VIDEOVIGILÂNCIA

COMPRESSÃO E CRIPTOGRAFIA

TESE/DISSERTAÇÃO DO MEEC

José Fernando de Oliveira Carvalho

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
2008

Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores

Candidato: José Fernando de Oliveira Carvalho, Nº 1040919, 1040919@isep.ipp.pt
Orientação científica: Prof. Doutor Jorge Mamede, jbm@isep.ipp.pt

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações
Departamento de Engenharia Electrotécnica
Instituto Superior de Engenharia do Porto
10 de Dezembro de 2008

Agradecimentos

O sucesso de qualquer trabalho, assim como o da própria aprendizagem, é uma consequência directa da dedicação. Não pode haver dedicação sem muito esforço e empenho. Agradeço aos meus pais que me educaram na honra e incutiram os verdadeiros valores da dedicação. Agradeço à minha esposa e aos meus dois filhos (mais velhos) todo o apoio e compreensão que me têm dado. Por fim agradeço aos meus Professores e de uma forma especial ao orientador da Tese, o Prof. Doutor Jorge Mamede, não só pelos conhecimentos mas também pela motivação transmitida.

O Autor,

José Fernando Oliveira Carvalho

Resumo

O estudo desta tese centra-se em torno de uma ideia que envolve condições aparentemente controversas, i.e. videovigilância segura, usando sistemas de baixo custo, através de redes baixo débito. Pretende-se com esta tese demonstrar que, recorrendo a determinados mecanismos de optimização e simplificação da compressão, se podem satisfazer simultaneamente as diversas condições, de forma a tornar a solução segura. Para esse efeito são estudados e implementados algoritmos de compressão (com e sem perdas) e de criptografia, num sistema de baixo custo, desenvolvido de raiz no projecto da Licenciatura, o sistema VideoVigi.

Em relação à compressão implementou-se uma versão adaptada da recomendação H.263 do ITU-T que inclui compressão de redundância espacial, temporal e de símbolos. No que diz respeito à criptografia foi implementado um algoritmo de encriptação simétrica, o standard AES. Os desenvolvimentos efectuados tiveram em conta o formato de vídeo CIF na sua versão monocromática.

Para se avaliar a nova solução VideoVigi realizaram-se diversos ensaios num ambiente semi-real. Para isso recorreu-se a um acesso ADSL de baixo débito (BW= 64 kbps) configurado num DSLAM ligado à rede IP. Nos testes realizados obteve-se uma taxa de 3 *frames* por segundo, com encriptação, e 6 *frames* por segundo, sem encriptação. A compressão atingiu valores superiores a 99%, assegurando um fluxo contínuo de imagens perfeito para a videovigilância. De outro modo, na rede usada para teste e sem recorrer à compressão, a taxa de *frames* por segundo não foi além de 0,06.

Os mecanismos de optimização da compressão e encriptação implementados, alteraram os dados do vídeo o suficiente, para poderem ser enviados por uma rede de baixo débito (64 kbps), e encriptados para poderem ser transferidos, de um modo seguro, sobre uma rede partilhada como a Internet. A simplificação destes mecanismos tornaram o seu algoritmo suficientemente "leve" para ser implementado em sistemas de baixo custo (30 MIPS). Desta forma, sistemas de baixo custo, como o VideoVigi, podem tornar a videovigilância de locais remotos viável através de redes de baixo débito (e.g. *General Packet Radio Service* – GPRS).

Palavras-Chave

Sistemas de videovigilância de baixo custo, compressão H263, criptografia simétrica, redes partilhadas de baixo débito.

Abstract

The study of this thesis is focused around an idea that involves conditions apparently controversial, i.e. secure video surveillance, low-cost solution implementations, through low-bandwidth networks. This thesis aims to prove that, through specific optimization and compression mechanisms, it is possible to satisfy the several conditions and build a secure solution. In order to make it possible studies were conducted, encryption and compression algorithms (with and without losses) were implemented and integrated in the low-cost VideoVigi system, developed from scratch during the final project of the graduation course.

Regarding the compression, an adapted version of the ITU-T H.263 recommendation is implemented that includes compression of space, time and symbol redundancies. In order to encrypt the video data, a symmetric encryption algorithm was also developted, the standard AES. The developments were carried out considering the CIF video format in its monochromatic version.

In order to evaluate the VideoVigi solution, several trials were realized in a semi-real environment. For that, a low-bandwidth ADSL network access (BW = 64 kbps) was configured in a DSLAM connected to the IP network. Rates of 3 encrypted frames per second and 6 non-encrypted frames per second were measured during the tests. The compression reached values above 99%. Otherwise, on the network used for tests and without compression, the rate of frames per second was not more than 0.06.

The mechanisms for optimizing the implemented compression and encryption algorithms, processed the video data the enough to enable its transmission via a low-bandwidth network (64 kbps), and encrypted to enable the transmission through a shared network, like the Internet, in a secure way. The simplification of these mechanisms made the algorithms sufficiently "light" enabling their implementation in low cost systems (30 MIPS). Thus, low-cost systems, such as VideoVigi, makes the video surveillance of remote sites possible through low-bandwidth networks (e.g. General Packet Radio Service - GPRS).

Keywords

Low cost video surveillance systems, H263 compression, symmetric encryption, low-bandwidth shared network.

Índice

Índice de Figuras

Índice de Tabelas

Acrónimos

ADC – Analog Digital Converter

ADSL – Asymmetric Digital Subscriber Line

AES – Advanced Encryption Standard

BW – Bandwidth

CCTV – Closed Circuit Television

CD – Compact Disk

CDM – Cable Data Modem

CIF – Common Intermediate Format

CPU – Central Processing Unit

CVBS – Composite Video Blanking and Sync

DCT – Discrete Cosine Transform

DES – Data Encryption Standard

DM – Data Memory

DPO – Digital Phosphor Oscilloscope

DPRAM – Dual Port Ram

DSLAM – Digital Subscriber Line Access Multiplexer

DSP – Digital Signal Processor

DVR – Digital Video Recorder

EOB – End Of Block

FIFO – First In First Out

GOB – Group Of Blocks

GPRS – General Packet Radio Service

HTML – Hyper Text Markup Language

IDMA – Internal Data Memory Access

IP – Internet Protocol

ISP – Internet Service Provider

ITU – International Telecommunication Union

JPEG – Joint Photographic Experts Group

LAN – Local Area Network

MAC – Multiply Accumulate

ME – motion estimation

MIPS – Millions of Instructions Per Second

MII – Media Independent Interface

MJPEG – Motion Joint Photographic Experts Group

MPEG – Moving Picture Experts Group

MSE – Mean Squared Error

MTU – Maximum Transmission Unit

NVR – Network Video Record

NVRAM – Non Volatile Random Access Memory

OSI – Open Systems Interconnection

PAL – Phase Alternating Line

PLD – Programmable Logic Device

PC – Personal Computer

PCB – Printed Circuit Board

PCR – Peak Cell Rate

PDA – Personal Digital Assistant

PLD – Programmable Logic Device

PM – Program Memory

PSNR – Peak Signal-to-Noise Ratio

PWM – Pulse Width Modulation

QCIF – Quarter CIF

RAM – Random Access Memory

RDIS – Rede Digital com Integração de Serviços

RF – Radio Frequency

RTC – Real Time Clock

SDSL – Symmetric Digital Subscriber Line

SHDSL – Symmetric High speed Digital Subscriber Line

SRAM – Static Random Access Memory

TCP – Transmission Connection Protocol

TDES – Triple Data Encryption Standard

UDP – User Datagram Protocol

UHF – Ultra High Frequency

UMTS – Universal Mobile Telecommunications System

VHS – Vídeo Home System

VLC – variable length coding

1. INTRODUÇÃO

Actualmente, o desenvolvimento das tecnologias de telecomunicações facilita a interligação de *Local Area Network* (LAN) entre empresas ou habitações. Existem diversas soluções, disponíveis no mercado empresarial, que asseguram estas ligações. A rede de acesso é normalmente, nestes casos, suportada por tecnologias *Symmetric Digital Subscriber Line* (SDSL) ou *Symmetric High speed Digital Subscriber Line* (SHDSL). Por outro lado, no mercado doméstico, os acessos à Internet suportados em tecnologias *Asymmetric Digital Subscriber Line* (ADSL), *Cable Data Modem* (CDM), ou *Universal Mobile Telecommunications System* (UMTS) têm vindo a sofrer uma forte penetração. Qualquer uma destas tecnologias permite a interligação de sistemas que podem ser utilizados em aplicações muito específicas. No entanto no mercado doméstico, como o acesso está vocacionado para a Internet, é mais valorizado o *downstream* do que o *upstream*; apesar de serem acessos de banda larga o *upstream* não vai além dos 1024 kbps.

A evolução contínua das telecomunicações conduz à necessidade crescente de aplicações que permitem substituir o deslocamento físico aos locais. Neste contexto, a videovigilância assume cada vez mais uma grande importância quer no mercado empresarial como no doméstico.

1.1. CONTEXTUALIZAÇÃO

A videovigilância, pelo facto de ser uma aplicação que requer apreciáveis recursos de largura de banda, pode ser inviável em redes de baixo débito. Além disso, é importante que

seja segura, principalmente se esta não for uma aplicação passiva (interacção com o observador).

O sistema *VideoVigi*, desenvolvido no âmbito do Projecto de Curso da Licenciatura de Engenharia Electrotécnica – Electrónica e Computadores, é uma solução que pretendia satisfazer algumas destas necessidades. No entanto quando se pretende industrializar um projecto, i.e. torná-lo um produto competitivo no mercado, é importante estudar, implementar e testar diversos algoritmos de forma a melhorar o produto desenvolvido.

