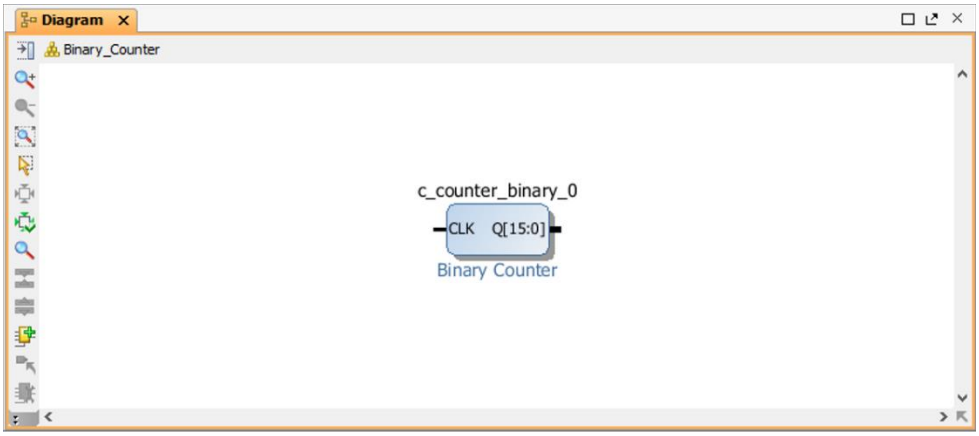


Figure 17 – Block Binary Counter.

5. Open the block Re-customize IP double clicking on “Block” (Figure 18)

Figure 18 – Block Binary Counter Selected.

6. Then the Re-Customize IP window will appear (Figure 19)

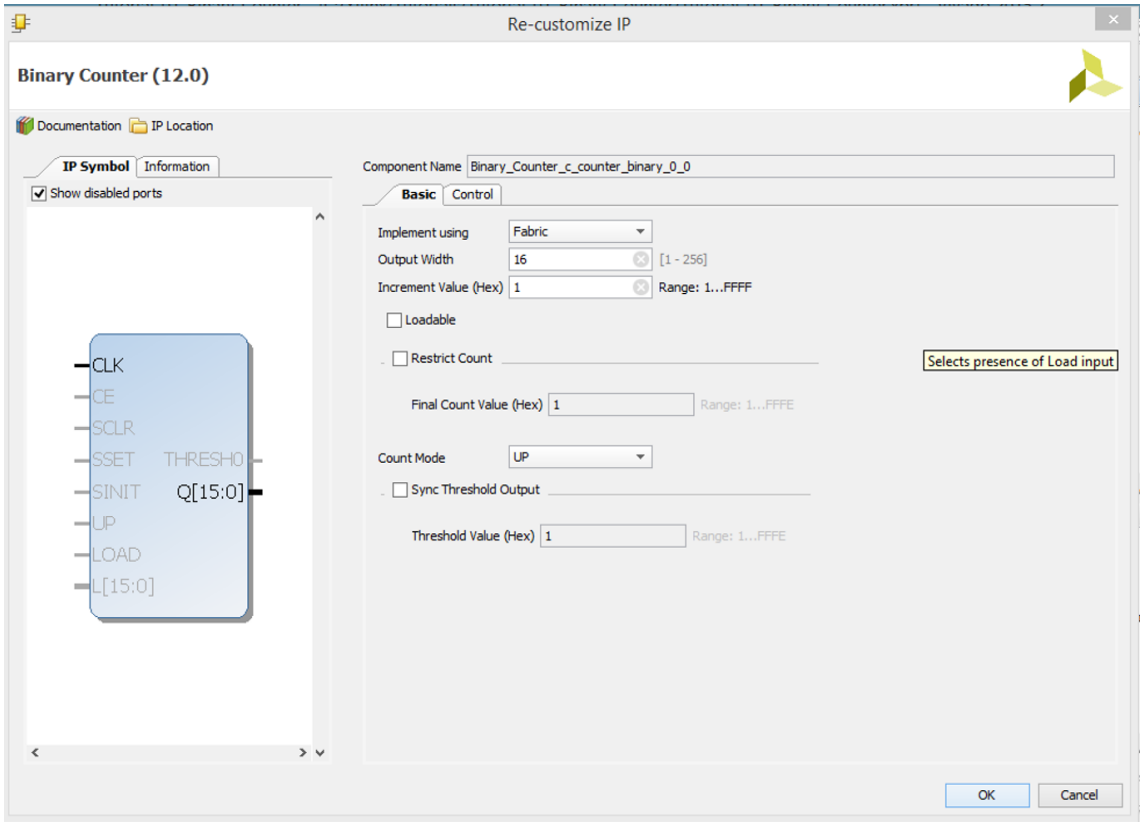


Figure 19 – Re-customize IP Binary Counter

7. As the project is a 4 Bits Binary Counter, edit the settings (Figure 20) and (Figure 21):
   - Output Width: 4
   - Count Mode: UPDOWN
   - Synchronous Clear: Checked (Figure 21)

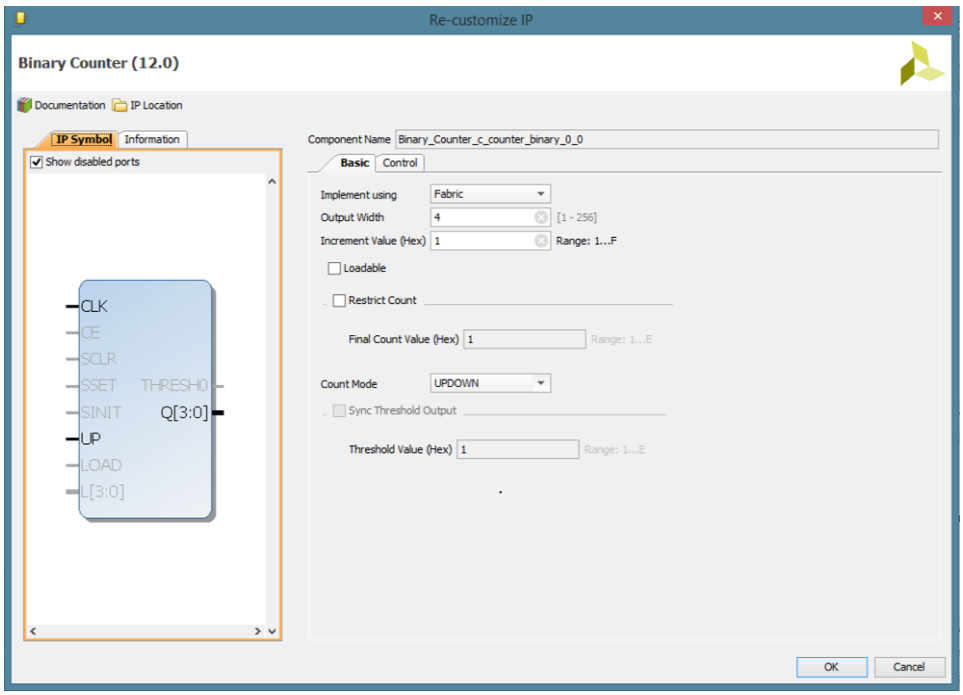


Figure 20 – Re-customize IP Basic Definitions.

8. And then click “OK” (Figure 21)

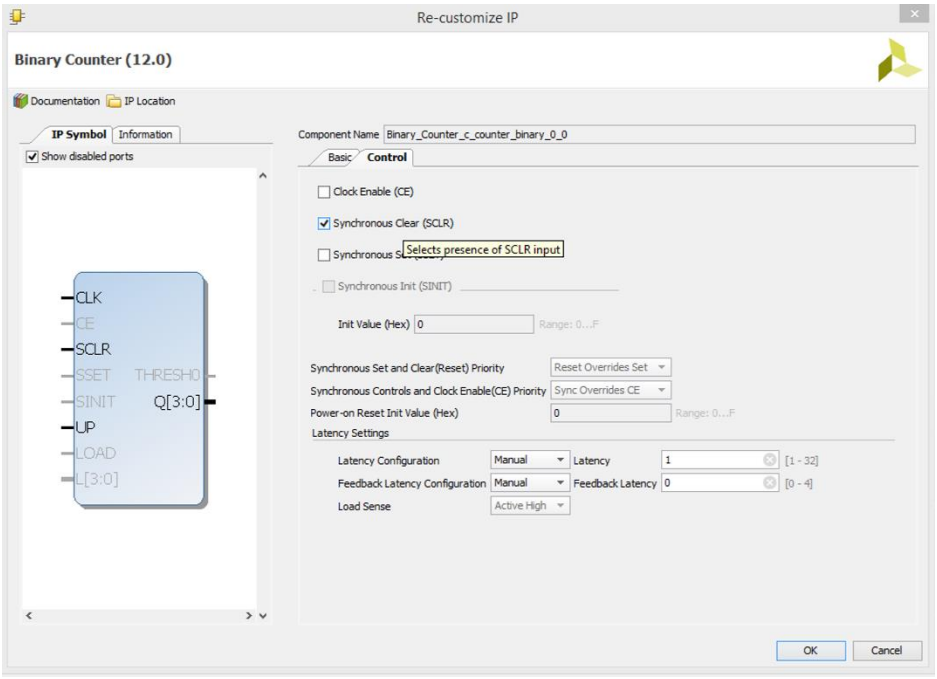


Figure 21 – Re-customize IP Definitions Control.

9. Then the Binary Counter Block will appear (Figure 22)

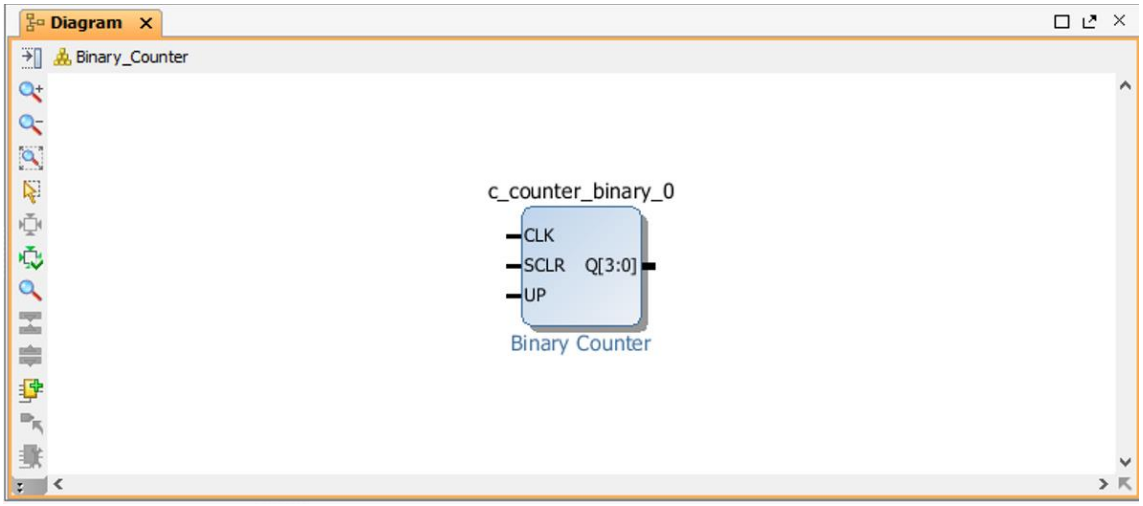


Figure 22 – Binary Counter Block edited.

Create Ports:

1. The next step is to create the Ports to connect the block and reference it to link it with the Zynq pins. To do so, right click within “Diagram” window and select “Create Port…” (Figure 23)

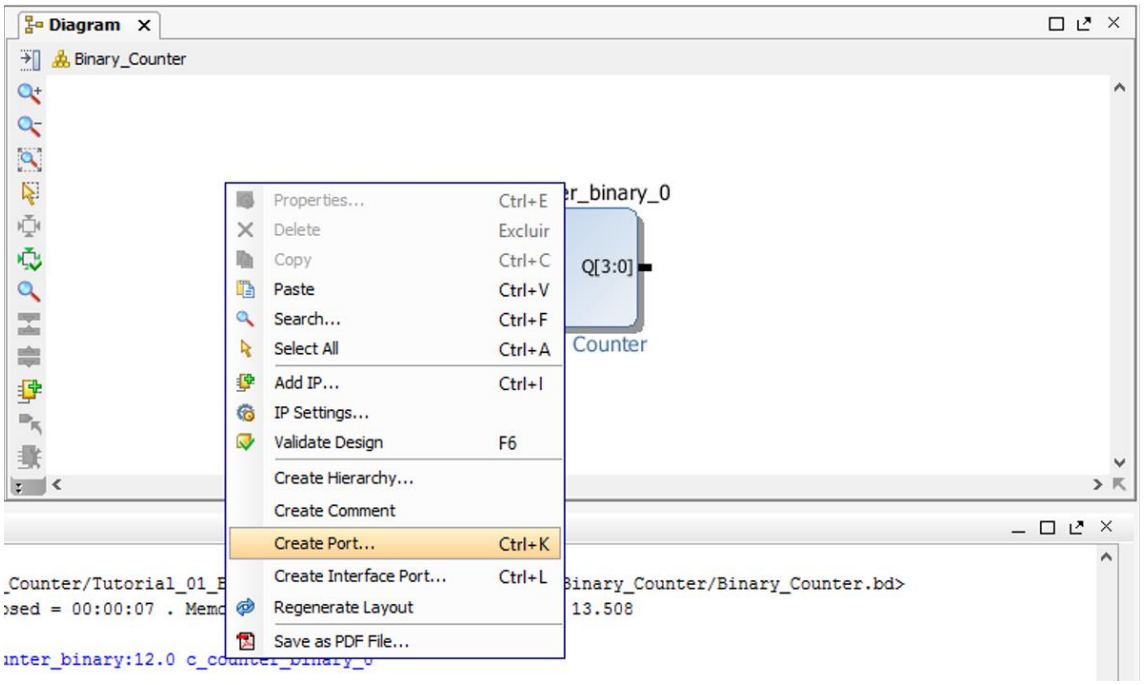


Figure 23 – Create Port.

2. A pop-up window, “Create Port”, appears (Figure 24). Fill in the settings of the ports shown on Figure 25. After finishing editing click “OK”. Return to step 1 of this section (Figure 23) and create the others Ports with their proper settings (Figure 26), (Figure 27) and (Figure 28) respectively

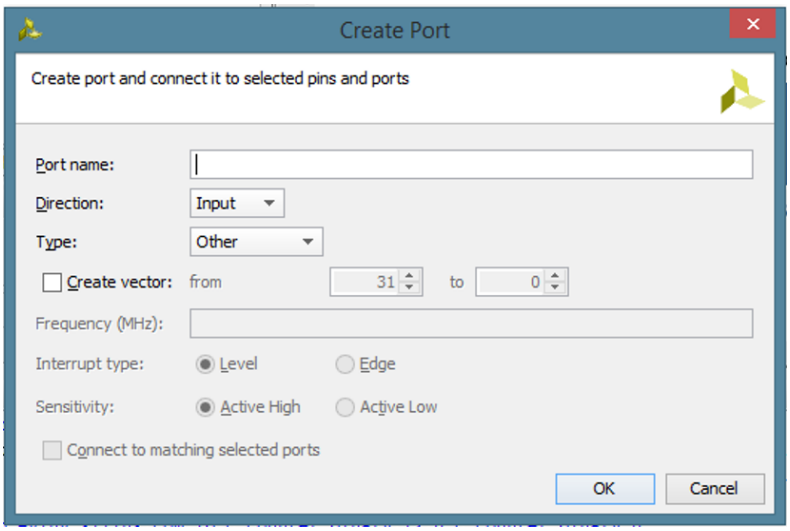


Figure 24 – Create Port Settings.

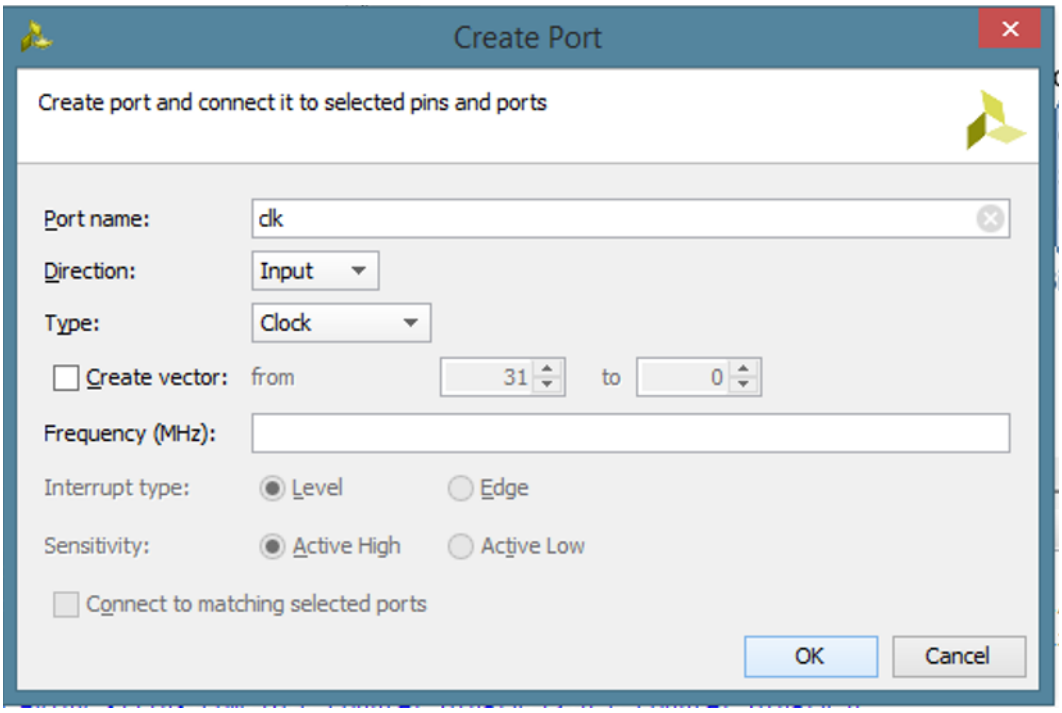


Figure 25 – Create Port clk Settings.

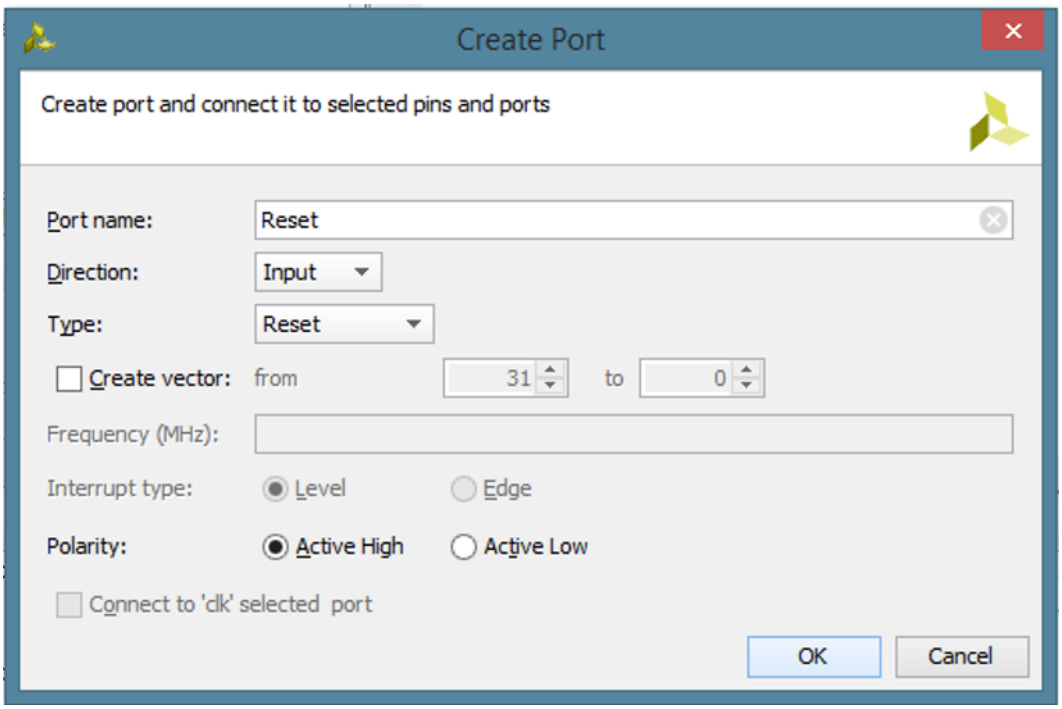


Figure 26 – Create Port Reset Settings.

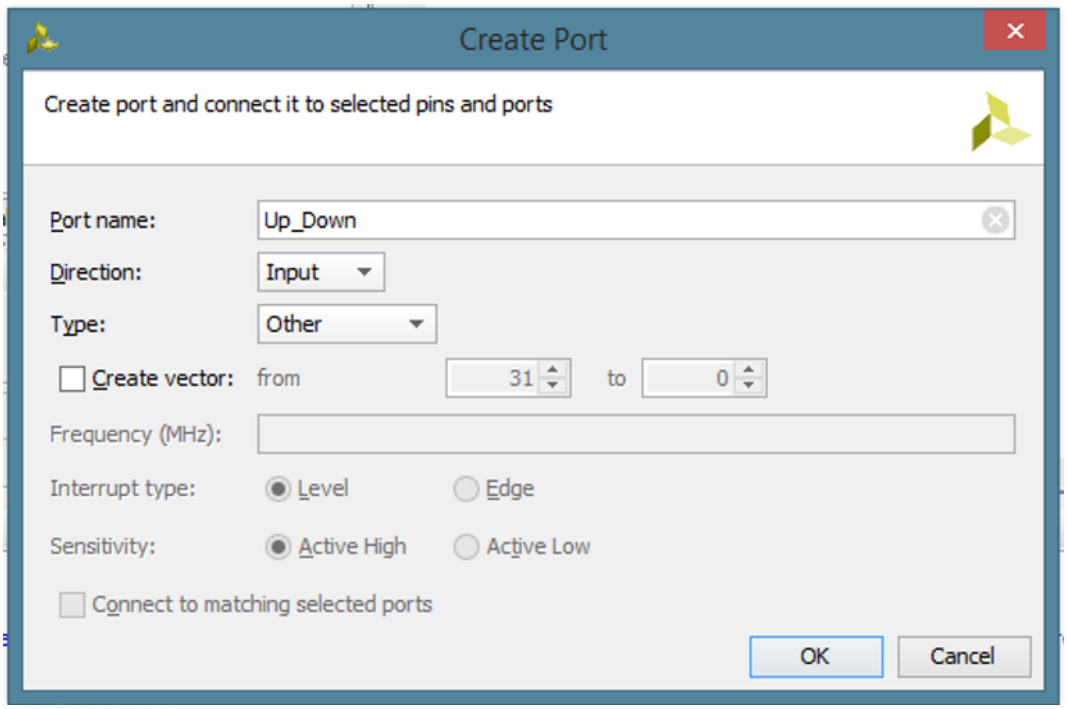


Figure 27 – Create Port Up_Down Settings.

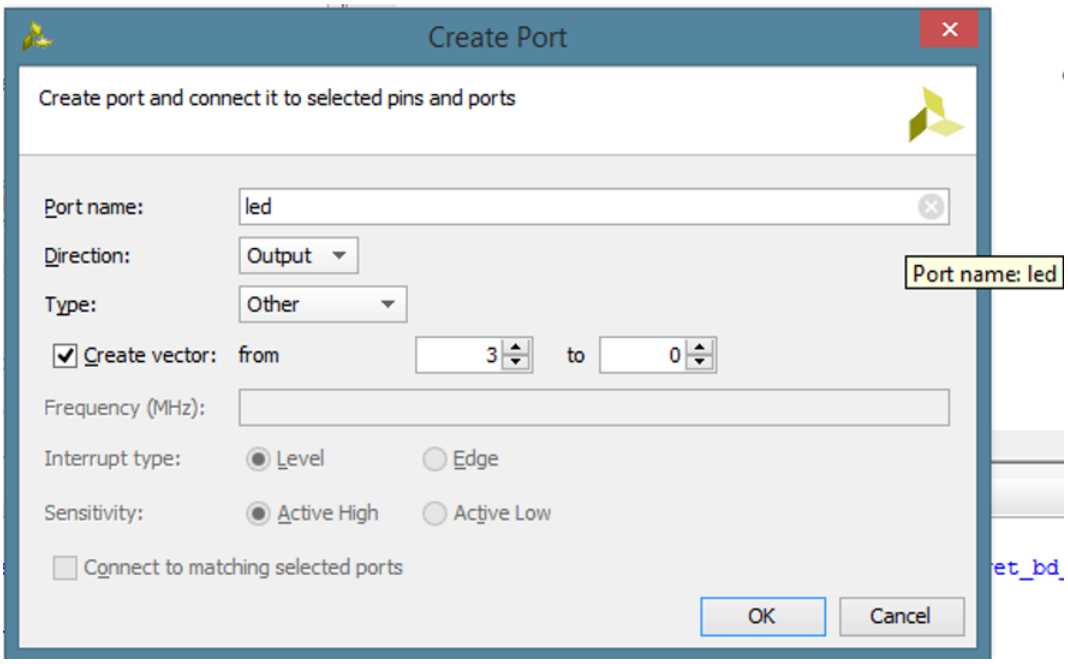


Figure 28 – Create Port led Settings.

3. With all the ports created (Figure 29), connect them to its respective pins by clicking and dragging a line (Figure 29), (Figure 30)

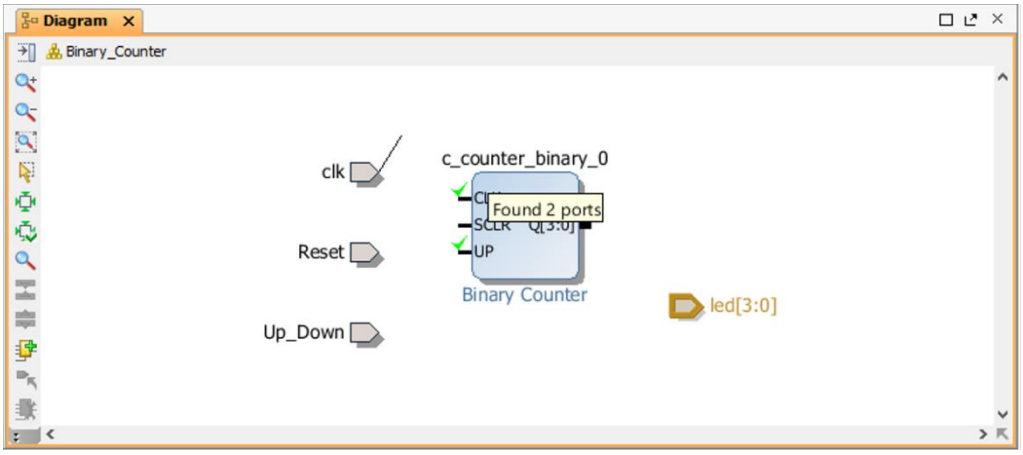


Figure 29 – Binary Counter Block, Ports Connecting.

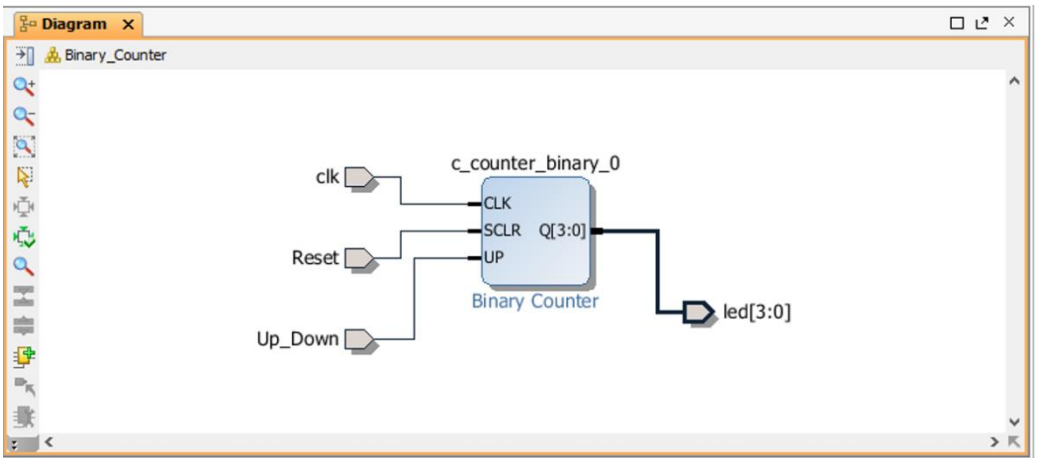


Figure 30 – Binary Counter Block, Ports Connected.

4. Just for aesthetics, right click within the “Diagram Window Tab” and click on the “Regenerate Layout” option (Figure 31). This action rearranges the block(s) and the port(s) (Figure 32)

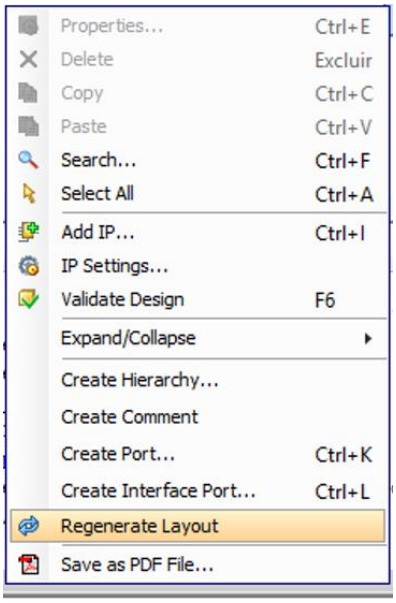


Figure 31 – Regenerate Layout.

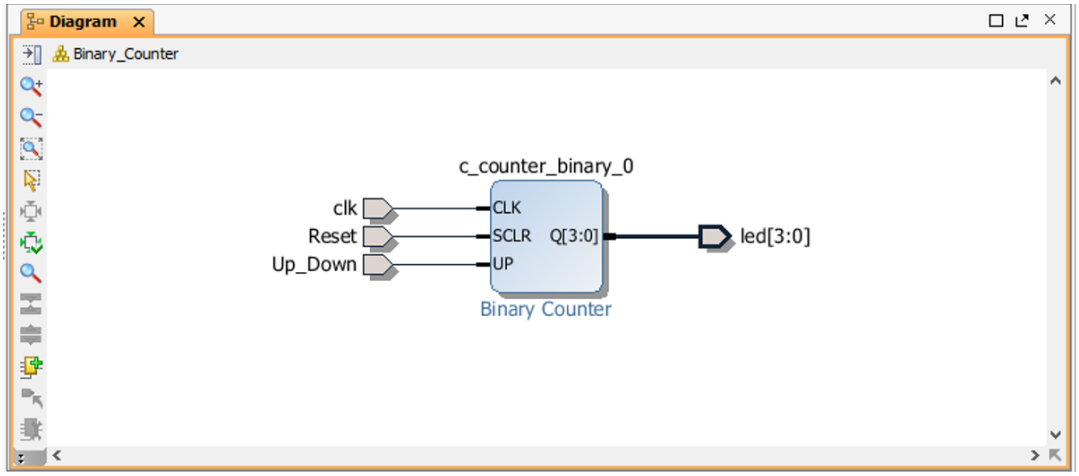


Figure 32 – Block Diagram, Binary Counter with ports.

**Note: After following all the above steps, it’s necessary to link the ports “clk” and “Reset”. Thus, the Reset pin is used as an adaptation of an asynchronous function that clears the Counter, which works as the reset.*

5. Then right click on “clk” port and select “External Port Properties” (Figure 33)

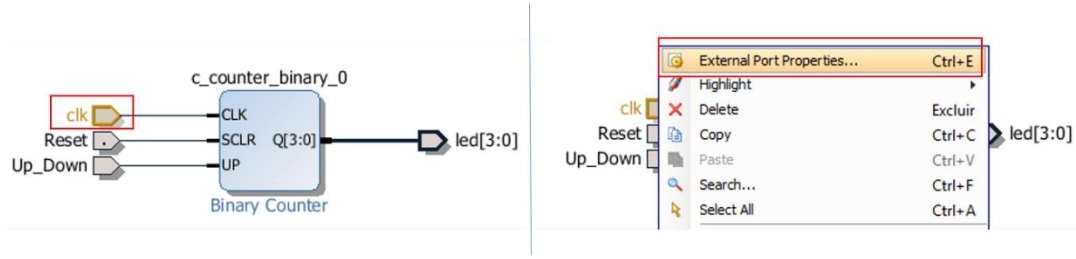


Figure 33 – External Port Properties option.

6. The window “External Port Properties” will be selected (Figure 34). Click on “Properties” tab (on the lower side) and click on “+” to visualize the options (Figure 35)

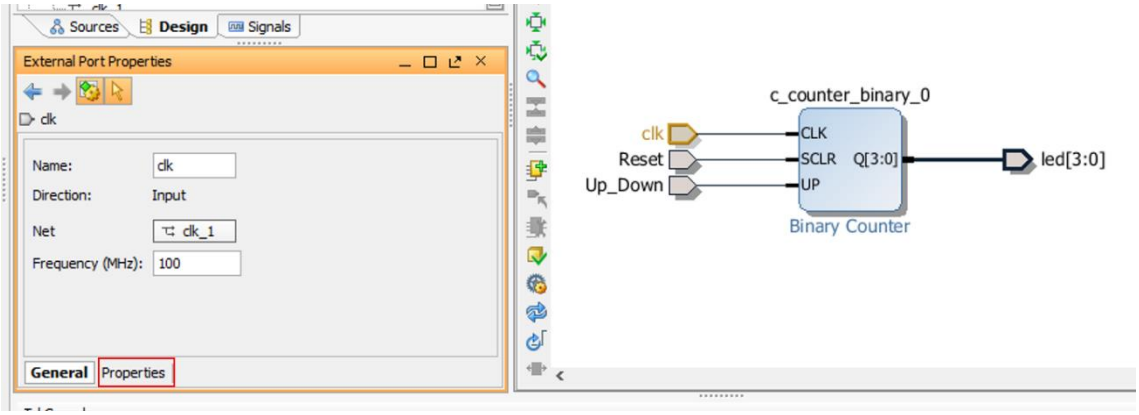


Figure 34- External Port Properties.

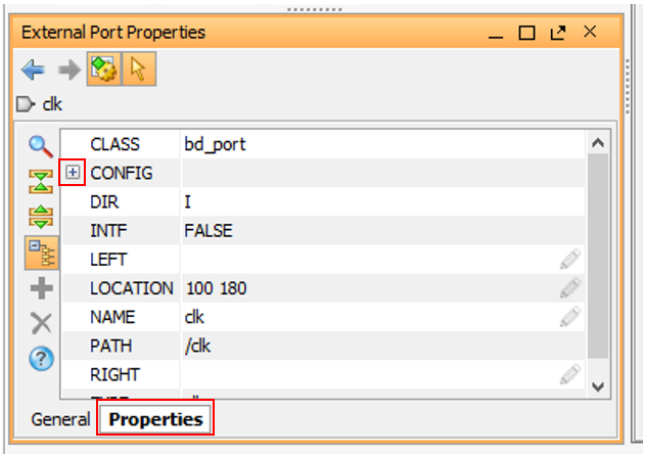


Figure 35 – External Port Properties (Properties).

7. Now within the “External Port Properties” window on the “Properties” tab, add the ‘Reset’ on the “CONFIG” → “ASSOCIATED_RESET” (Figure 36)

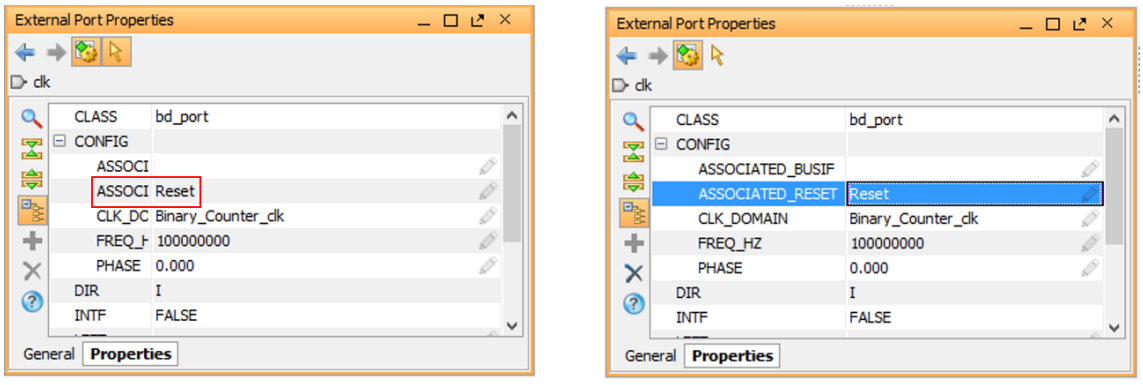


Figure 36 – Associated Reset.

8. Just to confirm the project design, right click within the “Diagram” window and click on “Validate Design” (Figure 37)

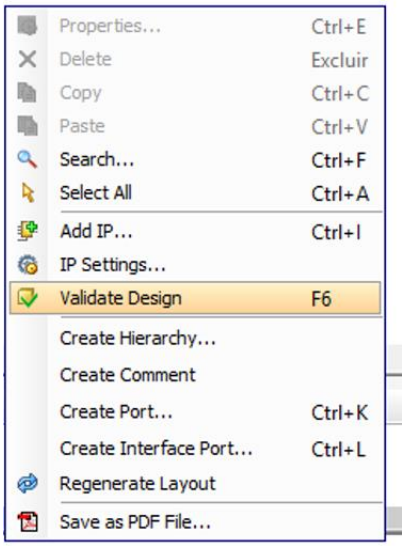


Figure 37 – Validate Design.

9. To finish this project save the block design by clicking on “Save button” (Figure 38)

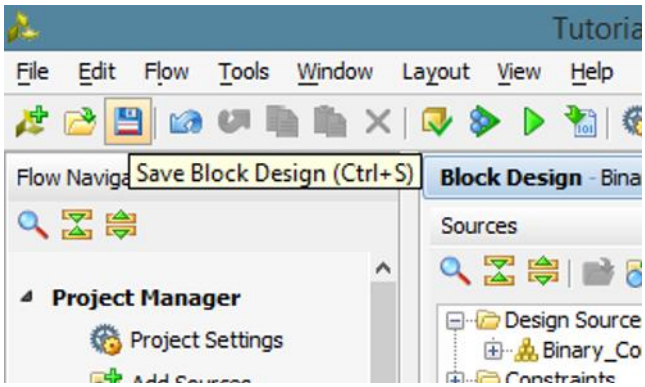


Figure 38 – Save Block Design.

Creating HDL Wrapper:

**Note: To continue the project in order to do some analyzes during this process, it is necessary to convert the simple Block Diagram created in a HDL (Hardware Description Language) Wrapper that will assign the Block Diagram to a main file of the project.*

1. On the “Sources window” right-click the Block Diagram created (“Binary Counter”) and then select “Create HDL Wrapper” (Figure 39)

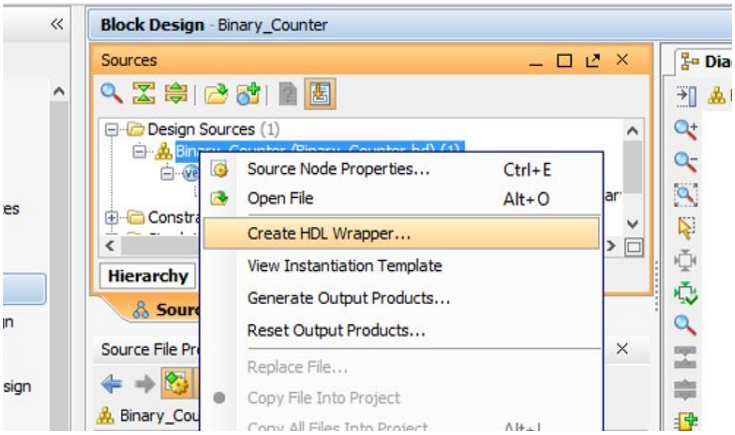


Figure 39 – Create HDL Wrapper.

2. Then Figure 40 will pop-up. Click “OK”

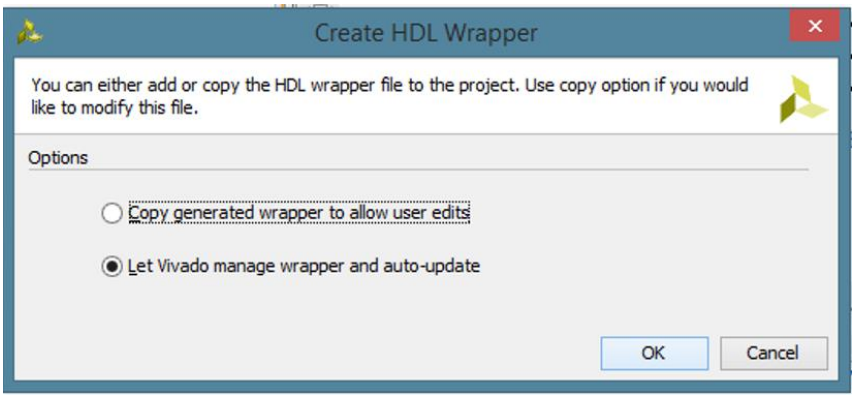


Figure 40 – Let Vivado Manage Wrapper and auto-update.

Simulating the Project:

**Note: To continue the design flow the next step is to verify the operation of the block by running Simulation. This example of simulation is an unusual basic mode that is used just when the project or module under development is small and simple, because it is based on the generation one by one of each signal. In another section of this tutorial, it will be presented another simulation method that is easier and more common for big projects.*

1. Click on the “Run Simulation” button within the Flow Navigator Project Manager (Figure 41) and then select “Run Behavioral Simulation” (Figure 42)

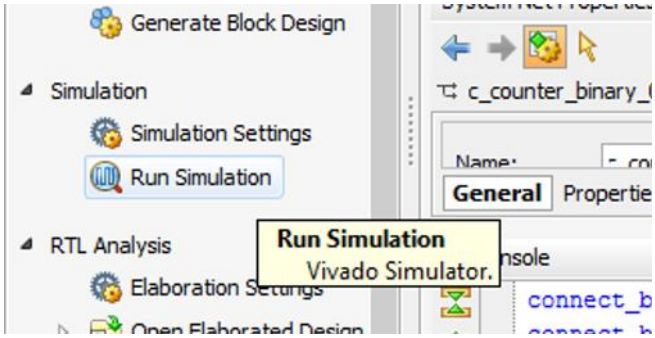


Figure 41 – Run Simulation.

**Note: The Behavioral Simulation is a simple operational simulation to verify if the block created is operational.*

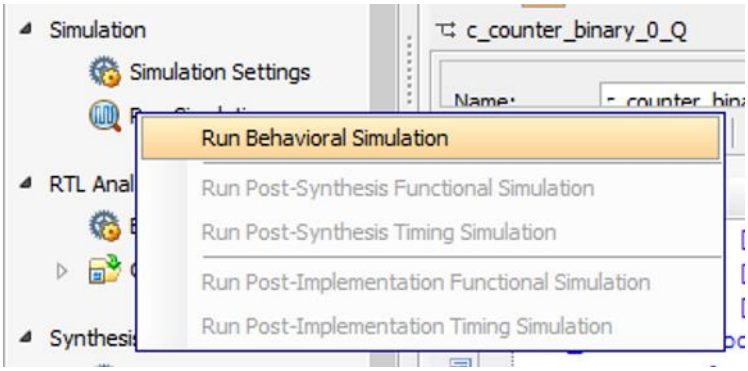


Figure 42 – Run Behavioral Simulation.

2. Now the simulation name can be changed or remain the same. Click on “OK” (Figure 43)

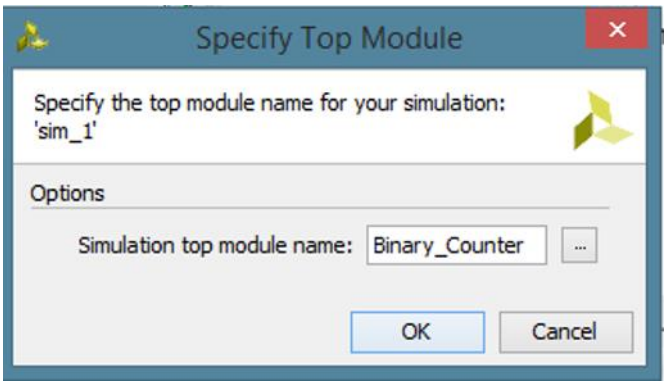


Figure 43 – Specify Top Module.

3. Then the Simulation Workspace will appear (Figure 44)

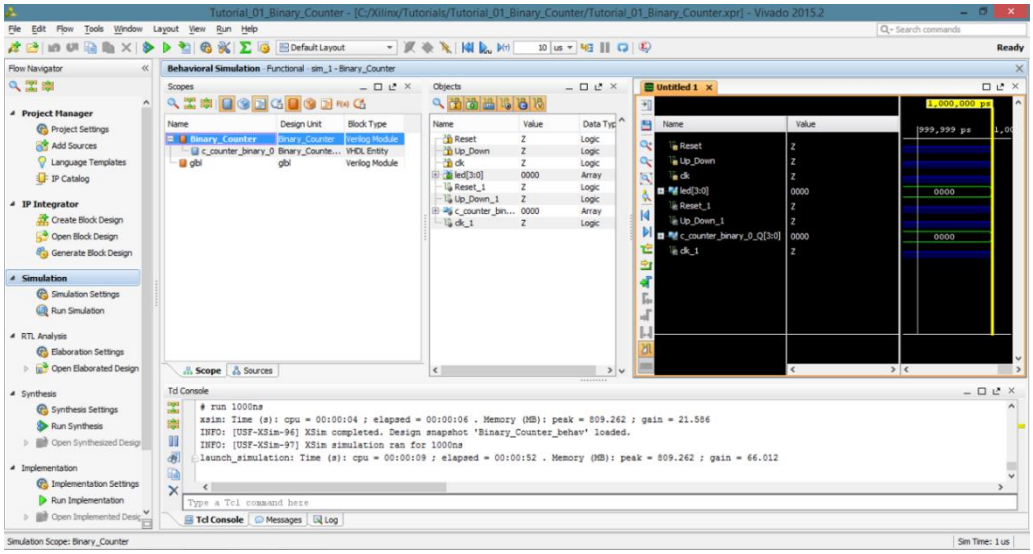


Figure 44 – Behavioral Simulation.

4. To analyze the operation, it is necessary to generate the input signals using the waves' window that is shown on Figure 45. This window's divisors can be settled to a better view of the signals

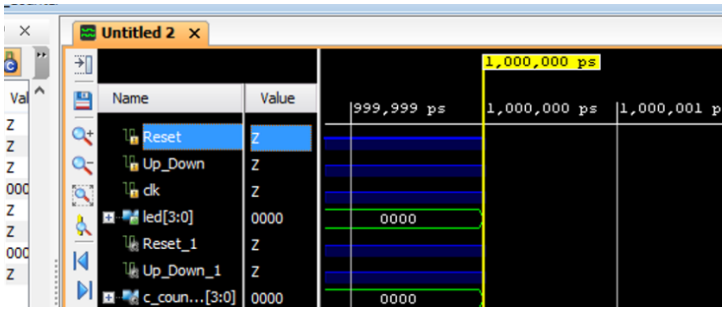


Figure 45 – Waves Window.

5. Right click on the "reference name" of the wave "Reset" selected on Figure 45 and then click on "Force Constant" (Figure 46). As it is just a simulation, the constant is used because a continuous signal is enough to drive this input

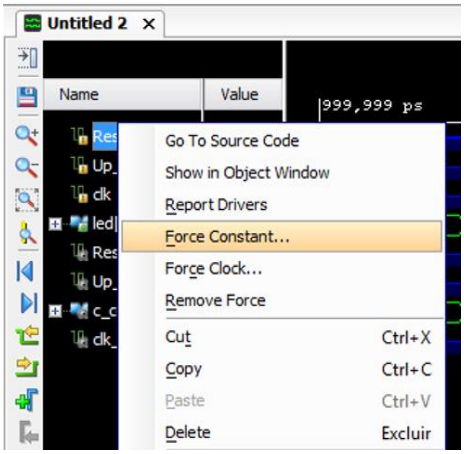


Figure 46 – Force Constant.

6. The Force Constant window will appear. On this step, configure the “Reset” signal (Figure 47) and click on “Apply” and then “OK”

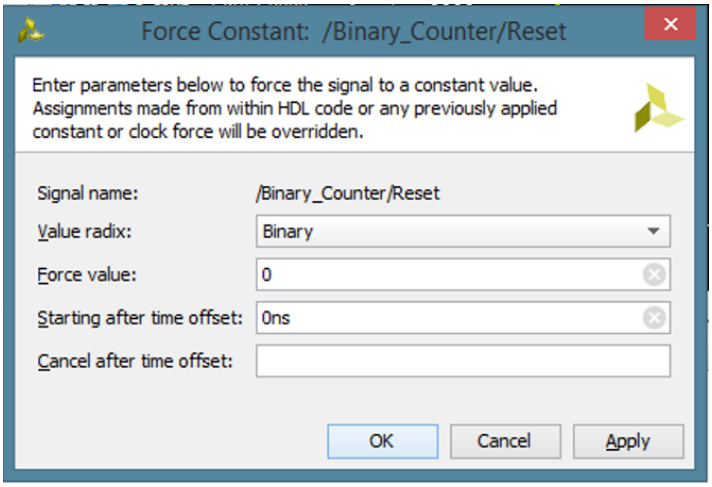


Figure 47 – Force Constant Reset Settings.

7. To verify if the counter works properly counting UP and DOWN, repeat the last step but now associating another type of signal, the “Force Clock”, to the next input “Up_Down” (Figure 48)

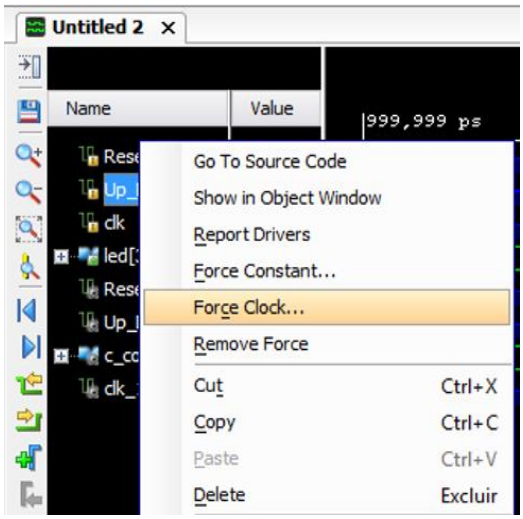


Figure 48 – Force Clock Up_Down.