Neste contexto, tendo em conta redes de baixo débito, é importante perceber o quanto influencia na rede de acesso algoritmos de compressão com e sem perdas. Por outro lado, atendendo às necessidades de segurança, nomeadamente em aplicações interactivas, é também importante analisar a influência de alguns métodos de criptografia de forma a optimizar a solução e determinar requisitos de *hardware* em futuros projectos nesta área.

Estas e outras questões motivaram o desenvolvimento desta Tese que, de alguma forma, dão continuidade ao referido projecto *VideoVigi* desenvolvido.

1.2. OBJECTIVO

O objectivo desta tese é o estudo da influência das variações de débito na rede de acesso em sistemas de videovigilância remota. Mecanismos de optimização da compressão e encriptação de dados serão estudados, implementados e avaliados procurando minimizar as referidas variações e aumentar a segurança em sistemas de videovigilância autónomos. Este estudo permitirá determinar e validar requisitos no desenvolvimento de sistemas de vigilância de baixo custo para utilizar em redes de baixo débito. Contudo não se pretende com este estudo desenvolver profundamente os conceitos matemáticos associados à compressão nem à criptografia.

A implementação prática terá como suporte físico o projecto desenvolvido na Licenciatura, i.e. o sistema VideoVigi. O desenvolvimento será efectuado de acordo com as seguintes tarefas:

- Estudo dos principais algoritmos de compressão de imagem com perdas;
- Estudo de algoritmos de compressão de dados sem perdas;

- Estudo de algoritmos criptográficos;
- Desenvolvimento de *firmware* referente aos algoritmos de compressão estudados;
- Desenvolvimento de *firmware* referente aos algoritmos criptográficos estudados;
- Analise da influência directa dos vários algoritmos desenvolvidos, individualmente e sequencialmente, na rede de acesso e conclusões;

1.3. ORGANIZAÇÃO DO RELATÓRIO

Este documento encontra-se estruturado em nove capítulos. O primeiro capítulo pretende enquadrar a tese e contextualizar o problema. O segundo descreve algumas arquitecturas e tecnologias usadas em sistemas de videovigilância, bem como o sistema VideoVigi desenvolvido. No terceiro descrevem-se alguns algoritmos de compressão com e sem perdas. O quarto refere-se aos principais algoritmos de criptografia simétrica. O quinto descreve o protocolo, o algoritmo de compressão e o respectivo *firmware* desenvolvido, e no sexto o *firmware* de encriptação. No sétimo capítulo descreve-se o algoritmo e o código em Java da aplicação cliente, desenvolvida. No oitavo fazem-se análises do desempenho do sistema desenvolvido em ambiente semi-real; no último tiram-se conclusões em relação ao formato de vídeo, à compressão, e à criptografia em sistemas de baixo custo; por fim retractam-se alguns desenvolvimentos futuros.

2. SISTEMAS VIDEOVIGILÂNCIA

Ao longo dos anos os sistemas de videovigilância têm vindo a sofrer fortes mutações, quer no respeitante às tecnologias de aquisição, como de transporte das imagens. Na década de setenta surgem os primeiros sistemas clássicos de videovigilância. Estes sistemas totalmente analógicos permitiam a visualização e gravação das imagens vindas de várias câmaras de vídeo a partir de um *Closed Circuit Television* (CCTV). Nos anos oitenta a electrónica de consumo trouxe o aparecimento de *kits* de videovigilância, ainda em ambiente analógico, que oferecem uma solução económica mas não dispõe de capacidade de gravação. Inicialmente, a interligação das câmaras ao monitor era realizada através de cabo, na década de 90 surgem sistemas *wireless*. No início dos anos noventa, com o aparecimento da Rede Digital com Integração de Serviços (RDIS) em Portugal, surgem os primeiros sistemas com ligação digital quer através de acessos básicos como primários. Em noventa e quatro, com a expansão da Internet em Portugal, vulgarizam-se os sistemas baseados em *Internet Protocol* (IP).

2.1. ARQUITECTURAS E TECNOLOGIAS

Os sistemas clássicos de videovigilância apresentam uma topologia do tipo "ponto multiponto" (Figura 1). Originalmente, as câmaras de vídeo com interface de vídeo composto interligavam directamente a um equipamento de gravação do tipo *Betamax* que por sua vez disponibilizava saída directa para o monitor. Actualmente, embora ainda existam sistemas de gravação analógicos em funcionamento, nomeadamente do tipo *Vídeo Home System* (VHS), estes têm vindo a ser substituídos por *Digital Video Recorder's* (DVR), e.g. "Simplex DVR-S4U". Este sistema permite gravar simultaneamente quatro entradas de vídeo *Composite Video Blanking and Sync* (CVBS) com compressão H.264.

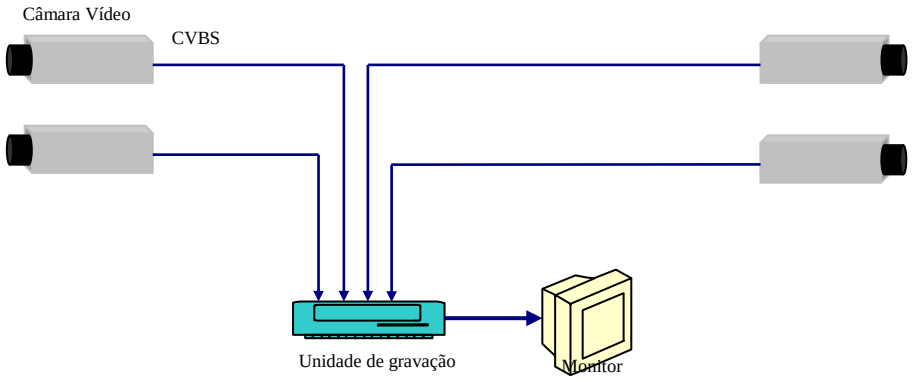


Figura 1 Arquitectura de referência do sistema clássico

Os *kits* de videovigilância apresentam também uma topologia do tipo "ponto multiponto". Mas nestes sistemas normalmente não existe qualquer capacidade de gravação. As câmaras CVBS interligam directamente ao monitor ou através de um sequenciador. Nesta gama de produtos existem soluções *wireless* que permitem instalações fáceis em locais que não disponham de qualquer infra-estrutura de rede para o efeito. As câmaras enviam os sinais *Radio Frequency* (RF) na faixa de *Ultra High Frequency* (UHF), e.g. "Rimax 30T".

Os sistemas de videovigilância baseados em interfaces RDIS, embora apresentem um funcionamento "ponto a ponto", a sua topologia de rede pode apresentar diversas configurações. Estes sistemas, como são suportados pela rede básica de telecomunicações, dispõem de uma flexibilidade muito elevada e permitem, ao contrário dos referidos anteriormente, a vigilância de locais que se encontram a elevadas distâncias. Um exemplo destes sistemas é o *VídeoPoint* da PTInovação. Este é utilizado para vigiar o tráfego rodoviário nas principais auto-estradas nacionais. Apesar das inúmeras vantagens, estes

sistemas também têm inconvenientes sendo o principal a sua largura de banda. Como um acesso básico dispõe apenas de dois canais de dados, estes garantem um débito de apenas 128 kbps. Considerando imagens a preto e branco, cada imagem da câmara de vídeo num formato intermédio *Common Intermediate Format* (CIF[1]) apresenta uma resolução de 811.008 *bits* (352x288 *Bytes*). O débito da RDIS limitaria o vídeo a 0,16 fps (131.072 / 811.008), i.e. seria actualizada cada imagem ao fim de 6,2 segundos. Por esta razão existe necessidade de compressão. A compressão utilizada em RDIS segue a norma do ITU H.261 ou H.263.

Os sistemas IP são os mais flexíveis de todos uma vez que podem ser suportados por qualquer rede física de telecomunicações. Estas redes também têm limitações na sua largura de banda, e.g. o *upstream* do ADSL, actualmente, disponível nos principais *Internet Service Provider's* (ISP) é de apenas 512 kbps ou 1024 kbps. Apesar deste valor poder ser algumas vezes superior ao da RDIS pode ser insuficiente, o que leva à necessidade de algoritmos de compressão. A compressão utilizada nestes sistemas é a *Motion Joint Photographic Experts Group* (MJPEG) definida na norma do ITU T.81[14] ou a compressão *Moving Picture Experts Group* (MPEG). Existem diversas soluções de hardware algumas mistas, i.e. integram câmaras analógicas a dispositivos IP e outras com todos os dispositivos IP. O sistema "AXIS 262" é um *Network Video Record* (NVR) que oferece uma solução totalmente IP com possibilidade de gravação local. Por outro lado, o sistema "AVC785" da *Bali CCTV* é um DVR com oito entradas de vídeo CVBS que, para além da gravação local em MPEG4, também permite a visualização das imagens via *browser* em MJPEG.

2.2. SISTEMA VIDEOVIGI

O sistema VideoVigi[1] desenvolvido no projecto da licenciatura é uma solução mista que permite integrar câmaras de vídeo analógicas numa rede IP. Este sistema não foi desenvolvido para suportar gravação, mas para permitir a visualização de imagens através de uma rede de baixo débito. A gravação poderá, no entanto, existir numa plataforma remota.

[1]O formato CIF é utilizado em MPEG1 no *Vídeo Compact Disc* (VCD). A sua resolução (352x288) é também muito utilizado em videovigilância.

A Figura 2 ilustra a arquitectura de referência do sistema desenvolvido. O sistema é composto por uma unidade de aquisição de vídeo que adquire o sinal proveniente de câmaras de vídeo (quatro no máximo), o processa e o envia em tempo real para uma estação de controlo remota através de uma ligação de baixo débito. Esta unidade foi projectada e desenvolvida de raiz no âmbito do projecto da licenciatura e foi denominada VideoVigi.

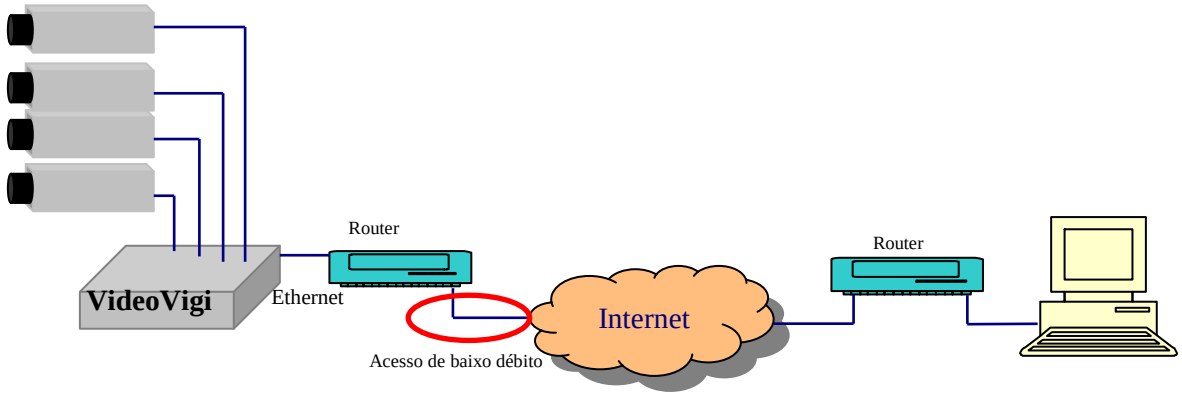


Figura 2 Sistema VídeoVigi [1]

O sistema desenvolvido assenta em duas componentes ou módulos: por um lado, todo o *hardware* e *firmware* de aquisição e compressão de vídeo, e por outro lado, uma aplicação de software que remotamente permite controlar e visualizar as imagens adquiridas pelas câmaras.

O módulo de hardware digitaliza e comprime as imagens da câmara de vídeo seleccionada no formato intermédio CIF e envia-as pela interface *ethernet.* Dispõe de quatro entradas de vídeo CVBS para as câmaras que poderão ser controladas remotamente através de uma porta série RS485 e de uma *Analog Digital Converter* (ADC). Para suporte ao desenvolvimento (mensagens de *debug*) e configurações locais do sistema, a unidade dispõe de uma porta série RS232. Dispõe ainda de oito entradas, que permitem receber alarmes de sensores e oito saídas que podem ligar a actuadores diversos, de forma flexível de acordo com as necessidades. O acesso ao VideoVigi assim como o envio das suas imagens comprimidas é efectuado através de uma porta *fast ethernet* (10/100).

O módulo de software instalado num computador desempenha a função de cliente, para controlo do local e visualização das imagens captadas remotamente. De forma a permitir executar o código em qualquer máquina, ele foi desenvolvido em *Java*. A aplicação pode ainda ser executada a partir de uma página *Hyper Text Markup Language* (HTML), sob a forma de *applet*, ou em s*tand alone*.

2.2.1. CONSTITUIÇÃO

O módulo de hardware, constituído por sete blocos funcionais é ilustrado na Figura 3. O primeiro bloco (Aquisição) é responsável pela aquisição de cada imagem, convertendo o sinal *Phase Alternating Line* (PAL) proveniente das câmaras em imagens digitais. O segundo bloco (Compressão) comprime as imagens adquiridas no formato JPEG de forma a obter vídeo MJPEG e disponibilizá-las ao bloco seguinte (Processamento). O bloco de Processamento controla todo o funcionamento do sistema e estabelece as ligações de rede, através do bloco *Ethernet.* Este assegura as funcionalidades de nível físico do modelo OSI. Para além destes blocos, existem mais dois, um bloco de Controlo que é usado para gerir as comunicações entre os três primeiros blocos e é essencialmente constituído por uma *Programmable Logic Device* (PLD) de forma a permitir alguma flexibilidade no hardware. O penúltimo bloco (Interface) assegura as comunicações com o exterior, i.e. controlo das câmaras (RS485 e ADCx8) e do local a vigiar (Alarmes e Comandos). O ultimo bloco disponibiliza as tensões de alimentação necessárias.

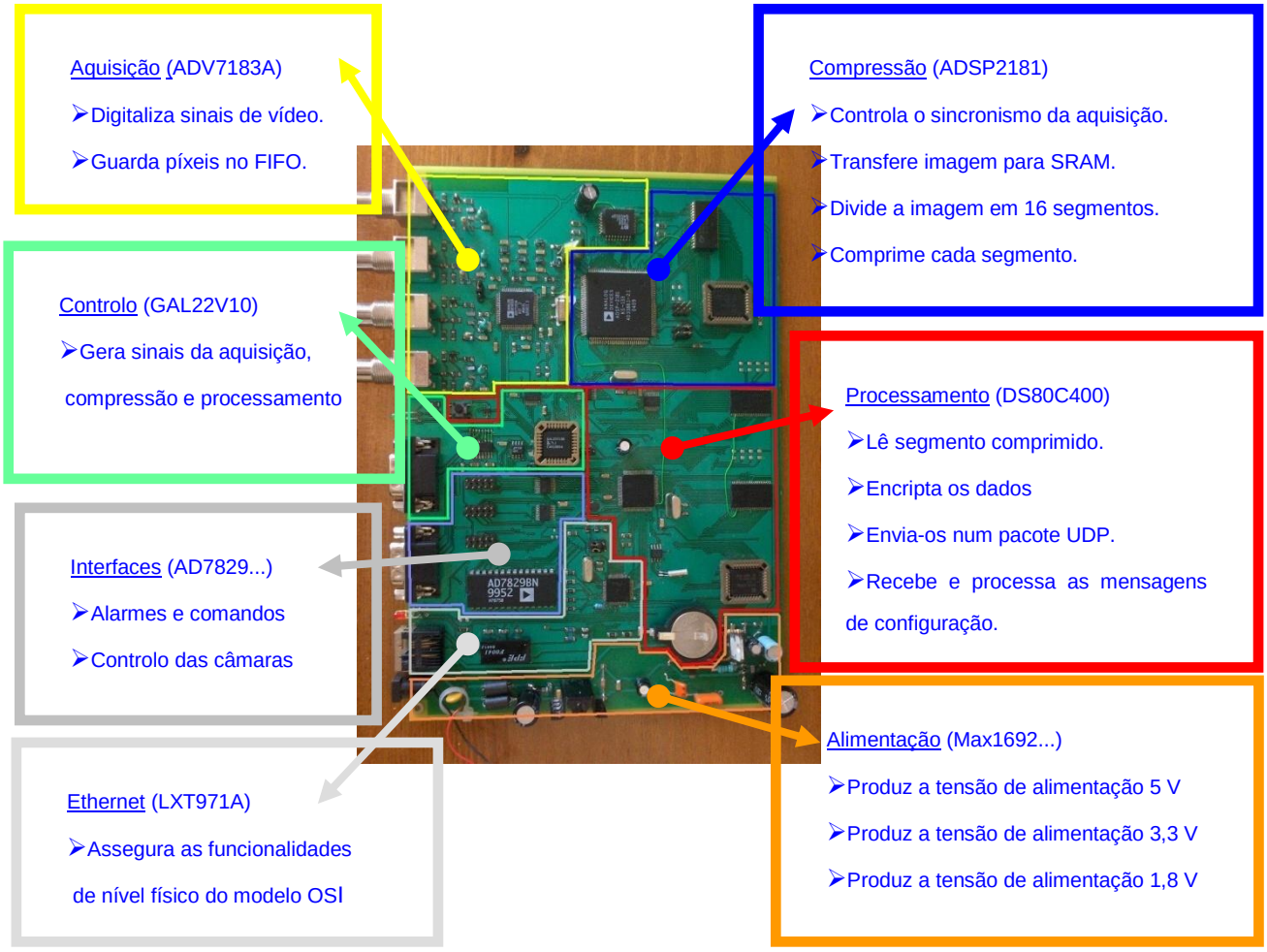


Figura 3 Módulo de hardware [1]

2.2.2. BLOCOS PRINCIPAIS

A aquisição e digitalização do vídeo é assegurada por um *Video Decoder* (VD-ADV7183A) e por um *First In First Out* (FIFO-IDT7202LA). De forma a evitar a ocorrência do efeito *Aliasing*, devido a sub-amostragem (de harmónicos), as entradas de vídeo analógico passam por um filtro activo.

A aquisição (352x256) é efectuada num único *field* e em metade dos pontos por linha. O *Digital Signal Processor* (DSP-ADSP2181KS133) controla a aquisição sempre no início do mesmo *field*, i.e. manipula os sinais do *video decoder* e do FIFO de forma a garantir. o sincronismo vertical. Enquanto o *video decoder* escreve os dados referentes a cada píxel no FIFO, o DSP vai simultaneamente lendo o FIFO e guardando metade dos pontos em *Static Random Access Memory* (SRAM), de acordo com a Figura 4.