8. Use the settings shown in Figure 49 to complete the “Up_Down” force clock window form

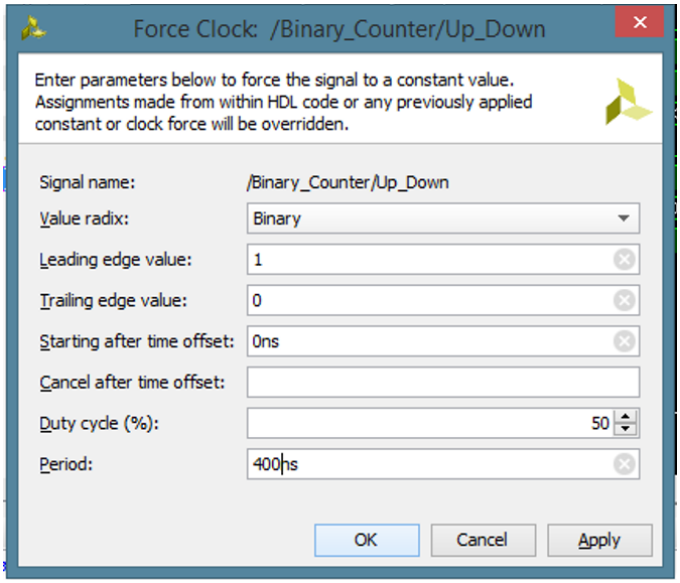


Figure 49 – Force Clock Up_Down.

9. For the last input signal “clk” use the “Force Clock” again (Figure 50) and (Figure 51)

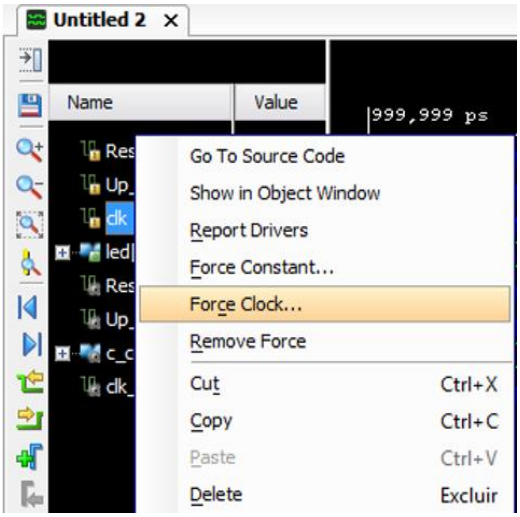


Figure 50 – Force Clock clk.

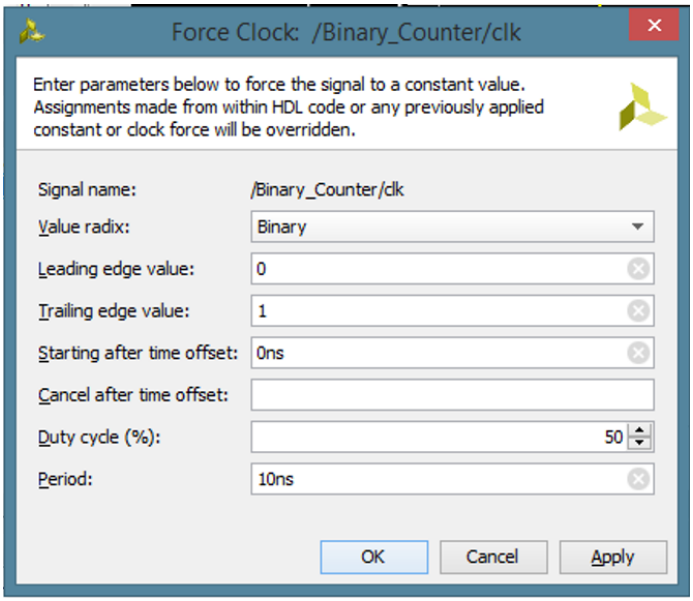


Figure 51 – Force Clock clk Settings.

10. After setting all the inputs, configure the simulation time to 2ms and click on "Run for 2ms" on the simulation menu (Figure 52), and wait

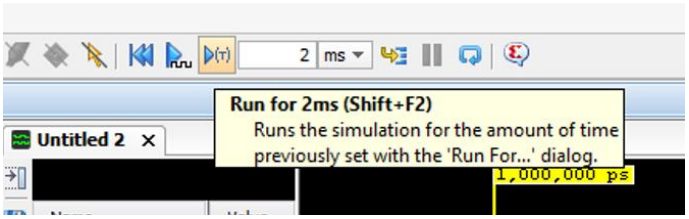


Figure 52 – Run for 2ms.

11. When the simulation finishes, the waveforms appear (Figure 53). Use the "Zoom" buttons to find the best scale to see the simulated signals

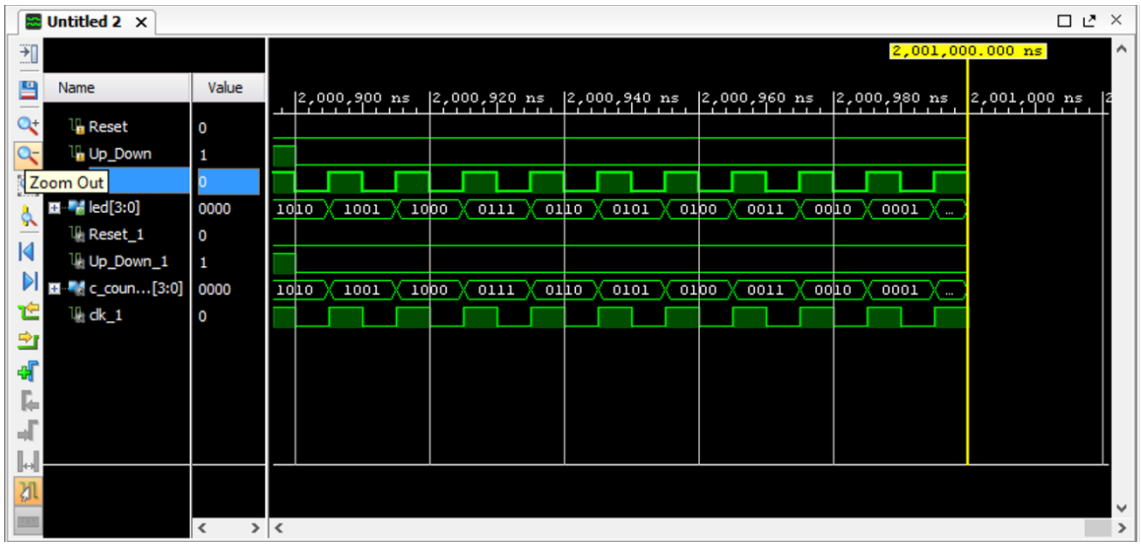


Figure 53 – Signals Simulated.

12. The output "led" is composed by four binary outputs that make the 4-bit counter. To verify if the Binary Counter is working properly, right click the "led" signal and then "Radix" → "Hexadecimal" (Figure 54), the numbers

displayed change from binary representation to a hexadecimal one (Figure 55)

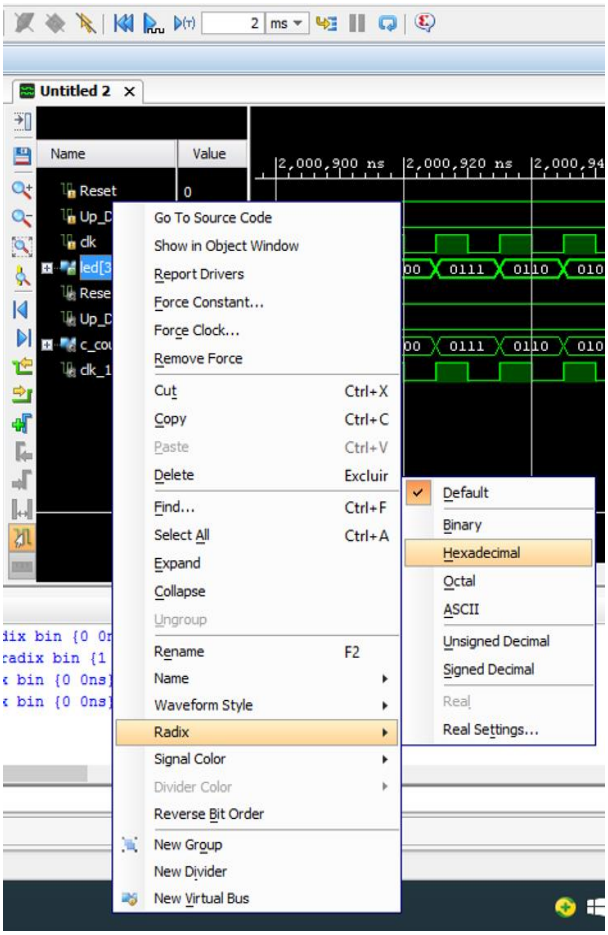


Figure 54 – Led Radix Hexadecimal.

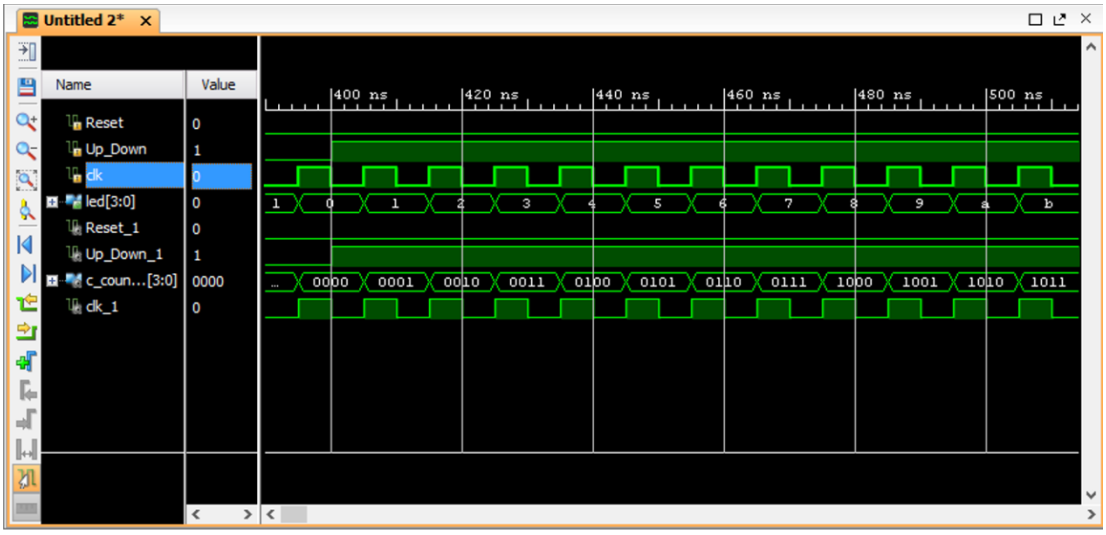


Figure 55 – Count Displayed in Hexadecimal.

13. After confirm that the counter is working according to the specifications, close the simulation workspace to continue the project (Figure 56)

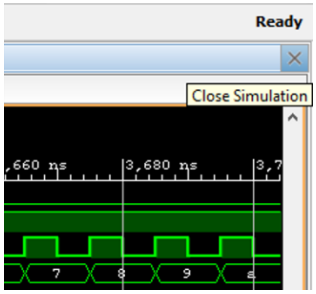


Figure 56 – Close Simulation.

RTL Analyzing:

**Note: The Vivado Design Suite enables you to take your design from full register-transfer level (RTL) creation to Bitstream generation. System-level design entry consists of setting up your design, including creating a project (if applicable), creating and adding source files, elaborating the RTL design, and inserting and configuring debug information* [3].

1. On the window “Flow Navigator” → “RTL Analysis” → “Open Elaborated Design” at any stage of the implementation process, you can generate a variety of reports, run design rule checks (DRCs), Report Noise and check the Schematic (Figure 57)

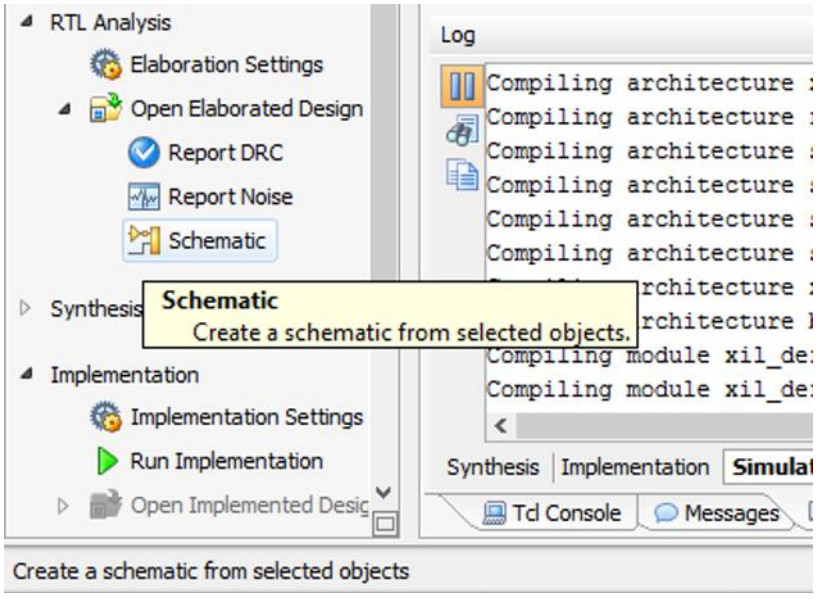


Figure 57 – Open Elaborated Design.

2. For example, to generate the Schematic click on the “Schematic” (Figure 57) and then click on “OK” (Figure 58)

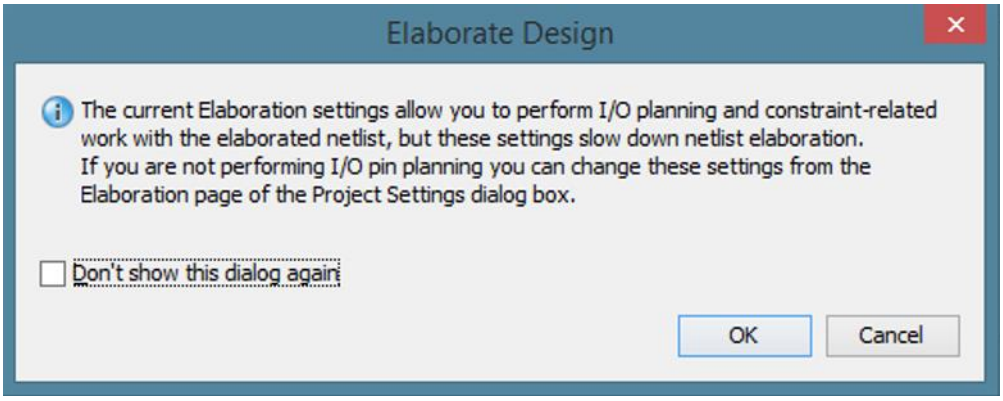


Figure 58 – Elaborate Design.

3. The Schematic Window will appear (Figure 59). It is possible to expand the Blocks to see the under blocks that were used to create the project’s main block, just by double-clicking over the Blocks (Figure 60)

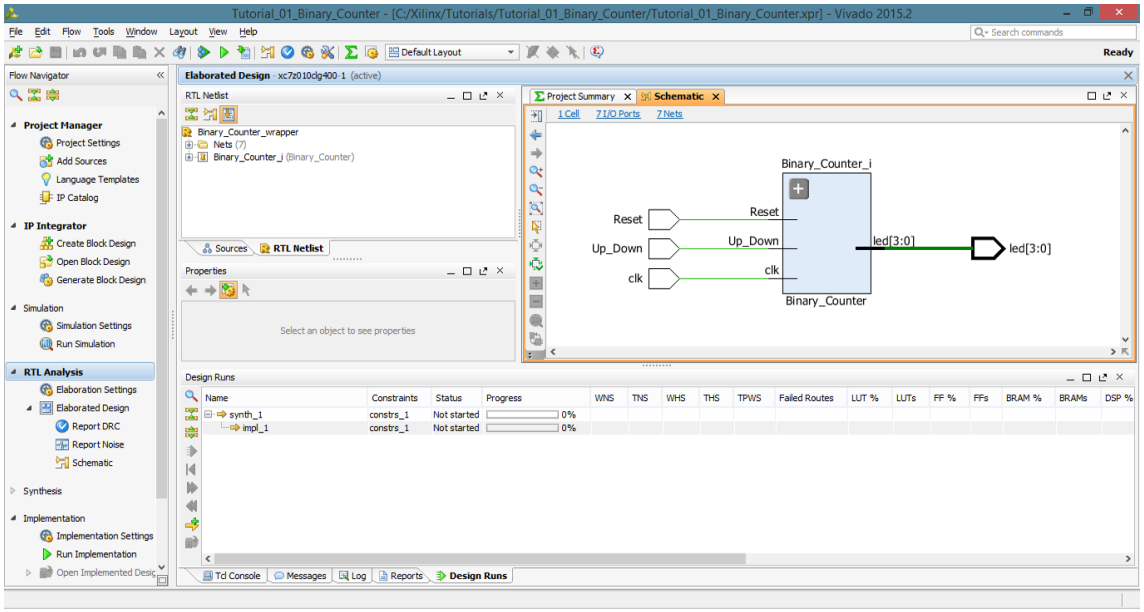


Figure 59 – Schematic Window.

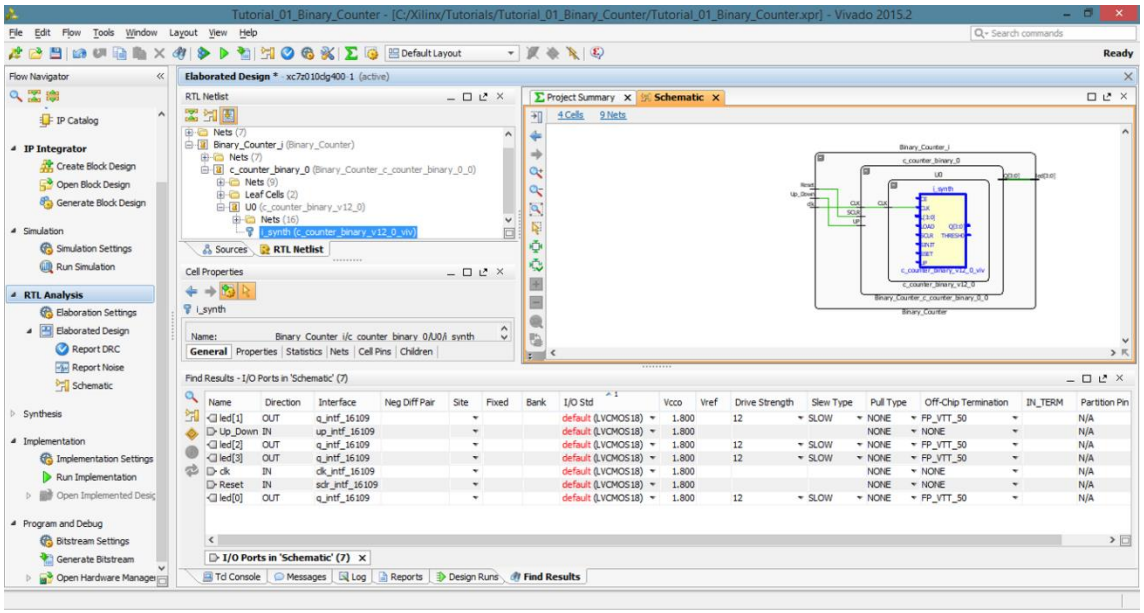


Figure 60 – Schematic Block Expanded.

Pins Assignment with the Schematic:

**Note: In this part, the "project" is almost finished. There are some options to do the Pins Assignment, which consists on linking the ports of the project with the pins of the Zynq Processor on the ZYBO board. The method shown can be used with small and simple projects, but in the next steps it will be shown a better method to do pins assignments for this project.*

1. If the "I/O Ports in 'Schematic'" window is open (Figure 56) look the ZYBO manual that is shown in the beginning of this tutorial and find the pin names where the buttons and leds from the ZYBO board are connected on the Zynq processor. On the section "Site" select the right pin to connect the Ports (Figure 56).
2. However if the "I/O Ports in 'Schematic'" window is not open, click on the Main menu "Window" → "I/O Ports" then the I/O Ports window will appear. Then follow the above steps

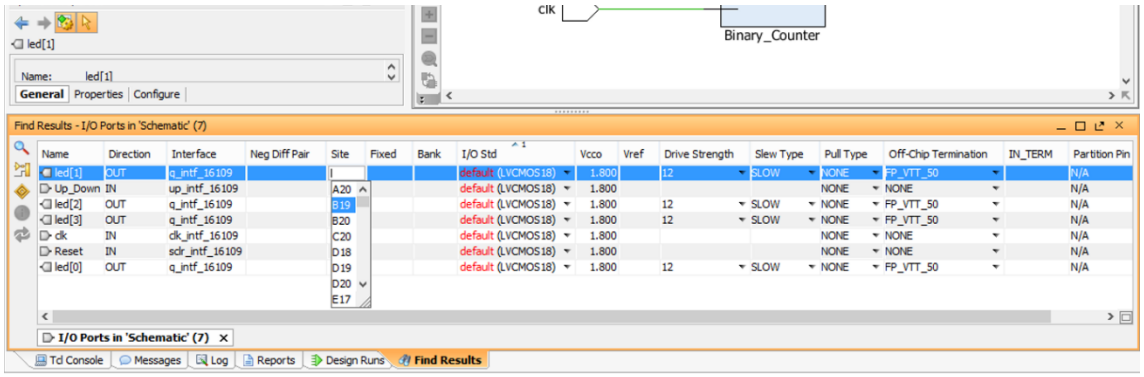


Figure 61 – I/O Ports in Schematics.

3. To continue, close the Elaborate Design (Figure 62)

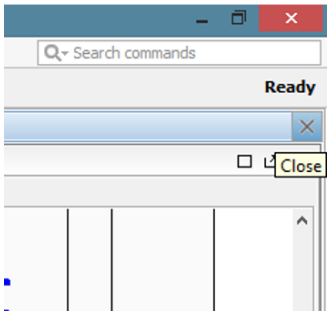


Figure 62 – Close Elaborate Design.

Run Synthesis:

**Note: According to* [4]*, Synthesis is the process of transforming an RTL-specified design into a gate-level representation. Vivado synthesis is timing-driven and optimized for memory usage and performance. Vivado synthesis supports a synthesizable subset of:*

- *SystemVerilog;*
  - *IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language (IEEE Std 1800-2012);*
- *Verilog;*
  - *IEEE Standard for Verilog Hardware Description Language (IEEE Std 1364-2005);*
- *VHDL;*
  - *IEEE Standard for VHDL Language (IEEE Std 1076-2002);*
- *Mixed languages*
  - *Vivado can also support a mix of VHDL, Verilog, and SystemVerilog.*

1. To Run the Synthesis, on “Flow Navigator” → “Synthesis” click on “Run Synthesis” (Figure 63)

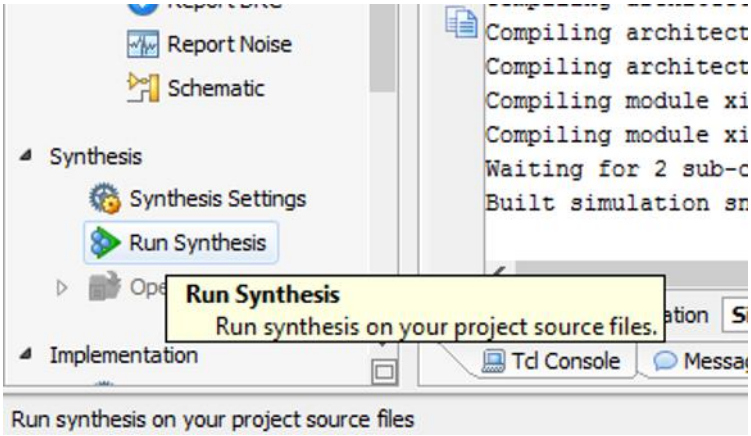


Figure 63 – Run Synthesis.

2. When the Synthesis is complete, a window (Figure 64) will appear, so click on “Cancel”

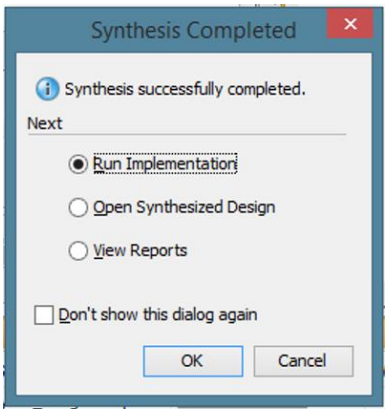


Figure 64 – Synthesis Completed.

Like in the RTL Analysis, the Synthesized Design View is also useful in debugging a design that has been synthesized. To open the Synthesized Design View after having the design synthesized, on “Flow Navigator” → “Synthesis” click “Synthesized Design” (Figure 65). This view is based on the Xilinx primitives that were used in creating the netlist. The synthesized design view can be used to view how the RTL was actually translated into the primitives. It lists all the properties of the primitives.

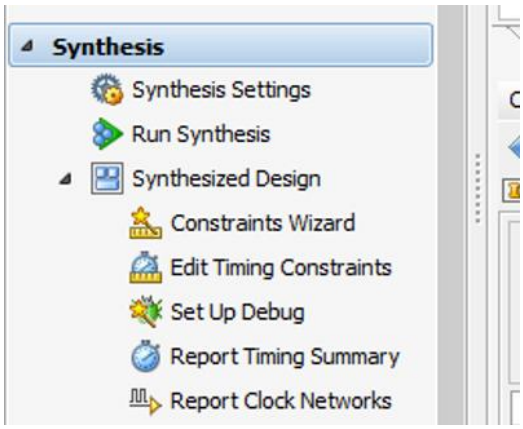


Figure 65 – Synthesized Design.

Pins Assignments with Constraints:

**Note: There are many ways to do the Pins Assignment in Vivado (link the Ports of the project created with the pins from the Zynq Processor). When there are a lot of Ports it is a waste of time to assign them one by one. The following example is based on including a Source File with some constraints that define it.*

1. In the “Flow Navigator” → “Project Manager” click on “Add Sources” (Figure 66)

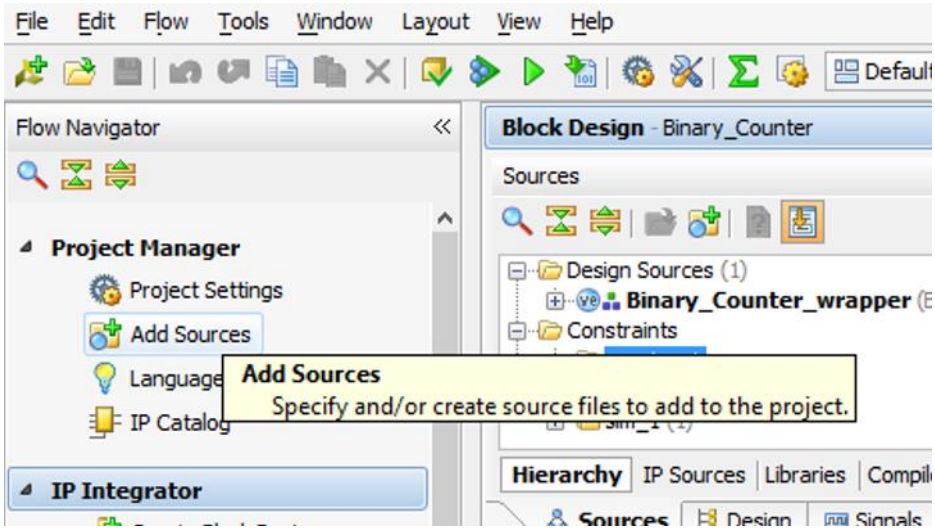


Figure 66 – Add Sources.

2. Select the option “Add or Create Constraints” and then click “Next >” (Figure 67)

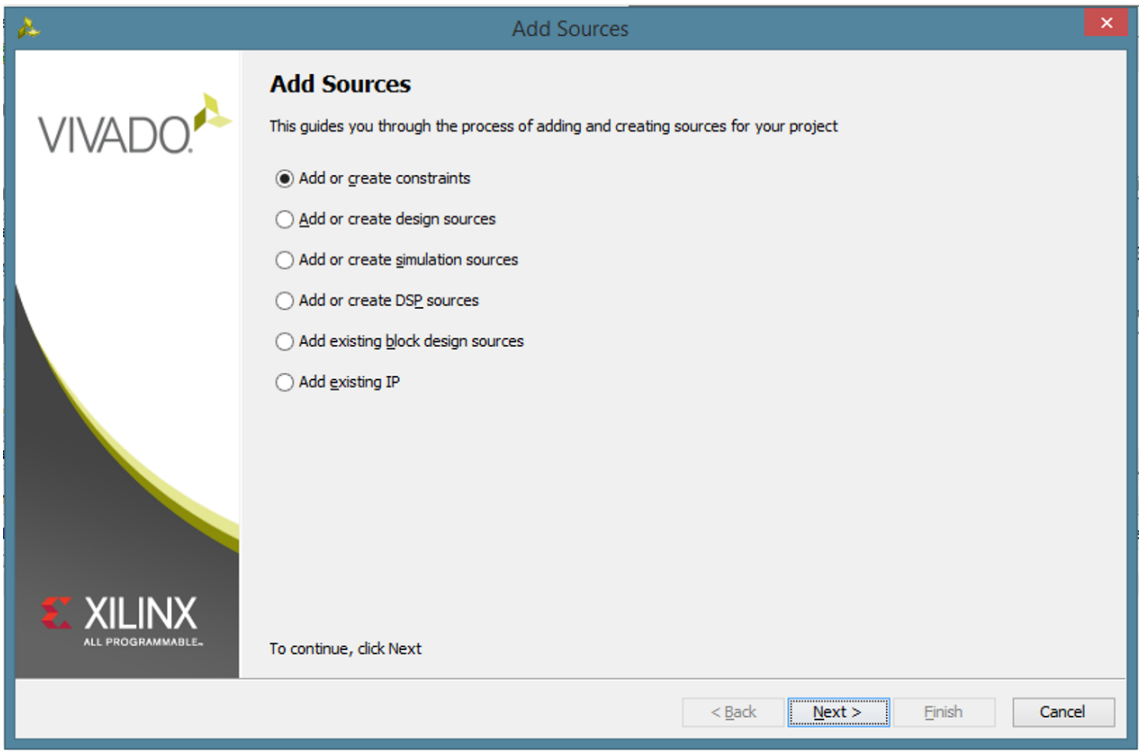


Figure 67 – Add or Create Constraints.

3. Click on the green plus signal → “Add Files” (Figure 68) and then select the file “ZYBO_Master.xdc” and click “OK” (Figure 69). The file is available to download in the Digilent website

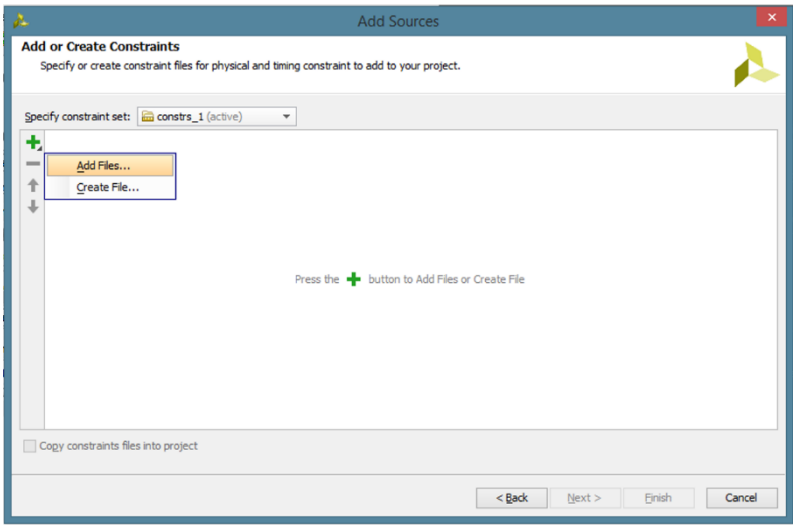


Figure 68 – Add or Create Constraints → Add Files….

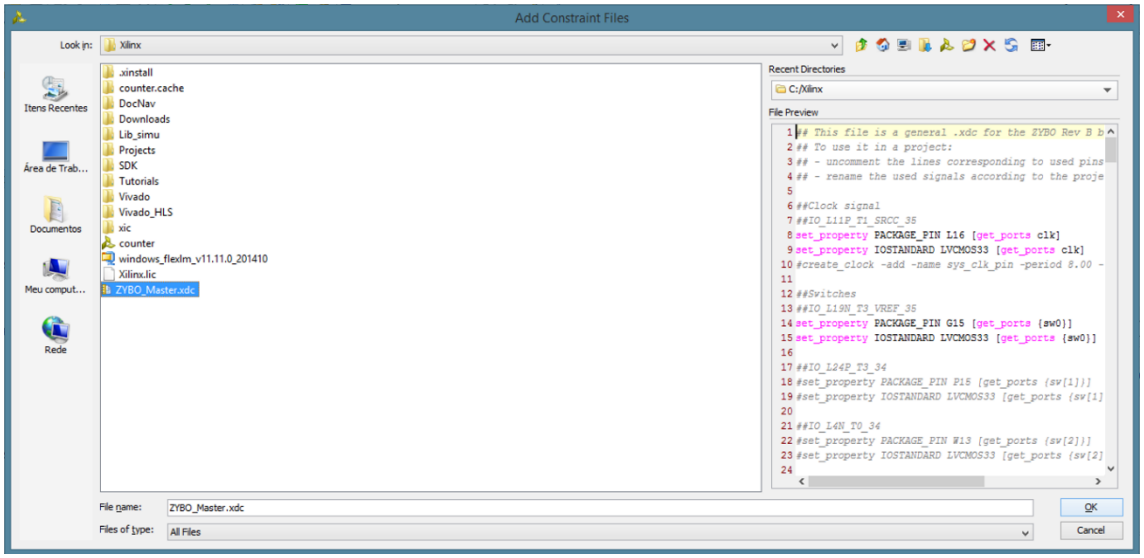


Figure 69 – ZYBO_Master.xdc.

4. On the next window click “Finish” (Figure 70):

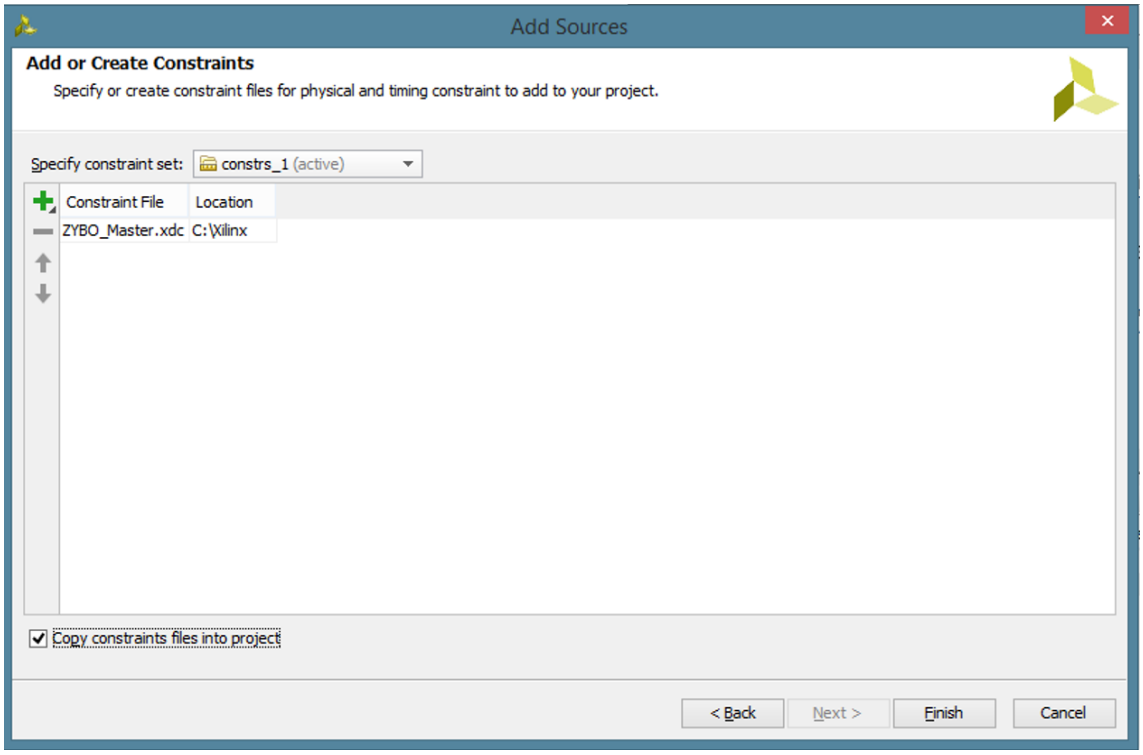


Figure 70 – Add or Create Constraints – ZYBO_Master.xdc

5. The file from the website is a generic normal text file (.xdc) that has some of the ZYBO board definitions in it, but it is necessary to edit the file to enable the constraints that will be used. On “Sources window” → “Constraints” → “Constrs_1” open the file added ZYBO_Master.xdc with a double click on the file (Figure 71)

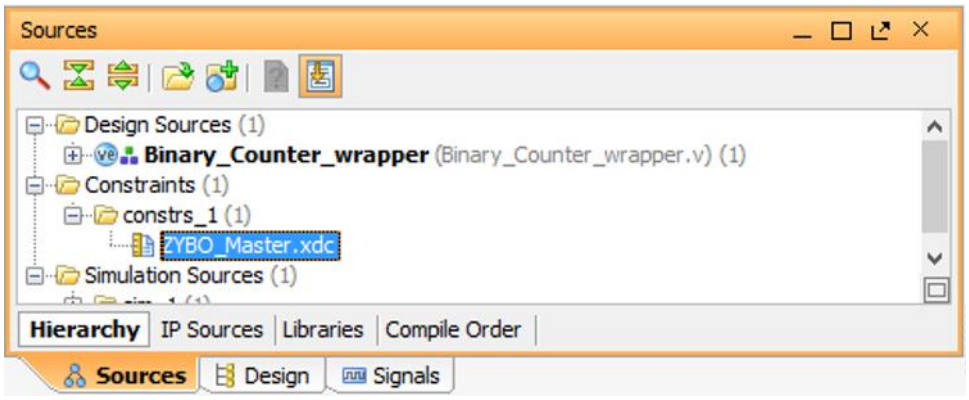


Figure 71 – Sources Constraints – ZYBO_Master.xdc.

6. After opening the file note that all lines are be commented with the character “#”. Uncomment the specific lines (Figure 72), (Figure 73) and (Figure 74), (removing the character “#” from the beginning of each necessary line) and edit the names in the end of the lines that have been changed in the figures

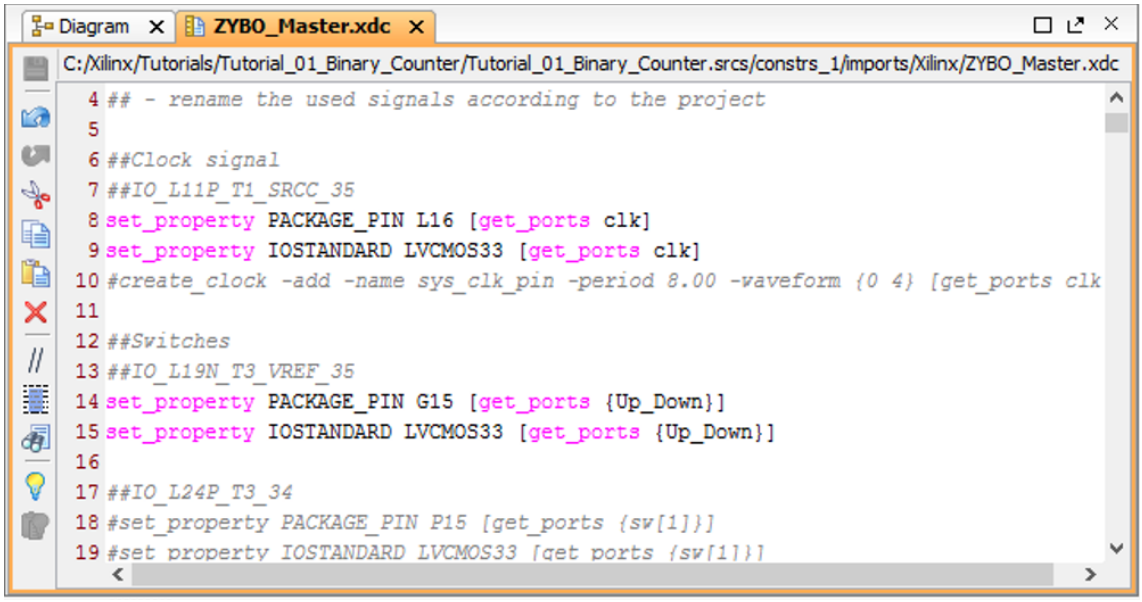


Figure 72 – ZYBO_Master.xdc edition one.

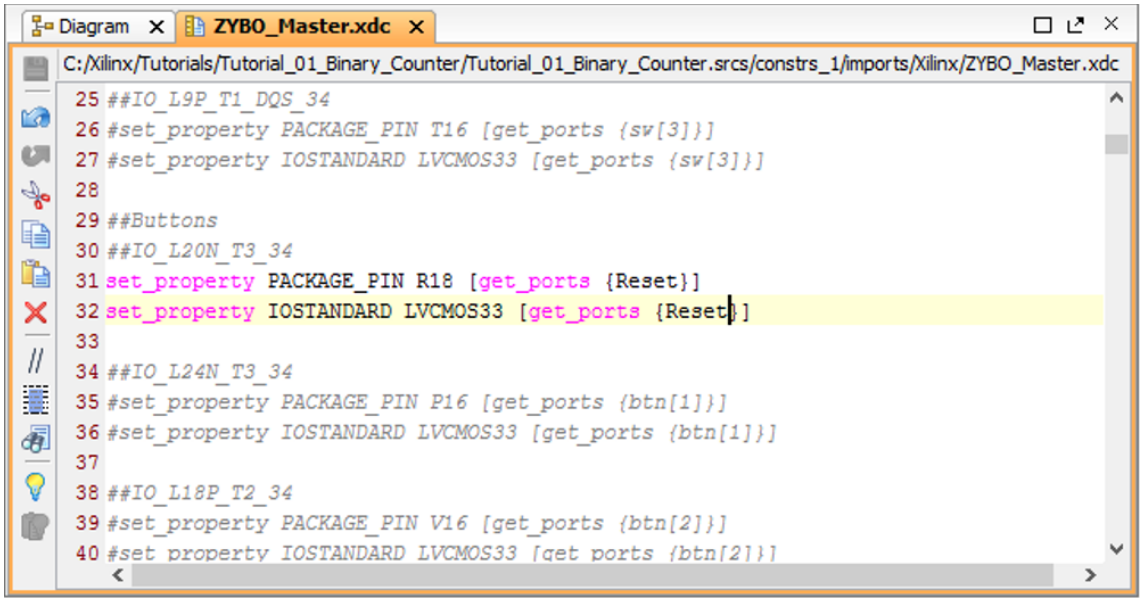


Figure 73 – ZYBO_Master.xdc edition two.

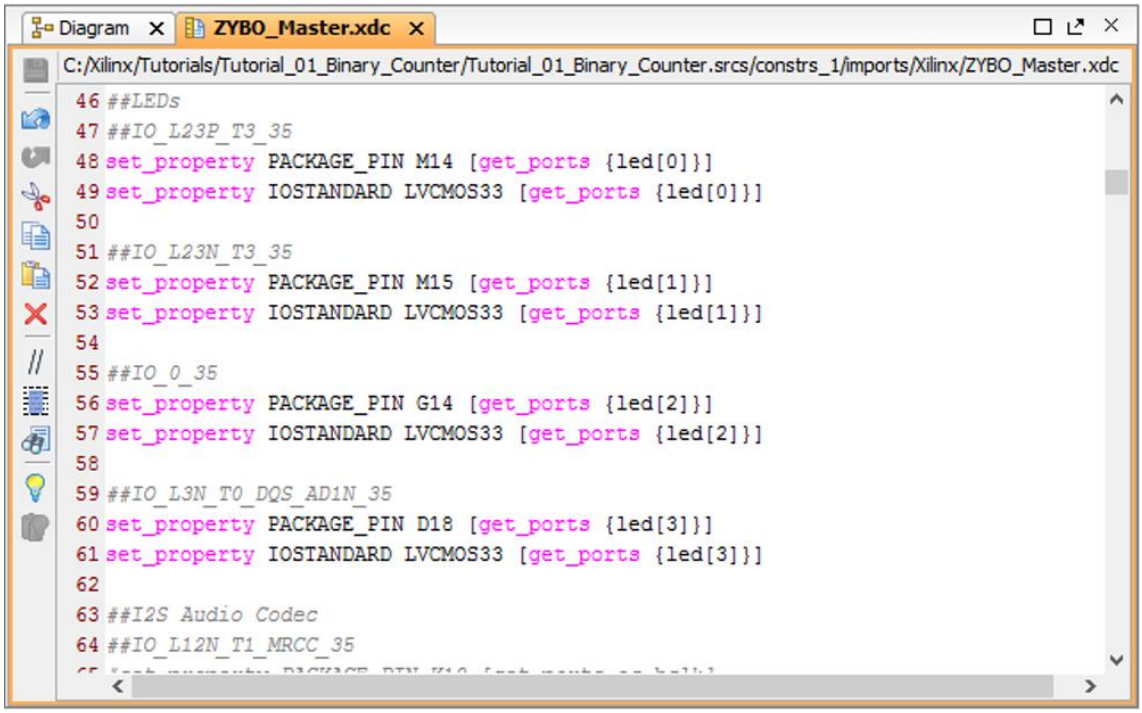


Figure 74 – ZYBO_Master.xdc edition tree.

7. After changing the file, save it by clicking on the button “Save” (Figure 75)

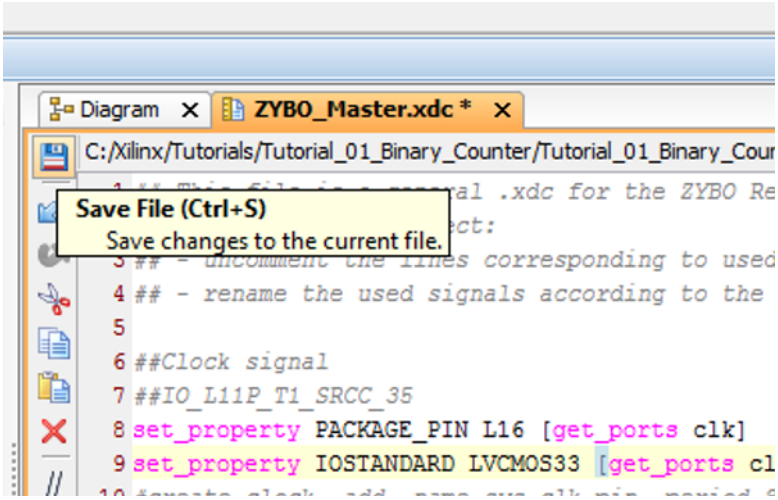


Figure 75 – Save File Editions.

**Note: Just to show some information that can be checked using the Synthesized Design Options like the Report Noise and Report Utilization, follow the next steps:*

8. Run the Synthesis again, on “Flow Navigator” → “Synthesis” click on “Run Synthesis” (Figure 76) and then click “Cancel” on the window that will appear (Figure 77)

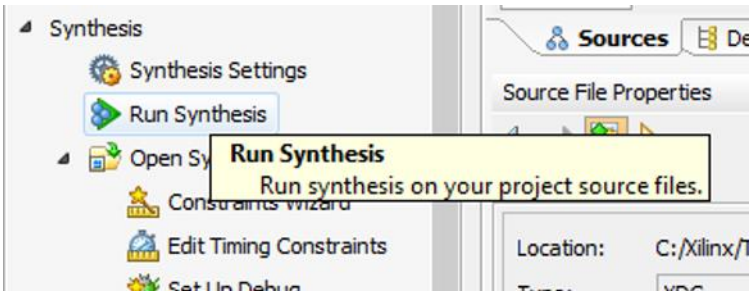


Figure 76 – Run Synthesis.

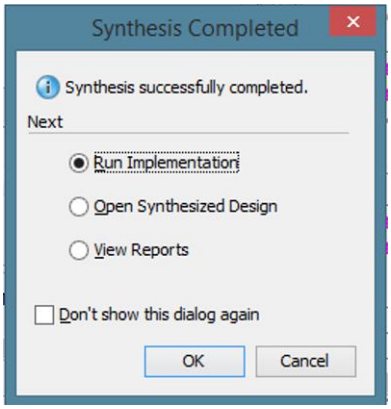


Figure 77 – Synthesis Completed.

9. For the “Report Noise”, on “Flow Navigator” → “Synthesis” → “Synthesized Design” click on “Report Noise” (Figure 78) and a window (Figure 79) opens with information obtained from simultaneous switching noise (SSN) analyzes

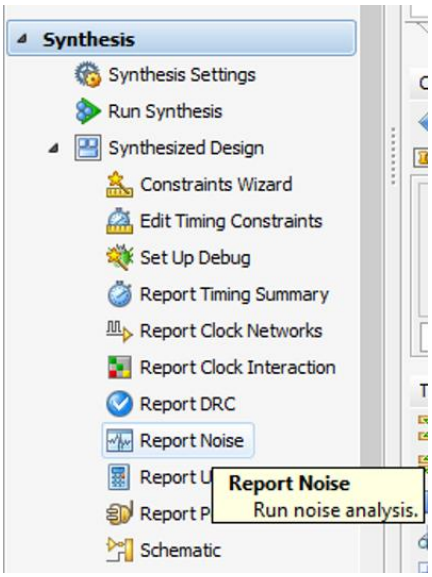


Figure 78 – Report Noise.

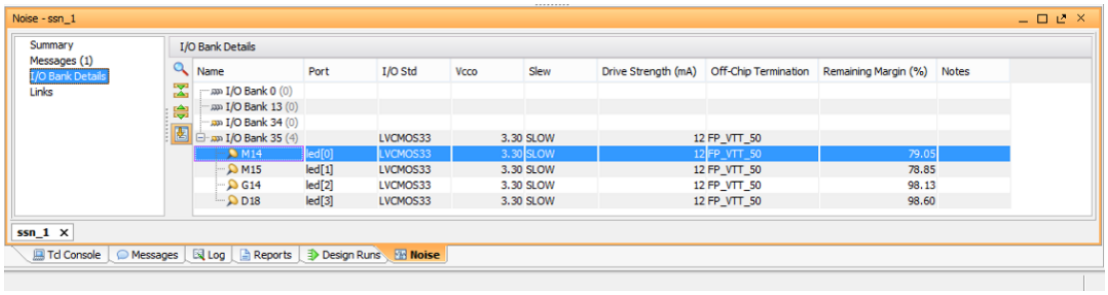


Figure 79 – Report Noise Window.

10. For “Report Utilization”, on “Flow Navigator” → “Synthesis” → “Synthesized Design” click on “Report Utilization” (Figure 80) and the information analyzed appears in a window (Figure 81)

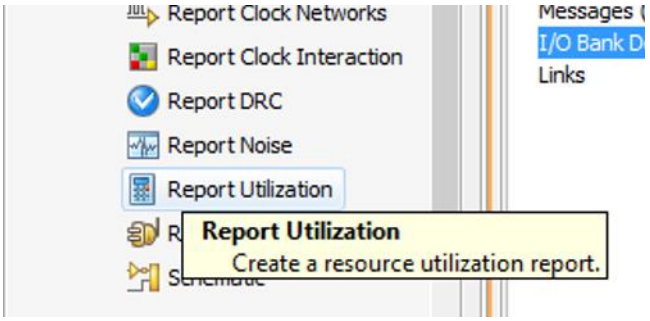


Figure 80 – Report Utilization.

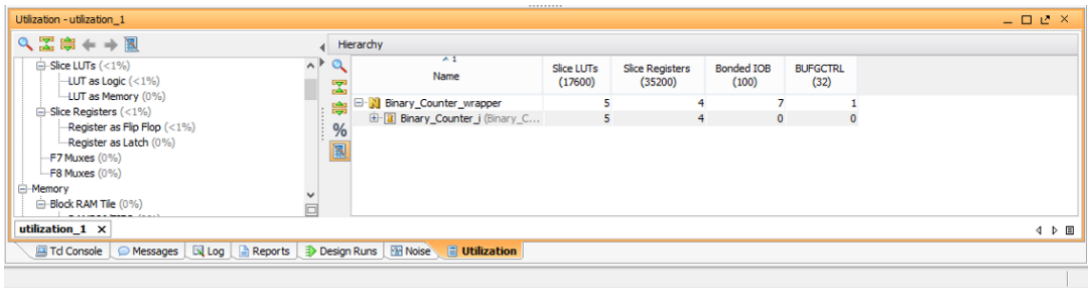


Figure 81 – Report Utilization Window.

Run Implementation:

**Note: The Vivado implementation includes all steps necessary to place and route the netlist onto device resources, within the logical, physical, and timing constraints of the design. To start the Implementation follow the next steps:*

1. On “Flow Navigator” → “Implementation” click on “Run Implementation” (Figure 82) and on the next window select the first option and click “OK” (Figure 83)

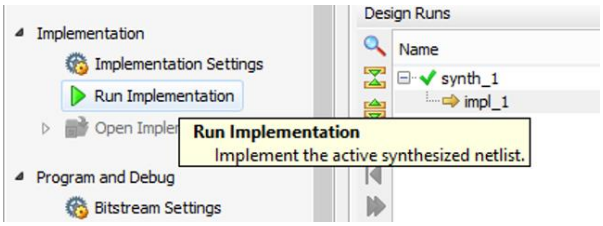


Figure 82 – Run Implementation.

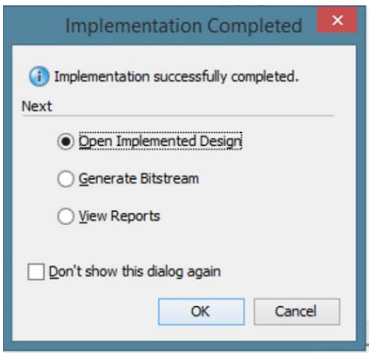


Figure 83 – Implementation Completed.

2. If necessary to see more information about the project, the options can be accessed on “Menu” → “Window” and/or on “Flow Navigator” → “Implementation” → “Implemented Design” (Figure 84)

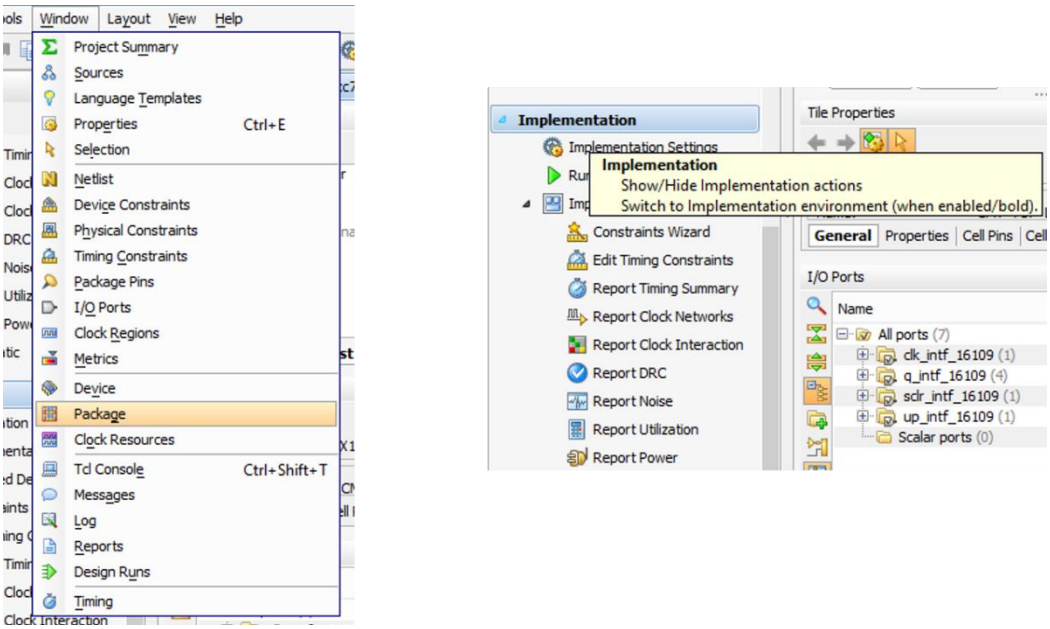


Figure 84 – Implemented Design Information.

3. After checking the information click on “x” to close the Implemented Design (Figure 85)

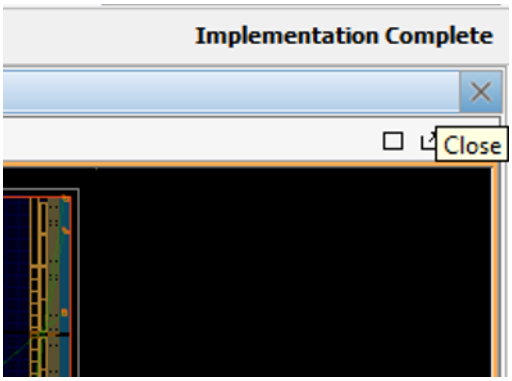


Figure 85 – Closing the Implemented Design.

Creating a Clock Divider for the 4 Bits Binary Counter:

Now the project is almost ready to program into the board, but if the actual project clock input uses the 125 MHz frequency clock from the board, that will do the Binary Counter to count really fast and it will be impossible to see the leds counting up or down. So it is necessary divide the clock before it enters in the port “clk”.

Creating a VHDL Source:

1. The Clock Divider will be created by a VHDL description, different from the method used on the design flow to create the Binary Counter. So don’t close the actual project that’s still open (The Binary Counter), but just follow again the design flow described on the topic **Create a New Project**. The only steps that will change is the first and the last, because the other project is still open. So, to create a new project in “Main Menu” → “File” click on “New Project” (Figure 86) and continue the design flow in the topic **Create a New Project.** A new window (Figure 87) appears. Click “No”

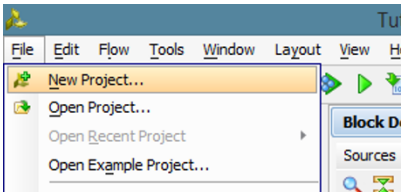


Figure 86 – New Project.

Note: Continue on the topic Create a new Project.

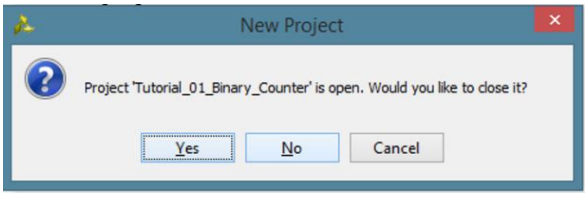


Figure 87 – Close the Open Project

2. Click on the “Add Sources” button within the “Flow Navigator” → “Project Manager” (Figure 88)

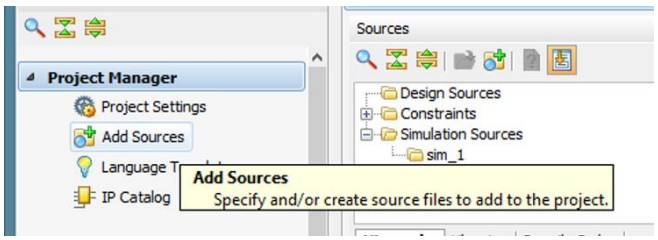


Figure 88 – Add Sources.

3. Then select the option to “Add or create design sources” and click on “Next” (Figure 89)

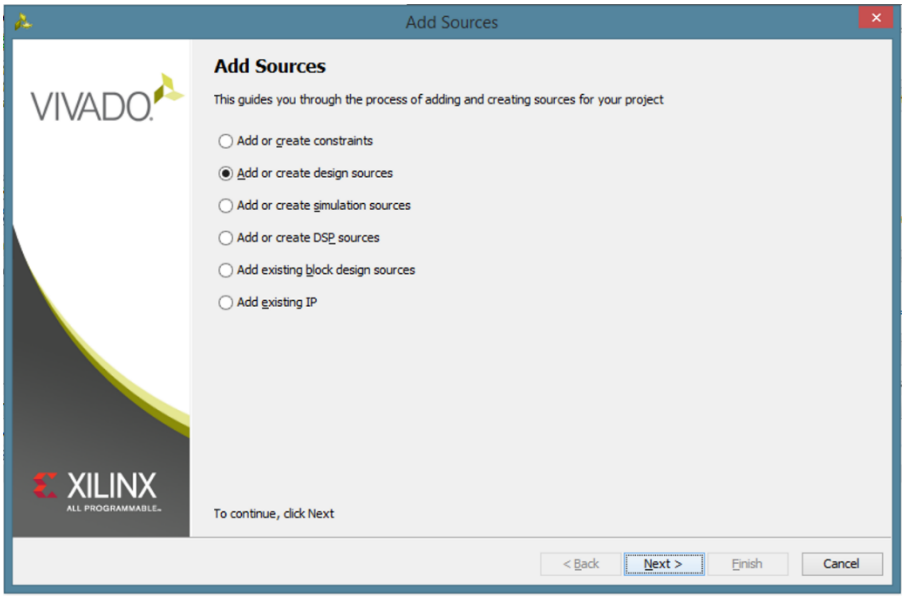


Figure 89 – Add or create design sources.

4. Now click on “+” and then “Create File” (Figure 90)

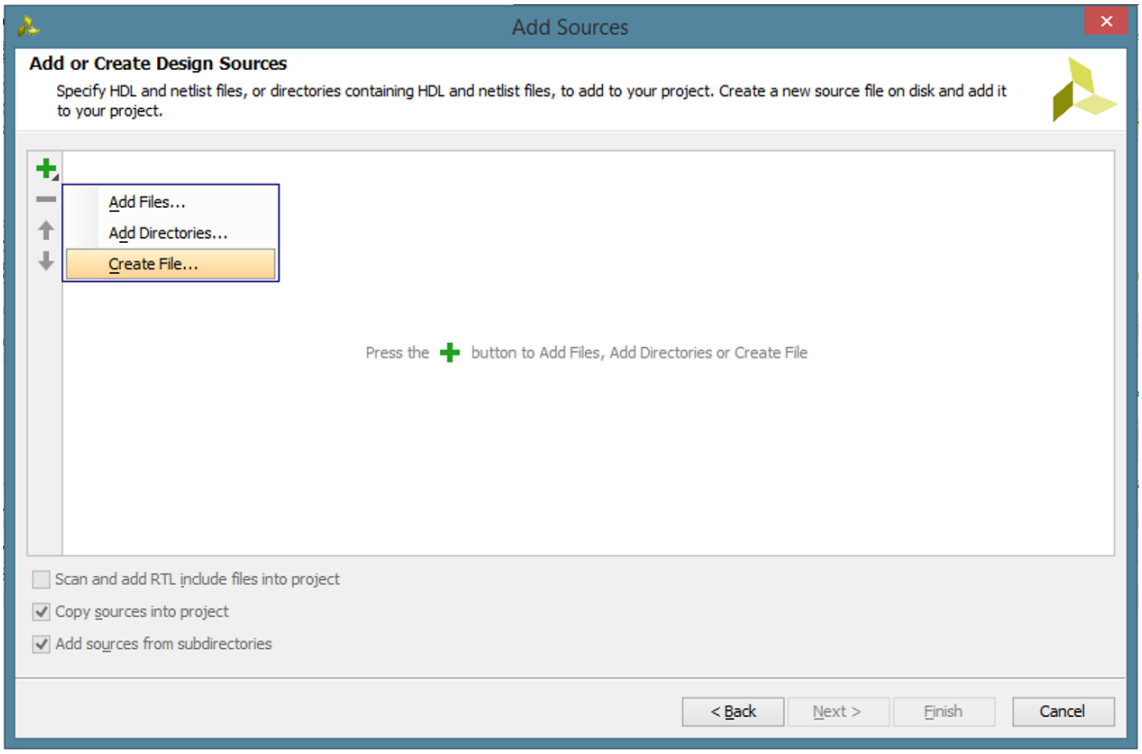


Figure 90 – Create File.

5. Choose now the type of the source file. In this example we will create a VHDL file. Then write the name of your source file l “Clock_div” and then click “OK” (Figure 91). In the next window click on “Finish” (Figure 92)

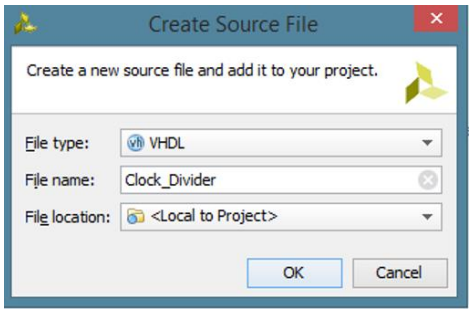


Figure 91 – Create a VHDL Source File.

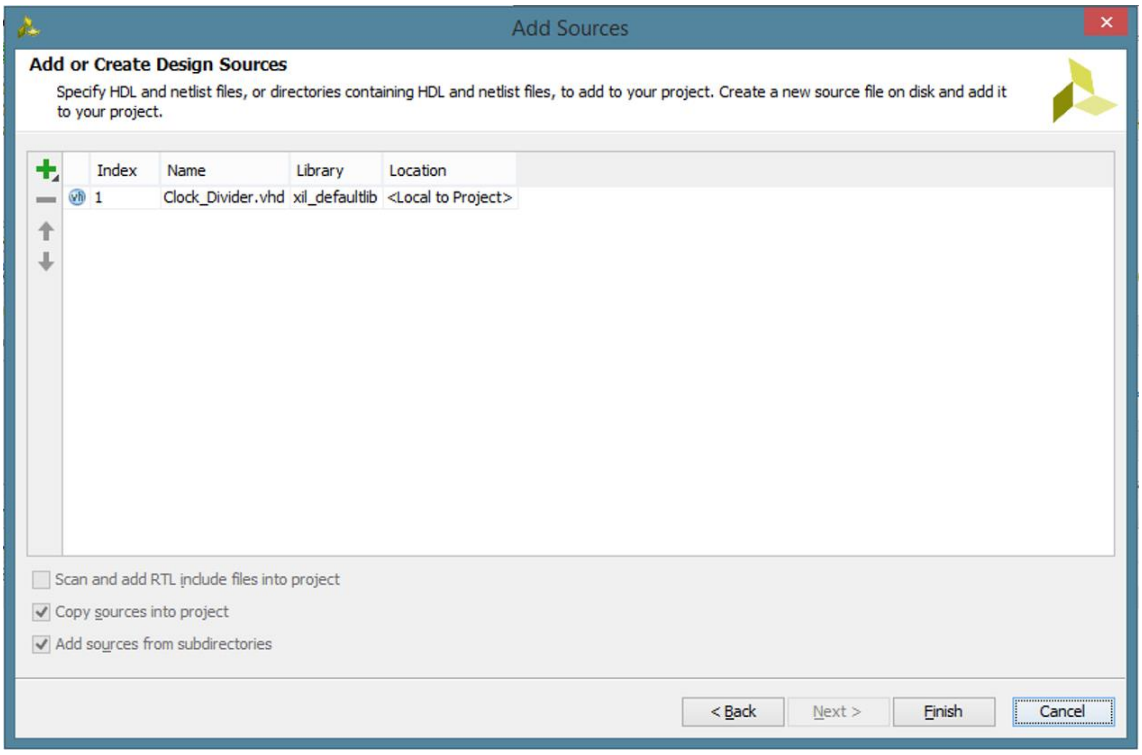


Figure 92 – Creating a Clock Divider Source.

6. A wizard window appears (Figure 93), where you can edit the automatic code for the Ports Definitions generated by Vivado. Click on the "+" to add another Port that this project need and edit the "Ports Names" and the "Direction" (Figure 93). Then click "OK" and in the next window click "Yes" (Figure 94)

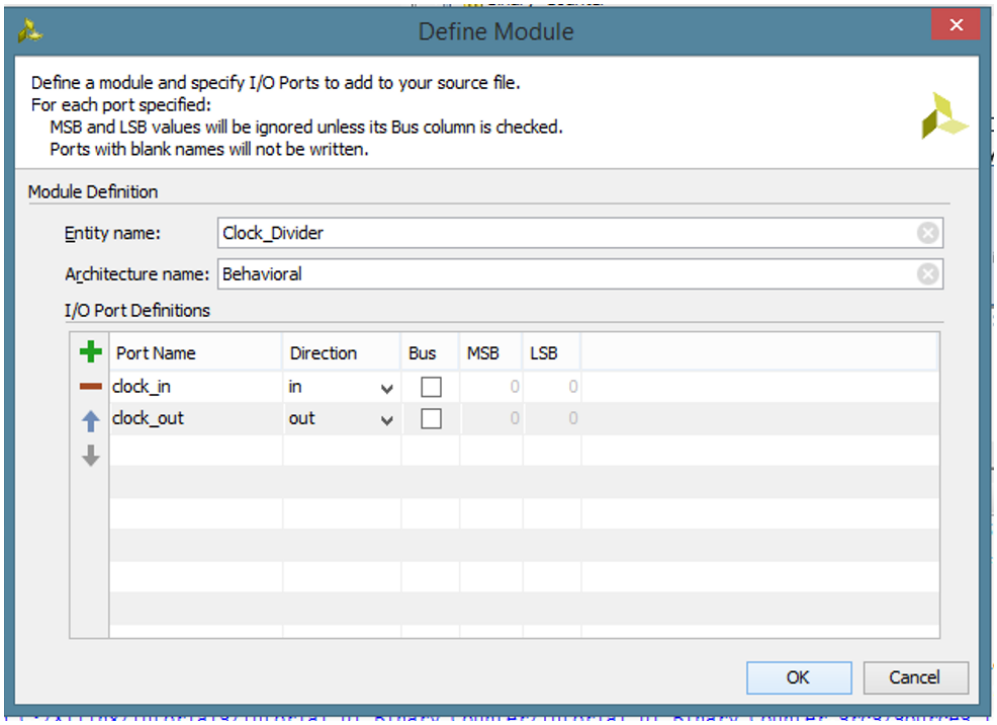


Figure 93 – Define Module.

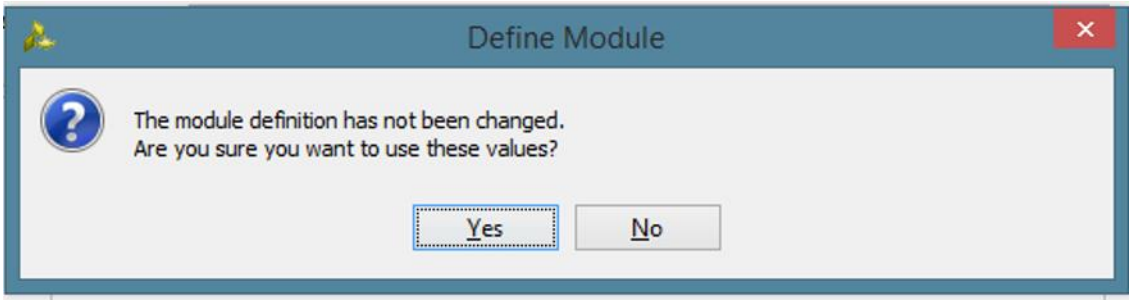


Figure 94 – Define Module Confirmation.

7. Open the Source File created. In “Sources” → “Design Sources” double click on “Source File Created” (Figure 95), and the file opens (Figure 96)

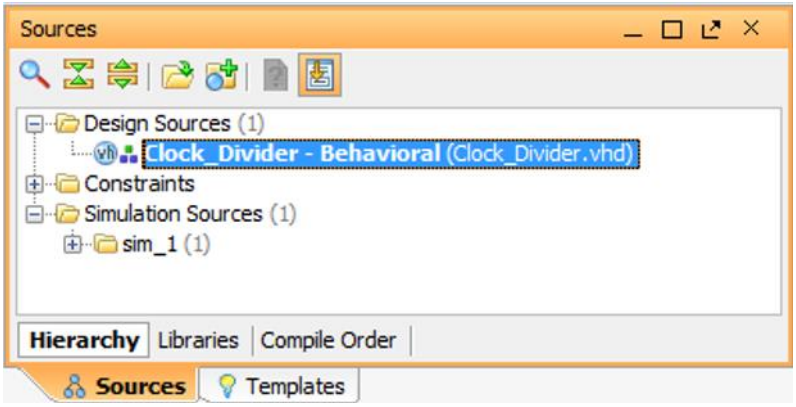


Figure 95 – Open The Design Source File Created.

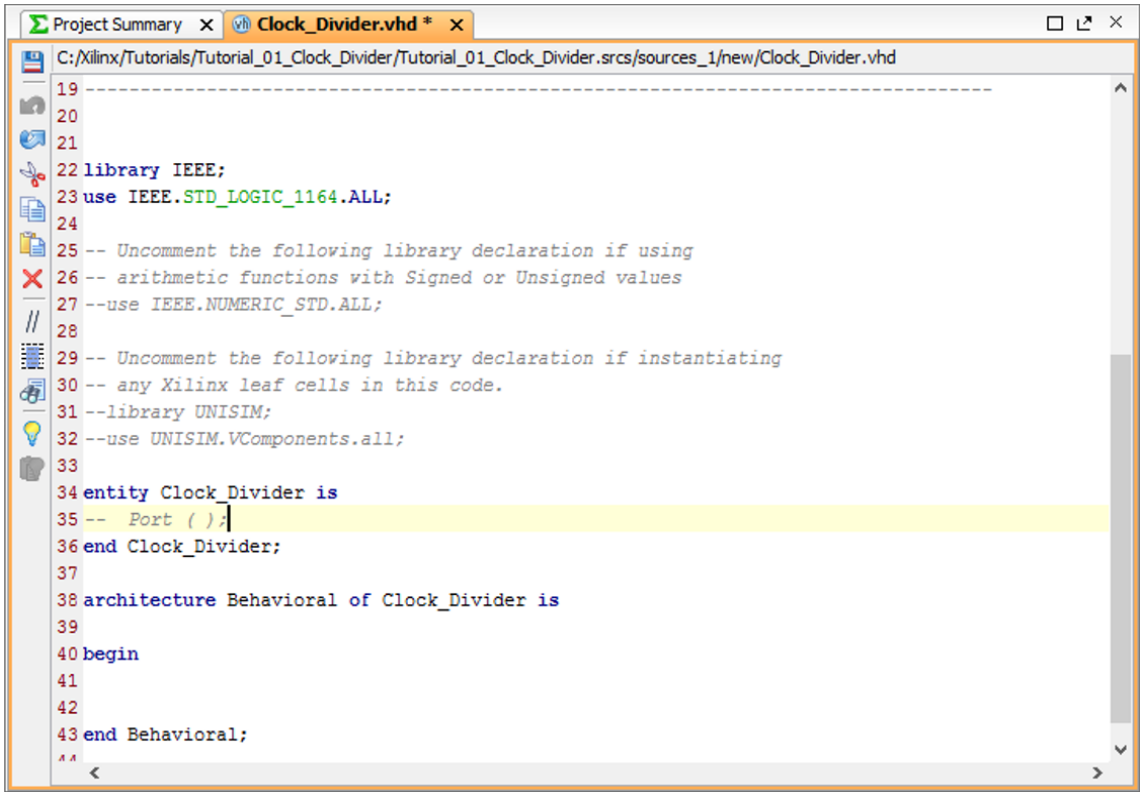


Figure 96 – Clock_Divider Source File.

8. Edit the code as the example (Figure 97), and after save the changes clicking on “Save” (Figure 98)

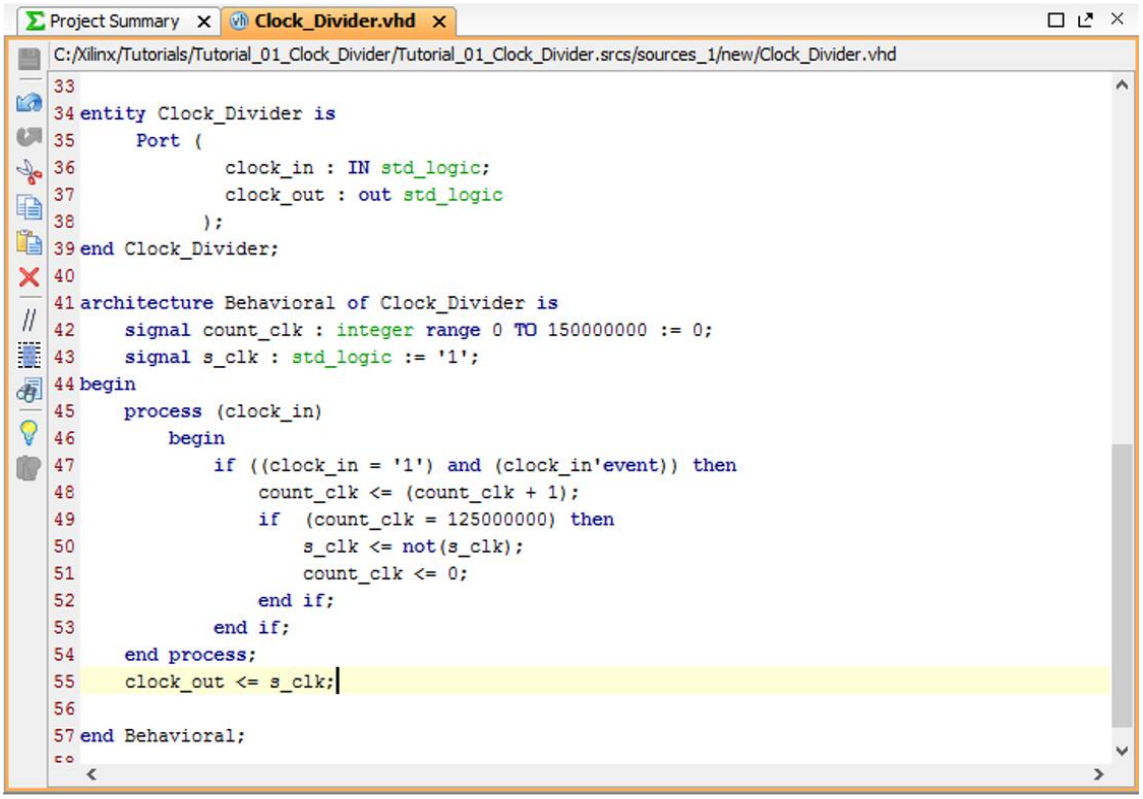


Figure 97 – Clock Divider VHDL Code.

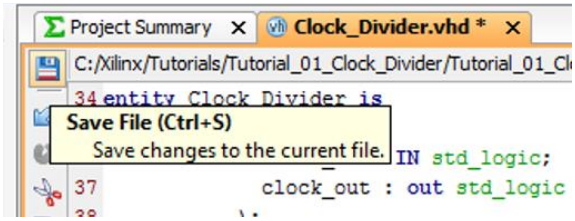


Figure 98 – Save File.

Simulating the Project II:

**Note: At this time another method to perform simulation is presented. This method is based on a “Test Bench File” in order to simulate the design. A test bench file is required to provide stimulus to the design. This version of the Vivado does not have an automatic Test Bench File generator, so you need to create and add it manually to your project.*

To create the Test Bench File you need follow the following steps:

1. Click again in Add Sources on the Flow Navigator (Figure 99)

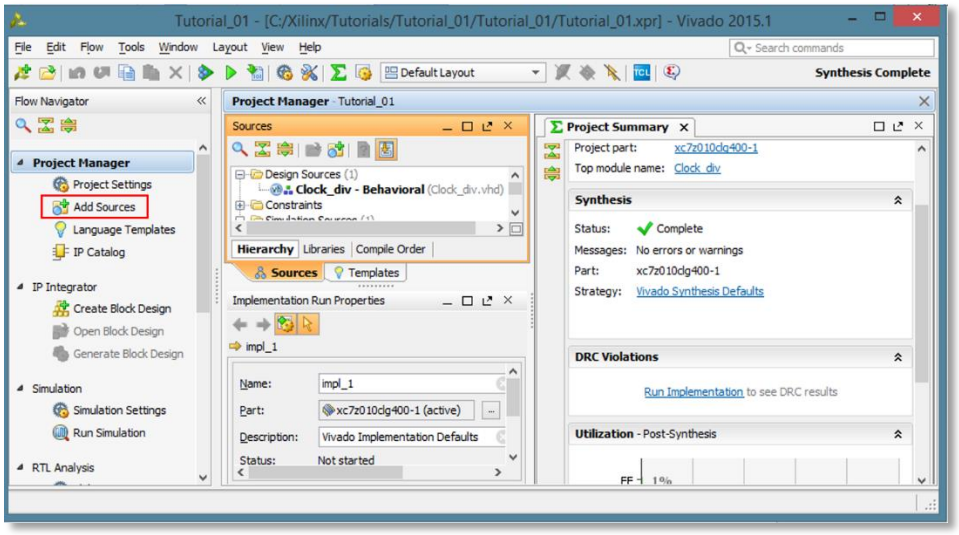


Figure 99 – Add Test Bench File.

2. Then select the option to Add or create simulation sources and click “next” (Figure 100)

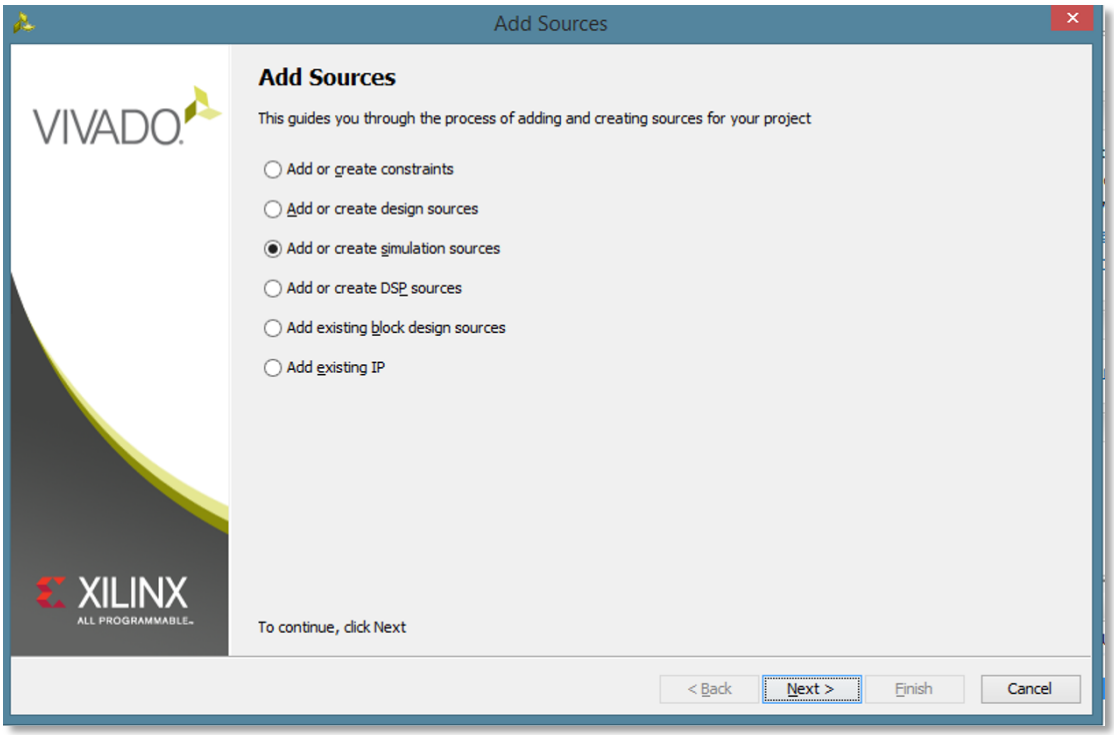


Figure 100 – Create Simulation Source.

3. Now click in the “+” (plus) and then click “Create File” (Figure 101). In this example is used ‘Clock_div_simu’ as File Name. Change the “File Type” to ‘VHDL”, click “OK” and then “Finish”. In the next page just click “OK” and then “Yes”

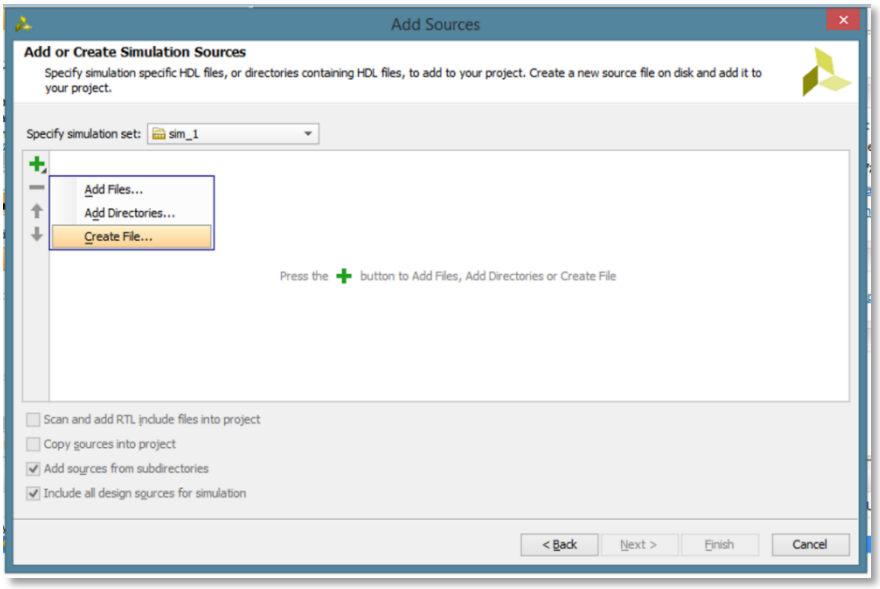


Figure 101 – Creating the Simulation File.

4. Now the “Test Bench Source File” was created. A window appears (Figure 102). Open the “File” with a double click in the testbench filename

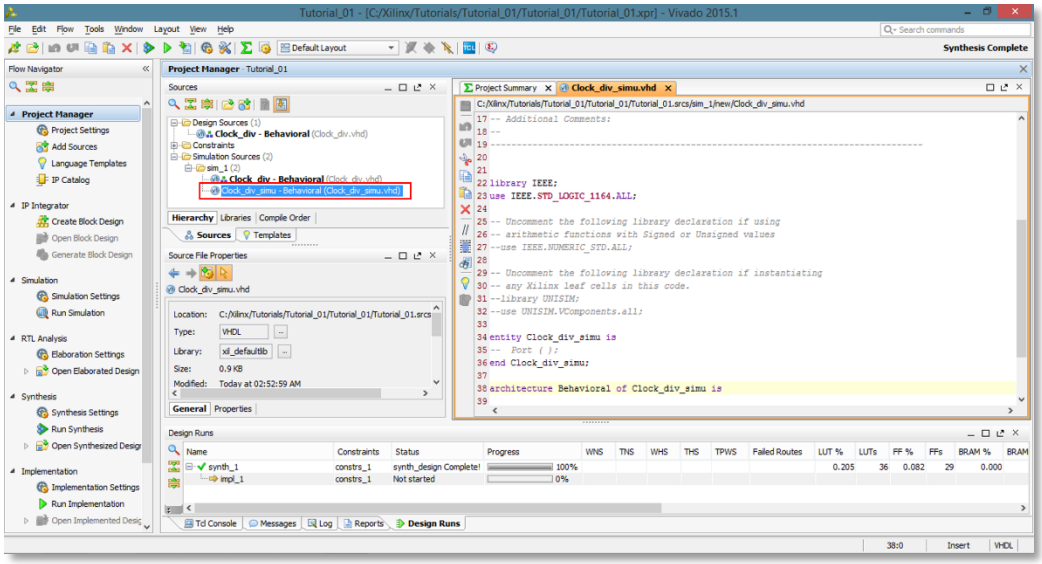


Figure 102 – Editing the Test Bench.

5. The code created is shown in Figure 103. You may create the simulation file yourself or copy this example

```
--use UNISIM.VComponents.all;

entity Clock_div_simu is
end Clock_div_simu;

architecture Behavioral of Clock_div_simu is
component Clock_div
port( clock_in : in std_logic; clock_out : out std_logic );
end component;
    signal clock_in : std_logic;
    signal clock_out : std_logic;
begin
uut: Clock_div port map( clock_in => clock_in, clock_out => clock_out );
    process begin
        clock_in <= '0';
        for I in 0 to 10000 loop
            clock_in <= '1';
            wait for 1ps;
            clock_in <= '0';
            wait for 1ps;
        end loop;
    end process;
end Behavioral;
```

Figure 103 – Test Bench Code.

6. Then click in the Simulation Settings, and check the configurations (Figure 104)

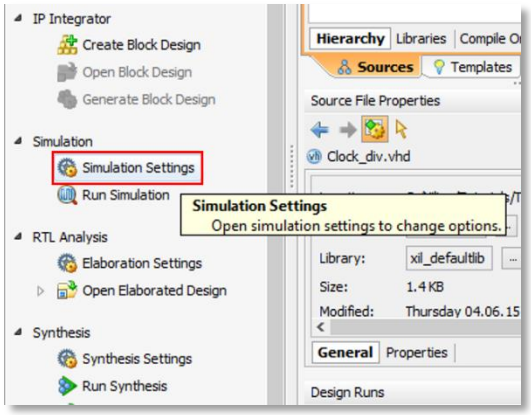


Figure 104 – Simulation Settings.

7. Confirm that the check box “Generate Scripts Only” is unchecked, and that in the “Advanced tab” the check box “Include all Design…” is checked. Then click “Apply” and “OK” (Figure 105)

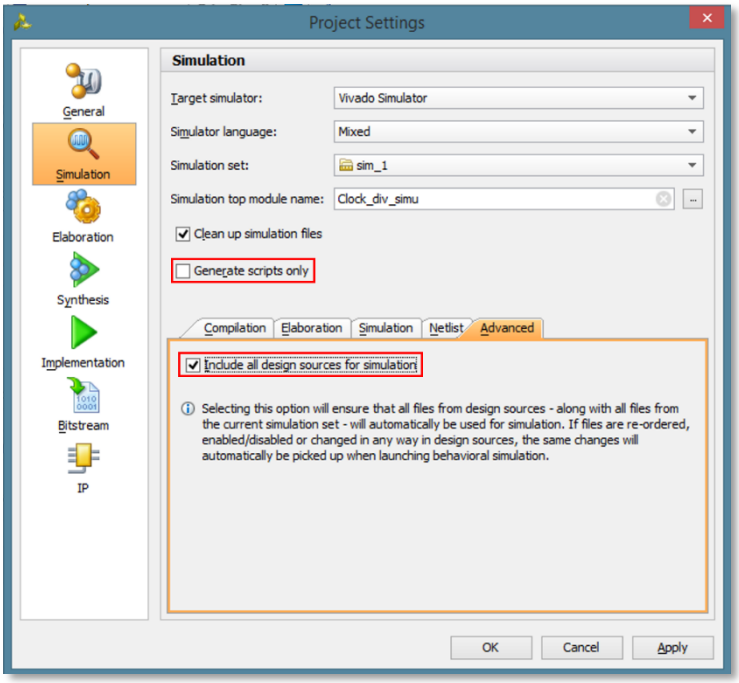


Figure 105 – Simulation Settings Window.

8. Now you can run the simulation. In “Flow Navigator” → “Simulation” click on “Run Simulation” and then “Run Behavioral Simulation” to simply simulate the functionality of your design, or click on the other options to run post-synthesis functional or temporal simulation, which will integrate temporal information on your design simulation (Figure 106)

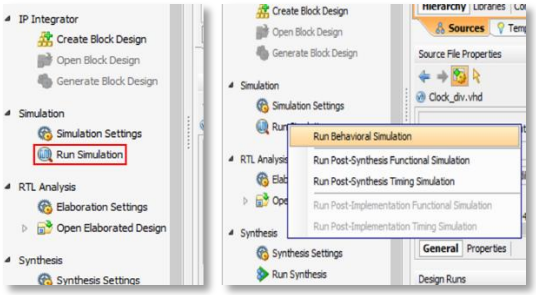


Figure 106 – Running the Architecture Simulation.

9. The “Simulation window” will appear (Figure 107), showing the simulation waveforms. To see and configure the signal simulated click on “maximize window” marked by a red box

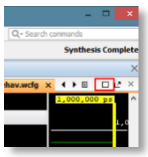


Figure 107 – Simulation.

10. To better analyze the generated signals, use the zoom button in the left-hand toolbar, and check if the design is working according to the specification (Figure 108)

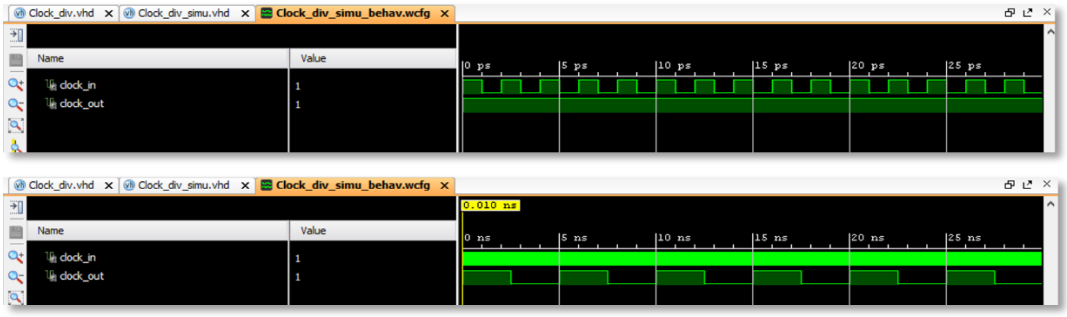


Figure 108 – Simulation Wave.

11. If you are using a slow computer edit some values in the original source code “Clock_div.vhd” before run the simulation or otherwise simulation will take a lot of time. This module divides the 125MHz clock to obtain a 1Hz clock. To accelerate simulation change the counter width from 125000000 to 1250 (Figure 109), and then click “Run Synthesis” again, and return to step 8

```
process (clock_in)
        begin
            if ((clock_in = '1') and (clock_in'event)) then
                count_clk <= (count_clk + 1);
                if  (count_clk = 125000000) then
                    s_clk <= not(s_clk);
                    count_clk <= 0;
                end if;
            end if;
    end process;
```

Figure 109 – Principal Source Editing.

Create and Package IP:

**Note: This section will show how to create an IP integrate (Intellectual Property integrate) from the current project that is the Clock Divider. The IPs can be created from all types of projects, like Block Diagrams or Design Sources. The steps are the same with small changes at the project begin. So follow the next steps:*

1. With the current project open, in “Main Menu” → “Tools” click on “Create and Package IP” (Figure 110) and click “Next” on the next window (not shown)

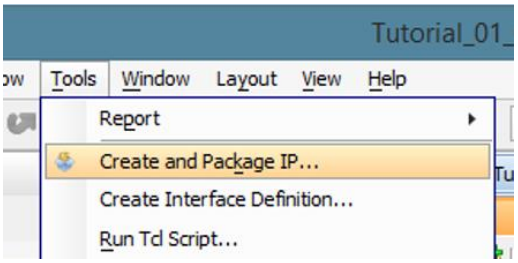


Figure 110 – Create and Package IP.

2. Check the option “Package your current project” and click “Next” (Figure 111)

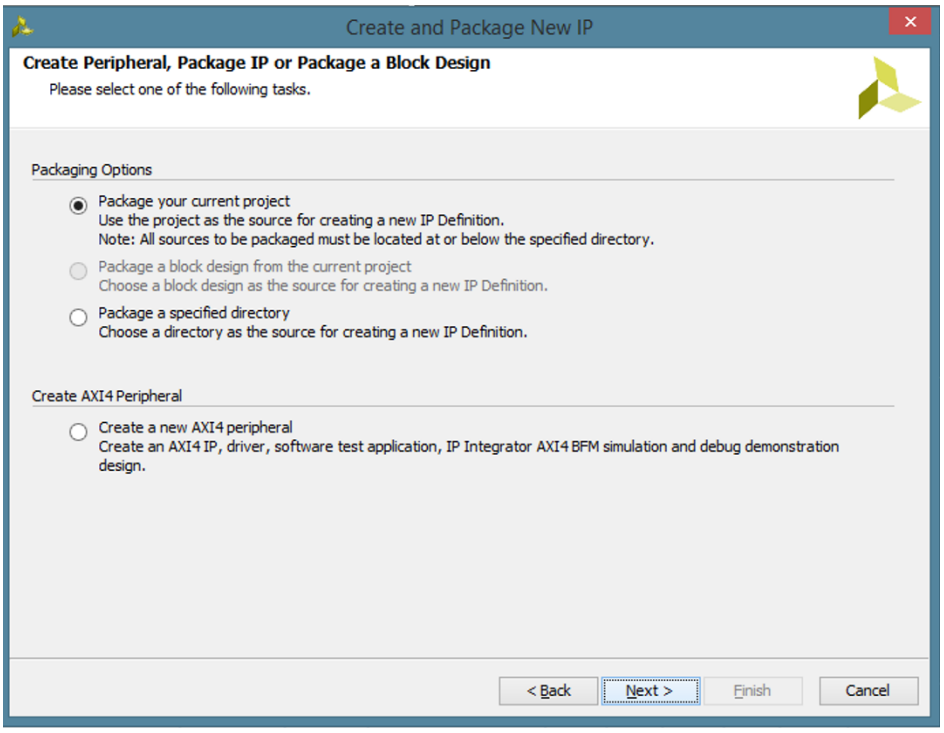


Figure 111 – Create Peripheral, Package IP or Package a Block Design.

3. In the next step just click “Next” (Figure 112) and then click “Finish” (Figure 113)

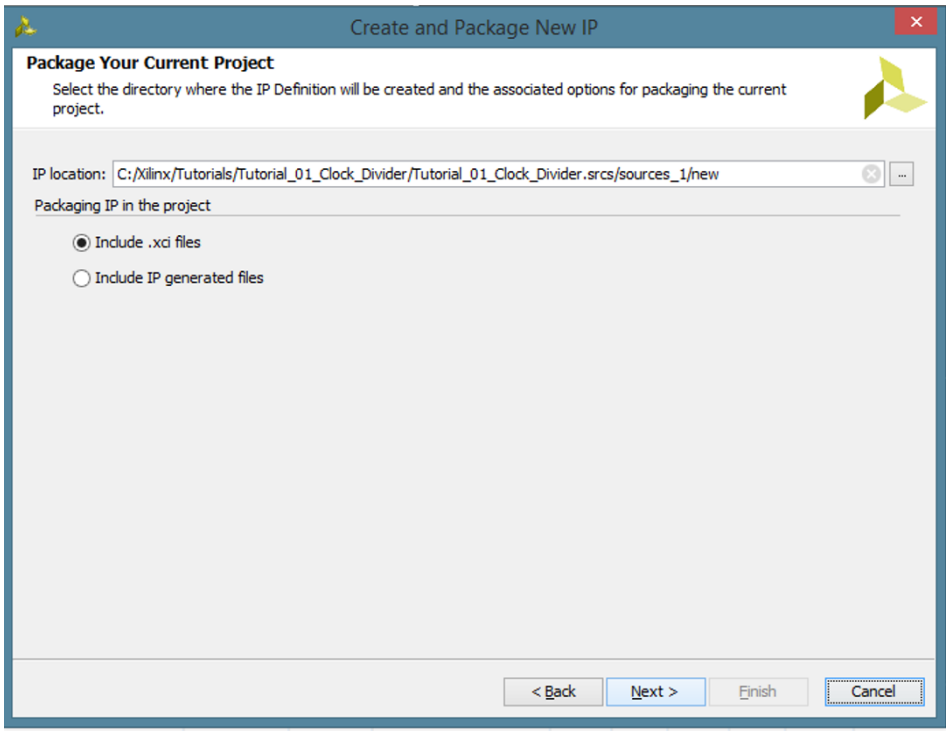


Figure 112 – Package Your Current Project.