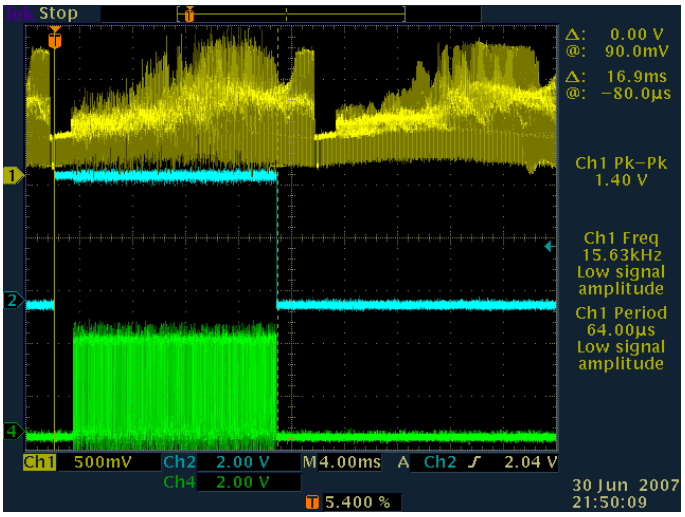


Figura 4 Aquisição síncrona [1]

Nesta figura, o canal 1 é o sinal de vídeo à saída do filtro activo (no emissor), o canal 2 é PIC_START, i.e um sinal que vem do DSP para garantir o início do *field* correcto, e o canal 3 é o /W (Write) do vídeo decoder que é também controlado pelo DSP e pela PLD.

Para ler o FIFO, o DSP utiliza o "espaço de endereçamento" I/O, embora não sejam necessário os endereços, faz-se uso do pino IOMS (I/O *Memory Select*). Para guardar uma imagem completa são necessários 128 Kbytes. Porém, o endereçamento que o ADSP2181 disponibiliza para a memória de dados(DM) externa é apenas de 14 bits, i.e. são dois segmentos de 8 Kbytes (16 Kbytes). Como para endereçar 128 Kbytes são necessário 17 bits (2^{17} = 128 Kbytes), para completar o endereçamento são utilizadas as três *Output flags*

que o ADSP disponibiliza (FL0:2), ficando com um espaço endereçável de 128 Kbytes organizado em 16 segmentos de 8 Kbytes (8x2x8Kbytes) de como se pode ver na Figura 5.

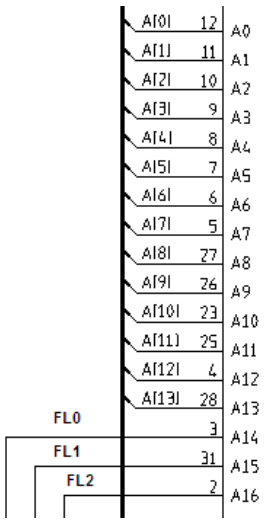


Figura 5 Endereçamento DM [1]

Cada imagem fica então dividida em 16 segmentos de 16 linhas, que contêm 88 blocos (8x8 píxeis). O DSP copia cada segmento de imagem num segmento de memória. Quando chegar ao fim do último segmento de imagem, o DSP desactiva o *decoder*.

A compressão é toda suportada pelo DSP ADSP2181KS133. Cada segmento é então copiado linha a linha para a DM interna do DSP, realizando sincronismo por software. Após configuração dos bytes de controlo (tipo de compressão, matriz de quantificação e posição do segmento), o DSP reordena o segmento por blocos (linhas de 8 píxeis). Inicia-se então a compressão para o segmento corrente que será descrita posteriormente. Embora este seja o funcionamento normal, o DSP também permite a disponibilização de imagens sem compressão.

No final da compressão, através de uma flag o DSP avisa o microcontrolador (DS80C400) que existe pacote novo e espera pela sua leitura. Após a leitura, confirmada por outra *flag* (do microcontrolador), reinicia a compressão do próximo segmento de imagem. O ADSP2181 dispões de um *bus* de 16 bits IDMA (*Internal memory Direct Memory Access)* que permite o acesso, de outros processadores, às suas memórias internas (DM e PM) e é através deste barramento que mcicrocontrolador efectua a leitua. Após a compressão e transferência de todos os blocos da actual imagem, o DSP vai activar o *decoder* que espera pelo início de uma nova *frame*, através das suas flags e da *Programmable Logic Device*

(PLD-GAL22V10). Assim que receber a informação de uma novo *field*, inicia-se um novo ciclo.

O processamento principal está assegurado pelo microcontrolador da família C51 (80DS400). Este microcontrolador é responsável pela realização das seguintes tarefas, a configuração e controlo de todo o sistema, a leitura do *bus* IDMA, a encriptação e a disponibilização de um servidor *User Datagram Protocol* (UDP).

A configuração do sistema envolve a inicialização do *video decoder*, através de um *bus* I^2C, a sinalização da *flag* que activa ou não a compressão no DSP (MIC_FL), o tipo de encriptação (com ou sem) e o modo de transmissão (*Fast, Slow, Frame* ou *Run*) descrito posteriormente. Estas configurações são efectuada quando é recebida uma mensagem de configuração por parte da aplicação cliente, e guardadas em memória na *Non Volatile Random Access Memory* (NVRAM) disponível do *Real Time Clock* (RTC-DS1307) via *bus* I^2C. No arranque do sistema o *video decoder*, a *flag* de compressão do DSP, a encriptação e o modo de transmissão do próprio micrcocontrolador são inicializados de acordo com as últimas configurações recebidas.

A leitura do *bus* IDMA do DSP permite o acesso do microcontrolador aos dados das imagens comprimidas ou sem compressão. Este recorre a *buffers* (LVT245) que servem por um lado para permitir a utilização do *bus* de dados do microcontrolador, e por outro adaptar os níveis de tensão (3V3 vs. 5V).

Após a leitura dos segmentos de imagem suficientes para o envio de um pacote UDP, se estiver activa a encriptação o microcontrolador inicia a cifra do pacote de forma a garantir o seu envio seguro pela rede partilhada.

O servidor UDP disponibilizado recorre a dois processos. O processo pai garante o atendimento da recepção de pacotes de configuração, enquanto o processo filho envia os pactotes de imagem.

2.2.3. BLOCOS AUXILIARES

O primeiro bloco auxiliar é o *Ethernet.* É neste bloco que os pacotes são enviados e recebidos pelo *bus Media Independent Interface* (MII), interligado entre o microcontrolador e o componente LXT971A. Este componente assegura as funcionalidades de nível físico do modelo *Open Systems Interconnection* (OSI).

O bloco de controlo assente essencialmente na PLD GAL22V10 serve por um lado para disponibilizar as *gates* necessárias ao funcionamento do *hardware*, e.g. entre o *vídeo decoder* e o FIFO e por outro dar alguma flexibilidade ao *hardware* e permitir o *debug* por sinalização. Por esta razão, a grande maioria das *flags* de controlo do *video decoder*, do DSP e do microcontrolador, passam por este componente assim como os dois *leds* de sinalização.

O bloco de interfaces assegura os comandos das câmaras, a recepção de alarmes e sinais de comando. Os controlos *pan*, *tilt* e *zoom* das câmaras de vídeo são feitos através da porta RS485 (LTC490), para as diversas câmaras móveis (se existirem) numa arquitectura do tipo *master/slave*. Para além disso, para ser possível colocar as câmaras em pré-posições, que são obtidas a partir de potenciómetros mecanicamente associados a cada motor, é utilizado uma *Analog Digital Converter* ADC para ler os seus valores. Como cada movimento necessita de uma entrada analógica, a ADC (AD7829) de oito entradas permite controlar oito pré-posições (e.g. duas câmaras com pré-posições de todos os seus movimentos e duas apenas o *pan* (movimento horizontal). A recepção de alarmes é realizada recorrendo a um *buffer* de oito entradas (LVT245) que possibilita a leitura de oito alarmes. Os oitos comandos são realizados através de uma *latch* LVT373, esta permite manter o estado das saídas, como se pretende.

Exceptuando a porta série RS485, as entradas analógicas da ADC, as entradas digitais de alarmes e os sinais de controlo, requerem um *Printed Circuit Board* PCB adicional, onde para além das fichas haverá a interface para ligar ao exterior de forma a "formatar" os sinais e proteger o resto do sistema. Esta separação entre a placa principal e uma placa secundária, para além de facilitar a concepção do projecto, permite que no caso de avaria devido a tensões ou correntes elevadas nas entradas, a reparação seja mais simples.

Em relação à alimentação, o sistema VideoVigi necessita de 3 tensões de alimentação. A tensão de 5V que é obtida através de um regulador linear (LM7805) é utilizada pelo DSP, pela sua SRAM e flash. Os restantes componentes utilizam a tensão de 3,3V. A tensão 1,8V.é utilizada no core do microcontrolador e *video decoder*. Como uma grande parte do sistema é alimentado a 3,3V, para evitar perdas de energia e aquecimento excessivo, a tensão de 3V3 é produzida através de um *step-down* a partir da tensão de 5V, para isso utiliza o controlador *Pulse Width Modulation* (PWM-MAX1692). A tensão de 1,8V é pelo

contrário, pouco utilizada sendo neste caso produzida através de um regulador linear ajustado (LM337).

3. COMPRESSÃO

A compressão de dados é um processo que recorre essencialmente à eliminação de informação redundante. Em sistema de videovigilância, os dados transmitidos são predominantemente as imagens captadas pelas câmaras de vídeo. A compressão de sinais de vídeo pode atingir valores elevados devida à forte correlação (informação redundante) entre imagens sucessivas e entre píxeis da mesma imagem. Em sinais de videovigilância há ainda a considerar uma característica muito própria que deve ser equacionada. As imagens recebidas, mesmo em câmaras móveis, apresentam-se durante uma grande parte do tempo em posições fixas. Esta qualidade faz da videovigilância um caso especial no que concerne a compressão de vídeo. Para além disso existem algumas considerações que devem ser tomadas de acordo com a aplicação que se pretende, como por exemplo:

- Qual o formato de vídeo pretendido?
- Pretende-se vídeo fluido ou apenas sequência de imagens?
- Qual a degradação máxima admitida?
- Qual o débito da rede disponível?
- Qual o orçamento disponível?

Embora idealmente para a aplicação pretendida fosse fácil responder a estas perguntas, na prática a resposta às três primeiras é limitada pelas duas últimas, i.e. tudo depende do débito da rede e do orçamento disponível. Entenda-se que o orçamento irá determinar características ao nível do *hardware* do sistema que limitarão a capacidade de compressão.

O sistema VideoVigi desenvolvido, assim com o estudo desta tese, está vocacionado para redes de baixo débito e sistemas de custo reduzido. Estas duas razões, de alguma forma contraditórias irão determinar um compromisso entre os objectivos da videovigilância pretendida e as limitações impostas.

3.1. FORMATOS VÍDEO

À semelhança da maioria dos sistemas de videovigilância, o sistema VideoVigi desenvolvido, dispõe de interfaces analógicas (vídeo composto) para ligação das câmaras de vídeo. Nestes sistemas, o sinal de vídeo analógico é obtido por uma sequência de imagens, sendo cada uma destas, adquirida linha à linha por varrimento horizontal e vertical, do canto superior esquerdo para o canto inferior direito. Cada varrimento completo designa-se por *field*; uma imagem completa gera uma *frame*. Uma *progressive frame* é uma *frame* obtida a partir de um único varrimento da imagem, por outro lado se a imagem for obtida por dois varrimentos de linhas entrelaçados (*field*) estamos perante uma *interlaced frame*. Estas duas formas de varrimento são apresentadas na Figura 6.

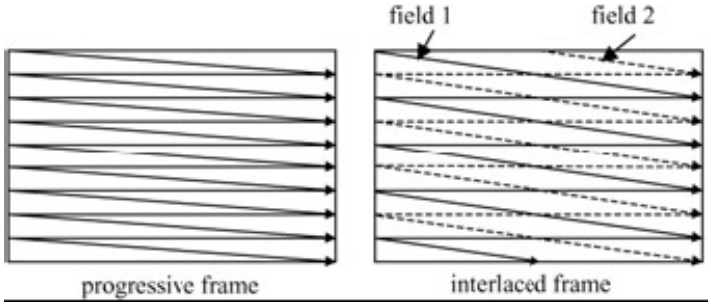


Figura 6 Interlaçamento [3]

As "imagens" no sistema PAL são formadas por *field* pares e impares interlaçados, i.e. *interlaced frame*. O *field* par é constituído por 312 linhas e o quadro ímpar por 313 (Figura 7). No entanto, do conjunto destas linhas, apenas 576 representam linhas visíveis as restantes são usadas para serviços como o teletexto.

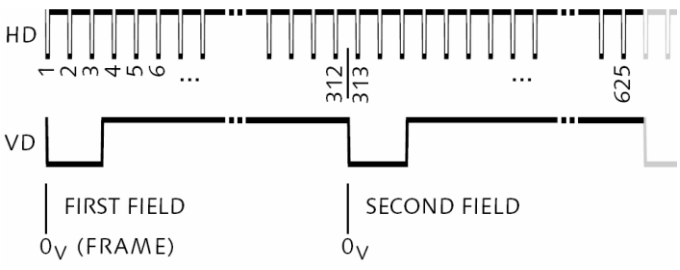


Figura 7 Sistema PAL [2]

A largura de faixa máxima do sistema PAL é 5,5 MHz, pelo teorema de Nyquist a frequência mínima de amostragem para à digitalização deveria ser 11 MHz. No entanto, o International Telecommunication Union na recomendação CCIR-601[2] aconselha a utilização de uma frequência de amostragem de 13,5 MHz. e define 720 píxeis por cada linha.

Considerando cada píxel representado por apenas oito bits (sistema monocromático), no formato CCIR-601 a rede necessária para transmitir vídeo sem compressão é 82.944.000 bps (576x720x25x8). Este valor é tão elevado que mesmo para uma compressão de 99% seria insuficiente para transmitir o vídeo numa rede de baixo débito. O formato de imagem *Common Intermediate Format* (CIF-352x288) que corresponde a cerca de um quarto da resolução anterior, é o mais aconselhável para redes de baixo débito. Esta obtêm-se considerando metade dos píxeis por linha e em apenas um *field*. Caso a rede seja de muito baixo débito, como RDIS pode ser interessante o formato *Quarter-CIF* (QCIF-176x144). Para sistemas policromáticos existem basicamente dois formatos que são o YUV ou YCbCr e RGB. No formato YCbCr, o "Y" representa a luminância, "Cb" a componente azul e "Cr" a componente vermelha da crominância. O formato RGB é formado pelas três cores vermelho, verde e azul que embora não sejam as três primárias permitem definir um triangulo cromático a partir do qual se reproduzem todas as outras cores. Enquanto o primeiro formato é usado no sistema PAL, o segundo é usado em monitores informáticos.

3.2. COMPRESSÃO VÍDEO

Num sistema de dados, a compressão deve ser feita de forma a garantir a recuperação da informação sem perdas. No entanto na compressão de vídeo, como é desenvolvida para ser observada por humanos, pode aceitar-se alguma perda de informação sem que isso comprometa significativamente as imagens recebidas. De facto a percepção humana é pouco sensível aos contornos de alta frequência contidos nas imagens. Esta informação pode então ser considerada supérflua ou visualmente redundante. A compressão de vídeo

baseia-se essencialmente na eliminação de informação redundante, que existe sob três formas fundamentais:

- Redundância espacial – existente entre píxeis adjacentes na mesma imagem.
- Redundância temporal – existente entre os mesmos píxeis de imagens sucessivas.
- Redundância de símbolos – existente por repetição dos mesmos símbolos na imagem.

Os dois *standards* de compressão mais conhecidos e difundidos são o JPEG e o MPEG. A compressão de vídeo pode ser efectuada *frame* a *frame* utilizando o standard JPEG, conhecida, neste caso, por *Motion JPEG* (MJPEG[2]). Este tipo de compressão, resolve a redundância espacial, i.e. a compressão é efectuada em cada *frame* isoladamente. A compressão MPEG para além de resolver a redundância espacial, também resolve a redundância temporal, i.e. a relação existente entre os mesmos píxeis de *frames* sucessivas. Em cada uma destas normas é ainda possível comprimir os dados obtidos usando um algoritmo *variable length coding* (VLC) como o *huffman* ou o algoritmo de codificação aritmética, ambos resolvem a redundância de símbolos. Estes dois algoritmos são conhecidos por *entropy coding*.

A compressão baseada na redundância espacial, como acontece no JPEG e no MPEG, é particularmente eficiente em imagens reais, imagem com transições suaves, e.g. obtidas a partir de uma câmara; uma vez que esta compressão se baseia na eliminação das componentes de alta frequência.

A compressão baseada na redundância temporal existente no MPEG, embora seja muito eficiente e conduza a compressões muito elevadas, pode ser inviável em sistemas de baixo custo, uma vez que o seu complexo algoritmo exige um processamento elevado assim como a capacidade de memória. Para além disso, em determinadas aplicações de videovigilância o número de imagens por segundo pode ser relativamente baixo, logo a relação entre *frames* pode não ser muito significativa no objecto em movimento na imagem, mas existirá sempre relação no cenário que se encontra imóvel.

[2] A compresão MJPEG (Motin JPEG), utiliza compressão JPEG imagem a imagem.

A compressão baseada na redundância de símbolos é eficiente em ambos *standards* uma vez que os seus algoritmos de compressão conduzem a valores numéricos baixos com elevadas probabilidades de repetição, como será visto posteriormente, e esta compressão é tanto mais elevada quanto maior for a repetição de símbolos.

3.3. MJPEG

A compressão no formato JPEG é definida na norma do *International Telecommunication Union* (ITU), recomendação T.81[14]. Esta é a compressão utilizada em cada uma das imagens disponíveis no MJPEG (Figura 8). A compressão de imagens a cores oferece melhores resultados em códigos de cores YUV ou YCbCr do que RGB. Isto porque estes códigos têm a luminância separada da crominância o que permite o favorecimento da luminância (compressão menor) em detrimento da crominância, uma vez que o olho humano é mais sensível à luminância do que à crominância[4]. Nestes casos as componentes de crominância podem ser sub-amostradas numa relação até 2:1, horizontal e verticalmente, obtendo-se componentes U e V com um quarto da dimensão da componente Y (luminância). A compressão de imagens a preto e branco corresponde à compressão da luminância numa imagem a cores.