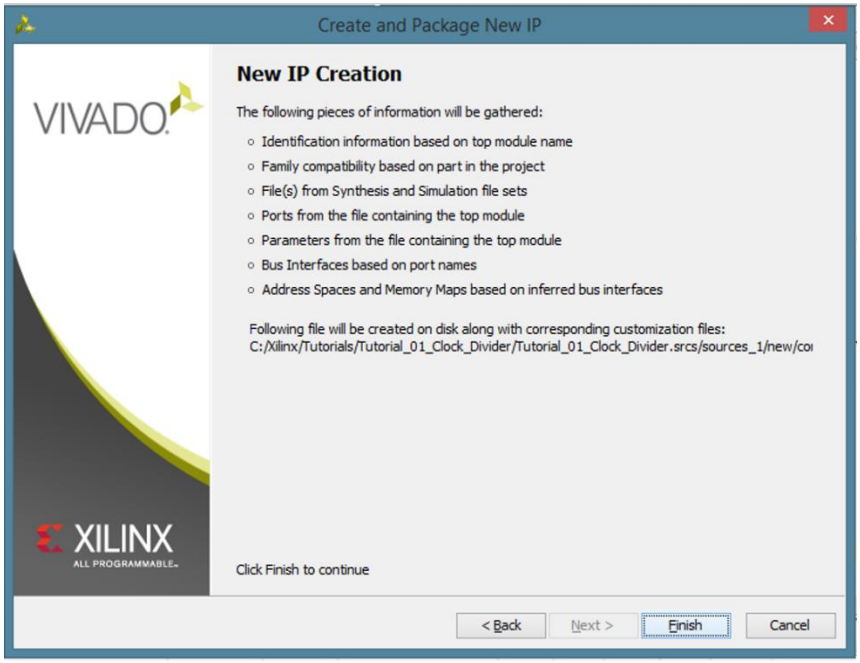


Figure 113 – New IP Creation.

4. Then a window will appear with the states of all the steps to Package IP. Confirm all steps and then in “Packaging Steps” → “Review and Package” click on the “Package IP” button (Figure 114). Click “OK” in the next window (Figure 115)

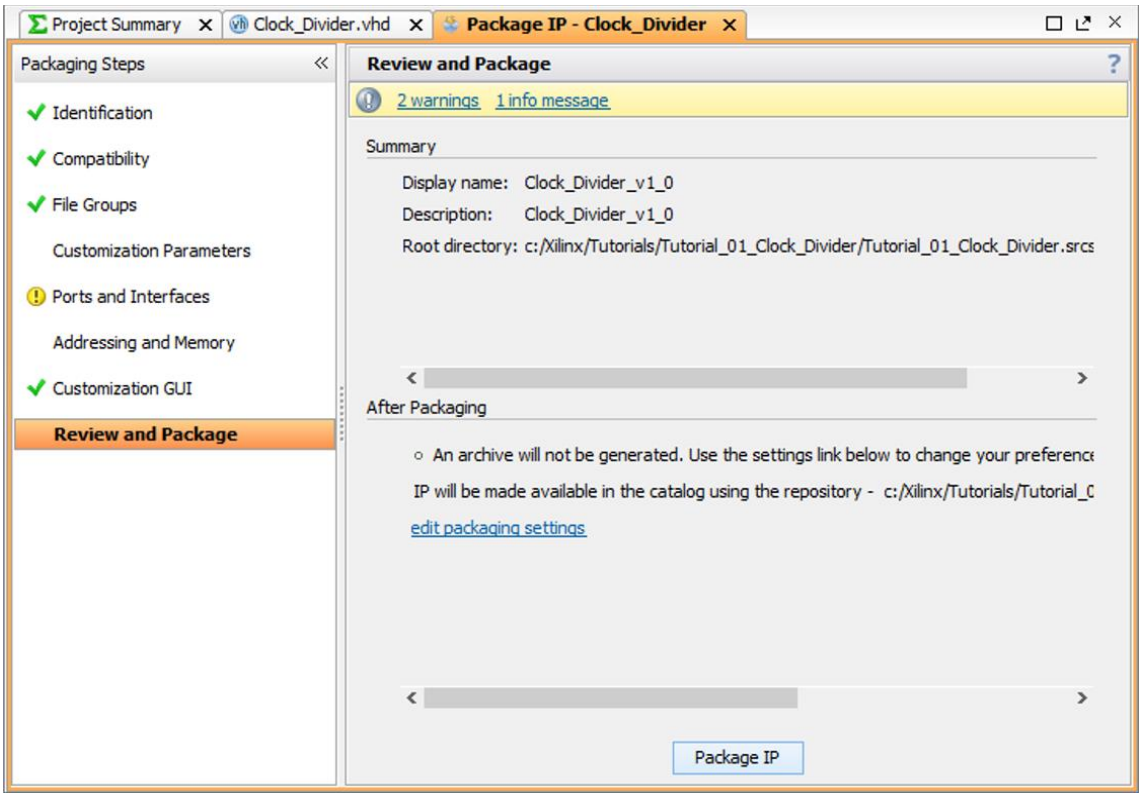


Figure 114 – Review and Package IP.

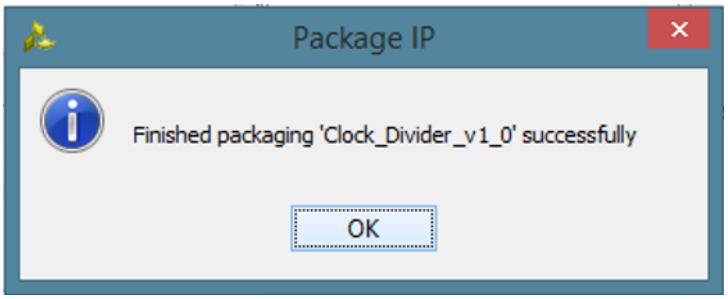


Figure 115 – Finished Packaging.

**Note: Now if you are following all the sections of this tutorial, close the project that was packaged, clicking on the "X" (Figure 116) and come back to the project "Binary Counter"*

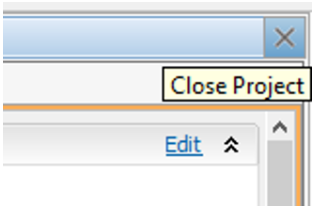


Figure 116 – Close Project.

Adding a New IP to the Project:

Now in the Design Project Binary Counter follow the next steps to add the IP created in the project Clock Divider.

1. In "Flow Navigator" → "Project Manager" click on "IP Catalog" (Figure 117)

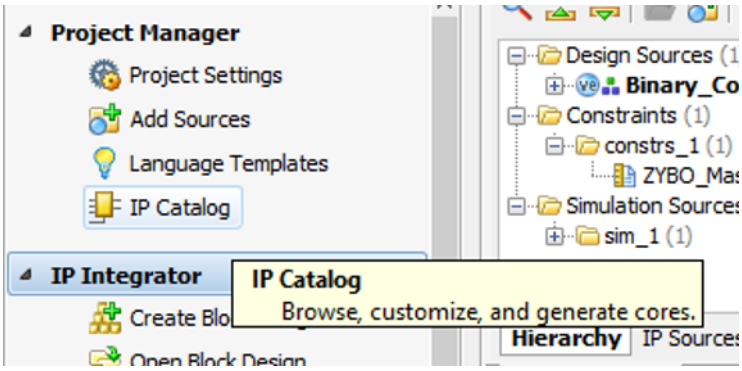


Figure 117 – IP Catalog.

2. So in the “IP Catalog” window, in the right menu, click on “IP Settings” (Figure 118)

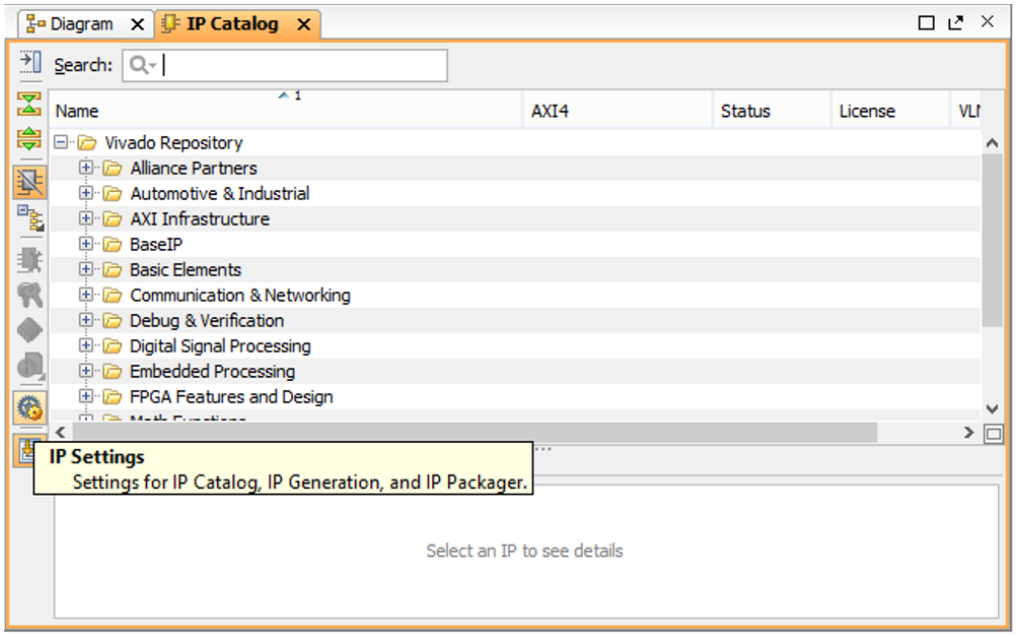


Figure 118 – IP Settings.

3. Now it is necessary to add the repository of the created IP to be able to use it. So click on the “+” (Add Repository) (Figure 119)

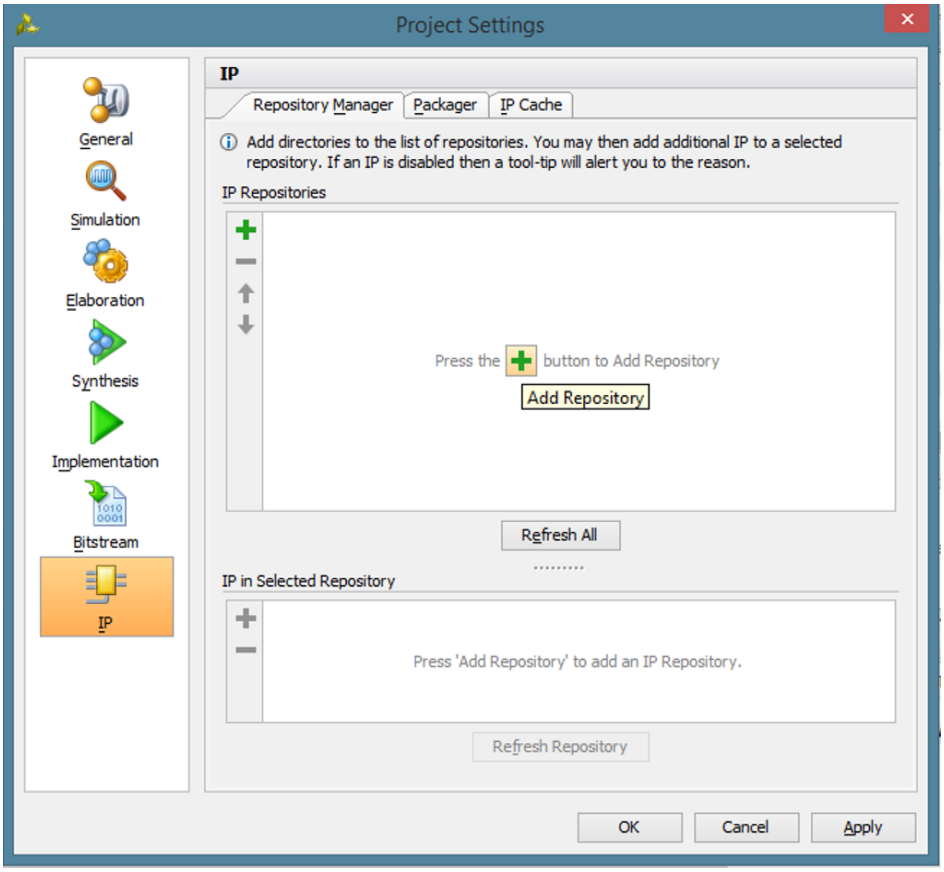


Figure 119 – Add Repository IP.

4. Then find the repository (folder) where the project Clock Divider was saved, and click “Select” (Figure 120)

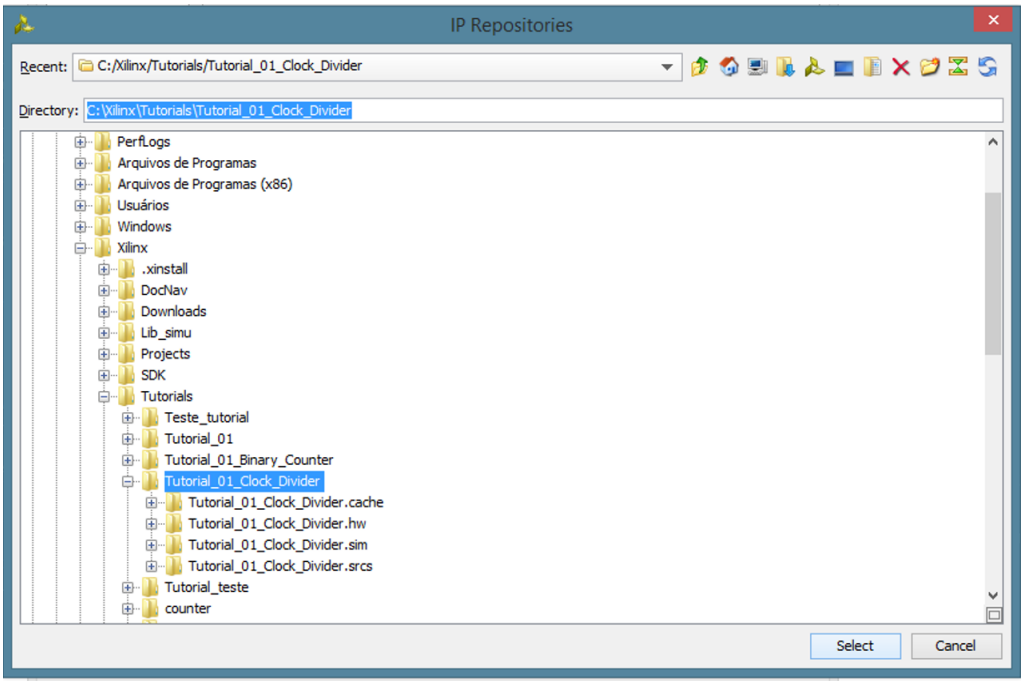


Figure 120 – IP Repositories.

5. Confirm if the directory and the IP are right and click on “Apply” and then on “OK” (Figure 121)

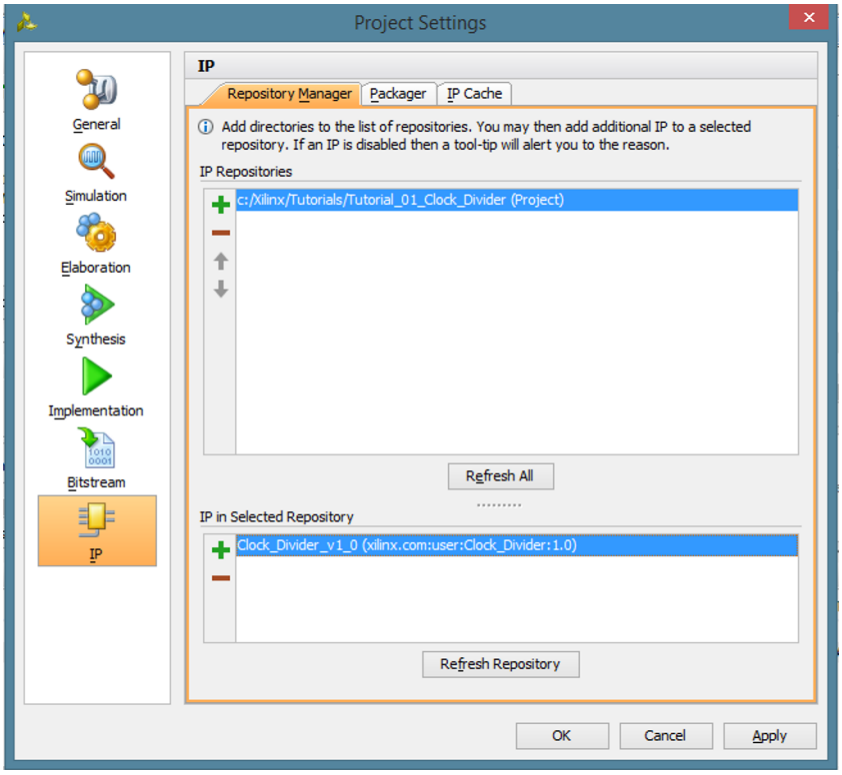


Figure 121 – Repository Manager.

6. In “IP Catalog” → “ User Repository” → “User IP” double click on “Clock_divider” as in Figure 122 and then click on “Add IP to Block Design” (Figure 123)

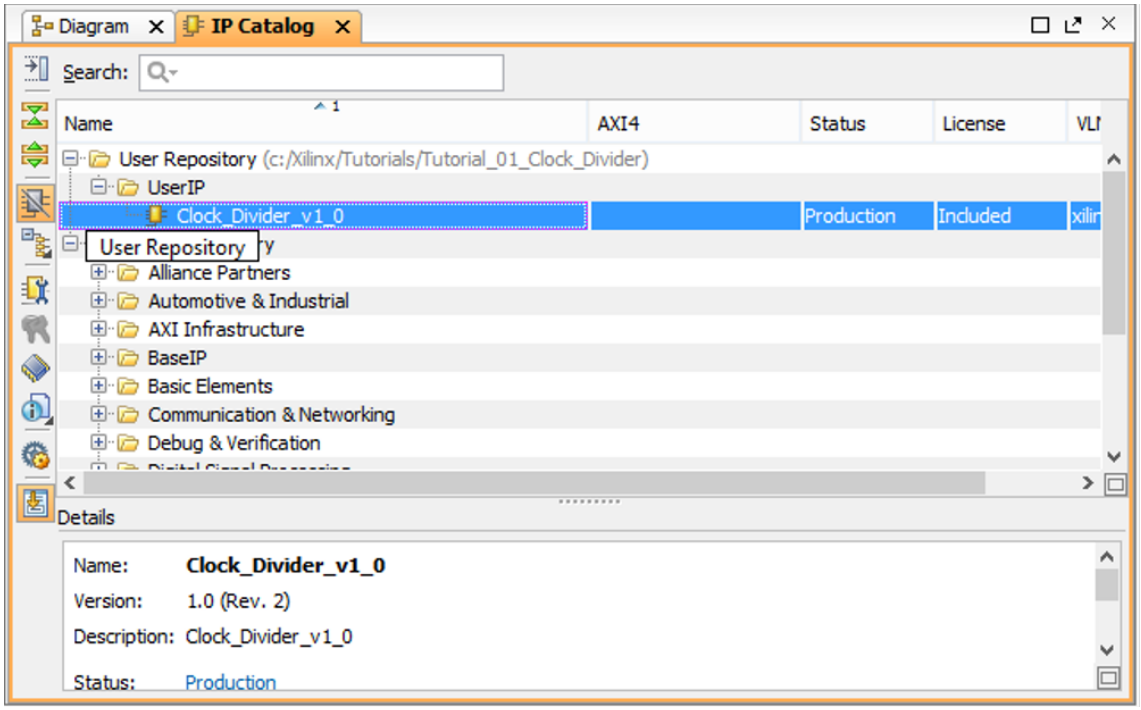


Figure 122 – IP Catalog.

Figure 123 – Add IP.

7. Then the Block IP Clock_Divider will appear (Figure 124)

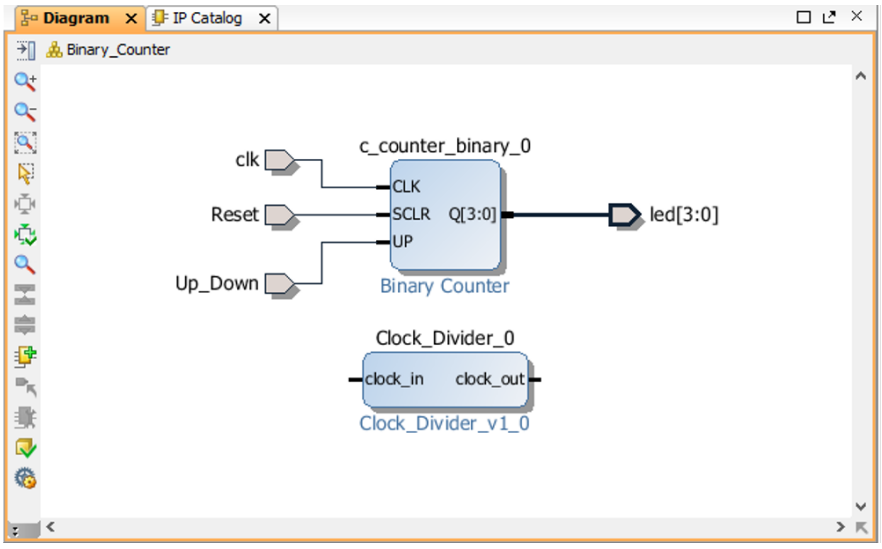


Figure 124 – Block Diagram.

8. Connect the block (Figure 125)

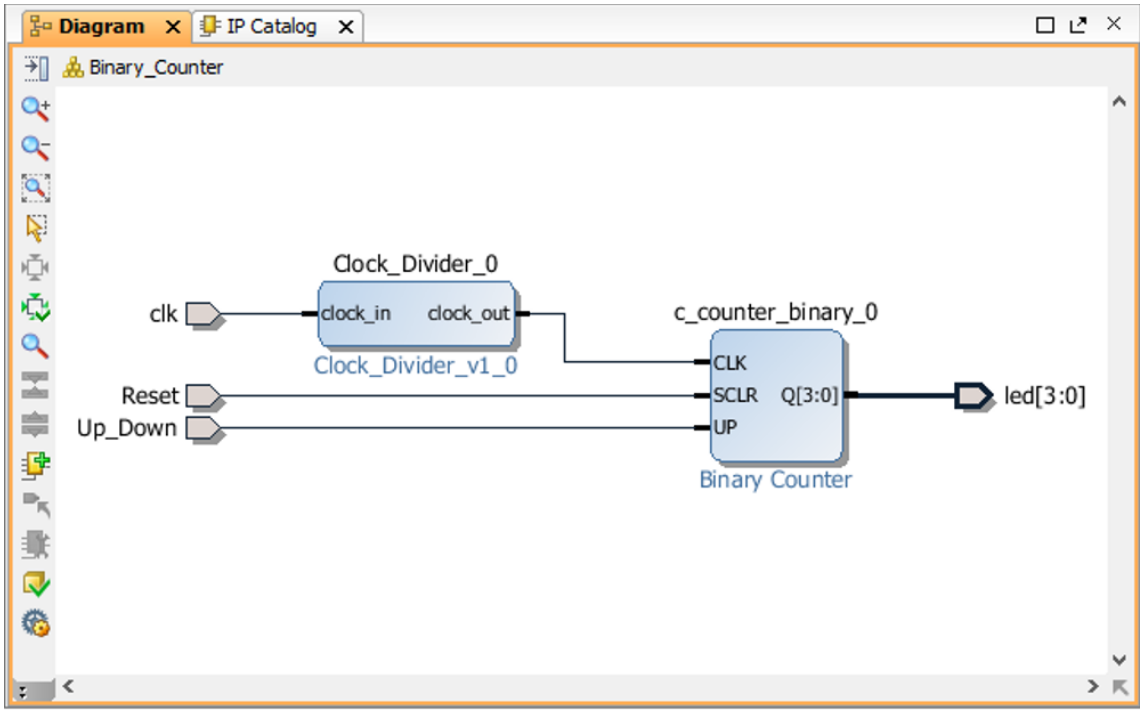


Figure 125 – Connecting the Clock_Divider.

Now the project is ready. Run all the steps to synthesize and then program the ZYBO board.

1. Run the synthesis again (Figure 126), and then click “OK” (Figure 127) and “Save” (Figure 128)

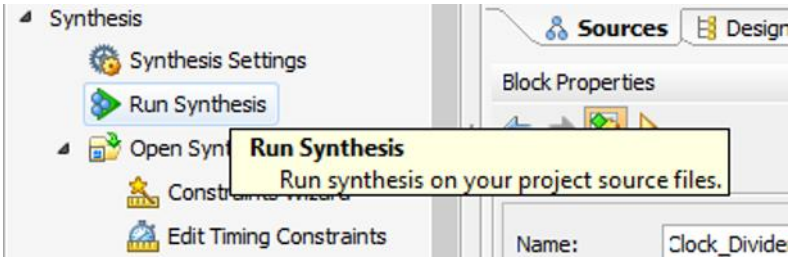


Figure 126 – Run Synthesis.

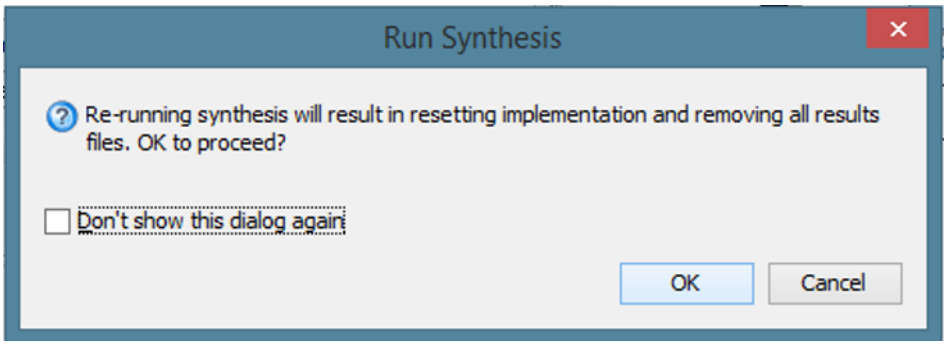


Figure 127 – Re-running Synthesis.

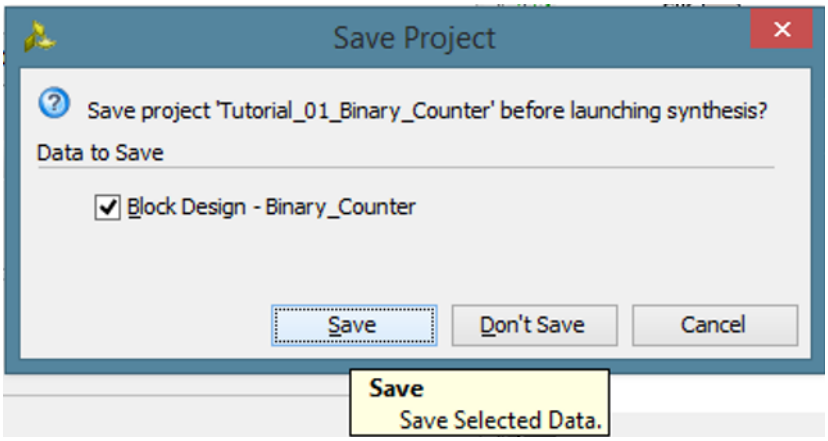


Figure 128 – Save Project.

2. When the synthesis is finished, check the option “Run Implementation” and click on “OK” (Figure 129):

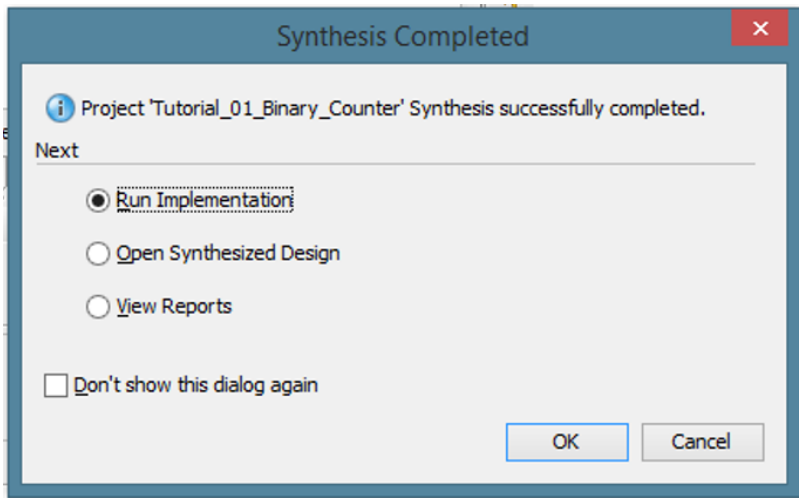


Figure 129 – Synthesis Completed.

Now the project is almost ready to be programmed into the ZYBO board. However, a final step is needed: Generate Bitstream, where the Vivado will convert what was created in a bit stream to program the board.

Generate Bitstream:

1. When the implementation is finished, check the option “Generate Bitstream” and click “OK” (Figure 130). Then check the option “Open Implemented Design” and click “OK” (Figure 131)

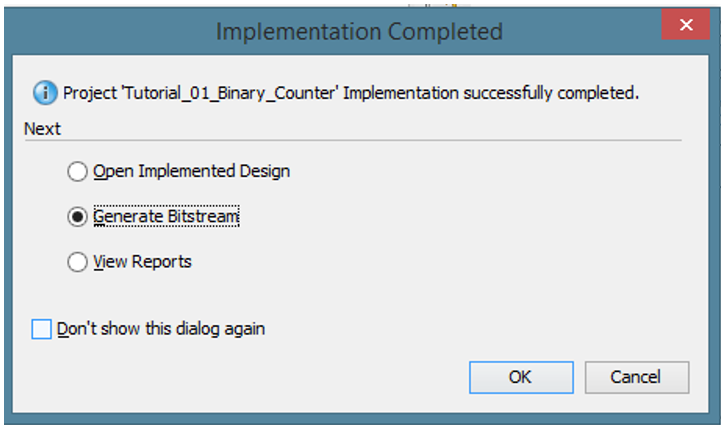


Figure 130 – Implementation Completed.

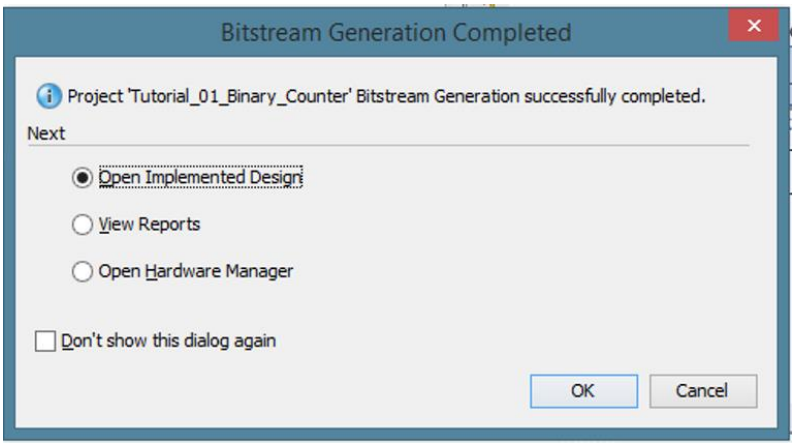


Figure 131 – Bitstream Generation Completed.

**Note: To Generate the Bitstream at any time, in “Flow Navigator” → “Program and Debug” click on “Generate Bitstream”.*

Programming the ZYBO Board:

After generating the bit stream to program the ZYBO board follow the next steps:

1. First, connect the port marked (Figure 132) in the board to the computer with an USB cable and turn ON the board

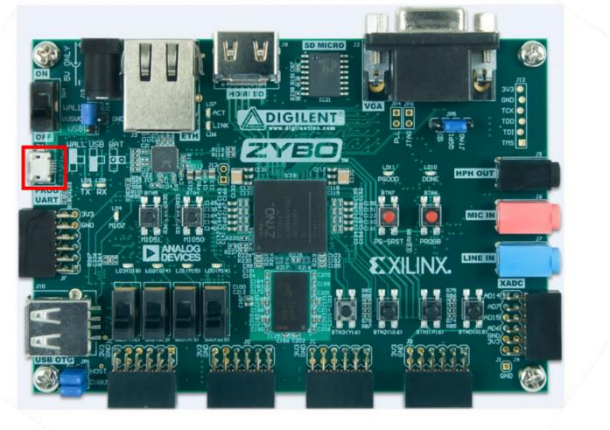


Figure 132 – USB Program Port of ZYBO Board.

2. On the “Flow Navigator” → “Program and Debug” → “Hardware Manager” → “Open Target” click on “Open New Target” (Figure 133), (Figure 134) and (Figure 135)

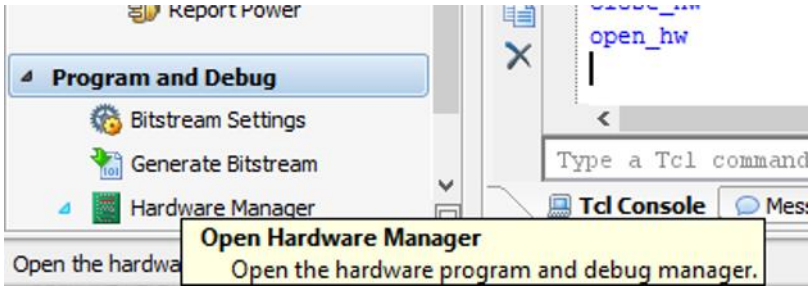


Figure 133 – Open Hardware Manager.

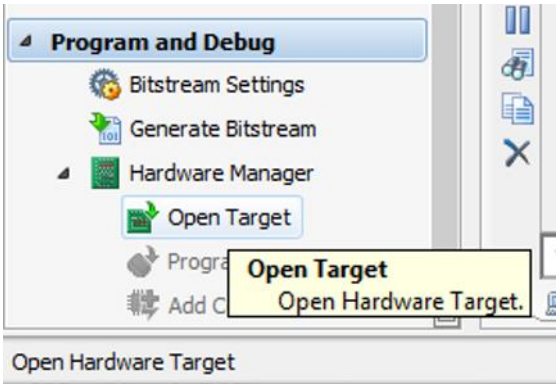


Figure 134 – Open Target.

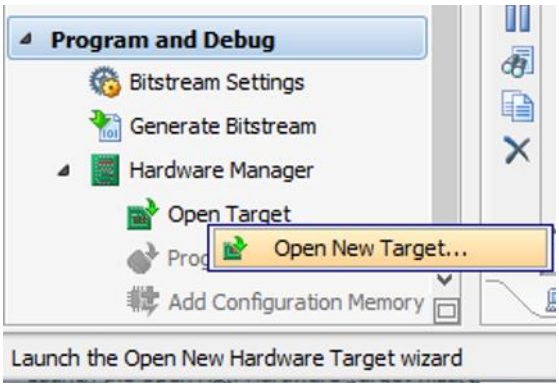


Figure 135 – Open New Target.

3. Click “Next” on the first window (Figure 136) and in the next select the option Connect to: “Local Server” and click “Next” (Figure 137):

Figure 136 – Open Hardware Target.

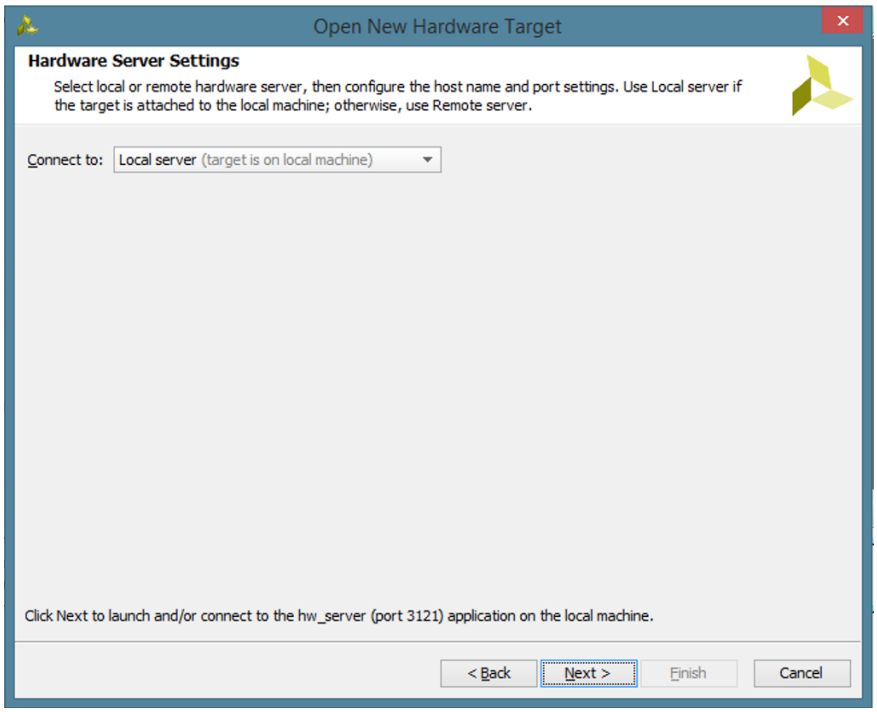


Figure 137 – Hardware Server Settings.

4. Select the hardware and confirm the information (Figure 138) and click "Next" and then click "Finish" (Figure 139):

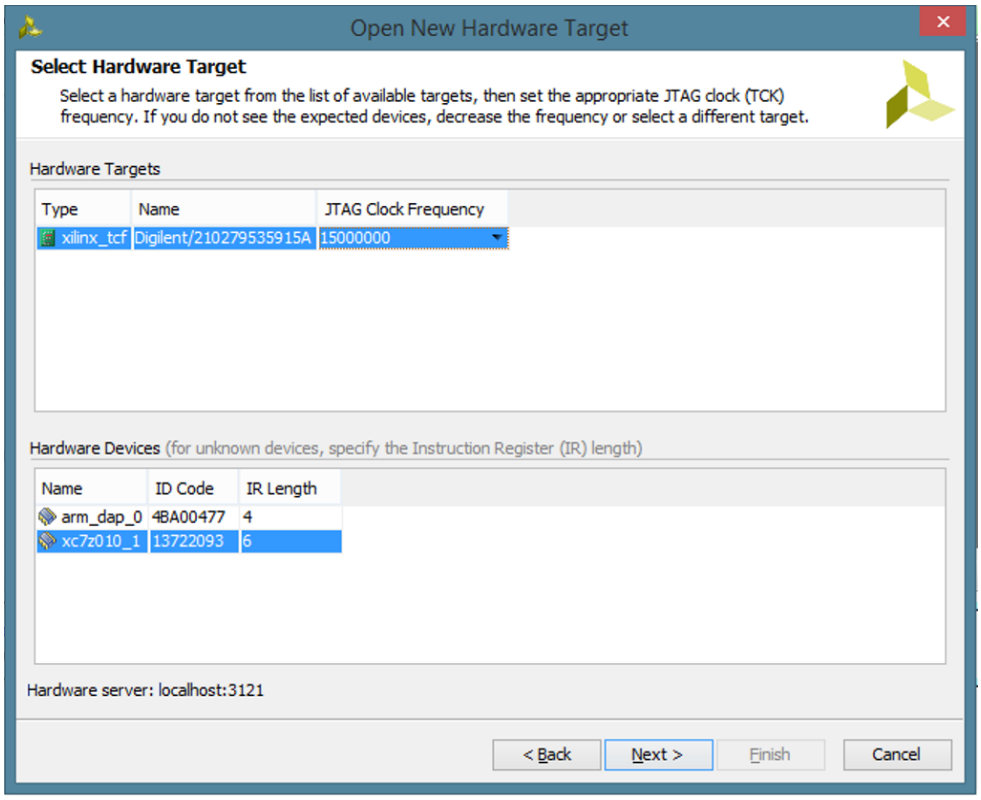


Figure 138 – Select Hardware Target.

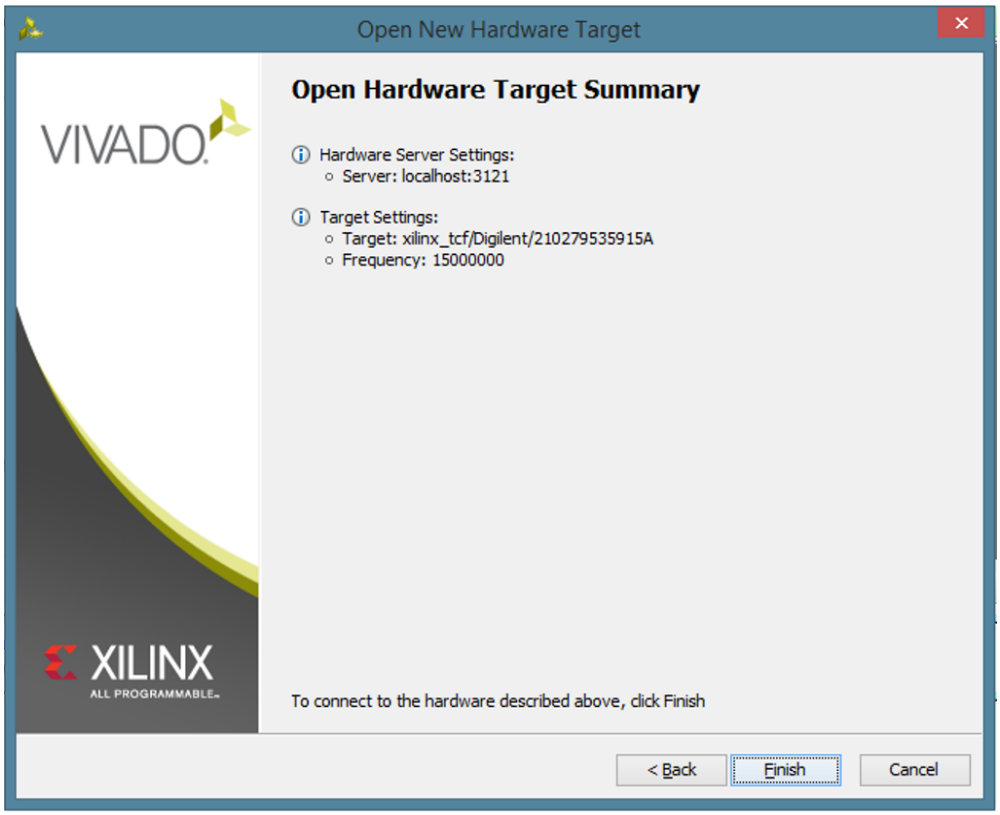


Figure 139 – Open Hardware Target Summary.

5. So in the “Flow Navigator” → “Program and Debug” → “Hardware Manager” click on “Program Device” and then click on the device “xc7z010_1” (Figure 140) and (Figure 141)

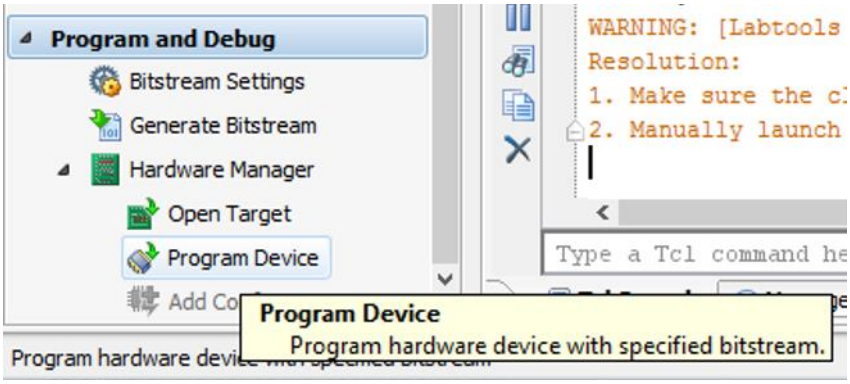


Figure 140 – Program Device.

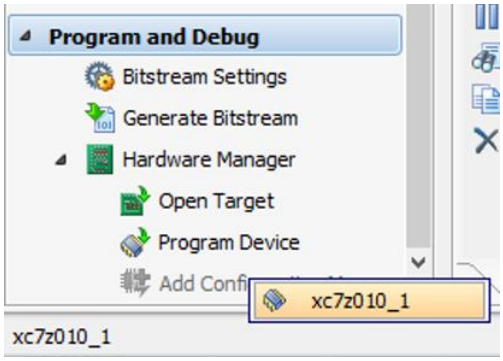


Figure 141 – Device xc7z010_1.

6. Click on “Program” (Figure 142) and wait a moment, then confirm the operation in the ZYBO Board (Figure 143), and using the button and the key that are marked is possible controller the counter:

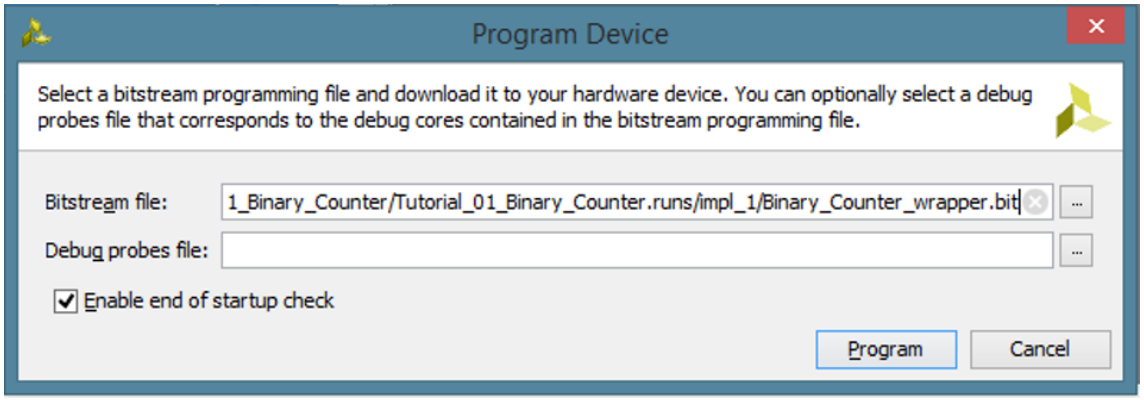


Figure 142 – Program Device.

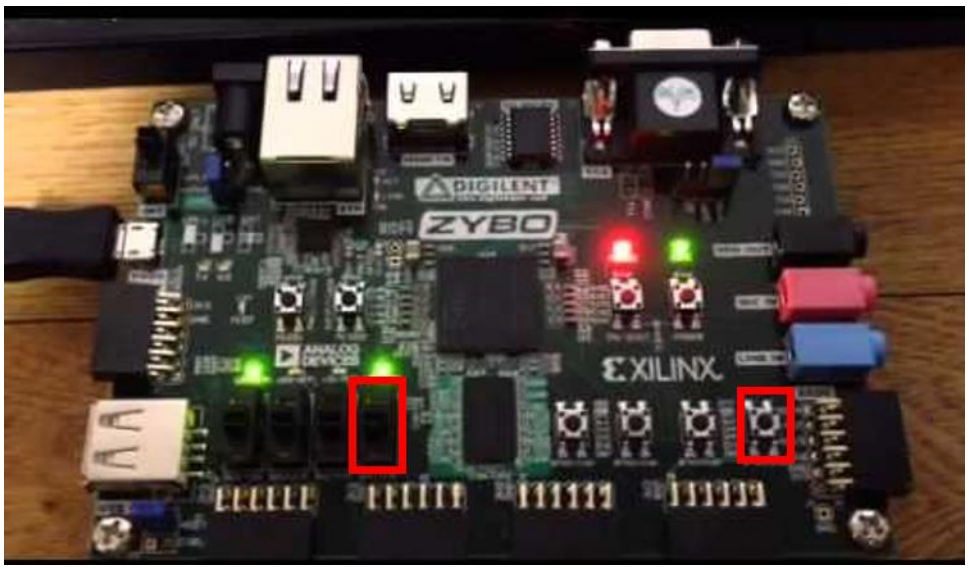


Figure 143 – ZYBO Board Binary Counter.

References:

[1] XILINX, "Vivado Design Suite," XILINX, [Online]. Available: http://www.xilinx.com/products/design-tools/vivado.html. [Accessed 14 Set 2015].

[2] J. Woldstad, "ZYBO Manual," 06 Mar 2015. [Online]. Available: https://reference.digilentinc.com/zybo:refmanual. [Accessed 11 Set 2015].

[3] XILINX, “Vivado Design Suite, User Guide: System-Level Design Entry,” 01 Jul 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug895-vivado-system-level-design-entry.pdf. [Accessed em 22 Oct 2015].

[4] XILINX, "Vivado Design Suite Synthesis," 24 June 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug901-vivado-synthesis.pdf. [Accessed 11 Set 2015].

[5] XILINX, "Vivado Design Suite Implementation," 24 June 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug904-vivado-implementation.pdf. [Accessed 11 Set 2015].

Anexo B. Tutorial II – Vivado and SDK 2014.x/2015.x Quick Start Tutorial to ZYBO Board

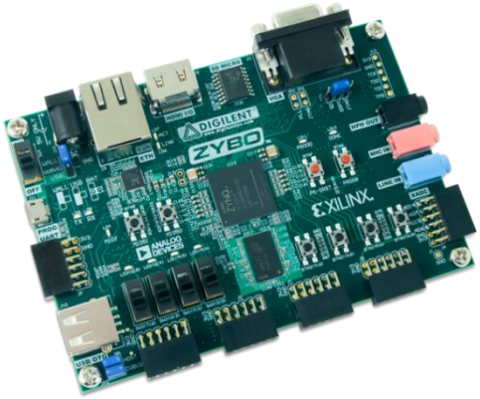
VIVADO AND SDK 2014.X/2015.X QUICK START TUTORIAL TO ZYBO BOARD

Héber Miguel dos Santos

Department of Electrical Engineering

Master in Electrical and Computer Engineering

NOV/2015



ZYNQ™ VIVADO™

Summary:

Figure List:

General software, hardware, files and Information needed:

- Xilinx Vivado 2014.x/2015.x;
- ZYBO Zync-7000 Development Board;
- ZYBO Board reference Manual;
- ZYBO Master XDC;
- Micro USB cable.

Software and hardware needed in specifics projects:

- SDK Webpack;
- Power supply 5V;
- Monitor with the VGA and HDMI interfaces;
- Micro SD-card minimal 4Gb;

Important Reference Links:

https://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,1198&Prod=ZYBO
http://www.zynqbook.com/

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug910-vivado-getting-started.pdf

Tutorial's Knowhow:

How to create a VHDL Design for a FPGA project with the Vivado Software, to be used by another project in C implemented in SDK from Xilinx, running in an ARM Cortex-A9 integrated in a Zynq processor part of the ZYBO board.

The project Implemented:

The project implemented in this tutorial is a basic 4-bit Binary Counter with the directions control UP/DOWN and Reset count. This implementation will use 4 (four) Leds, 1 (one) Push Button and 1 (one) switch available on the ZYBO board.