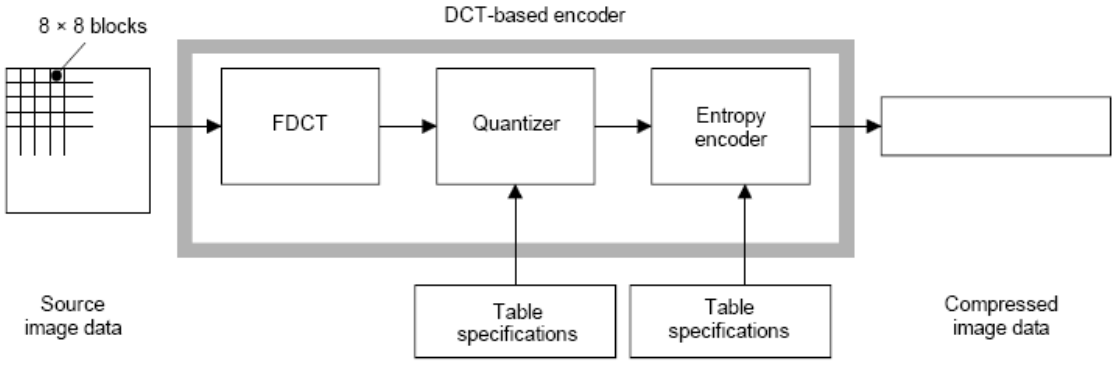


Figura 8 Algoritmo (M)JPEG [14]

No algoritmo MJPEG cada imagem é dividida em blocos de 8x8 píxeis, os coeficientes, que representam o valor de cada píxel, são transformados em valores com sinal através de uma subtracção de 128 ou 2048 (conforme os coeficientes sejam de 8 ou 12 bits). Seguidamente a imagem, ainda espacial, é transformada nas suas componentes espectrais através da *Discrete Cosine Transform* (DCT) de acordo com a primeira expressão da Figura 9. A DCT é uma variante da transformada de Fourier discreta, que permite passar a imagem do domínio do espaço para o domínio das frequências.

FDCT: $$S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

IDCT: $$s_{yx} = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S_{vu} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16}$$

$C_u, C_v = 1/\sqrt{2}$ for $u, v = 0$

$C_u, C_v = 1$ otherwise

Figura 9 FDCT e IDCT [14]

Na expressão "S_{yx}" corresponde ao píxel da matriz bloco corrente na posição yx (linha coluna). "S_{vu}" representa o coeficiente da matriz bloco no domínio das frequências na posição "vu" (linha coluna).

Assim obtém-se uma matriz que contém as frequências da imagem, o primeiro coeficiente corresponde à componente DC; os coeficientes seguintes correspondem à componente AC. As componentes de alta frequência são então eliminadas através de um quociente, elemento a elemento, com uma matriz de quantificação, esta possui coeficientes progressivos de forma a actuar como um filtro, é aqui que ocorre a compressão baseada na eliminação das componentes de alta frequência de amplitude reduzida. A matriz de quantificação é uma matriz definida na recomendação T.81[14] que permite obter resultados de compressão elevados e quase imperceptíveis na imagem (tabela seguinte).

Tabela 1 Matriz de quantificação [14]

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Por fim, a matriz resultante é ordenada em ziguezague (Figura 10) e são retirados os últimos zeros através da colocação de um carácter de fim de matriz. A ordenação em ziguezague permite juntar o maior número de zeros para o fim. Ainda é possível comprimir

os dados referentes a cada valor usando códigos de comprimento variável como o *huffman* que será descrito posteriormente.

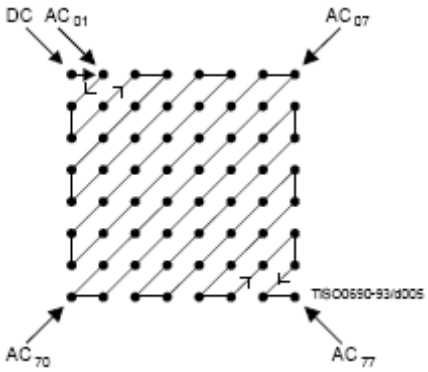


Figura 10 Ziguezague [14]

Os recentes desenvolvimentos do *standard* de compressão JPEG conduziram ao aparecimento do JPEG2000 e consequente ao equivalente MJPEG2000 no que se refere a sequência de imagens. O JPEG2000 faz uso da transformada de *wavelet*. Para comparar os dois formatos foram realizadas compressões através de uma imagem obtida directamente a partir do sistema VideoVigi. A imagem foi comprimida por software em JPEG e JPEG2000 utilizando uma taxa de compressão de aproximadamente 97% (Figura 11). Analisando as duas imagens nota-se que a principal vantagem da compressão JPEG2000 é o desaparecimento do "efeito de bloco" presente na compressão JPEG. Para além disso existe um favorecimento na qualidade da imagem JPEG2000 em relação à imagem JPEG, que só desaparece com uma diminuição na compressão em cerca de 25 % na compressão JPEG. No entanto a complexidade e o correspondente atraso de compressão, comparado com a melhoria de 25 % torna a compressão MJPEG2000 inviável em sistemas de baixo custo.

Figura 11 Lena – JPEG e JPEG2000 respectivamente (compressão 97%)

3.4. MPEG

Os primeiros *standards* desenvolvidos foram o MPEG-1, o MPEG-2 e o MPEG-4. Mais tarde foram desenvolvidos os *standards* de nova geração, i.e. o MPEG-7 e MPEG-21. O MPEG-1, inicialmente desenvolvido para suporte de vídeo em *Compact Disk* (CD), i.e. *videoCD,* está orientado para redes de 1,5 Mbps. O formato de vídeo utilizado é o CIF. O MPEG-2 foi projectado para oferecer uma resolução superior ao MPEG-1 CCIR-601 (720x576) e é utilizado em DVD. Consequentemente necessita de uma largura de faixa também superior. O MPEG-4 foi pensado para levar o vídeo a novas plataformas como telemóveis ou PDA's. Também utiliza o formato CCIR-601. As recomendações H.261 e H263 do ITU-T baseiam-se ambas no *standard* MPEG; correspondem a versões simplificadas deste *standard* orientadas para redes de baixo débito como acesso básicos e primários da RDIS no caso da recomendação H.261 e acessos de débito variável no caso da recomendação H.263. Apesar das recomendações H.261 e H263 não trazerem qualquer vantagem em termos de compressão, elas enquadram-se no âmbito desta tese, i.e. redes de baixo débito e sistemas de custo reduzido.

Na compressão MPEG são definidos três tipos diferentes de *frames*: as *I-frames,* as *P-frames* e as *B-Frames*. As *I-frames* são *frames* que não dependem de outras *frames,* estas resultam da compressão de uma imagem completa usando o algoritmo JPEG e servem de referência para a reconstrução de outras *frames*. As *P-frames* são *frames* reconstruídas a partir de uma *frame* anterior e também servem de referência para a reconstrução de outras *frames.* As *B-frames* resultam da interpolação das *frames* anteriores e posteriores. O conjunto de imagens criadas entre duas *P-frames* consecutivas chama-se *group of picture* (GOP) como se pode ver na figura seguinte.

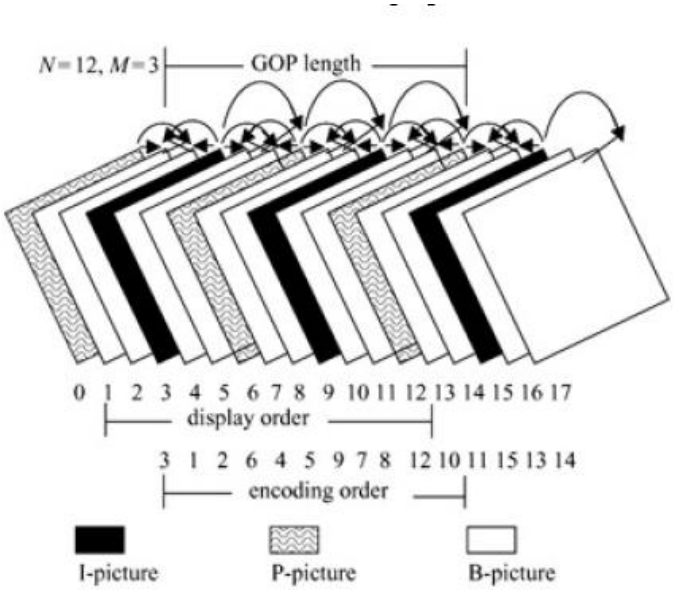
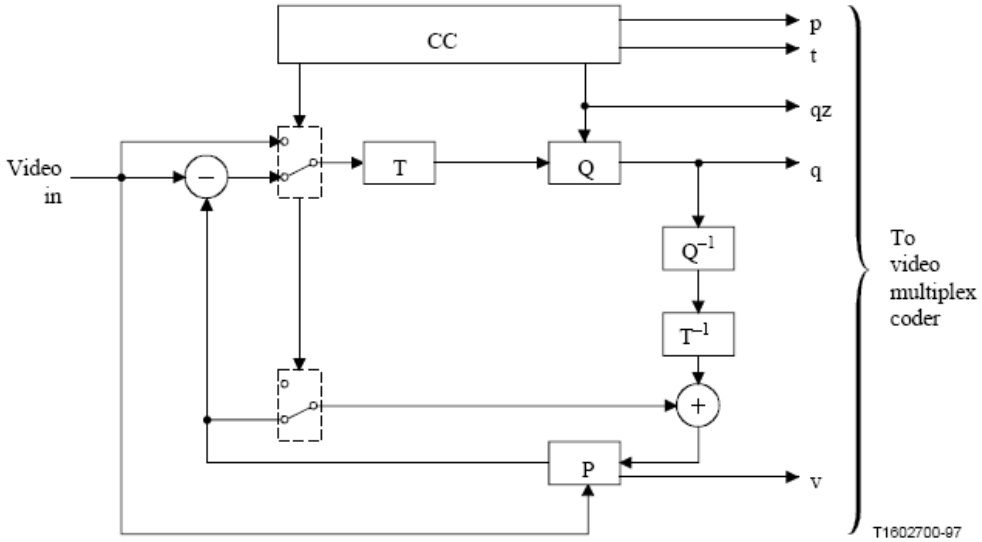


Figura 12 MPEG1 – GOP [3]

A principal diferença entre os algoritmos H.261/3 e o MPEG é o facto das *frames* reconstruídas serem-no sempre a partir de *frames* anteriores ou seja não existe o conceito de *B-frames.* No entanto, a recomendação H.263 prevê o modo de funcionamento *PB-frames* definido no anexo G. No algoritmo H.263, a imagem que corresponde a uma *P-frames* é dividida em blocos de 16 x 16 píxeis conhecidos por *macroblock*. A compressão resulta da informação obtida pela subtracção de cada *macroblock* da *frame* actual ao da *frame* anterior assim como a determinação do seu respectivo vector de movimento; em relação às *I-frames* a compressão é análoga ao MJPEG. A compressão H.263 é fruto do resultado obtido pela sequência de *P-frames* e *I-frames*.