Start the Vivado Software:

1. To start Vivado, double-click the desktop icon [1]

Figure 1 – Vivado Icon.

2. Or on the Start menu select: "Menu Start" → "All Programs" → "Xilinx Design Tools" and click on "Vivado 201x.x".

**Note: Your start-up path is set during the installation process and may differ from the one above.*

The main page looks like the one in Figure 2

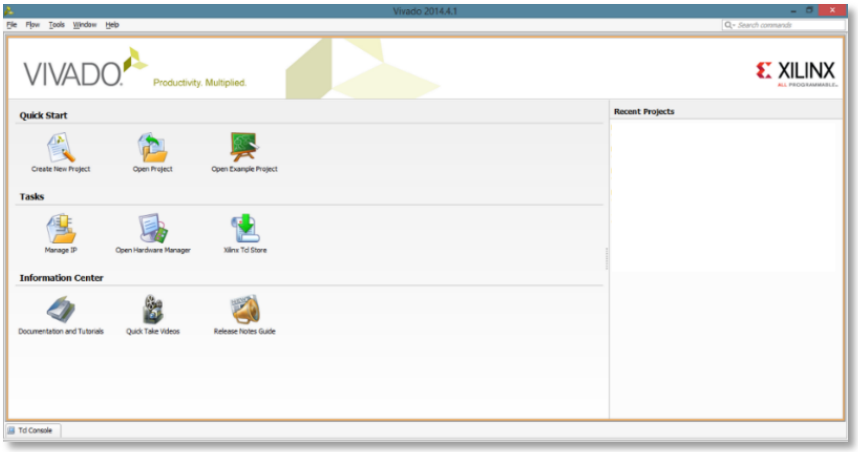


Figure 2 – Vivado Started.

Create a New Project Structure:

1. In the first window of Vivado *click* on "Create New Project" (Figure 3)

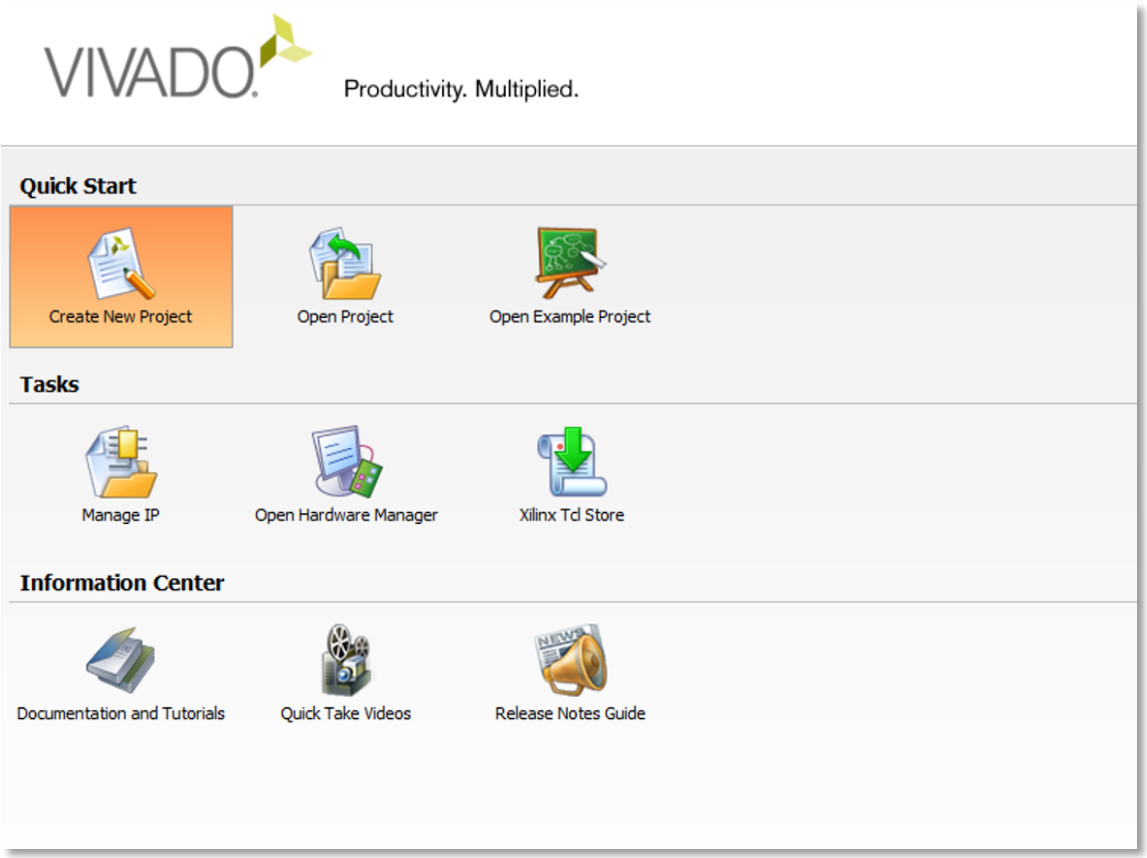


Figure 3 – Create New Project.

2. Click “Next” (Figure 4)

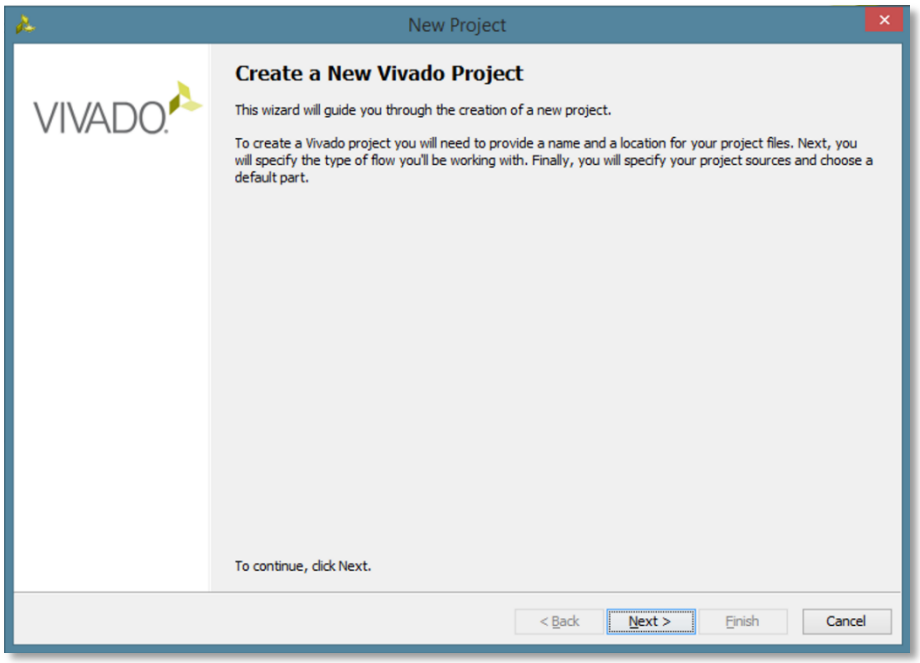


Figure 4 – Create a New Vivado Project.

3. In the next page, enter a project name along with its location and click “Next” (Figure 5)

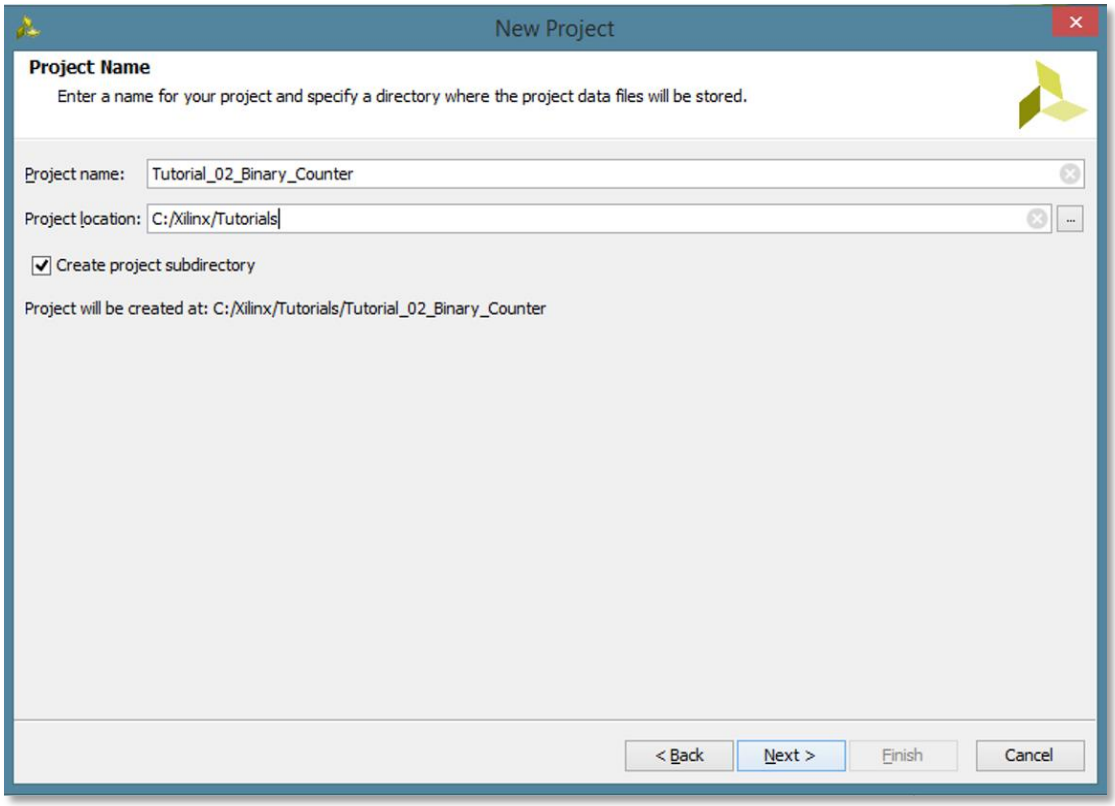


Figure 5 – Project Name.

4. Choose the correct project type: "RTL Project", verify that the "Do not specify sources at this time" box is checked (Figure 6) and click “Next”

   **Note: The “Register-Transfer-Level” create high-level representations of a circuit from the lower-level representations generated by the hardware description languages (HDLs) like Verilog and VHDL. It is a typical practice in modern digital design.*

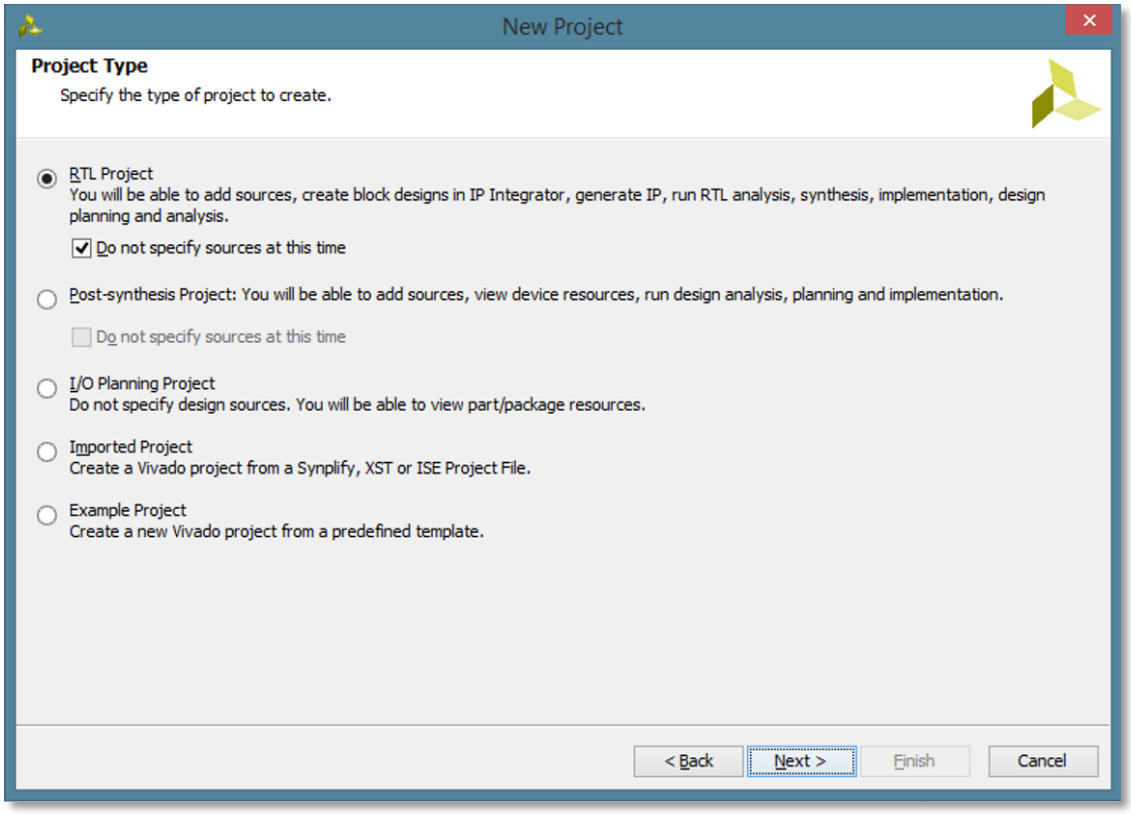


Figure 6 – Project Type.

5. Select the correct part. For this tutorial, we are using the Zynq-7000 with the specifications shown below (Figure 7). Click “Next” and after click “Finish” (Figure 8)
   - Product category: All
   - Family: Zynq-7000
   - Sub-Family: zynq-7000
   - Package: clg400
   - Speed Grade: -1
   - Temp Grade: C

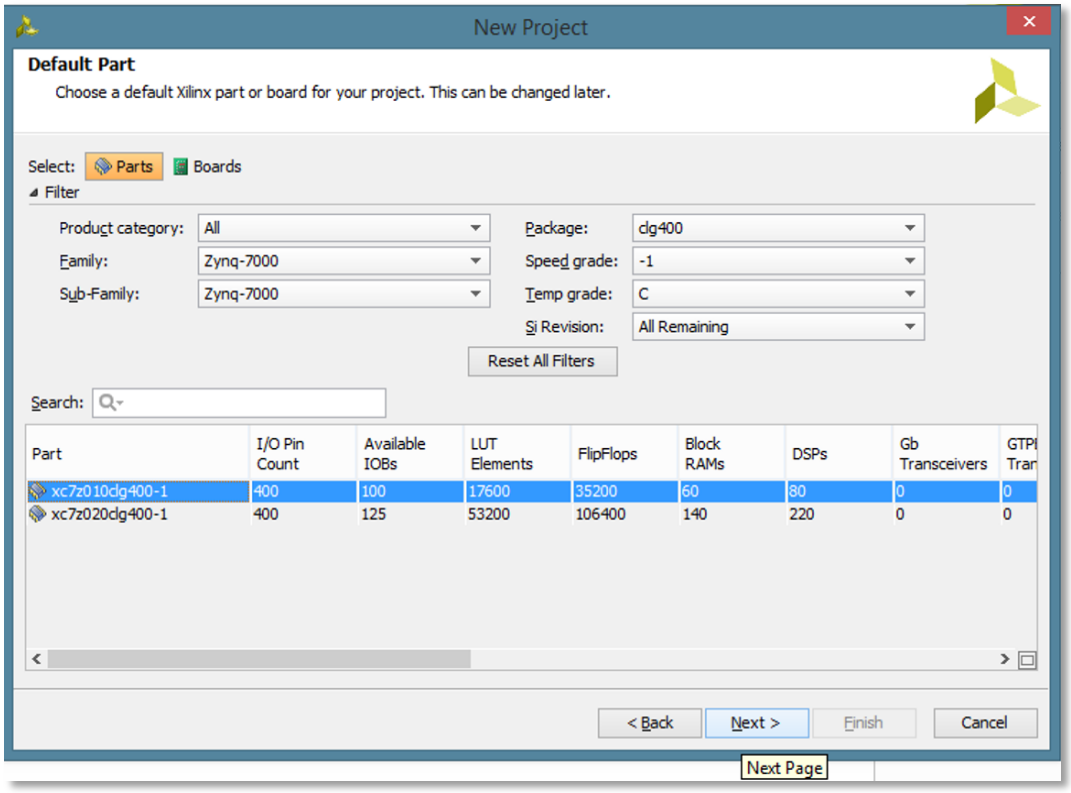


Figure 7 – Default Part.

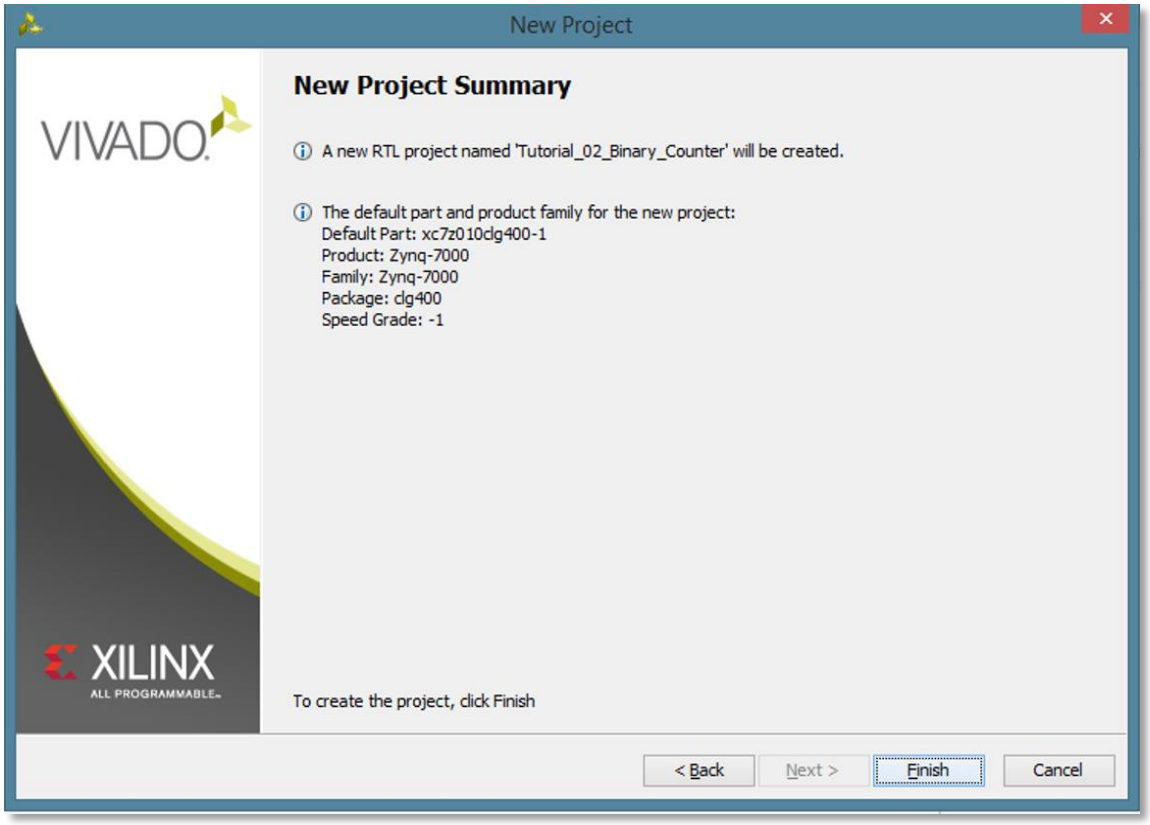


Figure 8 – New Project Summary.

6. After a few seconds, your project should be created and you look like the one in Figure 9

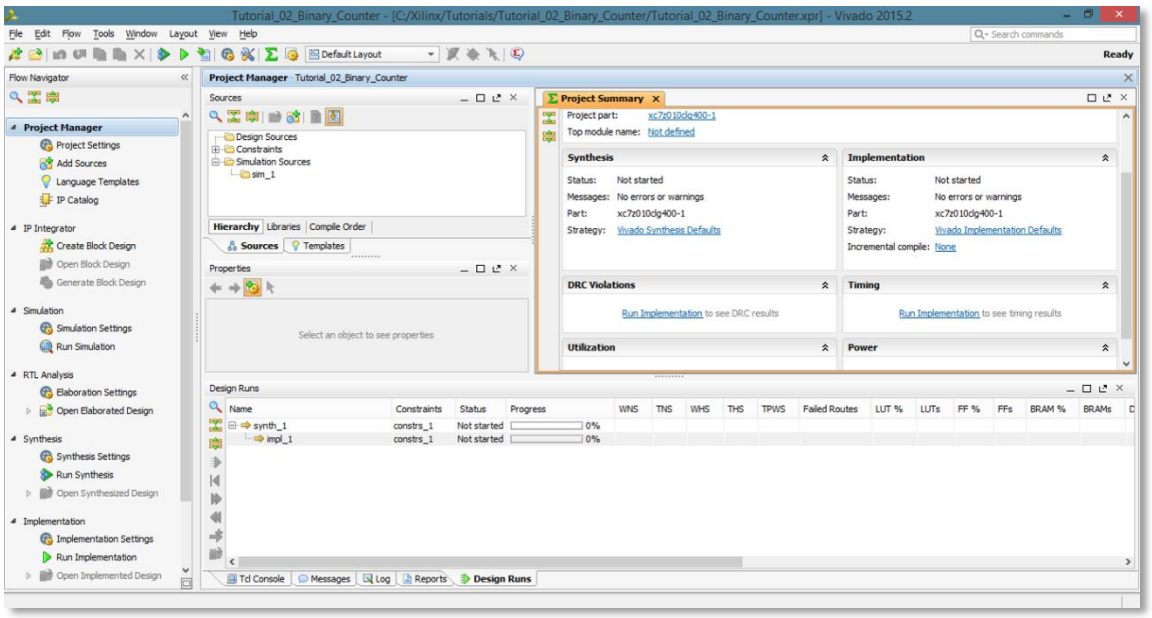


Figure 9 – Vivado Project Work Station.

Creating the Block Design:

**Note: The project will be structured in blocks, so it is necessary to create the design space.*

1. In “Flow Navigator” → “IP Integrator” click on “Create Block Design” (Figure 10):

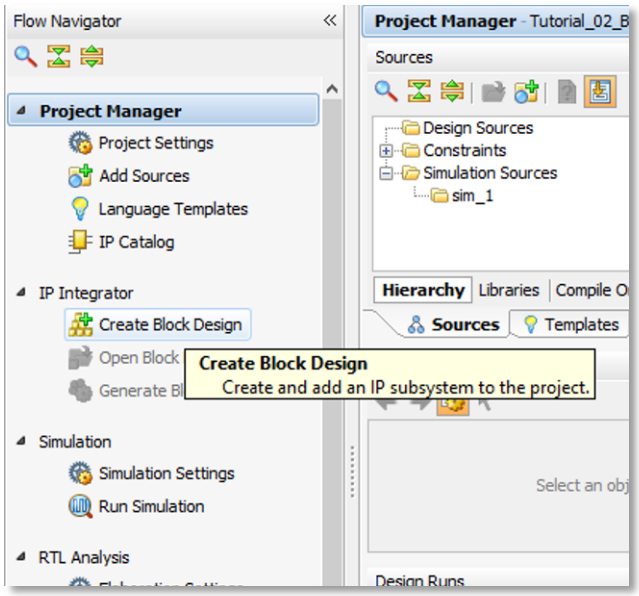


Figure 10 – Create Block Design.

2. On the new window specify the design name and click “OK” (Figure 11)

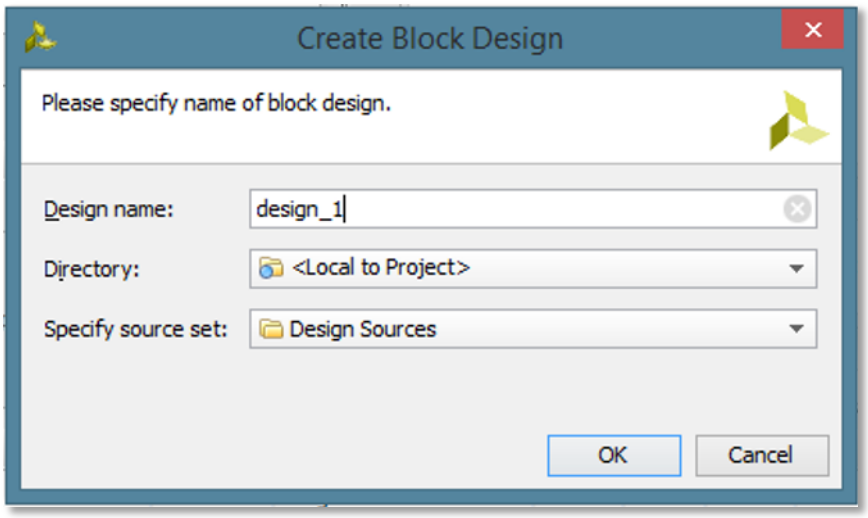


Figure 11 – Name of Block Design.

3. It will create a blank Block Diagram (Figure 12)

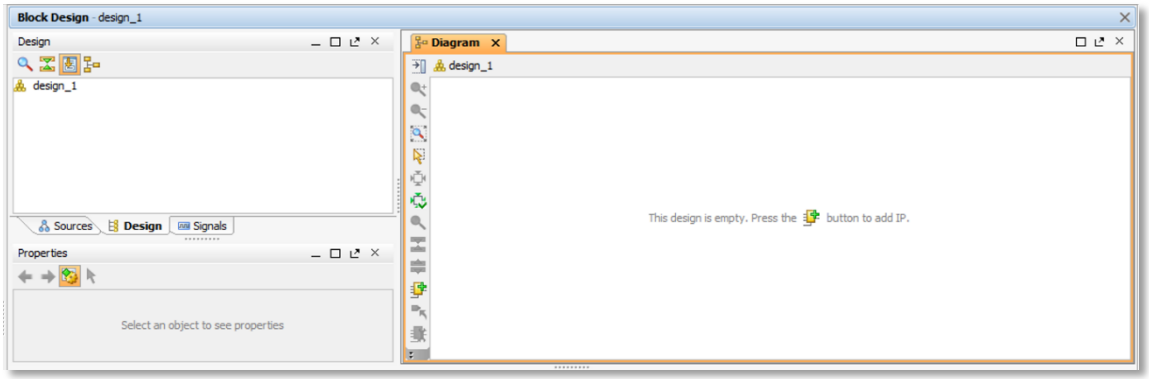


Figure 12 – Block Diagram Created.

4. Click on the "Add IP" image (Figure 13). Then a smaller window will appear within the Diagram window. In the search box, type "zynq" and double-click the “ZYNQ7 Processing System” option (Figure 14)

**Note: The focus of this project is to create a logical interface to the Zynq processor that will be programmed. The Zynq processor structure is instantiated using the IP Integrator from Vivado.*

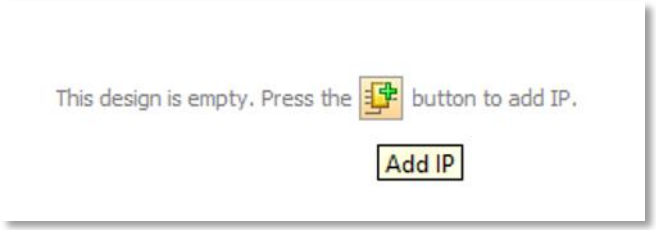


Figure 13 – Add IP.

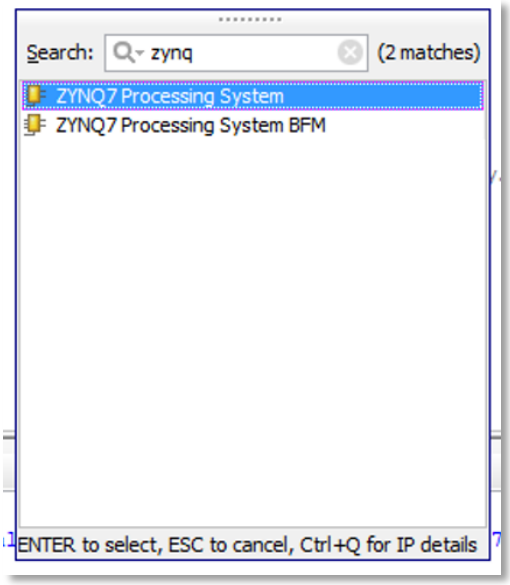


Figure 14 – Search IP ZYNQ Processor System.

5. The Zynq block will appear within the Diagram window (Figure 15)

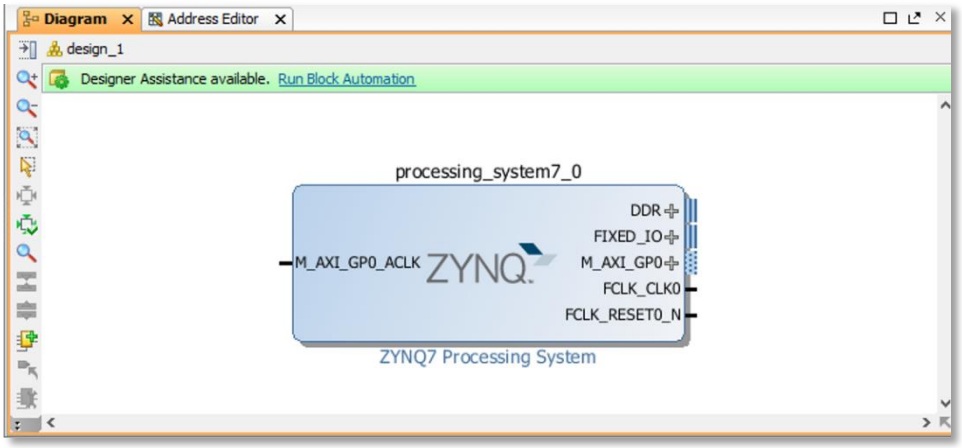


Figure 15 – IP Block of ZYNQ7 Processing System.

6. Next, click on “Run Block Automation” at the top of the Diagram Window Tab (Figure 16)

**Note: Vivado automatically creates the basic connections to the processor.*

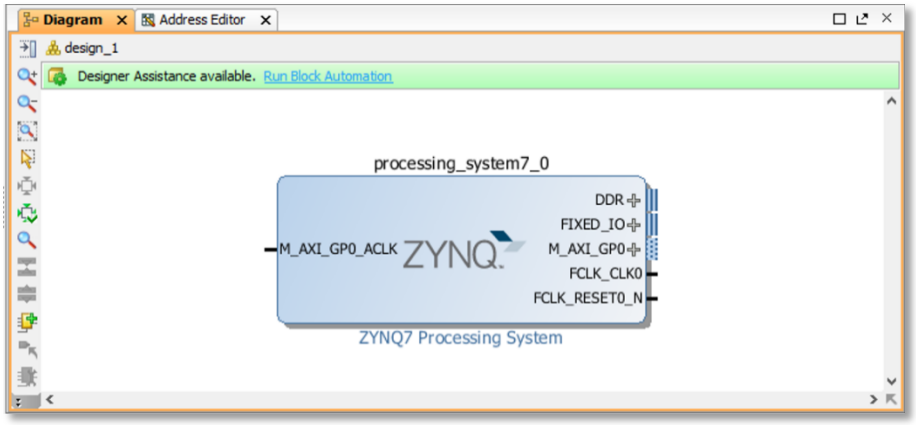


Figure 16 – Run Block Automation.

7. After that, a window will appear (Figure 17). Verify that the “processing_system7_0” checkbox is checked and that the “Cross Trigger In” and “Cross Trigger Out” drop down selections are both disabled and click “OK”

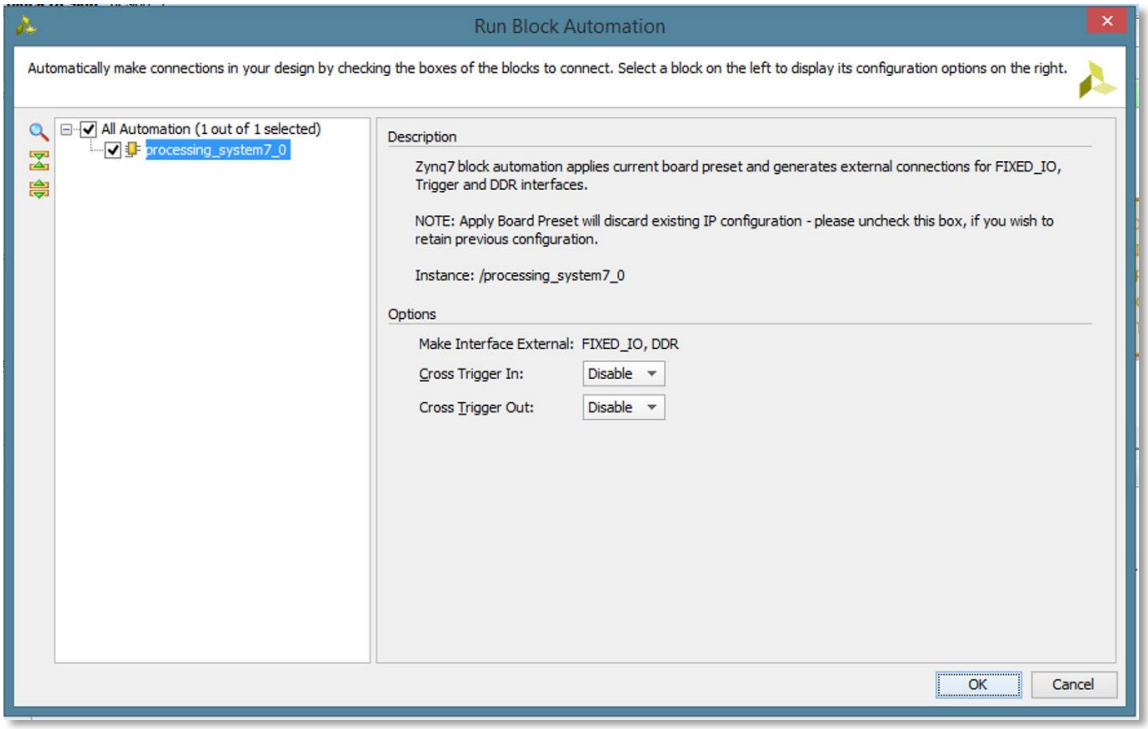


Figure 17 – Run Block Automation Settings.

8. Therefore, the “DDR” and “FIXED_IO” pins on the Zynq processing system block are connected to interface ports of the same name (Figure 18)

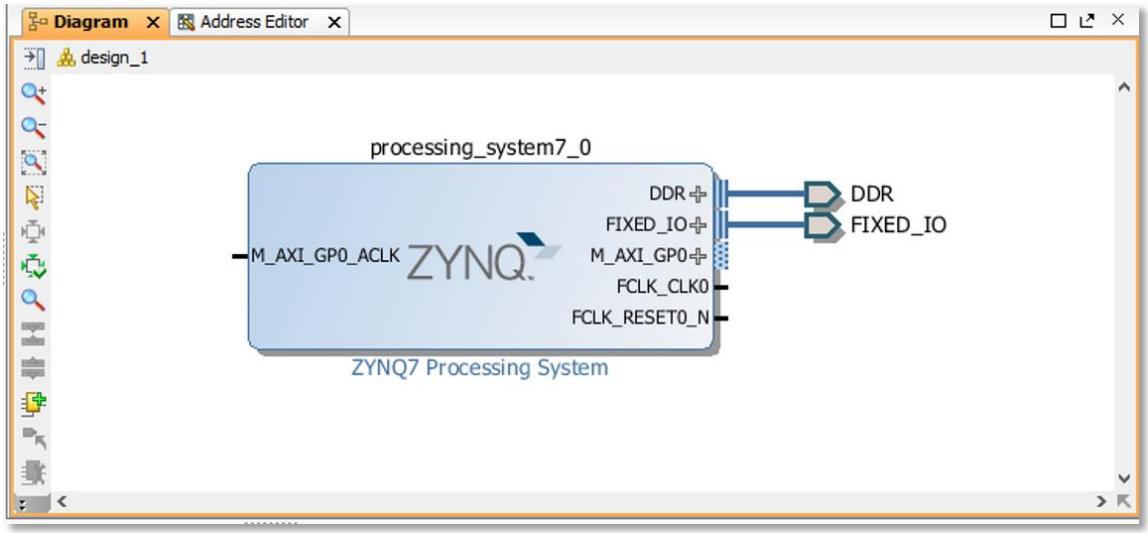


Figure 18 – IP Block ZYNQ with Ports Created.

9. Right click on anywhere within the “Diagram Window Tab” and click on “Add IP” (Figure 19)

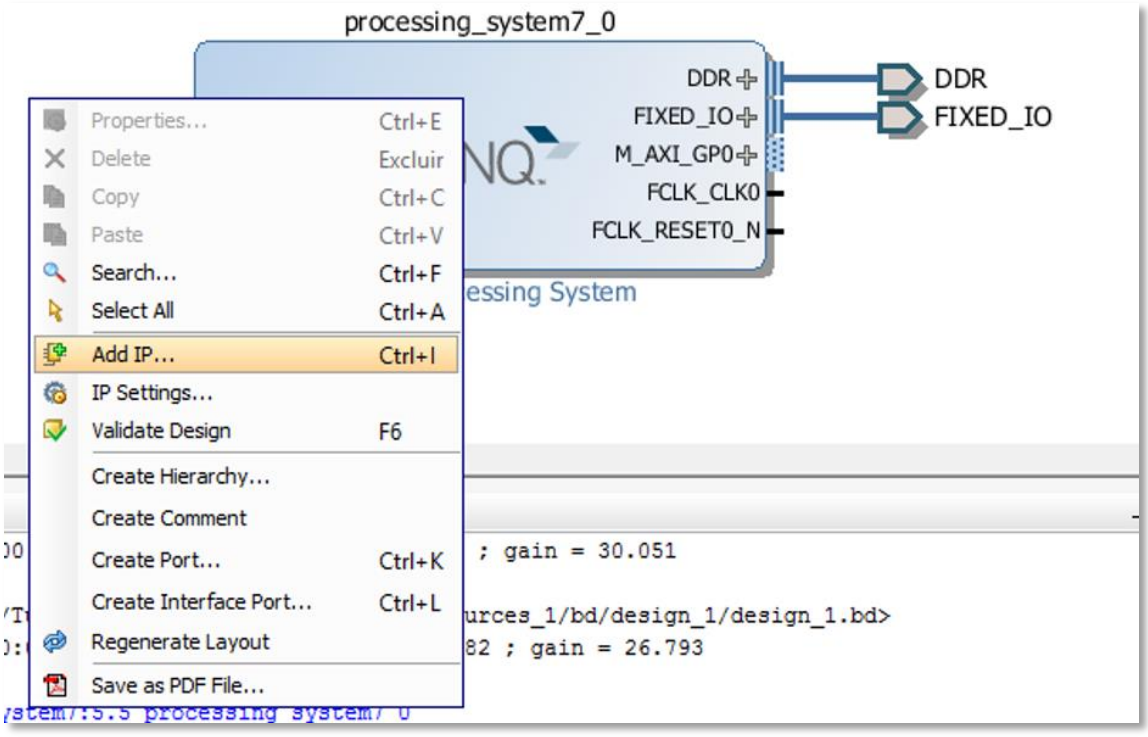


Figure 19 – Add IP.

10. Then a window will appear that is similar to that seen when we first created the Zynq processing system block. Within the search window, type “AXI_GPIO” (there should only be one item selected) (Figure 20). Double click on the selection. After some time, your Diagram Window Tab will appear (Figure 21)

**Note: The AXI IP integrator being added is a bus communication technology developed by Xilinx. The implemented bus facilitates the communication between the Processor and the logical interface. In this case, it enables the access to the GPIOs.*

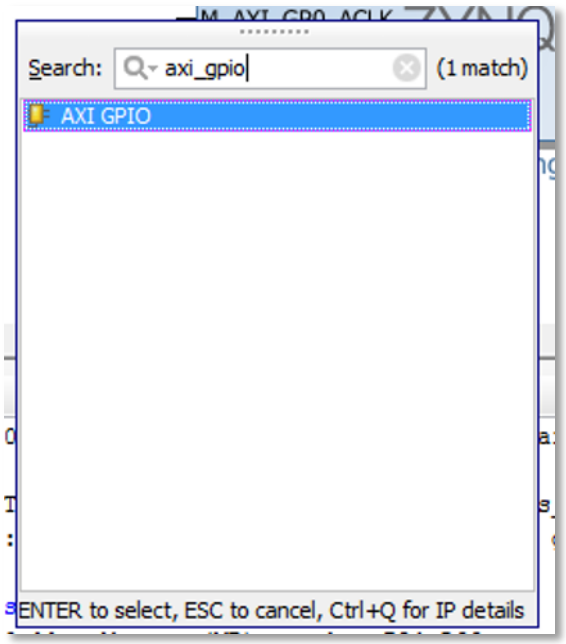


Figure 20 – Search IP AXI_GPIO.

11. When the AXI_BLOCK appear towards the top of the “Diagram Window Tab”, will see the “Run Connection Automation” link. Click on that link. After, a new Run Connection Automation window will appear. Check the checkbox “S_AXI” and verify that the option clock connection is set to "Auto", and click “OK” (Figure 22)

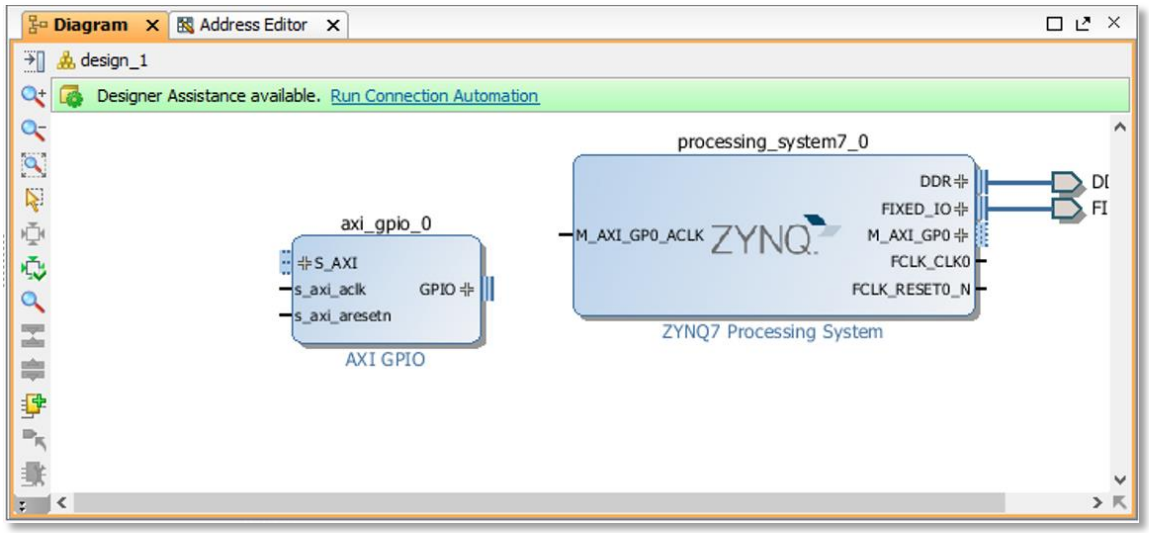


Figure 21 – AXI_GPIO IP Block Connection Automation to ZYNQ7 IP Block.

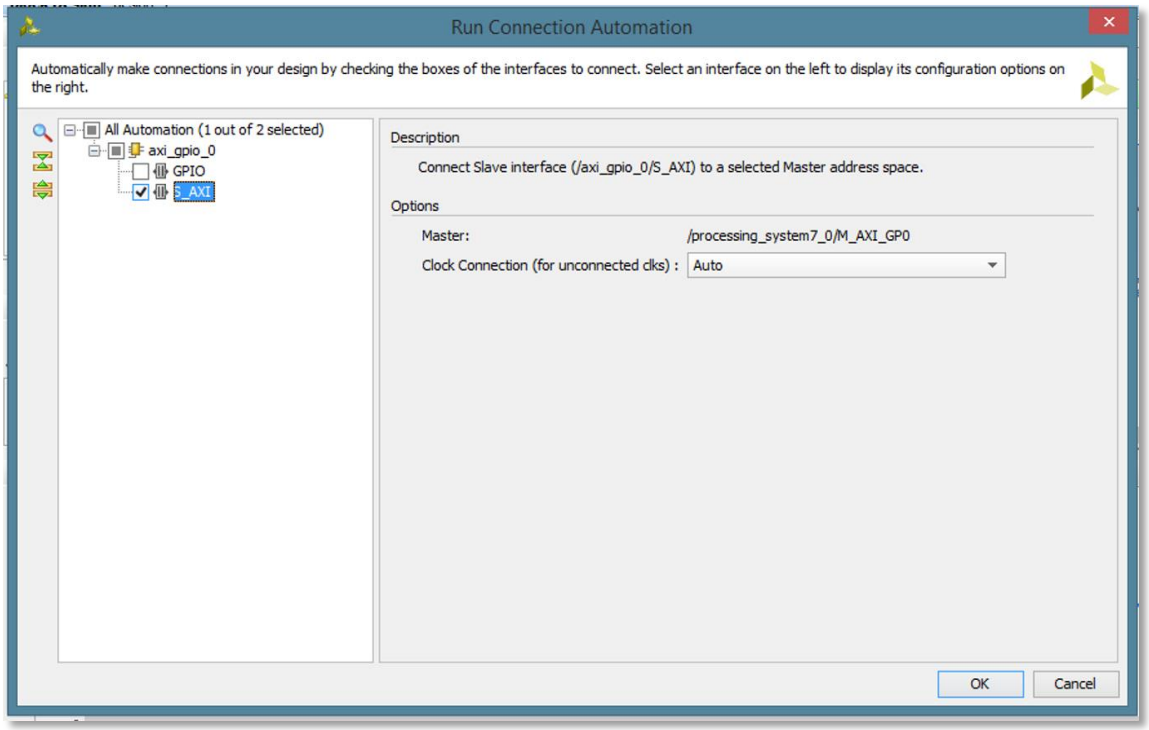


Figure 22 – Run Connection Automation.

12. After a few seconds, more system blocks should appear within the “Diagram Window Tab” (Figure 23). At the top of the window, click on “Run Connection Automation” link again

**Note: These complementary blocks created are needed to make the AXI bus work. The AXI Interconnect is the manager of the Communication Bus and the Processor System Reset is used to synchronize the communication.*

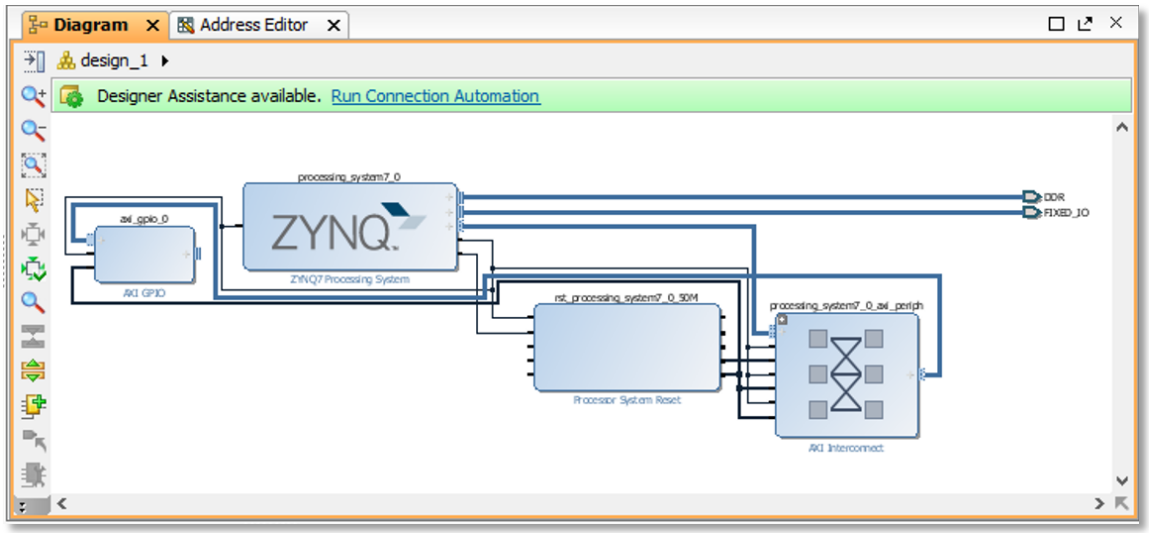


Figure 23 – Blocks Diagram after the First Automatic Connection.

13. This time, verify that the “GPIO” checkbox is checked and click on “OK” (Figure 24)

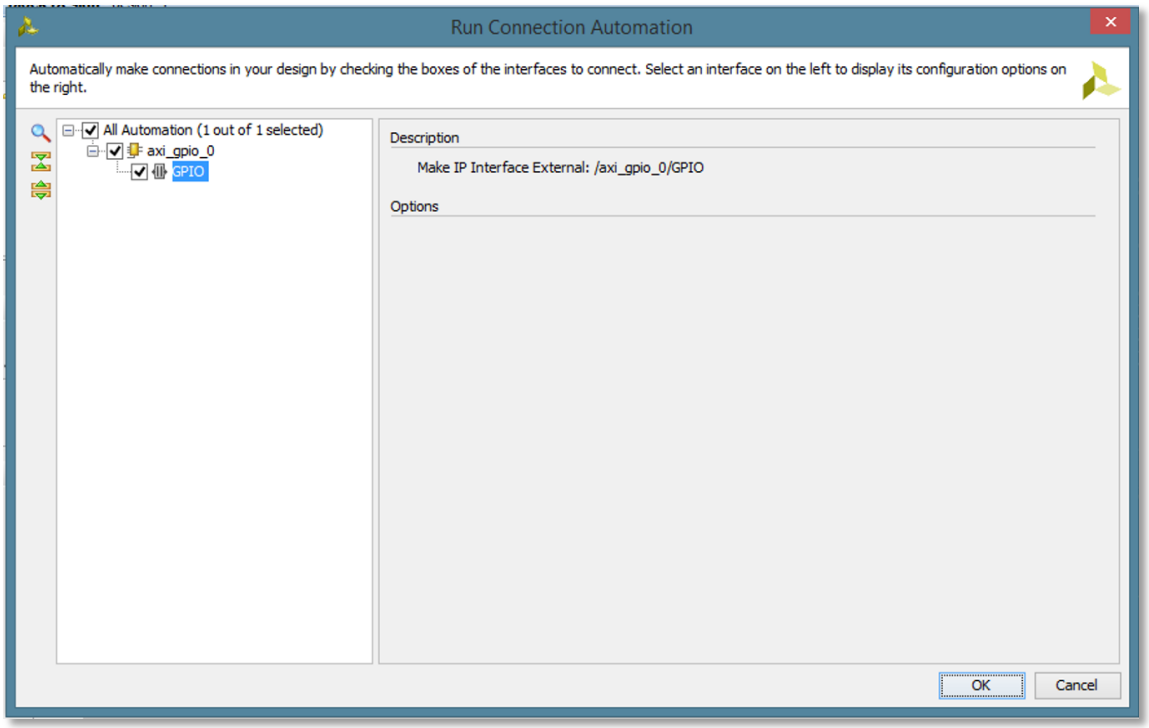


Figure 24 – Run Connection Automation.