O primeiro passo da compressão H.263 (Figura 13), conhecido por *Motion Estimation* (ME), consiste em estimar o movimento dentro de cada *macroblock* da *frame* anterior em relação à *frame* actual. A ME é efectuada comparando cada *macroblock* com o seu correspondente da *frame* anterior de forma a encontrar um vector de movimento, i.e. um valor que indica uma deslocação dos píxeis dentro dentro do *macroblock*. Se for encontrado um vector, a área dentro do respectivo *macroblock* sofre um movimento, dando origem a um *macroblock motion compensated* e consequentemente no final uma imagem *motion compensated*. As diferenças encontradas entre cada *motion compensated macroblock* e o actual *macroblock* são alvo de um processo semelhante à compressão JPEG, i.e. ao resultado da diferença é efectuada uma DCT e uma operação de quantificação. A matriz de quantificação é ajustada de acordo com a qualidade de imagem

e compressão desejada. Os dados comprimidos de cada *macroblok* assim como o respectivo vector de movimento sofrem ainda uma nova compressão baseada em códigos de comprimento variável como o *huffman*, antes de serem enviados pela rede. Para a reconstrução da próxima imagem são realizadas as operações inversas às diferenças (IDCT e Q^{-1}) e somadas à imagem anterior. É a partir desta imagem reconstruída que é realizado o *motion estimation.* Desta forma garante-se que a imagem do codificador e do descodificador na qual são efectuadas as operações são iguais evitando-se que ocorra acumulação de erro.



T	Transform
Q	Quantizer
P	Picture Memory with motion compensated variable delay
CC	Coding control
p	Flag for INTRA/INTER
t	Flag for transmitted or not
qz	Quantizer indication
q	Quantizing index for transform coefficients
v	Motion vector

Figura 13 Algoritmo H.263 [16]

3.5. CODIFICAÇÃO SÍMBOLOS SEM PERDAS

Para resolver a redundância de símbolos existem duas alternativas "vulgares", a codificação de *Huffman* e a codificação aritméticas, que são *variable length coding* (VLC). Estes códigos baseiam-se na probabilidade de ocorrência dos símbolos. Os símbolos com maior probabilidade de ocorrência são convertidos em valores binários de menor comprimento e consequentemente os símbolos de menor probabilidade são convertidos em valores binários de maior comprimento. Como na compressão JPEG que é utilizada em

MJPEG e MPEG existe uma elevada probabilidade de ocorrência de determinados valores (coeficientes baixos), estes algoritmos são particularmente eficientes. No caso particular dos elementos zero na componente AC, eles são contabilizados e associados directamente ao próximo elemento diferente de zero. O algoritmo de codificação aritmética embora seja ligeiramente mais eficiente (5 a 10%), envolve cálculos mais complexos que tornam a compressão mais lenta; por esta razão é preterido pelo algoritmo H*uffman* [4].

No algoritmo *Huffman* cada símbolo é convertido noutro de comprimento diferente de acordo com a sua probabilidade de ocorrência, i.e. os de maior probabilidade de ocorrência são convertidos em símbolos de comprimento menor e vice-versa. Para não ocorrer ambiguidades, cada símbolo tem de ser diferente do início dos outos símbolos (mais longos). Com excepção da componente DC, cada valor, para além da sua representação, é precedido de um código que representa o número de zeros que o antecede. Para as componentes DC e AC, a recomendação T.81 [14] do ITU define duas tabelas de valores (*Difference value*) onde cada valor é representado por um par de números, o primeiro (ainda em decimal) indexa a linha (SSSS na Figura 14) e o segundo a posição (para a respectiva linha). O número de bits para representar os valores em cada linha é crescente de zero até doze. Para a componente AC, a codificação de cada número é efectuado por mais um valor, o primeiro representa o número de zeros que o antecede, o segundo a linha onde se encontra e o terceiro a posição na linha. A posição na linha é representada em binário por ordem, o número de bits varia de acordo com a linha (Figura 14), i.e. o número de bits para representar a posição é igual ao da própria linha (valor SSSS).

SSSS	DIFF values
0	0
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023
11	–2 047..–1 024,1 024..2 047

SSSS	AC coefficients
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023

Figura 14 ***Difference magnitude categories for DC/ AC Coding*** **[14]**

Para a componente AC, os dois primeiros números formam um par (*run/size*) que são substituídos directamente a partir de outras tabelas definida na recomendação T.81. A Figura 16 representa apenas os primeiros 40 valores (de 162) da tabela para a componente AC da luminância. Nesta tabela o *run* representa o número de zeros e o *size* a linha. Em relação à componente DC, como esta não é precedida por nenhum zero, a tabela apresenta menos valores. A Figura 15 representa os valores para a componente DC da luminância.

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Figura 15 ***Luminance DC Coefficient Difference*** **[14]**

Exemplificando o algoritmo com uma matriz real quantificada, obtida a partir do sistema VideoVigi (o último *byte* é o *End Of Block* (EOB):

6,12,3,0,-4,-9,3,2,0,-1,0,0,0,-1,-1,EOB.

O primeiro passo consiste em agrupar o número de zeros que precedem cada valor da componente AC, fica:

(6);(0,12);(0,3);(1,-4);(0,-9);(0,3);(0,2);(1,-1);(3,-1);0,-1);(0,0).

De seguida cada valor é substituído pela posição na tabela da Figura 15 (linha e posição) em que o último é já representado directamente em binário, fica:

(3,110);(0,4,1100);(0,2,11);(1,3,011);(0,4,0110);(0,2,11);(0,2,10);(1,1,0);(3,1,0);(0,1,0);(0,0);

O último passo consiste em substituir o primeiro par pelo correspondente na tabela da Figura 15 para o coeficiente DC e da Figura 16 para os coeficientes AC, fica então:

100110;10 111100;01 11;111100 1011;1011 0110;0111; 0110;1100 0;1110100; 000;1010
(9A F1 FC BB 67 6C 74 14)

O que prefaz um número total de 63 bits. A informação inicial tinha um comprimento de 16 Bytes, ou seja 128 bits, logo neste exemplo houve uma compressão ligeiramente superior a 50 %. Esta é, de facto, a compressão média atingida por este algoritmo. Apesar de ser relativamente baixa, tem a grande vantagem de ser isenta de perdas.

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110
3/1	6	111010
3/2	9	111110111
3/3	12	111111110101
3/4	16	1111111110001111
3/5	16	1111111110010000
3/6	16	1111111110010001
3/7	16	1111111110010010
3/8	16	1111111110010011
3/9	16	1111111110010100
3/A	16	1111111110010101

Figura 16 ***Luminance AC Coefficient*** **[14]**

4. CRIPTOGRAFIA

A criptografia ou encriptação é um processo necessário para proteger os dados usados numa comunicação entre dois pontos de uma determinada rede ou sistema vulnerável. A segurança na comunicação é garantida cifrando os dados recorrendo a algoritmos matemáticos que usam uma "chave"; no sistema remoto os dados originais são recuperados por um processo inverso. Existem duas formas de criptografia, a simétrica ou de chave secreta, se os dados forem cifrados e decifrados pela mesma chave, e a assimétrica ou de chave pública, quando são realizados por chaves diferentes, i.e. chave pública e privada respectivamente.

Na criptografia assimétrica os dados cifrados por uma determinada chave pública não podem ser recuperados por esta mesma chave, i.e. os dados recebidos só podem ser decifrados com a chave privada o que significa que a chave pública pode ser conhecida sem que isso represente qualquer píxel de falha de segurança. A Figura 17 ilustra este conceito. Como em certos serviços o servidor precisa de enviar a chave ao cliente, na criptografia simétrica a chave poderia ser "descoberta" o que poria em causa o nível de segurança desejado. Esta é a razão pela qual a criptografia simétrica é preterida pela assimétrica em transacções comerciais.

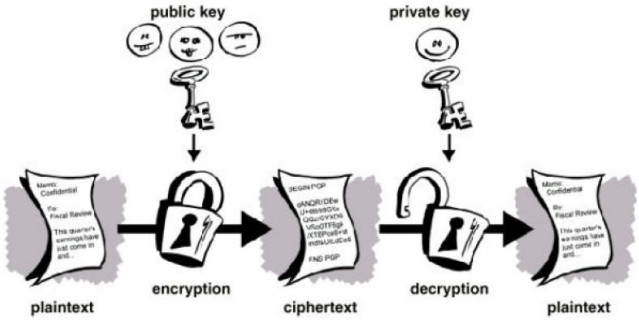


Figura 17 Criptografia assimétrica [10]

A criptografia simétrica (Figura 18) é no entanto válida em sistemas que conheçam, *a priori*, a mesma chave. A video vigilância entendida como uma comunicação entre um cliente "conhecido" por uma aplicação servidora enquadra-se perfeitamente neste requisito.