14. The “Diagram Window Tab” should look as the one in Figure 25

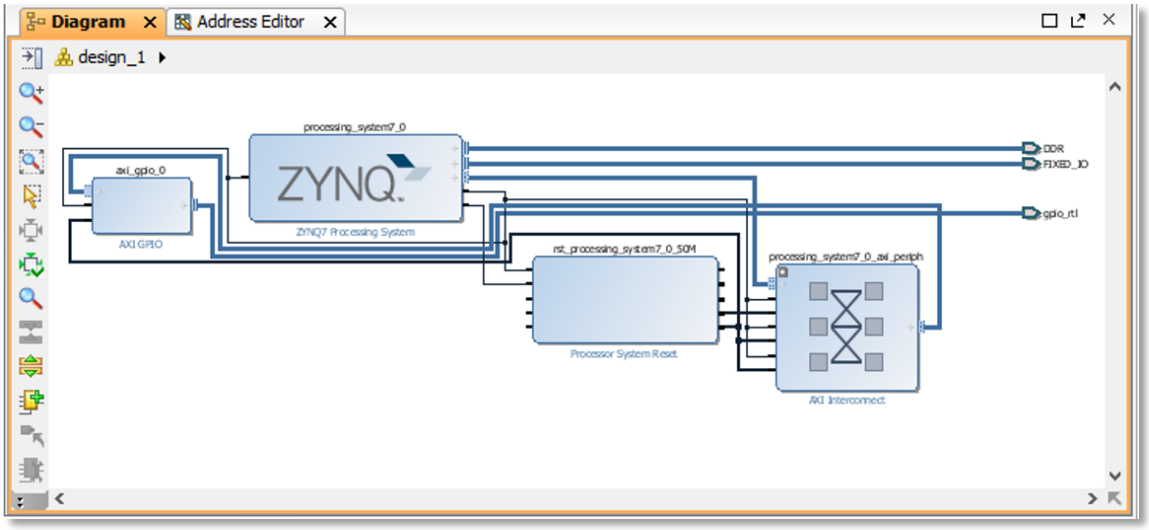


Figure 25 – The News Ports.

15. Then add one more AXI_GPIO that will be used by the Button and a Switch needed to control the counter state between counting Up or counting Down and Reset. So, follow again the last steps from step 9. When finished the “Diagram Window” will look as the one in Figure 26

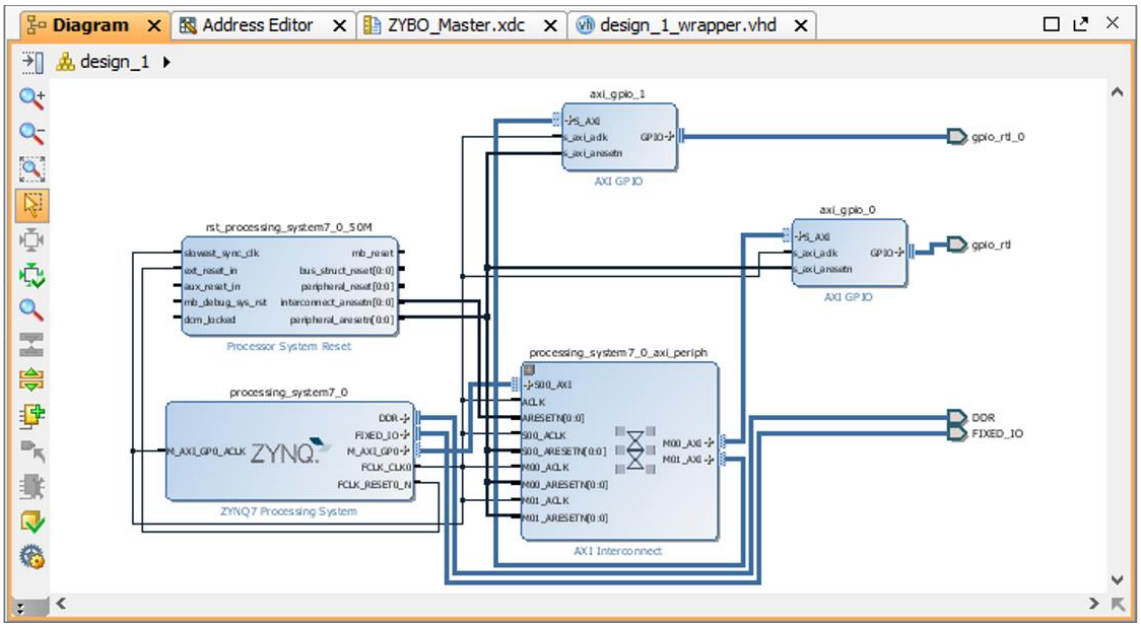


Figure 26 – Block Diagram Layout.

16. So just for aesthetics, right click within the “Diagram Window Tab” and click on the “Regenerate Layout” option (Figure 27)

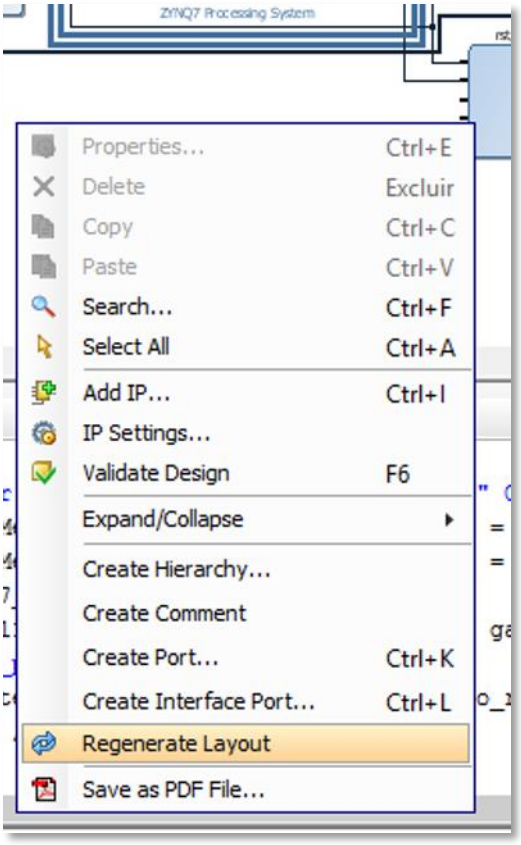


Figure 27 – Regenerate Layout.

17. This way the “Diagram Window Tab” will look as the one in Figure 28

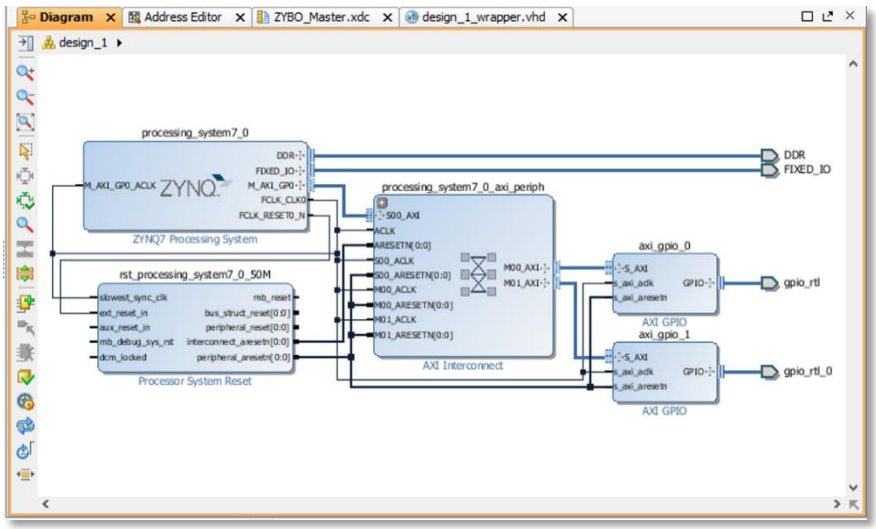


Figure 28 – Block Diagram Layout Regenerated.

18. Double click on the “AXI_GPIO_0” block that is selected in Figure 29

**Note: Now the AXI Blocks will be edited to make the logic interface to connect the Leds and Buttons of the ZYBO board to the Zynq processor*

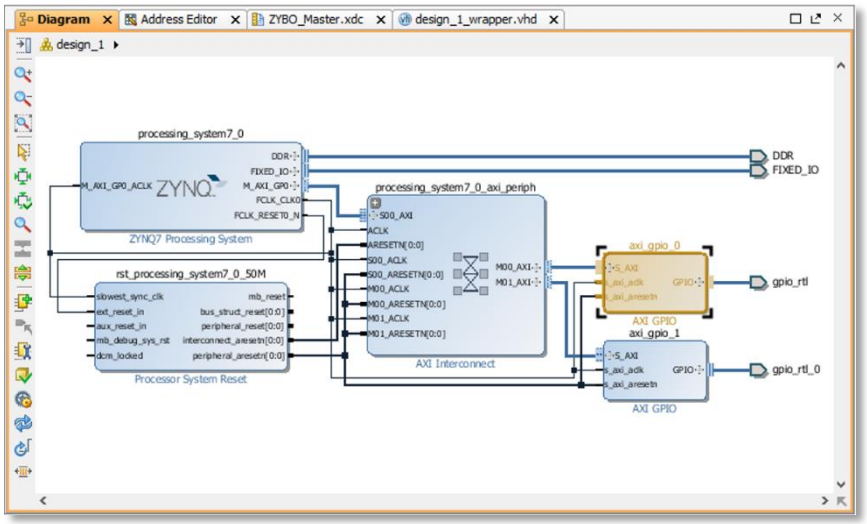


Figure 29 – Open AXI_GPIO Block.

19. Then a window similar to the one in Figure 30 is shown. Make the editions (Figure 31), verify that the “All Outputs” option is selected, the “GPIO Width” is 4, and that the “Default Output Value” is 0x0000000F, and click “OK”

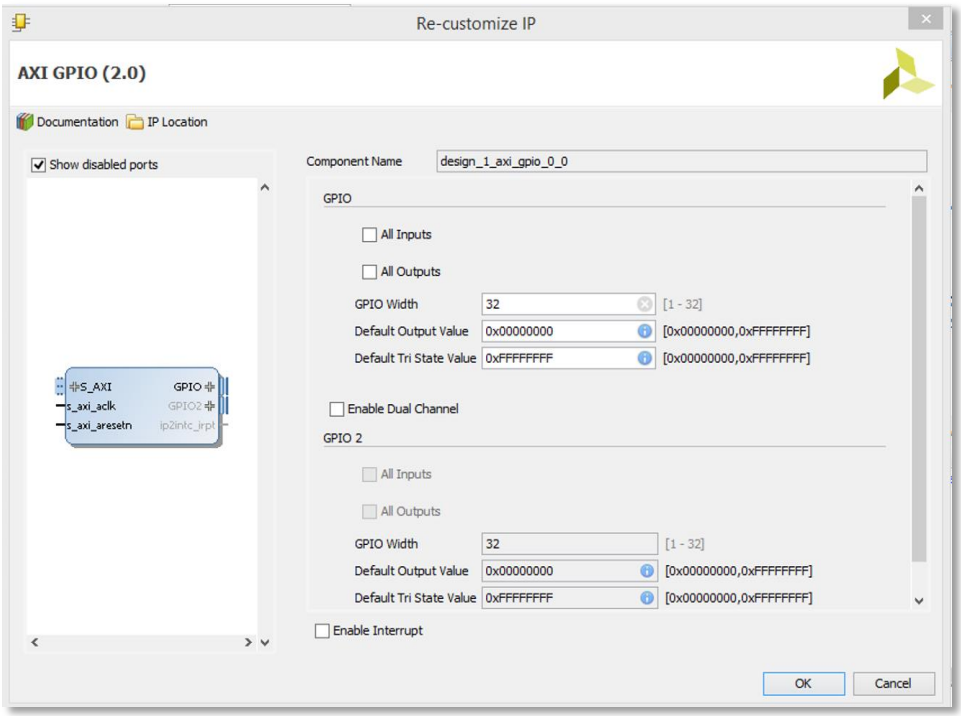


Figure 30 – Re-customize IP.

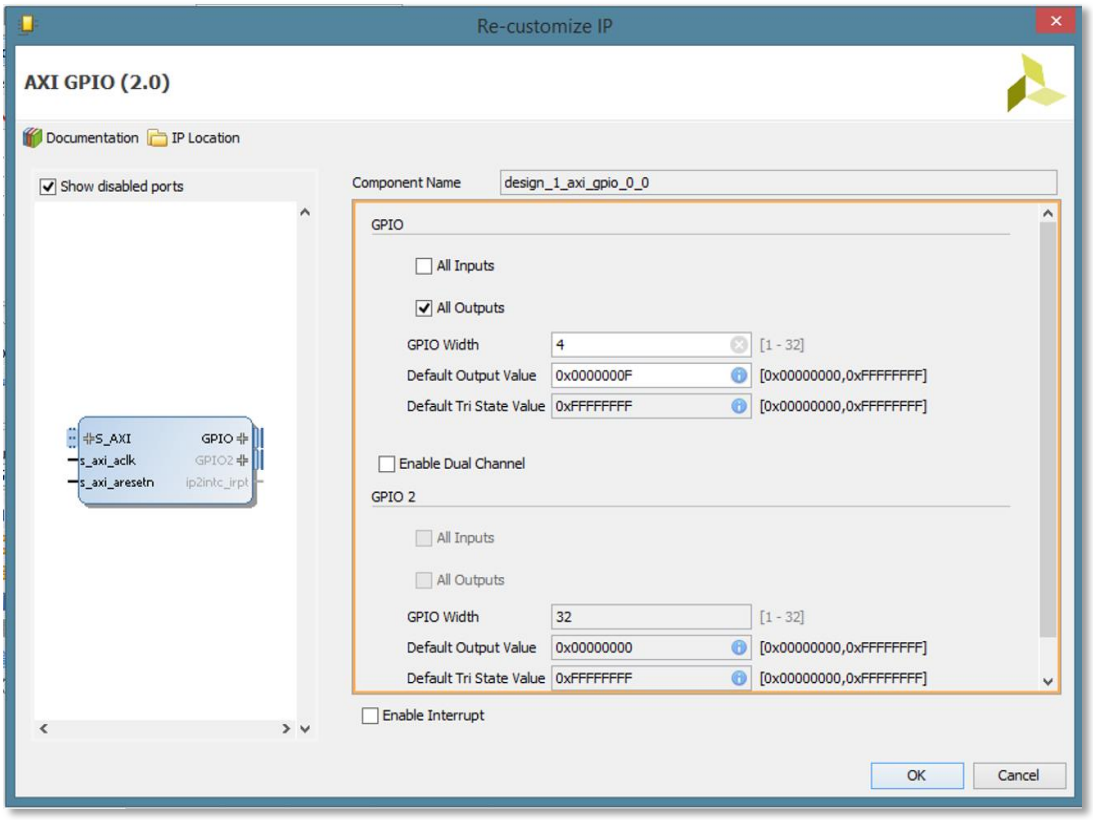


Figure 31 – Re-customize IP Settings.

20. Now edit the second AXI block. Double click on “AXI_GPIO_1” and make the configure it according to the values presented in Figure 32. Verify that the “All Inputs” is checked and “GPIO Width” box is ‘2’, and click “OK”

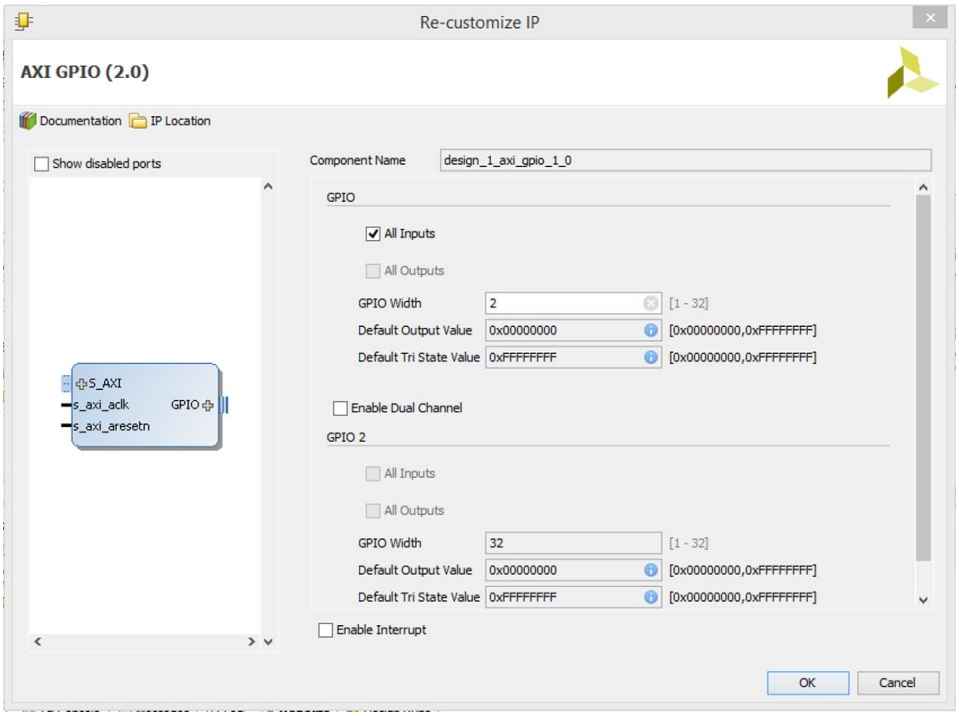


Figure 32 – Re-customizing IP Settings.

21. Next, right click within the “Diagram Window Tab” and click on “Validate Design” (Figure 33). In the end, a new window (Figure 34) appears. Click “OK”

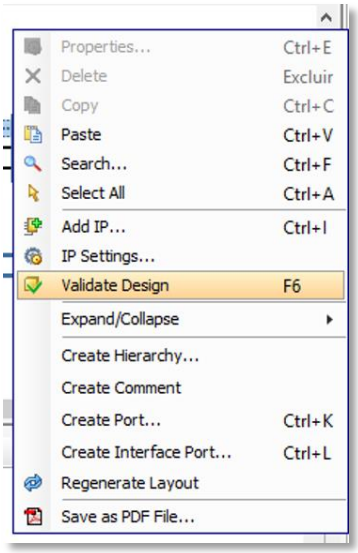


Figure 33 – Validate Design.

Figure 34 – Validation Successful.

22. Now in “Flow Navigator” → “Project Manager” click on “Project Settings” (Figure 35)

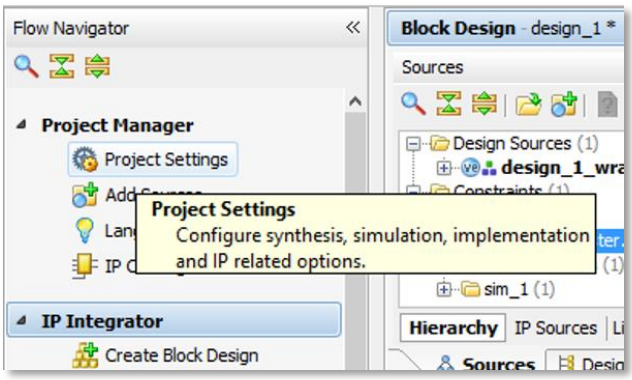


Figure 35 – Project Settings.

23. So on the option “Target Language” select the “VHDL” (Figure 36) and click “OK”

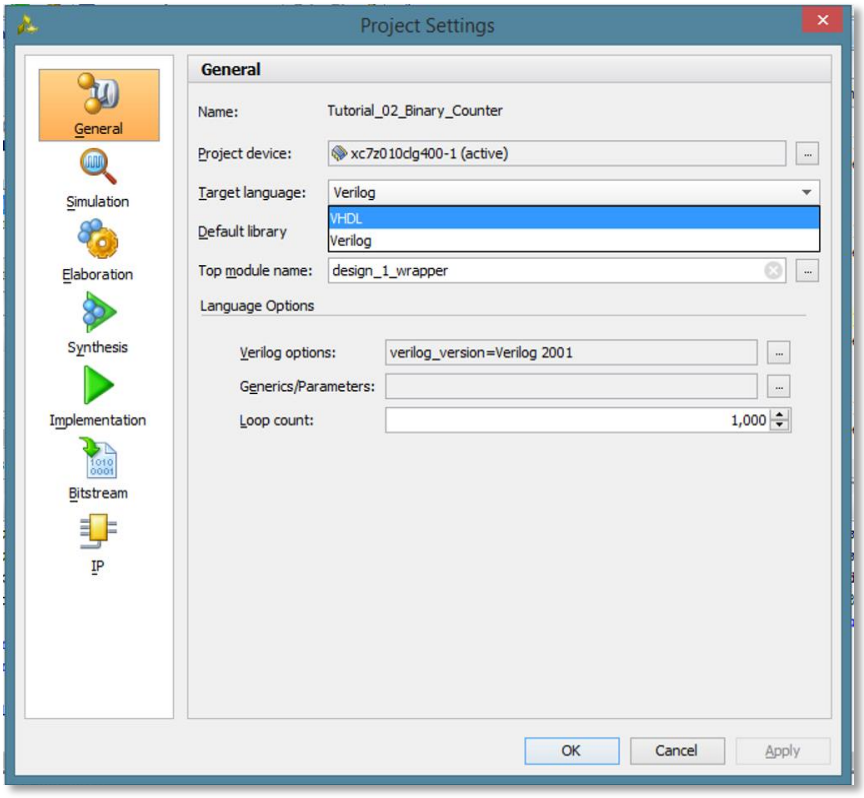


Figure 36 – Changing the Target Language.

24. Then, if step 18 - the Validate Design step - was successfully, select the “Sources Tab” → “Hierarchy” (Figure 37)

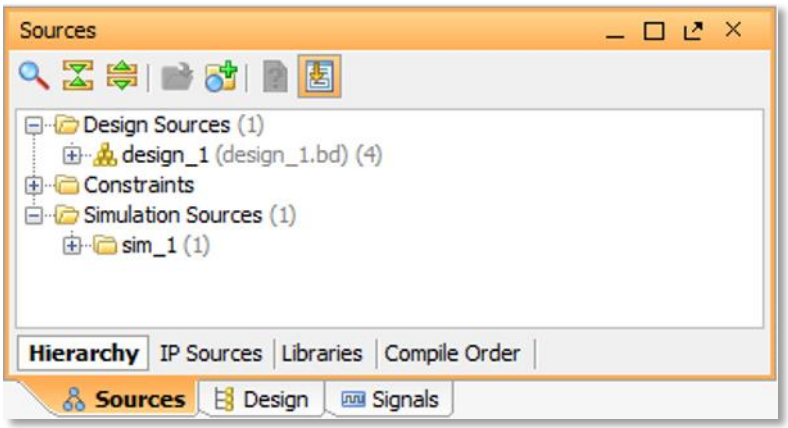


Figure 37 – Source Tab.

25. Within it choose “Design Sources” and right click on the “Block Design File”. In this case, it is “design_1.bd”. From the dropdown menu, click on the option “Create HDL Wrapper” (Figure 38)

**Note: The step Create HDL Wrapper is necessary to define the Design created as the Main device. This step does not create a block as the IP's Integrator, but does a Wrapper (main interface) with all blocks.*

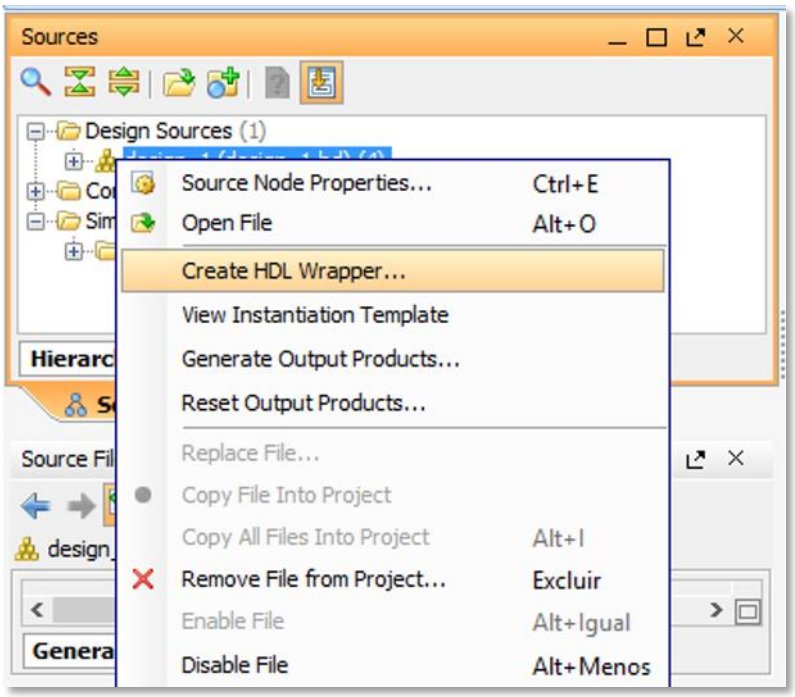


Figure 38 – Create HDL Wrapper.

26. Verify that the checkbox “Let Vivado manage wrapper and auto-update” is selected and click “OK” (Figure 39)

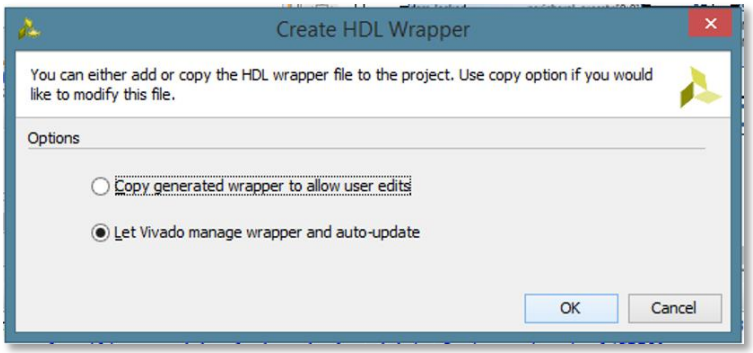


Figure 39 – Creating HDL Wrapper.

27. This step shows how to include the XDC file for the ZYBO development board, the ZYBO_Master.xdc. This is a constraint file that facilitates pin assignments, as all pin connections between the Zynq processor and the ZYBO board are there defined. This file can be downloaded from the Digilent website:
https://www.digilentinc.com/Data/Products/ZYBO/ZYBO_Master_xdc.zip
Once downloaded in "Flow Navigator" → "Project Manager" click on the "Add Sources" (Figure 40)

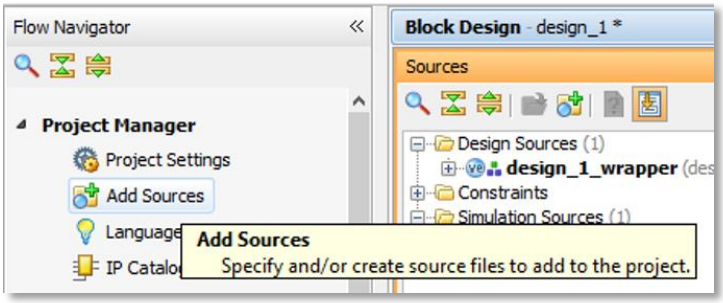


Figure 40 – Add Sources.

28. A window will appear (Figure 41). Select the option "Add or create constraints" and click "Next"

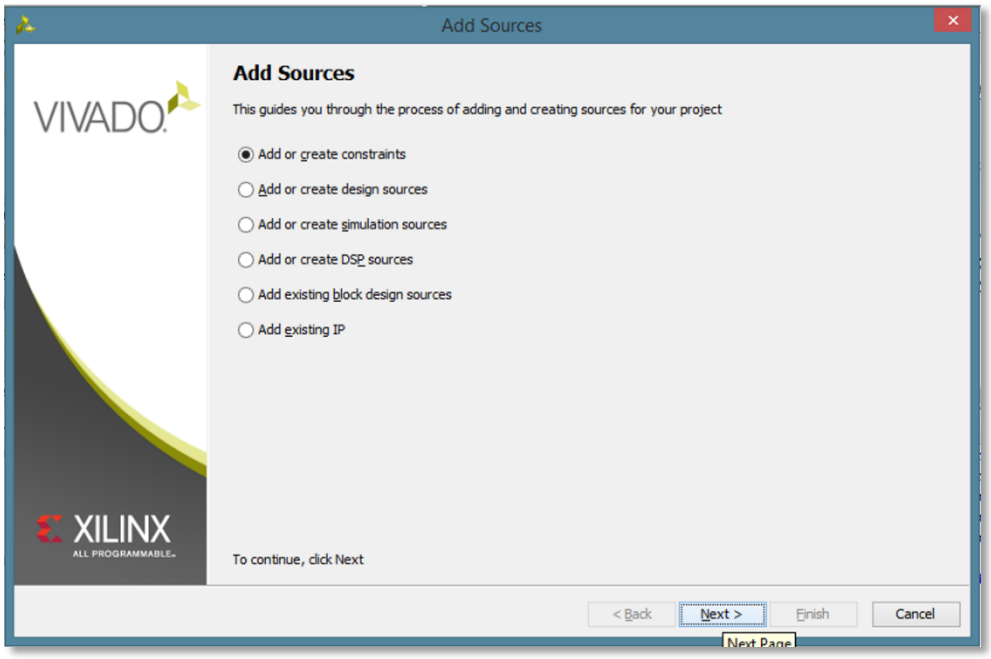


Figure 41 – Add or Create Constraints.

29. Within the next window, click on the “+” signal and click “Add Files” (Figure 42)

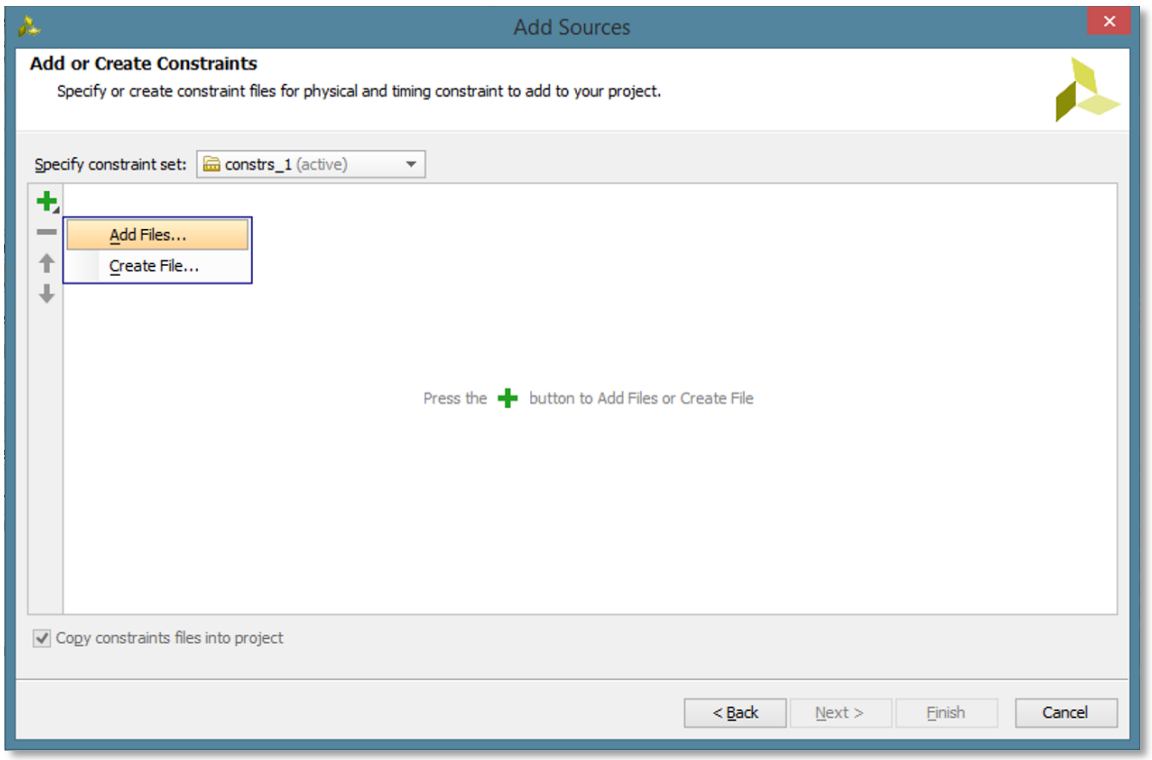


Figure 42 – Add Files Constraints.

30. Now navigate to the directory that you downloaded the ZYBO XDC file select the file and click “OK” (Figure 43).

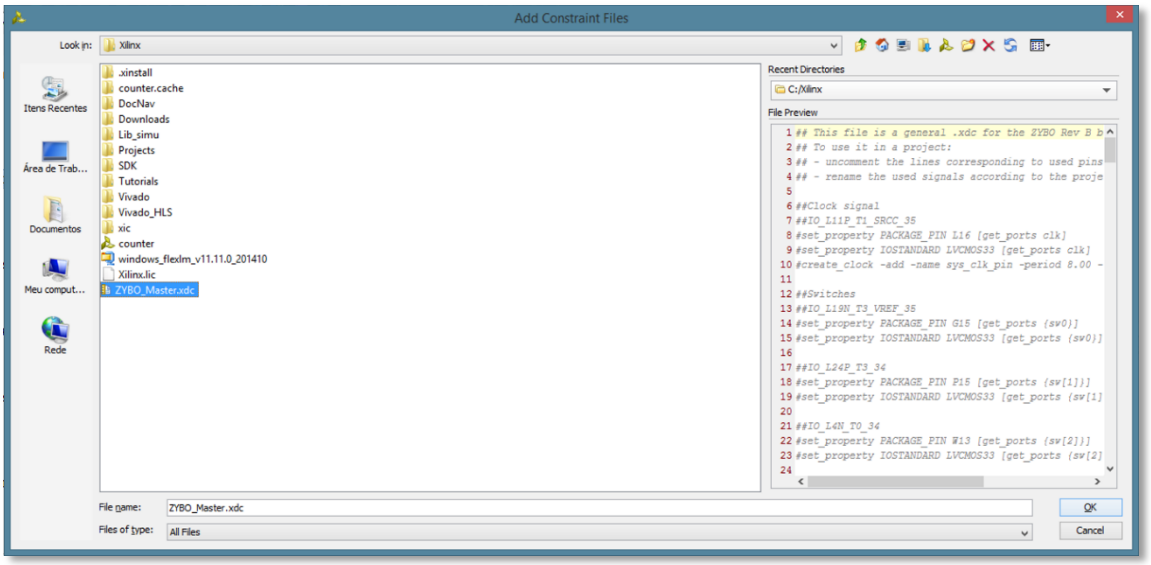


Figure 43 – ZYBO_Master.xdc File.

31. So check the checkbox “Copy constraints files into project” and click “Finish” (Figure 44)

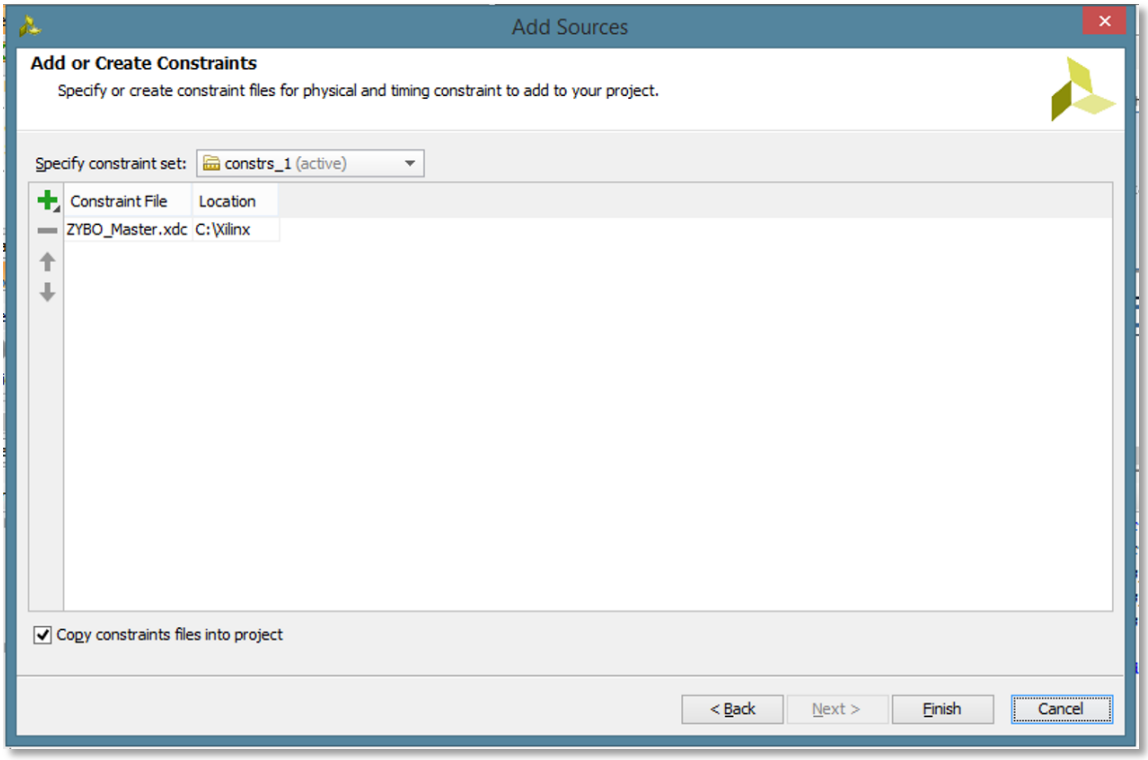


Figure 44 – Constraint File Added.

32. Within the “Sources Tab” → “Constraints” expand the folder and open this file with a double click on “ZYBO_Master.xdc” (Figure 45)

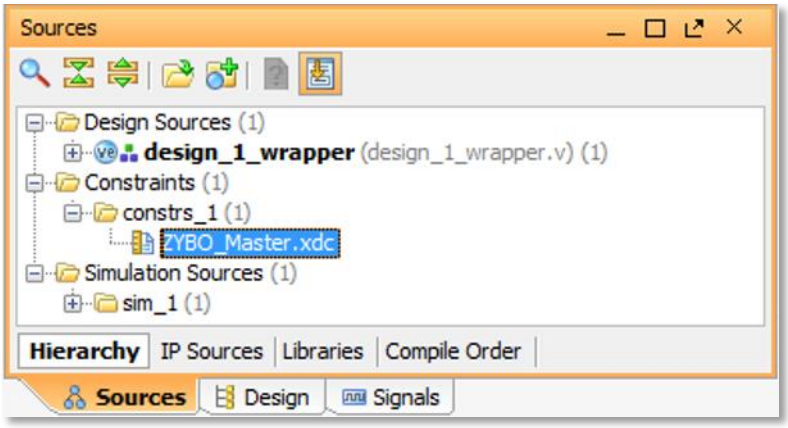


Figure 45 – Opening the Constraint ZYBO_Master.xdc.

33. When the window opens, notice that all lines have been commented out. Leave them commented except the ones that are shown uncommented in Figure 46, Figure 47 and Figure 48. Then click “Save” (Figure 49)

**Note: Now, the Lines that are being uncommented, match to the lines that assign the ports created to the pins on the Zynq that the peripherals are connected.*

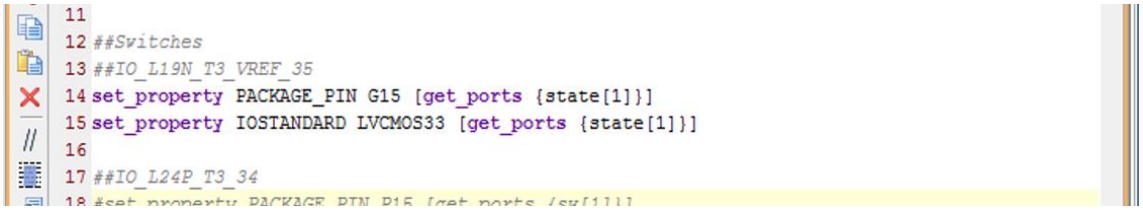


Figure 46 – Constraints Editions.

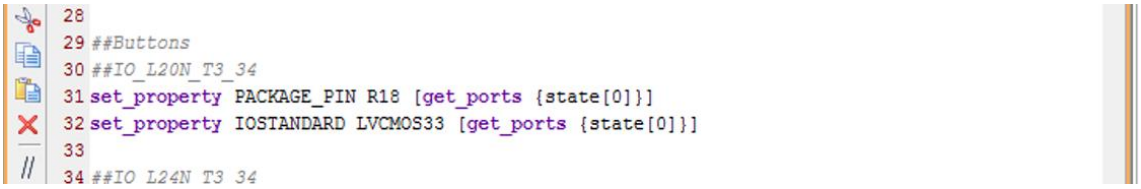


Figure 47 – Constraints Editions.

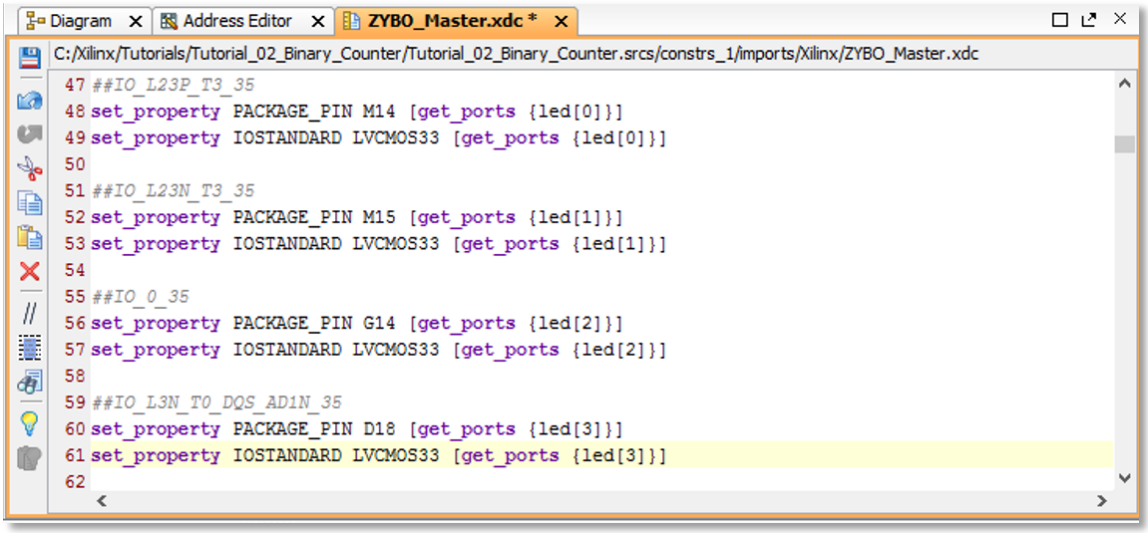


Figure 48 – Constraints Editions.

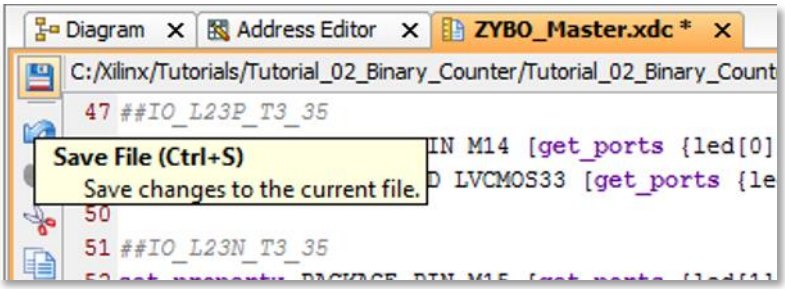


Figure 49 – Saving File.

34. Now open the VHDL wrapper file in the “Sources Tab” → “Design Sources” and double click on “design_1_wrapper.vhd” (Figure 50)

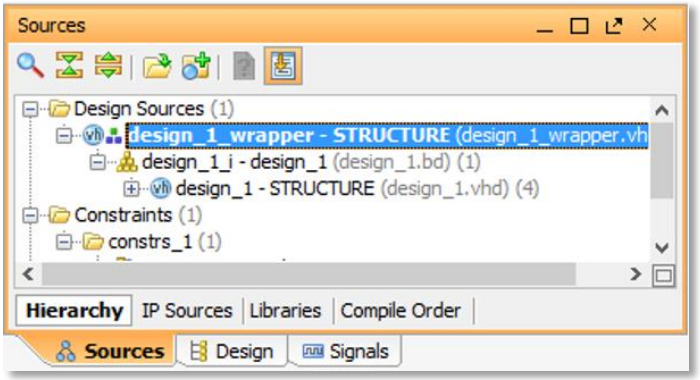


Figure 50 – Opening the Design Wrapper.

35. Then the window in Figure 51 appears

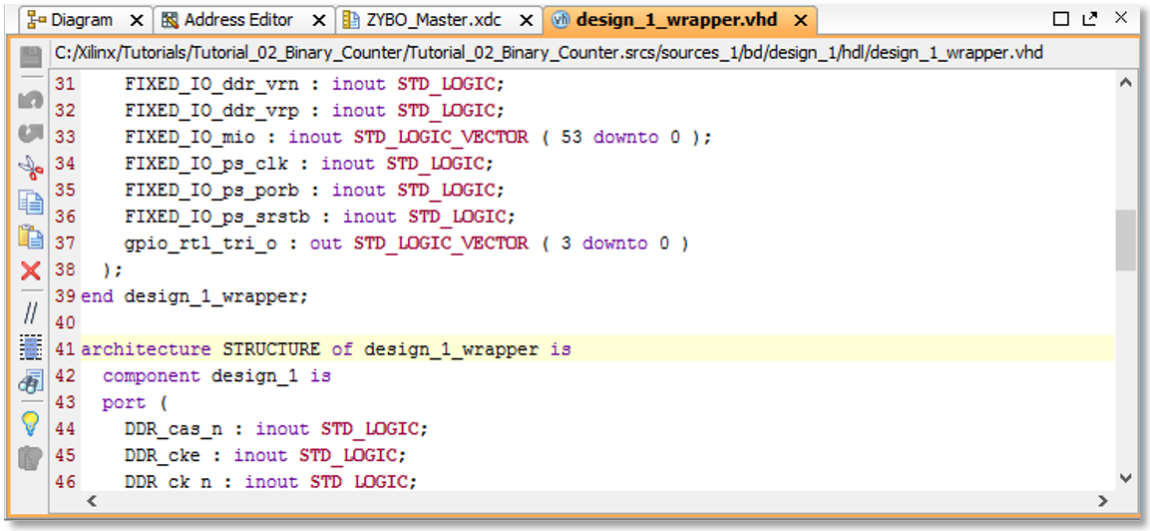


Figure 51 – Design Wrapper.

36. Make the following changes to the lines shown below and in Figure 52 and Figure 53

- LINE 37 (within design_1_wrapper entity): **CHANGE** gpio_rtl_0_tri_i : in STD_LOGIC_VECTOR (1 downto 0) **TO** state : in STD_LOGIC_VECTOR (1 downto 0)

- LINE 38 (within design_1_wrapper entity): **CHANGE** gpio_rtl_tri_o : out STD_LOGIC_VECTOR (3 downto 0) **TO** led : out STD_LOGIC_VECTOR (3 downto 0)

- LINE 94 (within component instantiation): **CHANGE** gpio_rtl_0_tri_i(1 downto 0) => gpio_rtl_0_tri_i(1 downto 0) **TO** gpio_rtl_0_tri_i(1 downto 0) => state(1 downto 0)

- LINE 95 (within component instantiation): **CHANGE** gpio_rtl_tri_o(3 downto 0) => gpio_rtl_tri_o(3 downto 0) **TO** gpio_rtl_tri_o(3 downto 0) => led(3 downto 0)

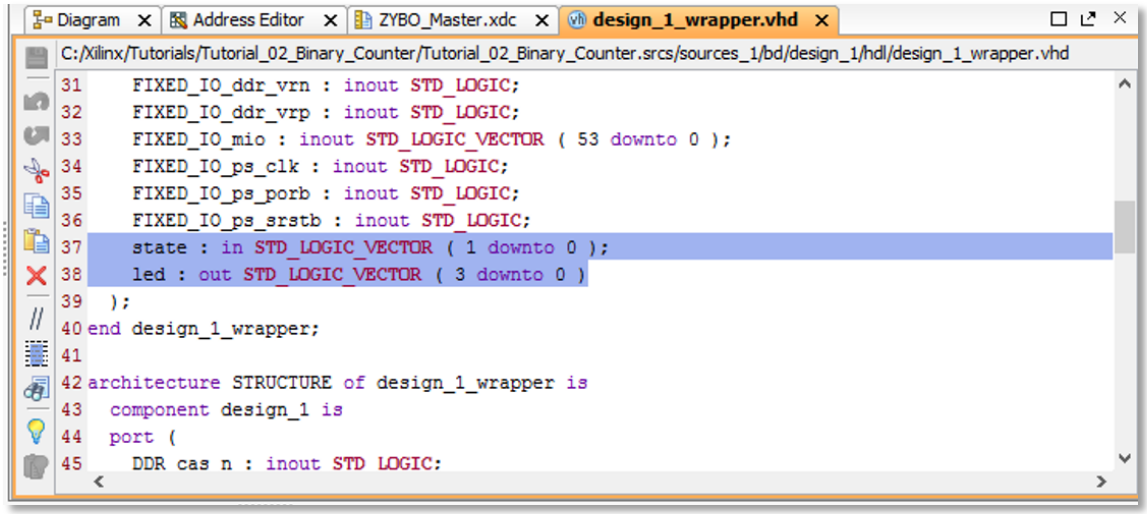


Figure 52 – Design Wrapper Edition 1.

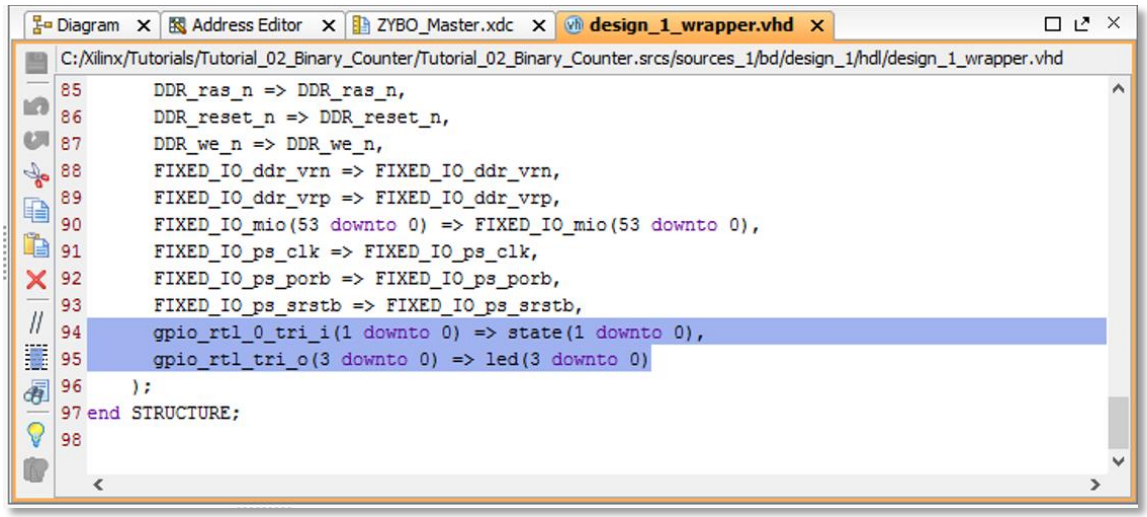


Figure 53 – Design Wrapper Edition 2.

37. Then save it by clicking on “Save File” (Figure 54)

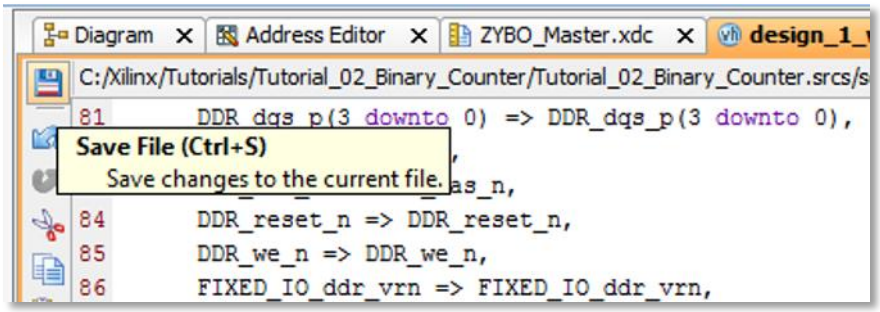


Figure 54 – Saving File.

38. On “Flow Navigator” → “Implementation” click “Run Implementation” (Figure 55)

**Note: The Vivado Design Suite implementation process transforms a logical netlist and constraints into a placed and routed design, ready for bitstream generation.* [1]

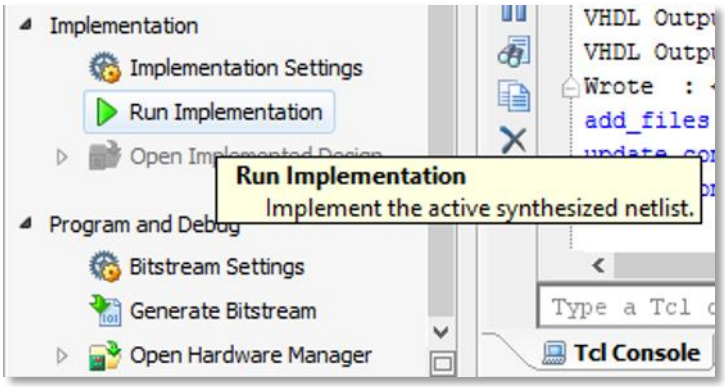


Figure 55 – Run Implementation.

39. Then a “Missing Synthesis Results” window appears (Figure 56). Click on “OK”

**Note: The Synthesis is the process of transforming an RTL-specified design into a gate-level representation. Vivado synthesis is timing-driven and optimized for memory usage and performance.* [2]

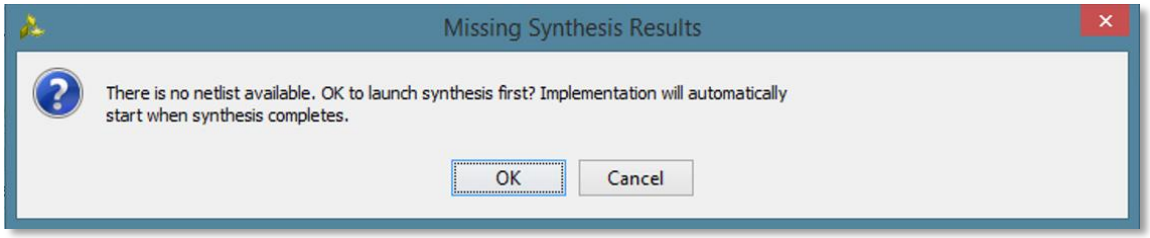


Figure 56 – Missing Synthesis Results.

40. If the design has not been saved, it will prompt to save again. Click “Save” (Figure 57)

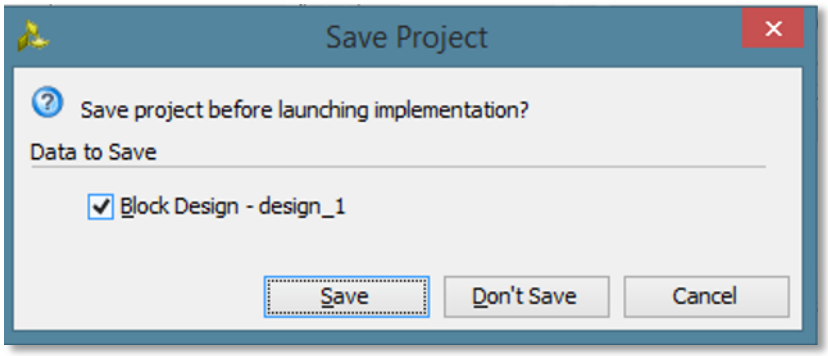


Figure 57 – Save Project.

41. Then after a while, two windows can appear. First, a window with a lot of warnings (Figure 58). This one appears because the VHDL wrapper file that was edited, have been reverted to its old version. Return and repeat all the steps from the step 31 and continue. If this window does not appear, just continue the tutorial

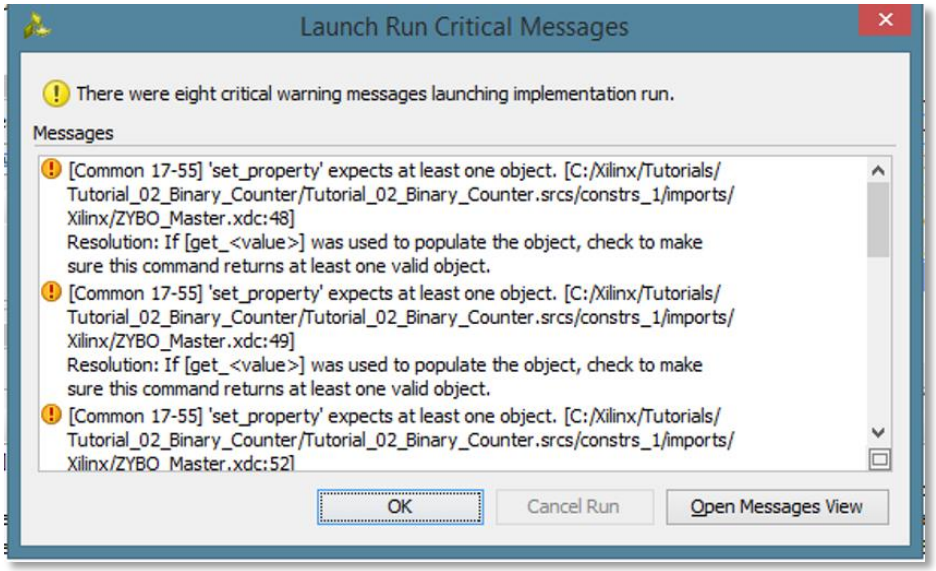


Figure 58 – Launch Run Critical Messages.

42. Therefore, the “Implementation Completed” window will appear (Figure 59). Select the option “Generate Bitstream”, and click “OK”

Note: The step Generate Bitstream will convert the entire project created in a bitstream to program the ZYBO Board.

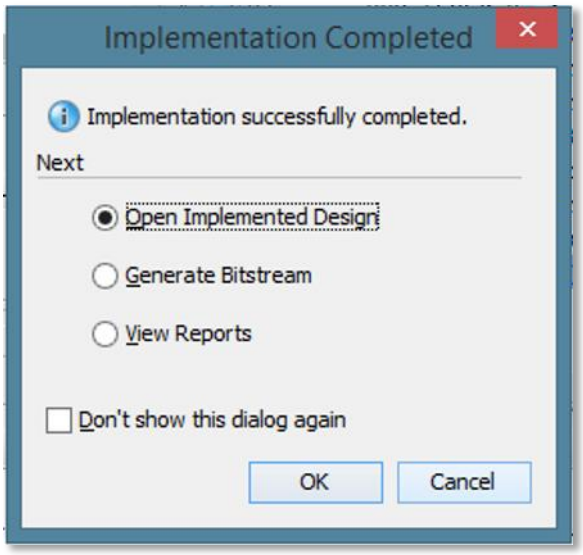


Figure 59 – Implementation Completed.

43. After the bitstream has been generated, a “Bitstream Generation Completed” window will appear. Choose the “Open Implemented Design” option and click “OK” (Figure 60)

Note: With the Implemented Design opened it is possible to have access to all the report analyzes that are shown in the VIVADO 2014.X/2015.X QUICK START TUTORIAL TO ZYBO BOARD. [3]

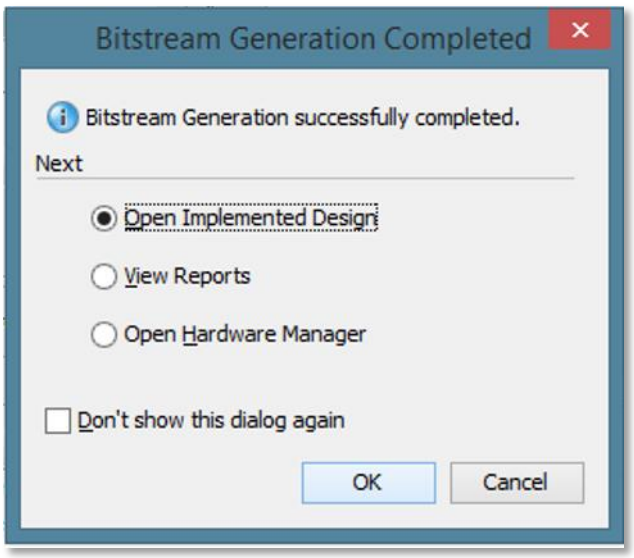


Figure 60 – Bitstream Generation Completed.

44. To finish the project it is necessary to export the hardware part to SDK to create the software part and to integrate them. In the main menu “File” → “Export” click on “Export Hardware” (Figure 61)

**Note: The Export Hardware generate a structure to be read by the SDK read, the application used to create the software to program the Zynq Processor.*

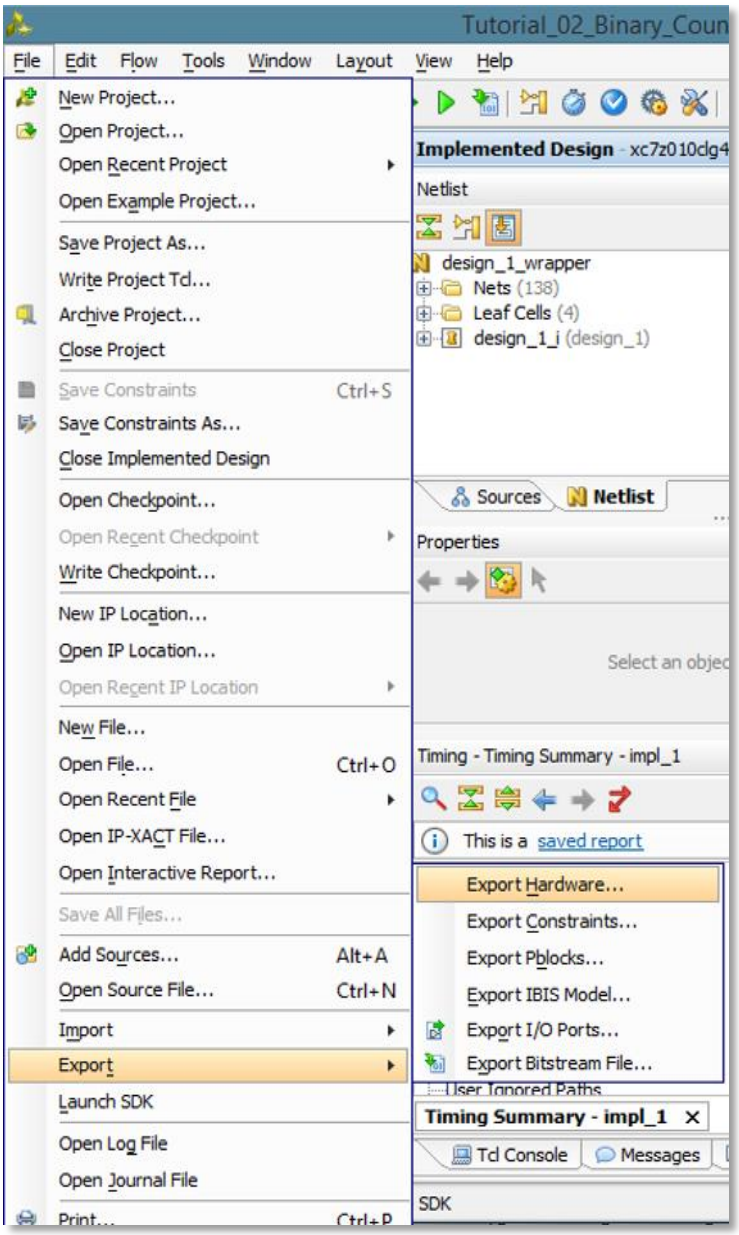


Figure 61 – Export Hardware.

45. Then an “Export Hardware” window appears. Check the “Include Bitstream” checkbox option and click “OK” (Figure 62)

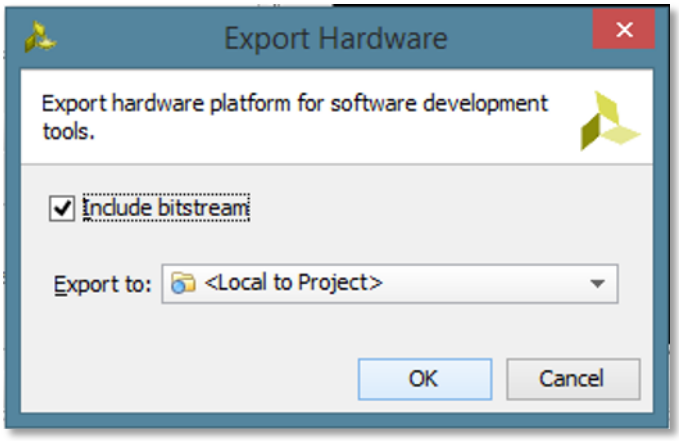


Figure 62 – Export Hardware Settings.

Programming the processor with SDK:

1. To launch SDK from Vivado go to the main menu “File” and click on “Launch SDK” (Figure 63). A “Launch SDK” window will appear (Figure 64). Just click “OK”. Sometimes, when the SDK is launch, the Windows Firewall can try to block this action. If this happens, just click “Allow Access”

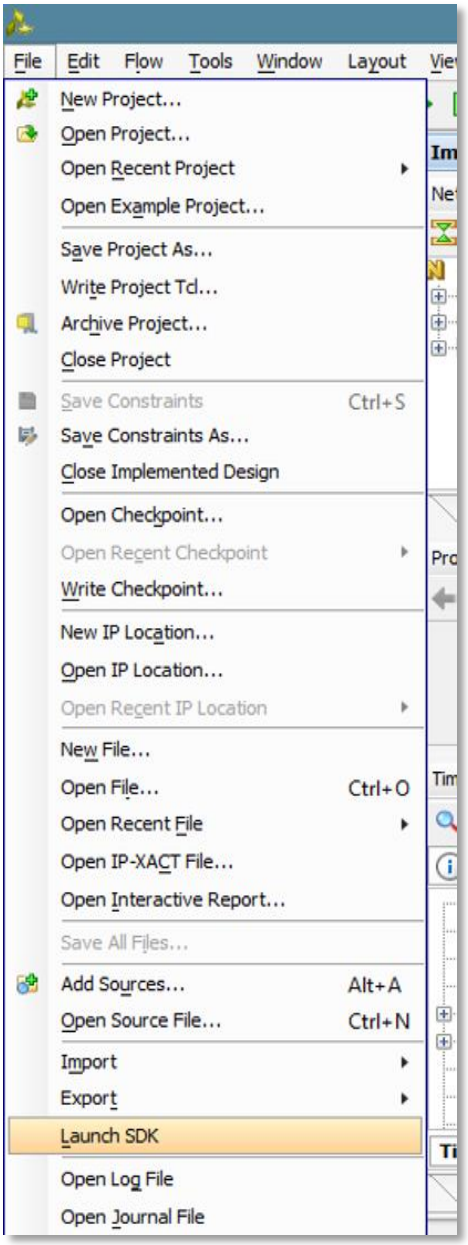


Figure 63 – Launch SDK.

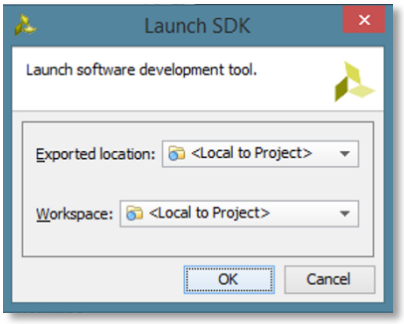


Figure 64 – Launch SDK Settings.

2. Figure 65 shows the SDK window

**Note: Now all the Project Structure created is opened and able to create the processor program.*

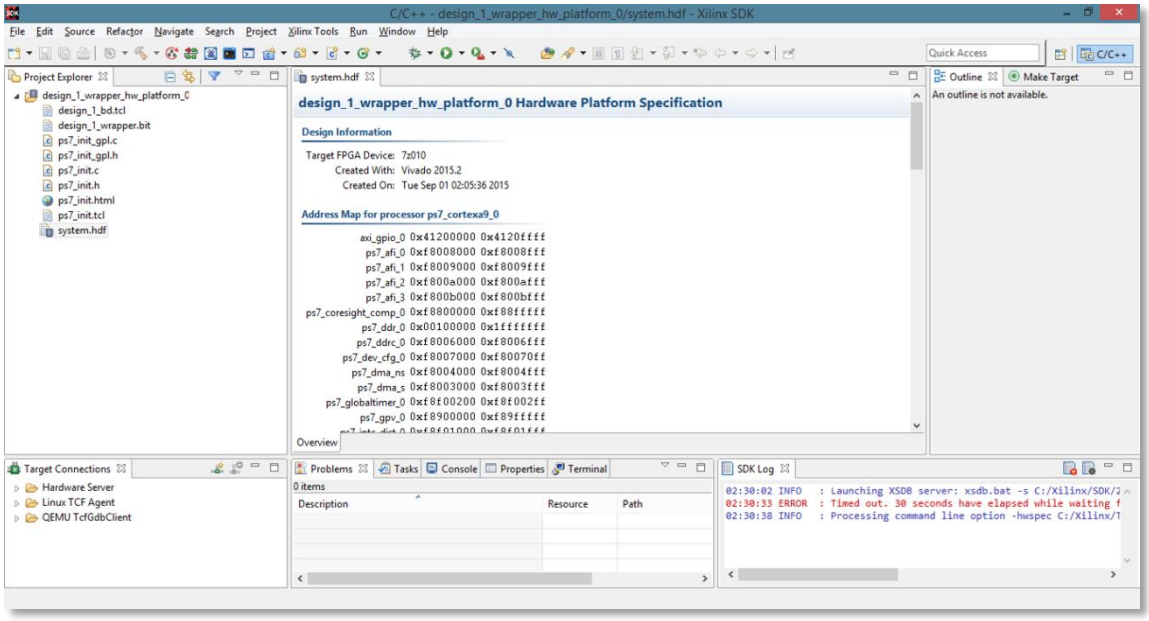


Figure 65 – Project Opened in SDK.

3. Go to main menu “File” → “New” and click on “Application Project” (Figure 66)

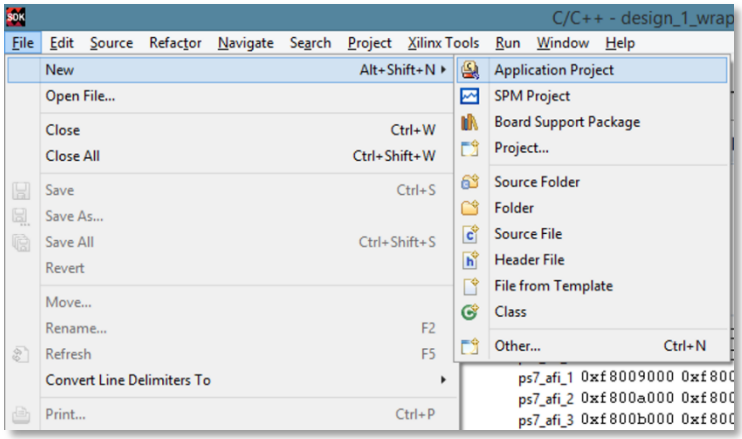


Figure 66 – New Application Project.

4. A new window opens (Figure 67). Choose a “Project name” and click on “Next”. In the next window, select the template “Empty Application” and click “Finish” (Figure 68)0

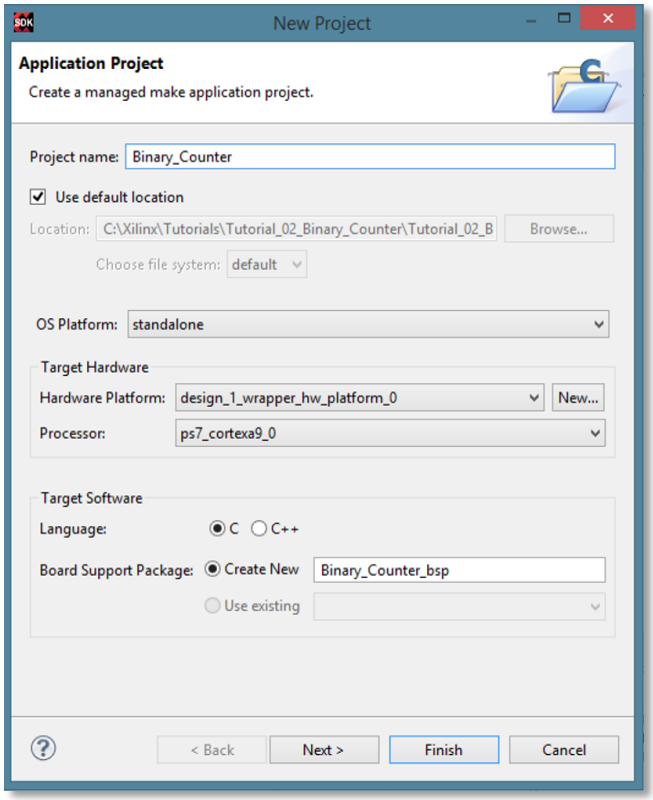


Figure 67 – Application Project Settings.

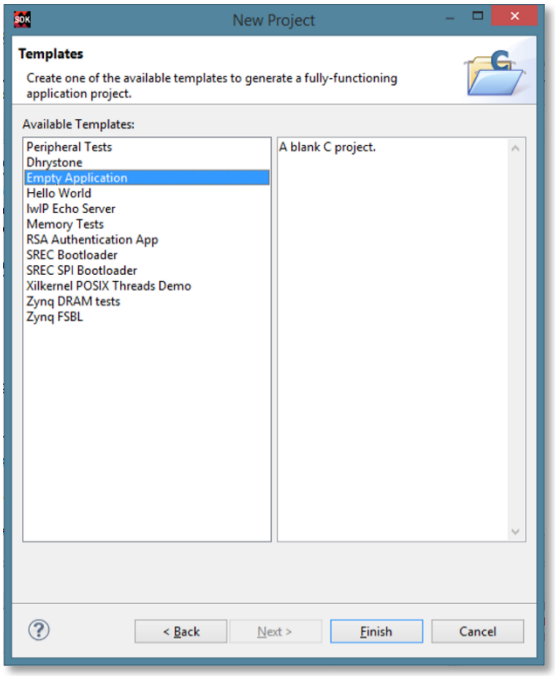


Figure 68 – Templates.

5. After this point pay attention to the bar in the right down side of the XDC main window because there will show the processing progress (Figure 69)

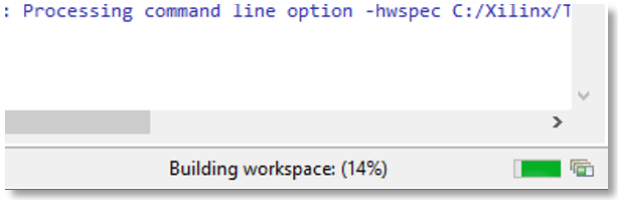


Figure 69 – Waiting the Processing.

6. After click Finish, there should be a C project folder within the “Project Explorer” - in this case the “Binary_Counter” folder. Click the arrow next to the new C project folder and you will see two more folders: “Includes” and “src” (Figure 70)

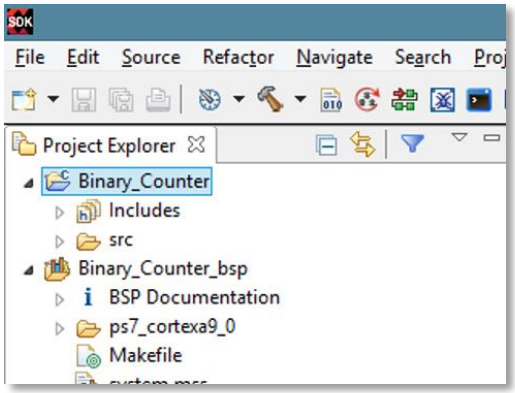


Figure 70 – Project Folder.

7. In “Project Explorer Tab” →”Binary_Counter” right click on “scr” folder and from the dropdown menu choose“New” and click “File” (Figure 71)

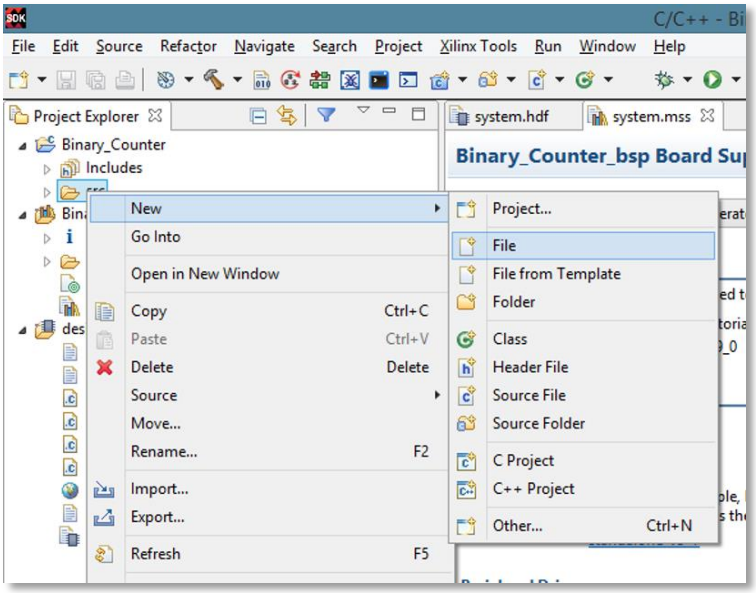


Figure 71 – New File.

8. In the new window keep the parent folder selected, and enter the name to the file followed by “.c” - in this case “Counter.c” (Figure 72). Then click “Finish”

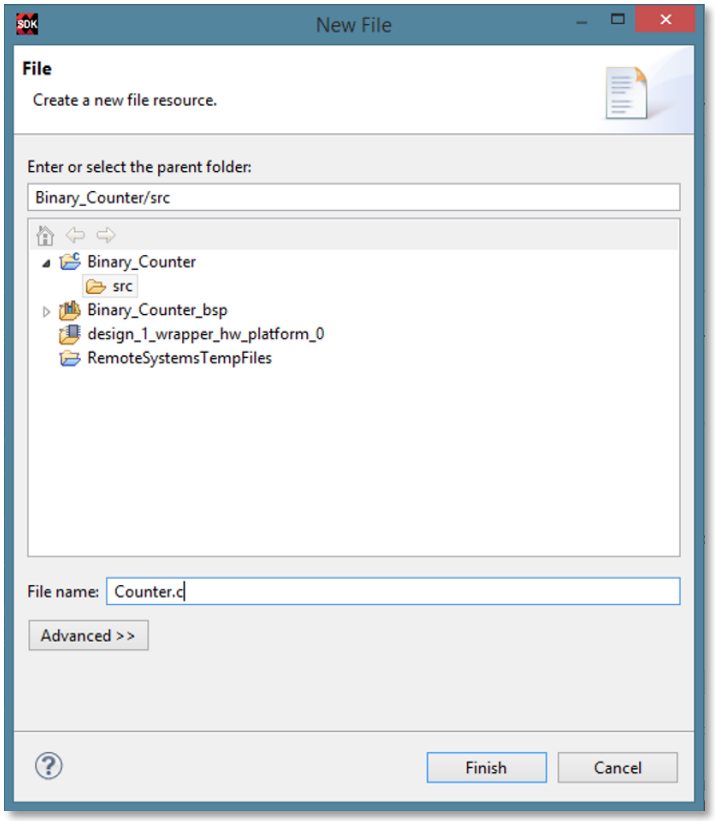


Figure 72 – Naming the New File.

9. The File window tab appears making it possible to implement the developed code. Use the code example (Figure 73), (Figure 74) and (Figure 75) and write the program

**Note: The code created makes the AXI communication initialization and uses this bus to do the necessary actions.*

```
// Borrowed from ZynqBook Tutorials
// Include Files
#include "xparameters.h"
#include "xgpio.h"
#include "xstatus.h"
#include "xil_printf.h"
// Definitions
#define LED_DEVICE_ID       XPAR_AXI_GPIO_0_DEVICE_ID// GPIO device that LEDs are connected to
#define STATE_DEVICE_ID     XPAR_AXI_GPIO_1_DEVICE_ID// GPIO device that STATEs are connected to
#define LED                 0x00                     // Initial LED value - 0000
#define LED_DELAY           100000000                // Software delay length
#define LED_CHANNEL         1                        // GPIO port for LEDs
#define STATE_CHANNEL       1                        // GPIO port for STATEs
#define printf              xil_printf               // smaller, optimized printf

XGpio LEDInst, STATEInst;                            // GPIO Device driver instance

int CounterLeds(void){
    volatile int Delay;
    u32 STATE;
    int led = LED; // Hold current LED value. Initialize to LED definition
            while (1) {
                //Receives the status of the buttons and the switch
                STATE = XGpio_DiscreteRead(&STATEInst, STATE_CHANNEL);
                //compares if the state is set to counter Up and if the Reset is press
                if((STATE == 0x02)||(STATE == 0x03)){
                    // Reset the count when the count finish or the Reset is press
```

Figure 73 – C Code Created.

```
    u32 STATE;
    int led = LED; // Hold current LED value. Initialize to LED definition
            while (1) {
                //Receives the status of the buttons and the switch
                STATE = XGpio_DiscreteRead(&STATEInst, STATE_CHANNEL);
                //compares if the state is set to counter Up and if the Reset is press
                if((STATE == 0x02)||(STATE == 0x03)){
                    // Reset the count when the count finish or the Reset is press
                    if((led == 0x0F)||(STATE == 0x03)){
                        led = 0x00;
                    }else{
                    // Increment the Counter.
                    led = led+1;
                    }
                }else{
                    //compares if the state is set to counter Down and if the Reset is press
                    if((STATE == 0x00)||(STATE == 0x01)){
                        // Reset the count when the count cycle  finish or the Reset is press
                        if((led == 0x00)||(STATE == 0x01)){
                            led = 0x0F;
                        }else{
                        // Increment the Counter
                        led = led-1;
                        }
                    }
                }
                // Write output to the LEDs.
```

Figure 74 – C Code Created.

```
                }
                // Write output to the LEDs.
                XGpio_DiscreteWrite(&LEDInst, LED_CHANNEL, led);
                // Wait a small amount of time so that the LED blinking is visible
                for (Delay = 0; Delay < LED_DELAY; Delay++);
            }
        return XST_SUCCESS; // Ideally unreachable
}
// Main function.
int main(){
    // Initialize the Led GPIO
    int status;
    status = XGpio_Initialize(&LEDInst, LED_DEVICE_ID);
    if (status != XST_SUCCESS)
        return XST_FAILURE;
    // Initialize the State GPIO
    status = XGpio_Initialize(&STATEInst, STATE_DEVICE_ID);
    if (status != XST_SUCCESS)
        return XST_FAILURE;
    // Set LEDs direction to outputs
    XGpio_SetDataDirection(&LEDInst, 1, 0x00);
    // Set all buttons direction to inputs
    XGpio_SetDataDirection(&STATEInst, 1, 0xFF);
    // Call the function Counter
    CounterLeds();
    return 0;
}
```

Figure 75 – C Code Created.

10. When the code is ready click “Save” (Figure 76). After the code have been saved it automatically compiles

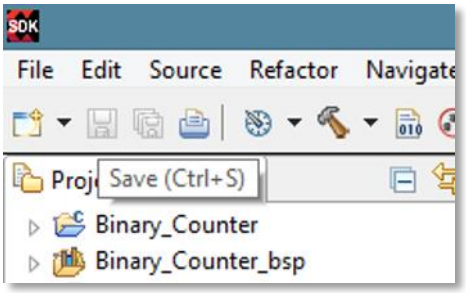


Figure 76 – Saving.

11. Now plug the ZYBO board into the computer, using the USB port (Figure 77)

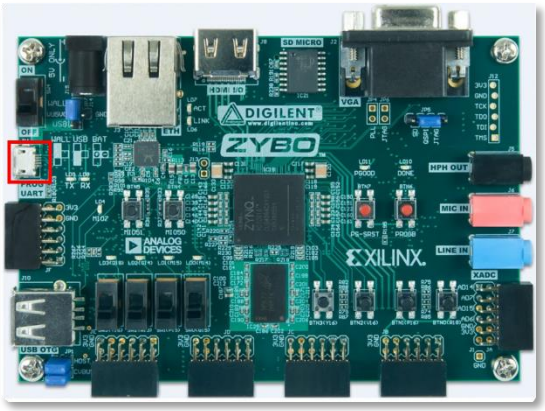


Figure 77 – USB Programmer Port of ZYBO Board.

12. To program the board using the SDK, go to the main menu “Xiling Tools” and click on “Program FPGA” (Figure 78)

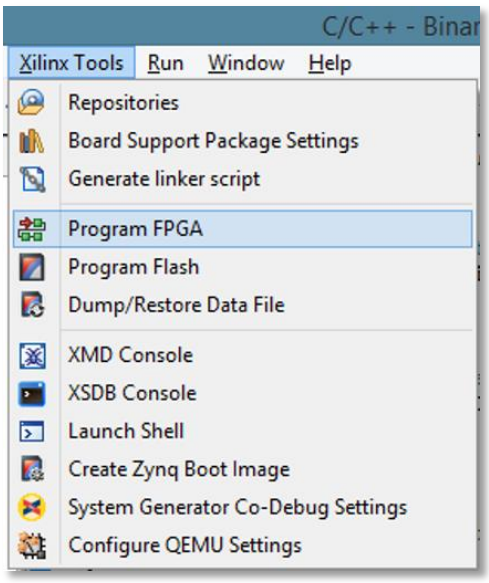


Figure 78 – Program FPGA.

13. A new window appears (Figure 79). Verify that the option “Device” is set to “Auto Detect”, that “Connection” is set to “Local”, and that the “.bit” file (that was generated in Vivado) is in the “Bitstream” box. Click “Program”. After completing, the four LEDs above the switches should be illuminated

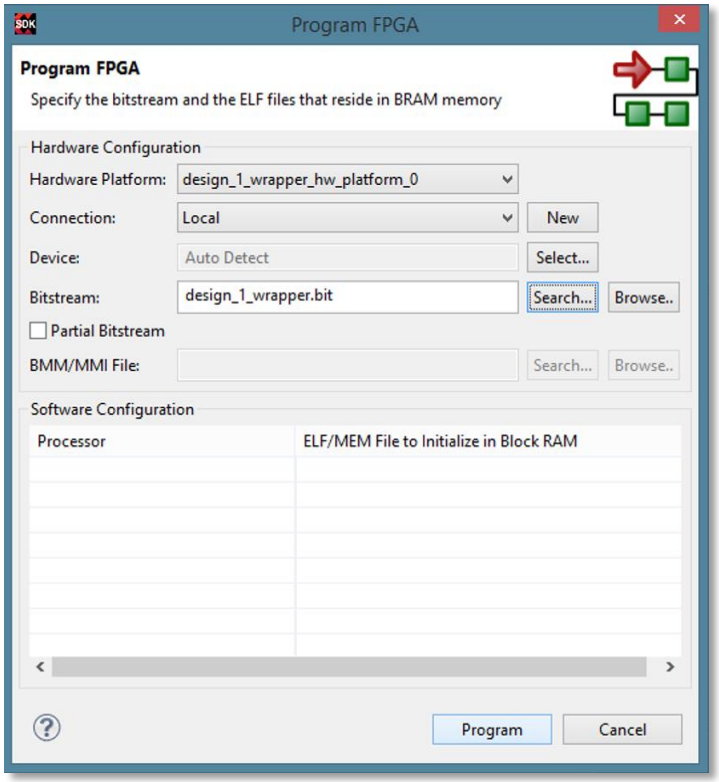


Figure 79 – Program FPGA Settings.

14. Next, in SDK main menu “Run” click on “Run Configurations…” (Figure 80)

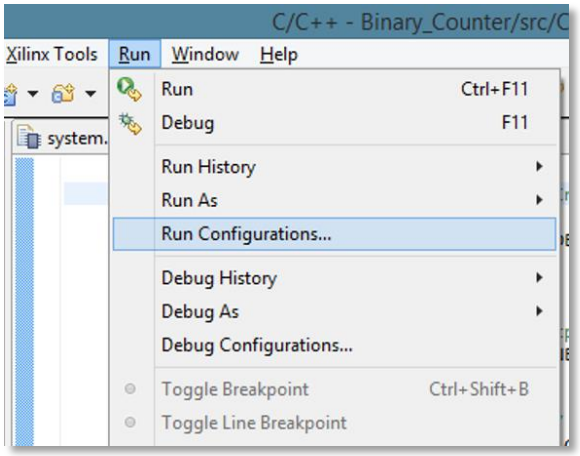


Figure 80 – Configurations of Run.

15. A new window appears (Figure 81). First, in the left drop down menu click on “Xilinx C/C++ application (GDB)” and double click to open the configuration window shown in Figure 81

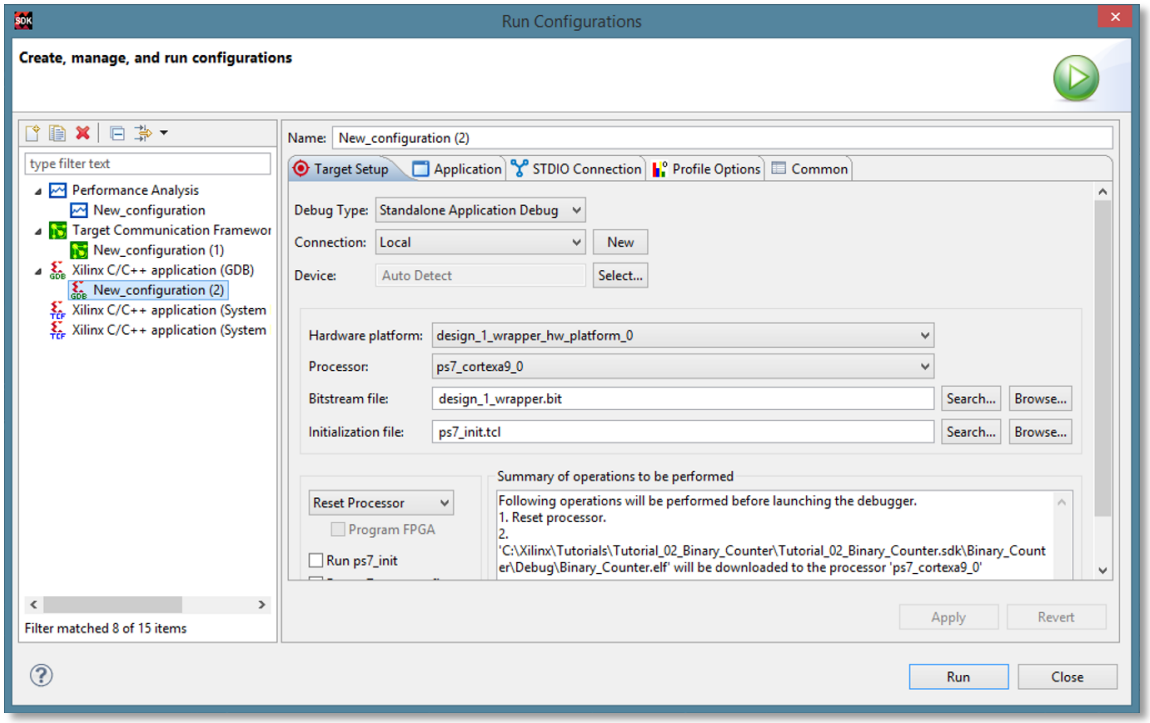


Figure 81 – Configurations of Run Settings.

16. In the “Target Setup” tab look for the “Bitstream file:” box and click “Search…” (Figure 82) and select the file (Figure 83)

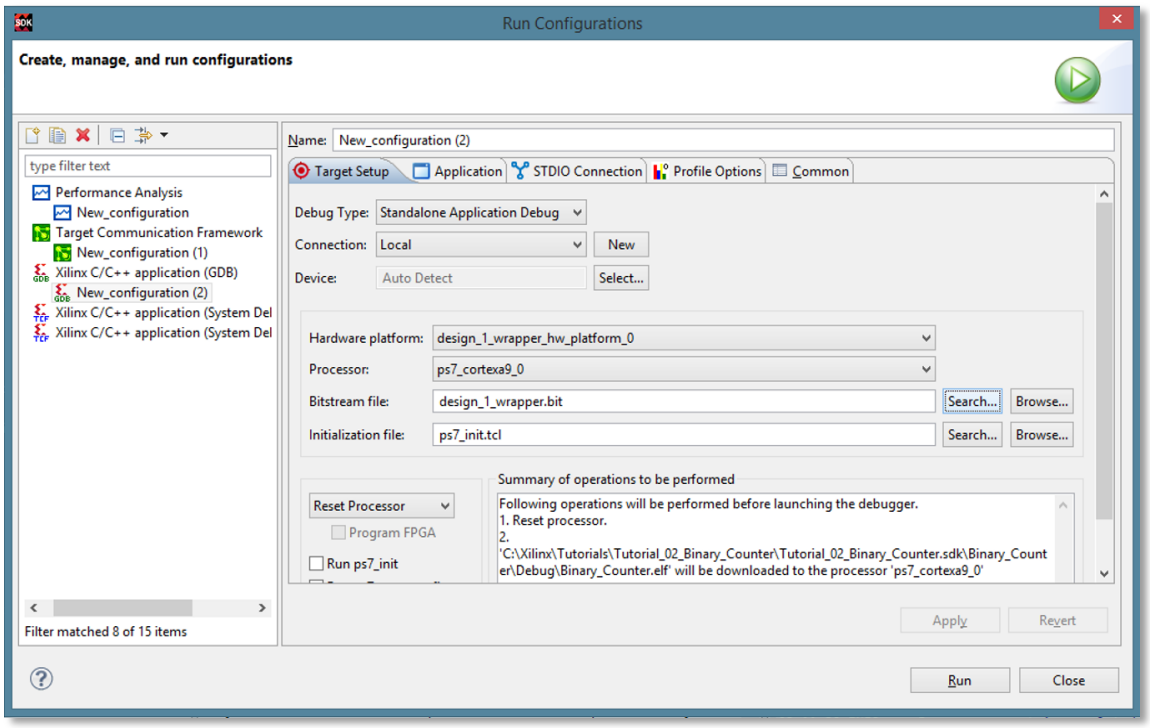


Figure 82 – Configurations of Run Settings.

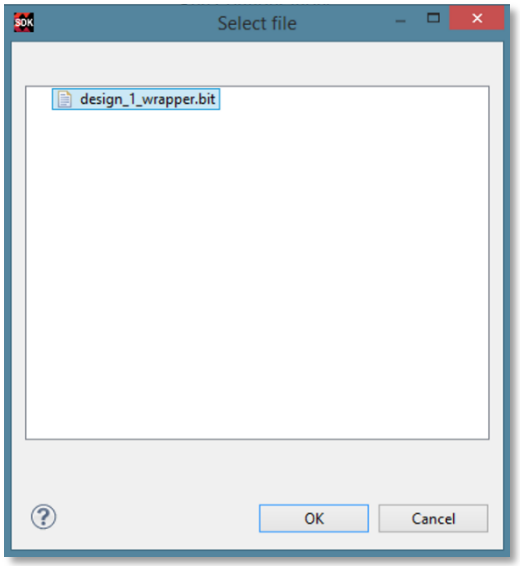


Figure 83 – Select File.

17. In the “Target Setup” tab look for the “Initialization File:” box, click “Search…” (Figure 84) and select the file “ps7_init.tcl” (Figure 85). Then uncheck the checkboxes “Run ps7_init” and “Run ps7_post_config” and click “Apply”

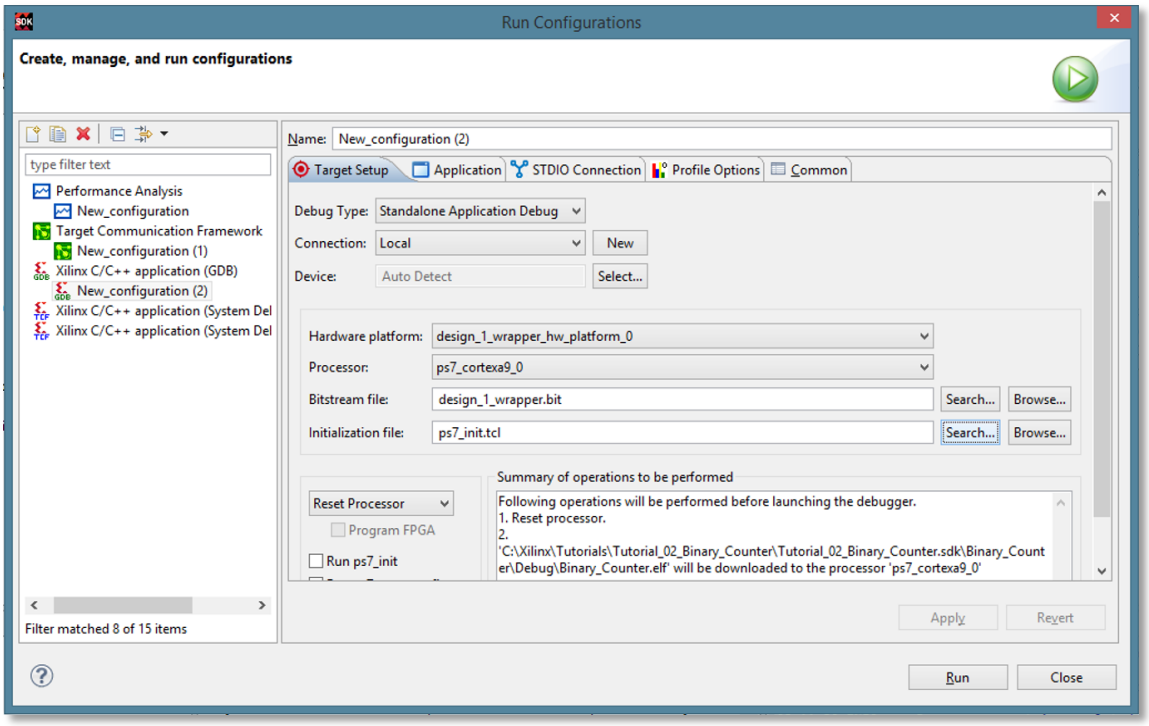


Figure 84 - Configurations of Run Settings.

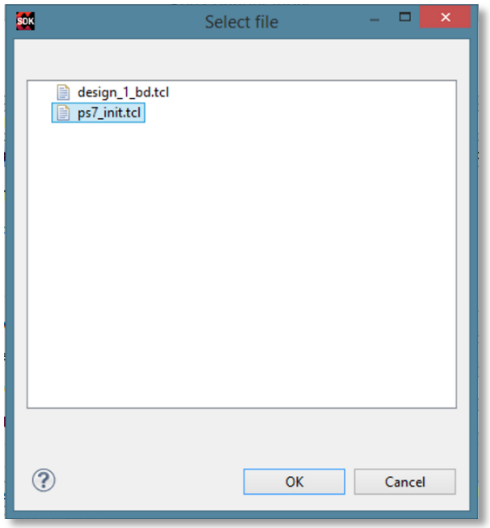


Figure 85 – Select File.

18. In the “Application” tab look for the “Project Name” box, click “Browse…” (Figure 86) and select the name of the project (Figure 87)

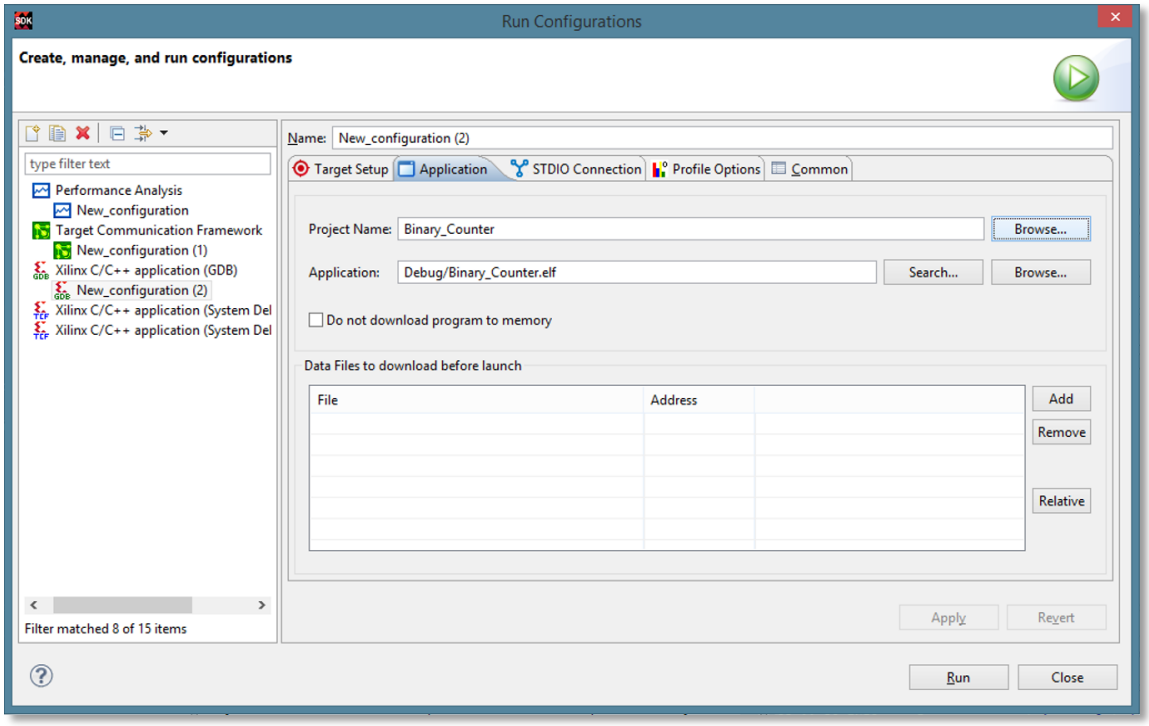


Figure 86 - Configurations of Run Settings.

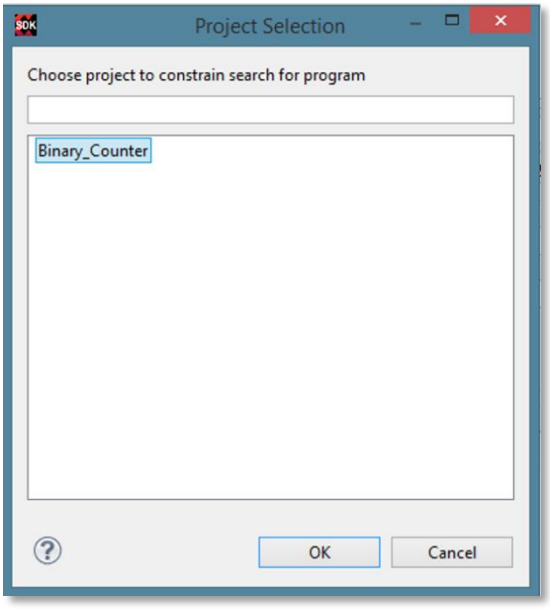


Figure 87 – Project Selection.

19. At last, check if the “Summary of operations to be performed” box in the “Target Setup” tab looks similar to that in Figure 88. Click on “Apply” and then “Run”

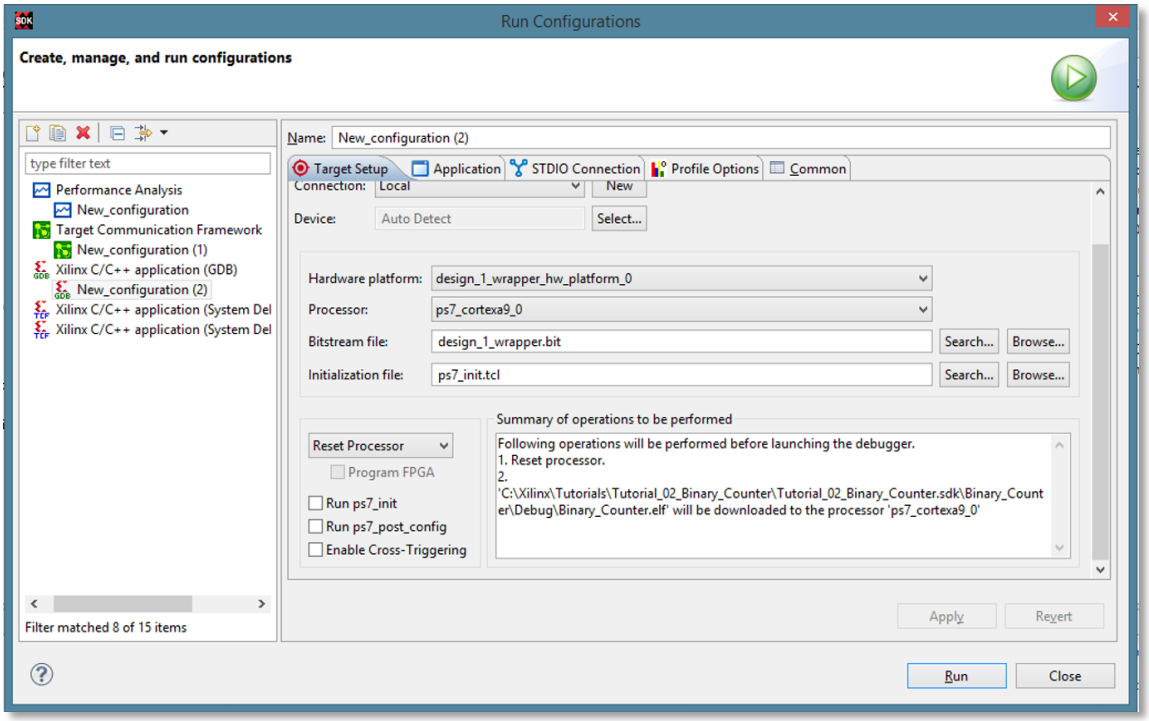


Figure 88 – Running the Project.

20. Finally it is possible to see the board counting (Figure 89)

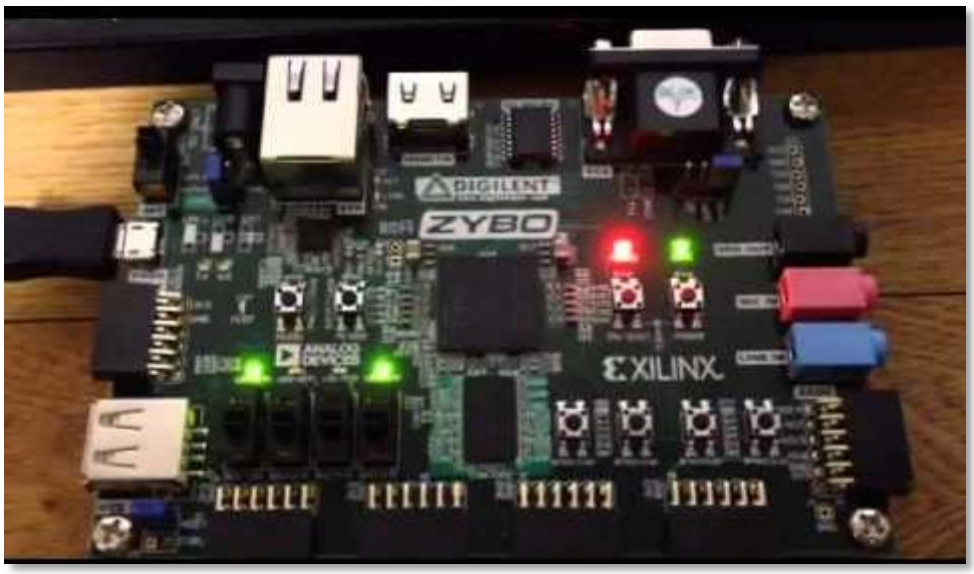


Figure 89 – ZYBO Board Leds Counting.

References:

[1] XILINX, "Vivado Design Suite Implementation," 24 June 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug904-vivado-implementation.pdf. [Accessed 11 Set 2015].

[2] XILINX, "Vivado Design Suite Synthesis," 24 June 2015. [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug901-vivado-synthesis.pdf. [Accessed 11 Set 2015].

[3] H. M. Santos, *Vivado 2014.x/2015.x Quick Start Tutorial with ZYBO Board Zynq 7000,* Porto, 2015.

Anexo C. Tutorial III – VGA-Out in VHDL at Vivado 2014.x/2015.x Quick Start Tutorial to ZYBO Board

VGA OUT IN VHDL VIVADO 2014.X/2015.X QUICK START TUTORIAL TO ZYBO BOARD

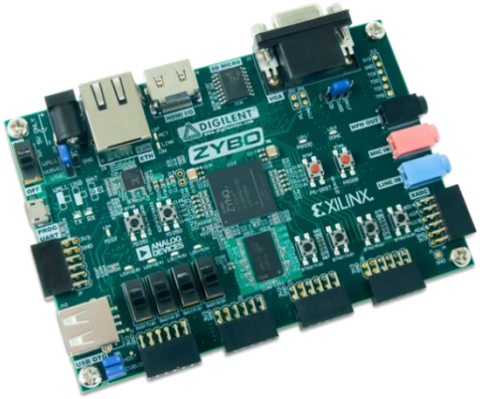
Héber Miguel dos Santos

Instituto Superior de Engenharia do Porto

Department of Electrical Engineering

Master in Electrical and Computer Engineering

NOV/2015



ZYNQ™ VIVADO™

Summary:

Figure List:

General software, hardware, files and Information needed:

- Xilinx Vivado 2014.x/2015.x;
- ZYBO Zync-7000 Development Board;
- ZYBO Board reference Manual;
- ZYBO Master XDC;
- Micro USB cable.

Software and hardware needed in specifics projects:

- SDK Webpack;
- Power supply 5V;
- Monitor with the VGA and HDMI interfaces;
- Micro SD-card minimal 4Gb.

Important Reference Links:

https://www.digilentinc.com/Products/Detail.cfm?NavPath=2,400,1198&Prod=ZYBO

http://www.zynqbook.com/

http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_1/ug910-vivado-getting-started.pdf

Tutorial's Knowhow:

How to create a logic VGA Design to a FPGA project in VHDL, with the Vivado Software, to be implemented on the ZYBO board.

The basic structure:

This design shows a basic VGA component that creates the VGA signal timing, written in VHDL to implement in FPGAs. Figure 1 shows a simple VGA signal generator system. The VGA Controller module needs a pixel clock with the frequency of the VGA monitor used, to generate all of the others signals necessary to the interface. It generates the pixel coordinates to synchronize an image source (the Image Generator) that generates the pixel values to the VGA RGB, where the VGA monitor is connected. Furthermore, it generates the sync signals for the VGA monitor too.

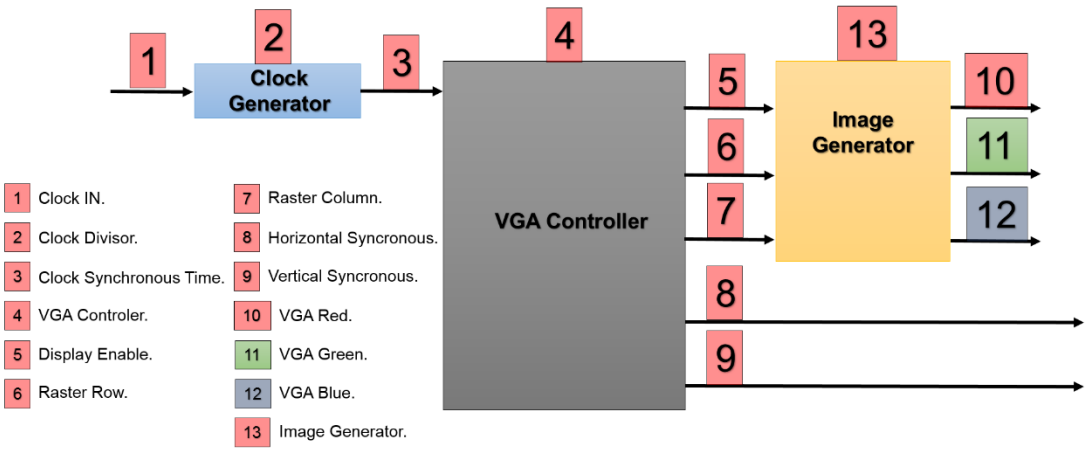


Figure 1 – VGA Generator controller Basic in VHDL.

"The VGA is a standard interface for controlling analog monitors. The computing side of the interface provides the monitor with horizontal and vertical sync signals, color magnitudes, and ground references." [1]

"The horizontal and vertical sync signals are 0V/5V digital waveforms that synchronize the signal timing with the monitor. Being digital, they are provided directly by the FPGA (3.3V meets the minimum threshold for a logical high, so 3.3V can be used instead of 5V)." [1]

"The color magnitudes are 0V-0.7V analog signals sent over the R, G, and B wires. (Alternatively, the green wire can use 0.3V-1V signals that incorporate both the horizontal and vertical sync signals, eliminating the need for those lines)." [1]

The ZYBO VGA Driver:

"The ZYBO board uses 18 programmable logic pins to create an analog VGA output port. This translates to 16-bit color depth and two standard sync signals (HS – Horizontal Sync, and VS – Vertical Sync). The digital-to-analog conversion is done using a simple R-2R resistor ladder. The ladder works in conjunction with the 75-ohm termination resistance of the VGA display to create 32 and 64 analog signal levels red, blue, and green VGA signals. This circuit, shown in Figure 2, produces video color signals that proceed in equal increments between 0V (fully off) and 0.7V (fully on). With 5 bits each for red and blue and 6 bits for green, 65,536 (32×32×64) different colors can be displayed, one for each unique 16-bit pattern." [2]

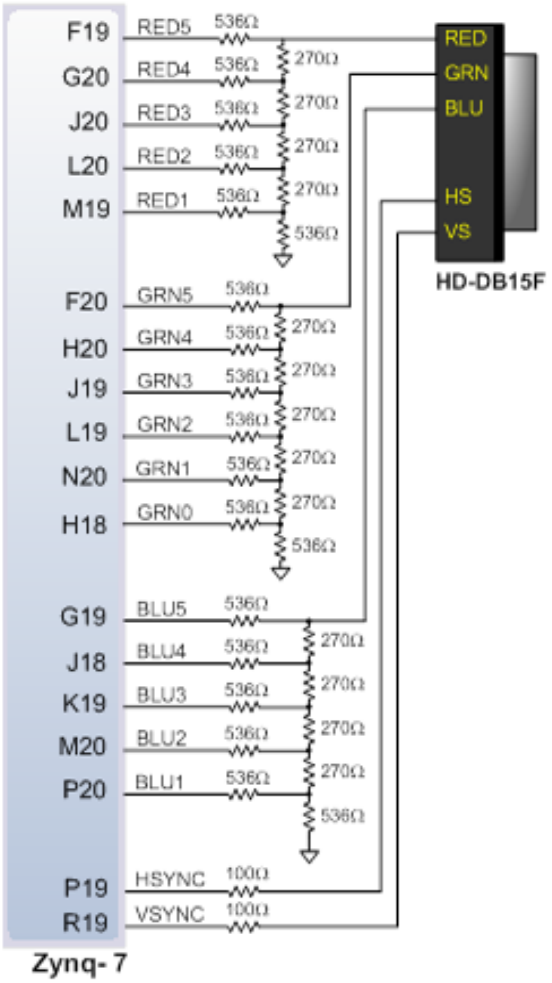


Figure 2 – VGA Driver by R-2R.

The VGA connector pins are shown in Figure 3.

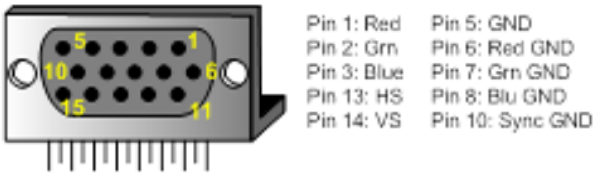
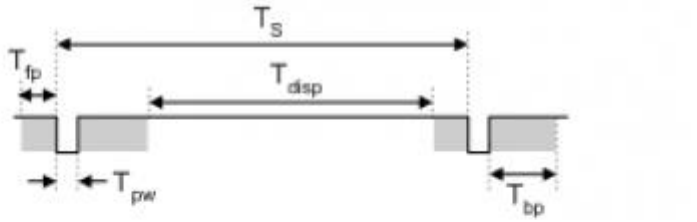


Figure 3 – VGA Connector Pins.

The Signal Timing:

“Modern VGA displays can accommodate different resolutions. A VGA controller circuit dictates the resolution by producing timing signals to control the raster patterns. Raster video displays define a number of “rows” that corresponds to the number of horizontal passes a cathode makes over the display area, and a number of “columns” that corresponds to an area on each row that is assigned to one “picture element”, or pixel. Typical displays use from 240 to 1200 rows and from 320 to 1600 columns. The overall size of a display and the number of rows and columns determines the size of each pixel.” [2]

“A VGA controller circuit must generate the HS and VS timings signals and coordinate the delivery of video data based on the pixel clock. The pixel clock defines the time available to display one pixel of information. The VS signal defines the “refresh” frequency of the display, or the frequency at which all information on the display is redrawn. The minimum refresh frequency is a function of the display’s phosphor and electron beam intensity, with practical refresh frequencies falling in the 50Hz to 120Hz range. The number of lines to be displayed at a given refresh frequency defines the horizontal “retrace” frequency. For example, for a 640-pixel by 480-row display using a 25MHz pixel clock and 60 +/-1Hz refresh, the signal timings shown in Figure 4 can be derived. Timings for sync pulse width and front and back porch intervals (porch intervals are the pre- and post-sync pulse times during which information cannot be displayed) are based on observations taken from actual VGA displays.” [2]



Symbol	Parameter	Vertical Sync			Horiz. Sync	
		Time	Clocks	Lines	Time	Clks
T_S	Sync pulse	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	Pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	Front porch	320 us	8,000	10	640 ns	16
T_{bp}	Back porch	928 us	23,200	29	1.92 us	48

Figure 4 - Signal timings for a 640-pixel by 480 row display using a 25MHz pixel clock and 60Hz vertical refresh.

“There is a wide variety of standard VGA modes, each with a specific resolution and refresh rate. Table 1, in the annex, shows the signal timing specifications for numerous VGA modes.” [1]

The Implemented Design:

Based on the information reported, the project has the module structure shown in Figure 5. Apart from the basic modules needed to create the VGA controller, there are two other modules: a Clock Divider module; and an Image Generator module with some input buttons. They are used in this example to generate a the image of a ball on the VGA monitor that may be moved around using those buttons, as shown in Figure 6.

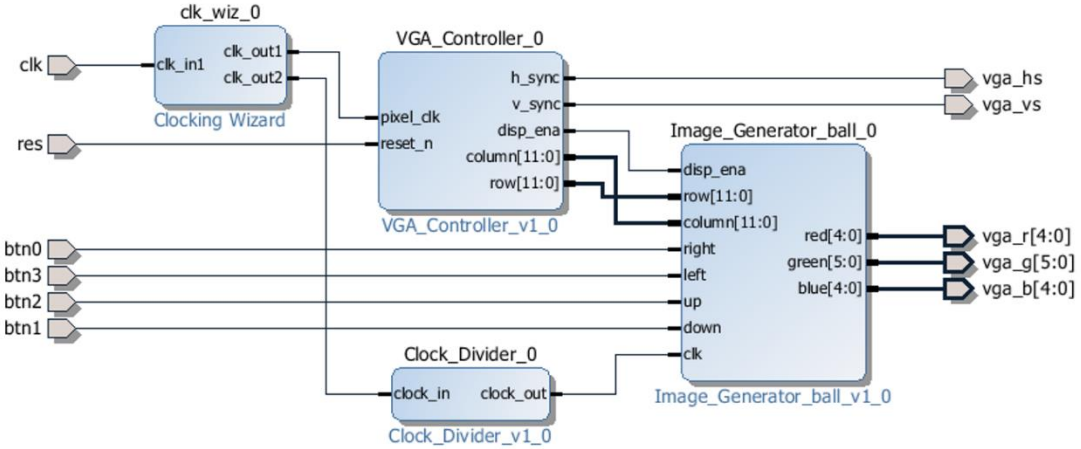


Figure 5 – VGA Design Module Driver.

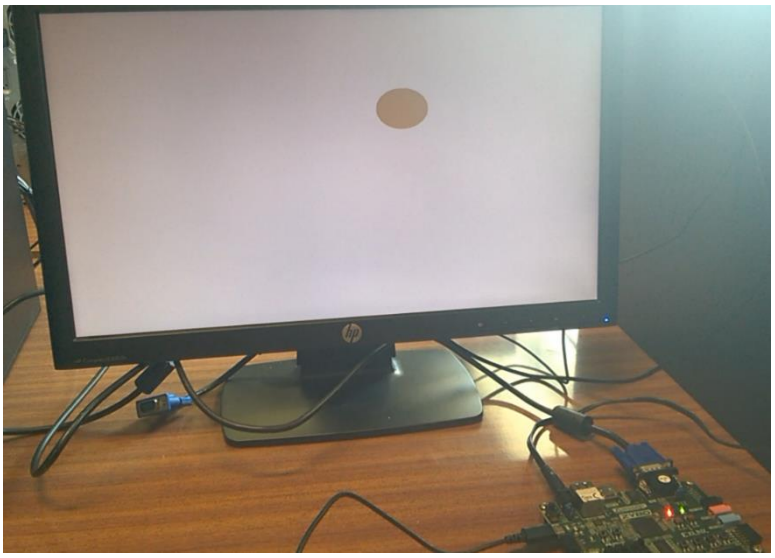


Figure 6 – Running the created design.

The first stage of the project, as seen in Figure 1, comprises the design of the Clock Generator, basically a clock divider module needed to generate the refresh frequency of the VGA display from the clock available in the ZYBO board. This project uses two modules to divide the clock as shown in Figure 7. The first module is an IP module named Clocking Wizard, which is an IP from the Vivado library. This is an accurate way to divide and generate the clocks needed in oru project needs. The only limitation of the IP Clocking Wizard is that the minimum frequency it is able to generate is, in our case, 4.687MHz. However, to read the buttons we need a frequency close to 150Hz. So, this is the reason we need to add another Clock Divider module in VHDL.

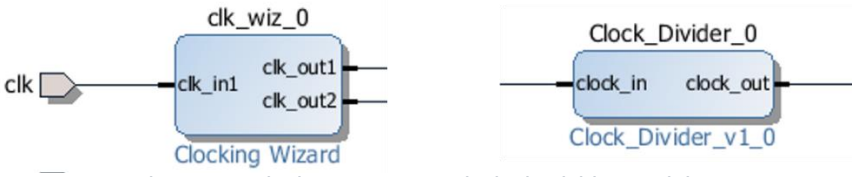


Figure 7 – Clock Generator and Clock Divider modules.

The second stage is the VGA Controller module shown in Figure 8. This module receives the refresh clock and a reset signal and generates the sync signals: Horizontal

Sync (h_sync); and Vertical Sync (v_sync), the Display Enable (Disp_ena), the Raster Column (column) and the Raster Row (row) signals. These signals are used to sync the VGA monitor output. The h_sync and v_sync are connected directly into the connector because this signals control the raster time of the VGA monitor. The disp_ena is connected to the module Image Generator to disable the monitor (send black to all the RGB colors outputs) when the sync signals are sent. The column and row are two vectors connected to the Image Generator module, with the coordinates of the current pixel being refreshed, needed to correctly transfer and display the desired image in the VGA monitor. The Image Generator uses GENERIC statements, declared in the ENTITY statement, so any timing specifications, except for the pixel clock which is generated by the Clock Module, may be set by the user according to his monitor's characteristics.

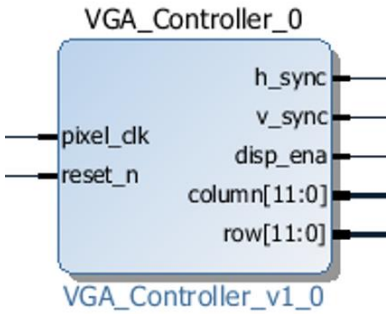


Figure 8 – VGA Controller.

The third stage is the Image Generator module, shown in Figure 9. This module receives the raster signals from the VGA Controller module and processes the position of the pixel scan generating the RGB output color to all the positions forming the image, a ball in this example. The four input buttons enable the user to specify the current display position of the ball.

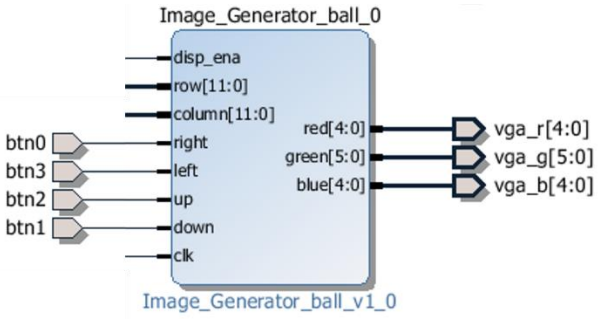


Figure 9 – Image Generator.

Tips on how to Create this Project:

The focus here is not to show step-by-step how to implement this project, but rather to give the user some insights on how to accomplish this development task.

For more detailed views on how to use the tools, the user should follow the "VIVADO 2014.X/2015.X Quick Start Tutorial to ZYBO Board".

Continue this design flow first by opening Vivado and by creating a new project design. After, follow the listed tips:

1. The first module is the Clocking Wizard, which is an IP from the IP Vivado Library. To create this module, right click anywhere within the "Diagram Window" tab and

search for “Clocking Wizard”. Double click on the name selected and the module will be added. Edit the module by double clicking on it, and set the module parameters according to the ones shown in Figure 10, Figure 11 and Figure 12

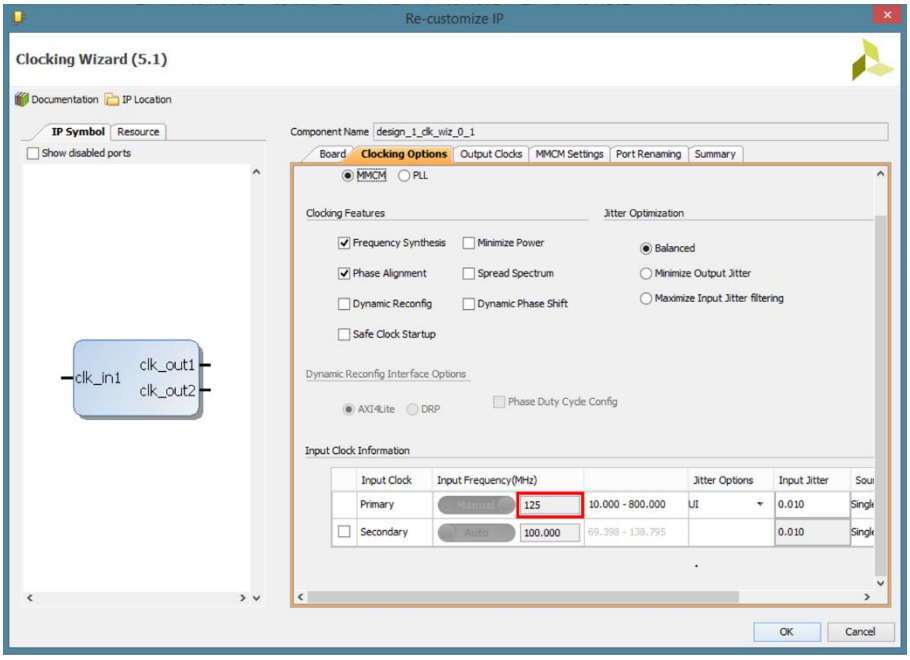


Figure 10 – Re-customize IP Input Frequency.

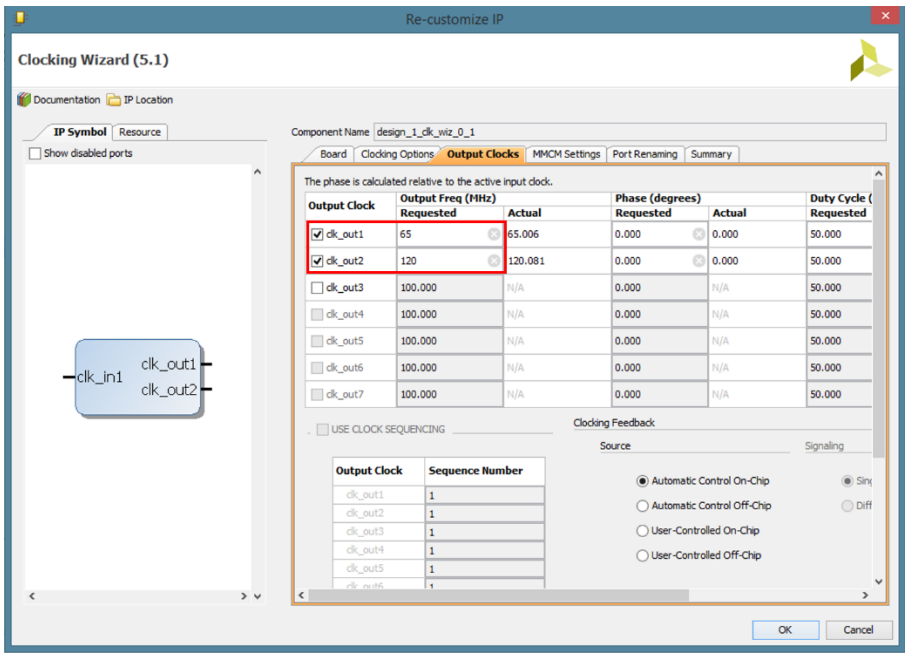


Figure 11 – Re-customize IP Clock output 1 & 2.

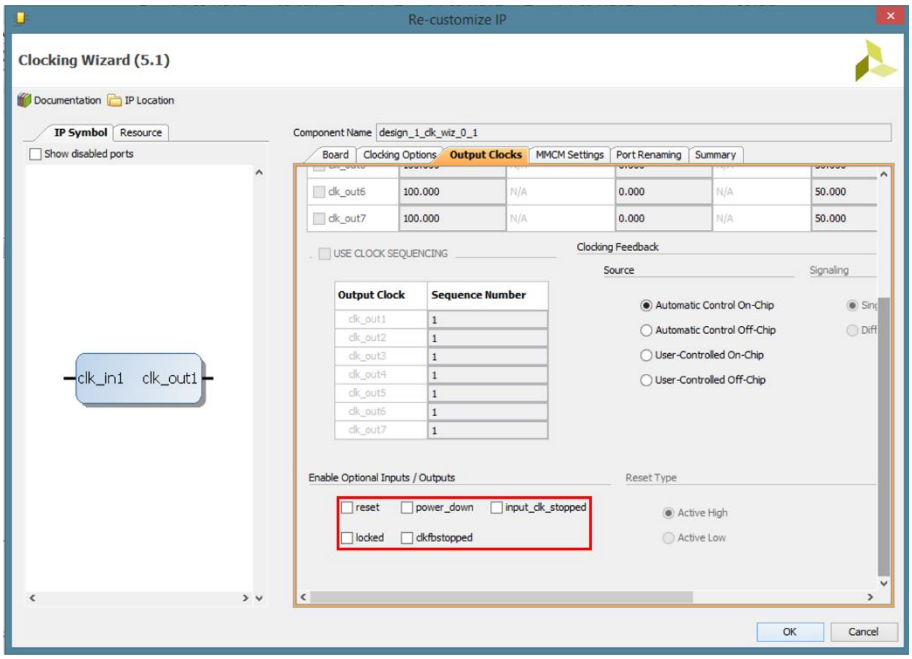


Figure 12 – Re-customize IP Enable Optional Inputs.

2. The others modules are developed using the design flow explained in the "VIVADO 2014.X/2015.X Quick Start Tutorial to ZYBO Board". Create a project in Vivado and add a design Sources in VHDL. The VHDL descriptions for the remaining module are shown in the annex of this document: Code 1, Code 2 and Code 3.
3. To Create and Package IP for the modules "VGA Controller" and "Image Generator" it is necessary to make a change on the Ports and Interfaces because the Vivado does not convert directly the integer output to a vector, in this case, the Ports "column" and "row" declared in both modules. To do the change start the "Creating and Package IP" and then, when the "Package IP Window Tab" appears, on the "Packaging Steps" → "Ports and Interfaces" click on the Port "column". A window "IP Port Properties", like the one shown in Figure 13, appears . In this window, first set the check box "Is Vector" on the down window, and add the number of bits used by this port, in this case "11", as shown in Figure 14.

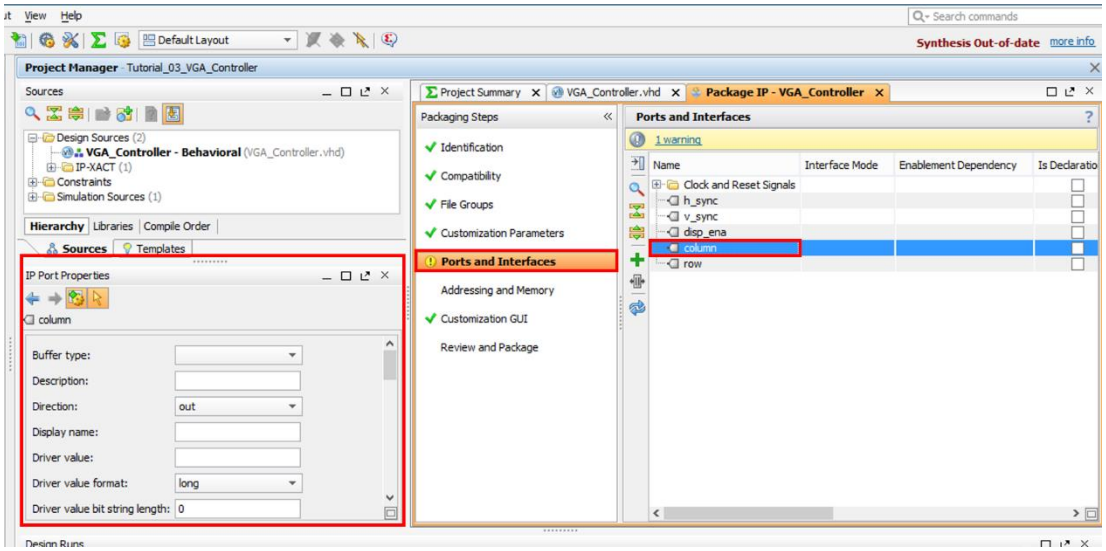


Figure 13 – IP Port Properties.

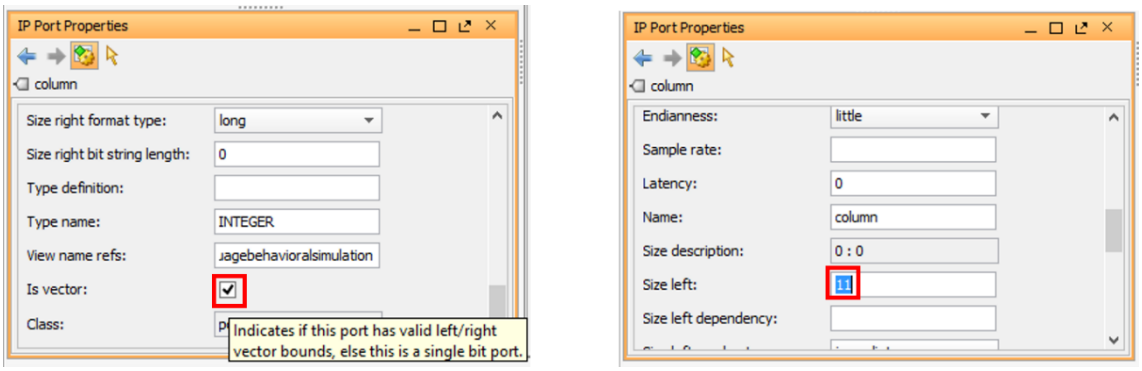


Figure 14 – IP Port Properties editions.

4. After all the modules have been created, use Figure 5 as a base to make the connections and create the necessary Ports.
5. Add the constraints file ZYBO_Master.xdc, available on the Digilent website, and then uncomment the necessary lines and edit the necessary Port names to correctly assign all pins.
6. Compile the project jumping directly to Generate Bitstream and program the ZYBO board following the same steps that are shown on the “VIVADO 2014.X/2015.X Quick Start Tutorial to ZYBO Board”. [3]

References:

[1] S. Larson, "VGA Controller (VHDL)," 01 Aug 2013. [Online]. Available: https://eewiki.net/pages/viewpage.action?pageId=15925278. [Accessed 11 Set 2015].

[2] J. Woldstad, "ZYBO Manual," 06 Mar 2015. [Online]. Available: https://reference.digilentinc.com/zybo:refmanual. [Accessed 11 Set 2015].

[3] H. M. d. Santos, *Vivado 2014.x/2015.x Quick Start Tutorial with ZYBO Board Zynq 7000,* Porto, 2015.

Annex:

Table 1 - Timing Specifications for Various VGA Modes

Resolution (pixels)	Refresh Rate (Hz)	Pixel Clock (MHz)	Horizontal (pixel clocks)				Vertical (rows)				h_sync Polarity	v_sync Polarity
			Display	Front Porch	Sync Pulse	Back Porch	Display	Front Porch	Sync Pulse	Back Porch		
640x350	70	25.175	640	16	96	48	350	37	2	60	p	n
640x350	85	31.5	640	32	64	96	350	32	3	60	p	n
640x400	70	25.175	640	16	96	48	400	12	2	35	n	p
640x400	85	31.5	640	32	64	96	400	1	3	41	n	p
640x480	60	25.175	640	16	96	48	480	10	2	33	n	n
640x480	73	31.5	640	24	40	128	480	9	2	29	n	n
640x480	75	31.5	640	16	64	120	480	1	3	16	n	n
640x480	85	36	640	56	56	80	480	1	3	25	n	n
640x480	100	43.16	640	40	64	104	480	1	3	25	n	p
720x400	85	35.5	720	36	72	108	400	1	3	42	n	p
768x576	60	34.96	768	24	80	104	576	1	3	17	n	p
768x576	72	42.93	768	32	80	112	576	1	3	21	n	p
768x576	75	45.51	768	40	80	120	576	1	3	22	n	p
768x576	85	51.84	768	40	80	120	576	1	3	25	n	p
768x576	100	62.57	768	48	80	128	576	1	3	31	n	p
800x600	56	36	800	24	72	128	600	1	2	22	p	p
800x600	60	40	800	40	128	88	600	1	4	23	p	p
800x600	75	49.5	800	16	80	160	600	1	3	21	p	p
800x600	72	50	800	56	120	64	600	37	6	23	p	p
800x600	85	56.25	800	32	64	152	600	1	3	27	p	p
800x600	100	68.18	800	48	88	136	600	1	3	32	n	p
1024x768	43	44.9	1024	8	176	56	768	0	8	41	p	p
1024x768	60	65	1024	24	136	160	768	3	6	29	n	n
1024x768	70	75	1024	24	136	144	768	3	6	29	n	n
1024x768	75	78.8	1024	16	96	176	768	1	3	28	p	p
1024x768	85	94.5	1024	48	96	208	768	1	3	36	p	p
1024x768	100	113.31	1024	72	112	184	768	1	3	42	n	p
1152x864	75	108	1152	64	128	256	864	1	3	32	p	p
1152x864	85	119.65	1152	72	128	200	864	1	3	39	n	p

Resolution (pixels)	Refresh Rate (Hz)	Pixel Clock (MHz)	Horizontal (pixel clocks)				Vertical (rows)				h_sync Polarity	v_sync Polarity
			Display	Front Porch	Sync Pulse	Back Porch	Display	Front Porch	Sync Pulse	Back Porch		
1152x864	100	143.47	1152	80	128	208	864	1	3	47	n	p
1152x864	60	81.62	1152	64	120	184	864	1	3	27	n	p
1280x1024	60	108	1280	48	112	248	1024	1	3	38	p	p
1280x1024	75	135	1280	16	144	248	1024	1	3	38	p	p
1280x1024	85	157.5	1280	64	160	224	1024	1	3	44	p	p
1280x1024	100	190.96	1280	96	144	240	1024	1	3	57	n	p
1280x800	60	83.46	1280	64	136	200	800	1	3	24	n	p
1280x960	60	102.1	1280	80	136	216	960	1	3	30	n	p
1280x960	72	124.54	1280	88	136	224	960	1	3	37	n	p
1280x960	75	129.86	1280	88	136	224	960	1	3	38	n	p
1280x960	85	148.5	1280	64	160	224	960	1	3	47	p	p
1280x960	100	178.99	1280	96	144	240	960	1	3	53	n	p
1368x768	60	85.86	1368	72	144	216	768	1	3	23	n	p
1400x1050	60	122.61	1400	88	152	240	1050	1	3	33	n	p
1400x1050	72	149.34	1400	96	152	248	1050	1	3	40	n	p
1400x1050	75	155.85	1400	96	152	248	1050	1	3	42	n	p
1400x1050	85	179.26	1400	104	152	256	1050	1	3	49	n	p
1400x1050	100	214.39	1400	112	152	264	1050	1	3	58	n	p
1440x900	60	106.47	1440	80	152	232	900	1	3	28	n	p
1600x1200	60	162	1600	64	192	304	1200	1	3	46	p	p
1600x1200	65	175.5	1600	64	192	304	1200	1	3	46	p	p
1600x1200	70	189	1600	64	192	304	1200	1	3	46	p	p
1600x1200	75	202.5	1600	64	192	304	1200	1	3	46	p	p
1600x1200	85	229.5	1600	64	192	304	1200	1	3	46	p	p
1600x1200	100	280.64	1600	128	176	304	1200	1	3	67	n	p
1680x1050	60	147.14	1680	104	184	288	1050	1	3	33	n	p
1792x1344	60	204.8	1792	128	200	328	1344	1	3	46	n	p
1792x1344	75	261	1792	96	216	352	1344	1	3	69	n	p
1856x1392	60	218.3	1856	96	224	352	1392	1	3	43	n	p
1856x1392	75	288	1856	128	224	352	1392	1	3	104	n	p
1920x1200	60	193.16	1920	128	208	336	1200	1	3	38	n	p

Resolution (pixels)	Refresh Rate (Hz)	Pixel Clock (MHz)	Horizontal (pixel clocks)				Vertical (rows)				h_sync Polarity	v_sync Polarity
			Display	Front Porch	Sync Pulse	Back Porch	Display	Front Porch	Sync Pulse	Back Porch		
1920x1440	60	234	1920	128	208	344	1440	1	3	56	n	p
1920x1440	75	297	1920	144	224	352	1440	1	3	56	n	p

Code 1 – Clock Divider:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity Clock_Divider is

     Port (  clock_in : IN std_logic;

             clock_out : out std_logic

           );

end Clock_Divider;

architecture Behavioral of Clock_Divider is

    signal count_clk : integer range 0 TO 150000000 := 0; -- Incremental
“variable”

    signal s_clk : std_logic := '1'; -- state variable

begin

    process (clock_in) -- sensibility

        begin

            if ((clock_in = '1') and (clock_in'event)) then – select the up
edge

                count_clk <= (count_clk + 1);  -- count to wait the time
defined

                if  (count_clk = 100000000) then

                    s_clk <= not(s_clk);      -- generate the clock out

                    count_clk <= 0;

                end if;

            end if;

       end process;

    clock_out <= s_clk;

end Behavioral;
```

Code 2 – VGA Controller:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity VGA_Controller is

    GENERIC(

        h_pulse  :  INTEGER   := 136;   --horiztonal sync pulse width in
pixels

        h_bp     :  INTEGER   := 160;   --horiztonal back porch width in
pixels

        h_pixels :  INTEGER   := 1024;  --horiztonal display width in pixels

        h_fp     :  INTEGER   := 24;   --horiztonal front porch width in
pixels

        h_pol    :  STD_LOGIC := '0';   --horizontal sync pulse polarity (1 =
positive, 0 = negative)

        v_pulse  :  INTEGER   := 6;     --vertical sync pulse width in rows

        v_bp     :  INTEGER   := 29;    --vertical back porch width in rows

        v_pixels :  INTEGER   := 768;  --vertical display width in rows

        v_fp     :  INTEGER   := 3;     --vertical front porch width in rows

        v_pol    :  STD_LOGIC := '0');  --vertical sync pulse polarity (1 =
positive, 0 = negative)

      PORT(

        pixel_clk :  IN   STD_LOGIC;  --pixel clock at frequency of VGA mode
being used

        reset_n   :  IN   STD_LOGIC;  --active low asycnchronous reset

        h_sync    :  OUT  STD_LOGIC;  --horiztonal sync pulse

        v_sync    :  OUT  STD_LOGIC;  --vertical sync pulse

        disp_ena  :  OUT  STD_LOGIC;  --display enable ('1' = display time,
'0' = blanking time)

        column    :  OUT  INTEGER range 0 to 2048;    --horizontal pixel
coordinate

        row       :  OUT  INTEGER range 0 to 2048    --vertical pixel
coordinate

            );

end VGA_Controller;

architecture Behavioral of VGA_Controller is

     CONSTANT  h_period  :  INTEGER range 0 to 2048 := h_pulse + h_bp +
h_pixels + h_fp;  --total number of pixel clocks in a row

     CONSTANT  v_period  :  INTEGER range 0 to 2048 := v_pulse + v_bp +
v_pixels + v_fp;  --total number of rows in column

begin

     PROCESS(pixel_clk, reset_n)
```

```
          VARIABLE h_count  :  INTEGER RANGE 0 TO h_period - 1 := 0;  --
horizontal counter (counts the columns)
          VARIABLE v_count  :  INTEGER RANGE 0 TO v_period - 1 := 0;  --
vertical counter (counts the rows)
     BEGIN
          IF(reset_n = '0') THEN  --reset asserted
            h_count := 0;         --reset horizontal counter
            v_count := 0;         --reset vertical counter
            h_sync <= NOT h_pol;  --deassert horizontal sync
            v_sync <= NOT v_pol;  --deassert vertical sync
            disp_ena <= '0';      --disable display
            column <= 0;          --reset column pixel coordinate
            row <= 0;             --reset row pixel coordinate
          ELSIF(pixel_clk'EVENT AND pixel_clk = '1') THEN
            --counters
            IF(h_count < h_period - 1) THEN    --horizontal counter (pixels)
              h_count := h_count + 1;
            ELSE
              h_count := 0;
              IF(v_count < v_period - 1) THEN  --veritcal counter (rows)
                v_count := v_count + 1;
              ELSE
                v_count := 0;
              END IF;
            END IF;
            --horizontal sync signal
            IF(h_count < h_pixels + h_fp OR h_count > h_pixels + h_fp +
h_pulse) THEN
              h_sync <= NOT h_pol;    --deassert horiztonal sync pulse
            ELSE
              h_sync <= h_pol;        --assert horiztonal sync pulse
            END IF;
            --vertical sync signal
            IF(v_count < v_pixels + v_fp OR v_count > v_pixels + v_fp +
v_pulse) THEN
              v_sync <= NOT v_pol;    --deassert vertical sync pulse
            ELSE
              v_sync <= v_pol;        --assert vertical sync pulse
```

```
            END IF;
            --set pixel coordinates
            IF(h_count < h_pixels) THEN  --horiztonal display time
              column <= h_count;         --set horiztonal pixel coordinate
            END IF;
            IF(v_count < v_pixels) THEN  --vertical display time
              row <= v_count;    --set vertical pixel coordinate
            END IF;
            --set display enable output
            IF(h_count < h_pixels AND v_count < v_pixels) THEN  --display
time
              disp_ena <= '1';              --enable display
            ELSE                             --blanking time
              disp_ena <= '0';              --disable display
            END IF;
          END IF;
     END PROCESS;
end Behavioral;
```

Code 3 – Image Generator:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Image_Generator_ball is
    PORT(
        disp_ena :  IN   STD_LOGIC;  --display enable ('1' = display time, '0'
= blanking time)
        row      :  IN   INTEGER range 0 to 2048;    --row pixel coordinate
        column   :  IN   INTEGER range 0 to 2048;    --column pixel coordinate
        red      :  OUT  STD_LOGIC_VECTOR(4 DOWNTO 0) := (OTHERS => '0');  --
red magnitude output to DAC
        green    :  OUT  STD_LOGIC_VECTOR(5 DOWNTO 0) := (OTHERS => '0');  --
green magnitude output to DAC
        blue     :  OUT  STD_LOGIC_VECTOR(4 DOWNTO 0) := (OTHERS => '0');
        right    :  IN   STD_LOGIC;  --move direita
        left     :  IN   STD_LOGIC;  --move lefterda
        up       :  IN   STD_LOGIC;  --move up
        down     :  IN   STD_LOGIC;  --move down
        clk      :  IN   STD_LOGIC     --clock para proteger move
```

```
        ); --blue magnitude output to DAC
end Image_Generator_ball;
architecture Behavioral of Image_Generator_ball is
    signal  move_x :  INTEGER range 0 to 1024 :=300;
    signal  move_y :  INTEGER range 0 to 1024 :=300;
    signal  rad        :  INTEGER range 0 to 1024 := 50;
begin
    PROCESS(disp_ena, row, column, right, left,up, down, clk)
        BEGIN
------- bloco de movimento início
        IF(CLK'EVENT AND CLK='1')THEN --adicionado
                IF (right = '1' and move_x < 934 ) then  --right'event and
                    move_x <= move_x+1;
                elsif (left = '1' and move_x > 0) then --left'event and
                    move_x <= move_x-1;
                end if;
                IF (up = '1' and move_y < 700) then
                    move_y <= move_y+1;
                elsif(down = '1' and move_y > 0) then --left'event and
                        move_y <= move_y-1;
                end if;
        end if; --adicionado
------- bloco de movimento fim
        IF(disp_ena = '1') THEN        --display time
            IF(((column-move_x-15)*(column-move_x-15))+((row-move_y-15)*(row-
move_y-15)) <= (rad*rad))THEN -- Ball generate
                red     <= ("10111");
                green   <= ("100000");
                blue    <= ("00111");
              ELSE
                red <= (OTHERS => '1');
                green  <= (OTHERS => '1');
                blue <= (OTHERS => '1');
              END IF;
            ELSE                           --blanking time
              red <= (OTHERS => '0');
              green <= (OTHERS => '0');
```

```
            blue <= (OTHERS => '0');
        END IF;
    END PROCESS;
end Behavioral;
```