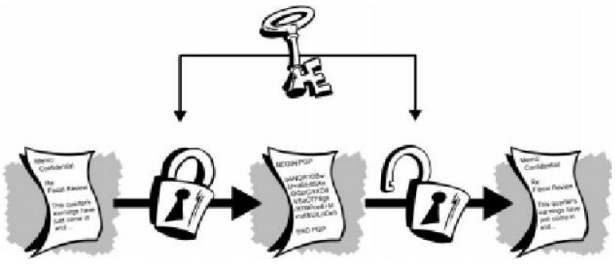


Figura 18 Criptografia simétrica [10]

4.1. DATA ENCRYPTION STANDARD (DES)

O *Data Encryption Standard* (DES) é um standard de criptografia simétrica baseado numa chave de 64 bits. Dos 64 bits da chave, 56 bits são usados pelo algoritmo para cifrar os dados (Figura 19). Os restantes 8 bits são de paridade, i.e. cada grupo de 8 bits da chave usa um bit de paridade impar (bits de paridade: 8, 16,24,..,64).

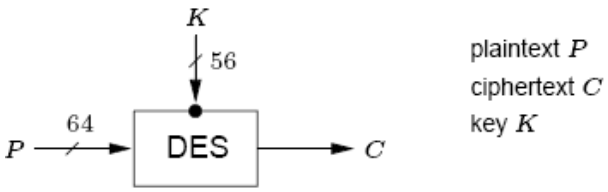


Figura 19 DES [7]

Os dados cifrados só podem ser recuperados a partir da primeira chave usada. A segurança dos dados cifrados depende directamente do conhecimento dessa chave, havendo desconhecimento da chave não é possível recuperar os dados originais. No entanto existem

formas para tentar descobrir a chave como o ataque "força bruta" em que são testada, sequencialmente, as várias combinações possíveis da chave. Os 56 bits utilizados ao cifrar os dados garantem cerca de $7x10^{16}$ possibilidades. Apesar do elevado número de combinações, actualmente a capacidade de processamento das máquinas viabiliza o descobrimento da chave ao fim de algum tempo. Esta é a principal "lacuna" do DES. Uma das formas para aumentar a segurança é utilizar o *triple* DES (TDES), ou TDEA no que se refere ao algoritmo, que utiliza sequencialmente três chaves. A outra forma é alterar periodicamente a chave (quanto menor for o período maior será a segurança).

A função de *Feitsel* [8] é um algoritmo iterativo desenvolvido por *Horst Feistel* para cifrar dados no qual o DES se baseia (Figura 20)

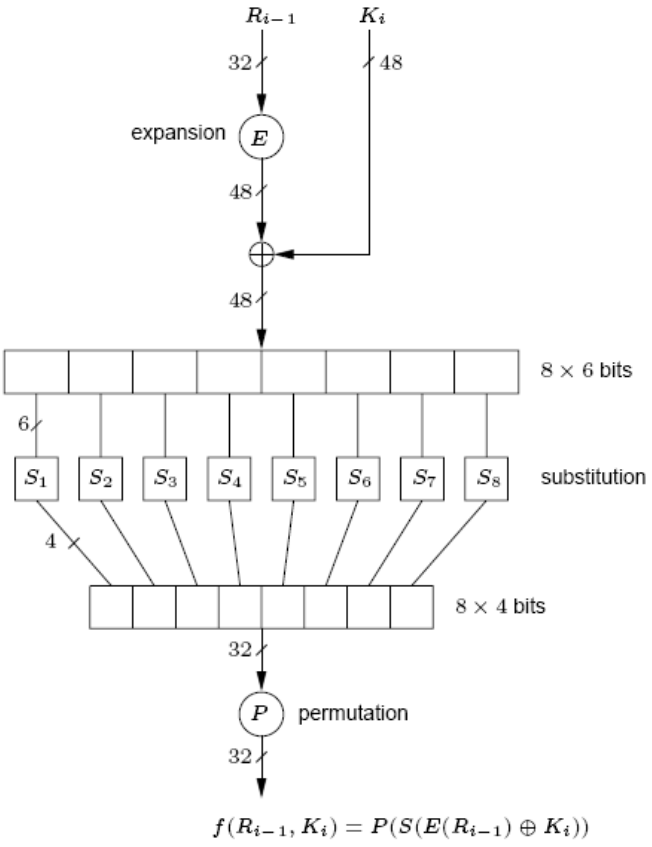


Figura 20 Função *f* [7]

Este foi desenhado para cifrar e decifrar os dados, em blocos de 64 bits. Em cada bloco (ao ser cifrado) realiza-se uma permutação inicial, sendo depois a informação sujeita a operações complexas, que dependem da chave, orientadas ao bit e finalmente uma permutação inversa da inicial. Esta sequência está ilustrada na Figura 21. Para decifrar são realizadas operações inversas semelhantes.

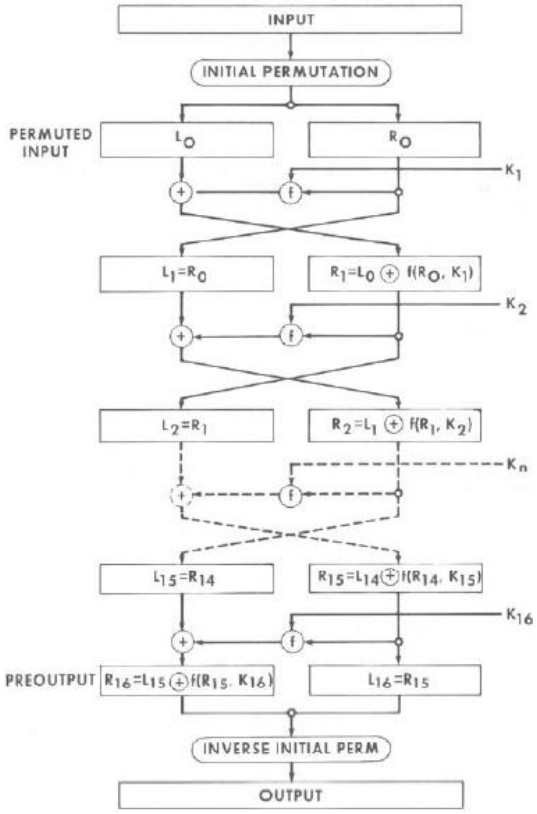


Figura 21 Algoritmo DES [7]

Considerando o texto a cifrar $P=m_1... m_{64}$ e a chave $K=k_1... k_{64,}$ o algoritmo DES é realizado nos seguintes passos:

1. Criação de sub-chaves: a partir da chave, usando os 56 bits de cifra (os bits 8, 16, 24,.., 64 são de paridade) são criadas dezasseis sub-chaves de 48 bits que irão ser utilizadas individualmente.

   1.1. Definir o vector $v_i=\{1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1\}$.

   1.2. No passo seguinte os 28 primeiros bits da chave são separados dos restantes 28 bits em $C_0=k_{57}k_{49}... k_{36}$ e $D_0=k_{63}k_{55}... k_4$ de acordo com a tabela PC1 (Tabela 2).

   1.3. Para i=1 até 16, é realizada uma rotação circular esquerda em C_0 e D_0 com o respectivo valor de bits de v_i, i.e. $C_i \leftarrow C_{i-1} << v_i$ e $D_i \leftarrow D_{i-1} << v_i$. Para cada valor i é obtida uma sub-chave $K_i= k_{14}k_{17}, ..., k_{32}$ que resulta da concatenação dos valores (C_i,D_i) de acordo com a tabela PC2 (Tabela 2).

Tabela 2 PC1 e PC2 [10]

PC1						
57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
above for C_i; below for D_i						
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2					
14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

2. Permutação: neste passo os bits a cifrar são reordenados de acordo com a Tabela 3 (IP). Os 32 primeiros bits são separados dos restantes 32 bits em $L_0=m_{58}m_{50}... m_8$ e $R_0=m_{57}m_{49}... m_7$.

Tabela 3 Permutação IP e Permutação Inversa IP^{-1} [10]

IP							
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

IP^{-1}							
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

3. Em seguida são aplicadas recursivamente (para i=1 até 16) as seguintes fórmulas baseadas num XOR (Figura 21):

$L_i = R_{i-1}$

$R_i = L_{i-1} \oplus f(R_{i-1},K_i)$, onde $f(R_{i-1},K_i) = P(S(E(R_{i-1}) \oplus K_i))$

Onde E é uma expansão fixa de R_{i-1} de 32 para 48 bits (alguns bits são usados duas vezes). De acordo com a tabela E (Tabela 4), e.g. o bit 32 é repetido na primeira posição e na penúltima (ver tabela E).

P é uma permutação ($P=p_{16},p_7, ..., p_{25}$) de acordo com a tabela P.

Tabela 4 Expansão e Permutação [10]

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

S é uma substituição de cada 6 bits (resultante da divisão dos 48 bits por 8) em 4 bits de acordo com a Tabela 5. Esta substituição é uma transformação não linear que aumenta o nível de segurança.

Tabela 5 DES *S-Box* [10]

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
S_1																
[0]	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
[1]	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
[2]	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
[3]	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2																
[0]	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
[1]	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
[2]	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
[3]	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3																
[0]	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
[1]	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
[2]	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
[3]	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4																
[0]	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
[1]	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
[2]	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
[3]	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5																
[0]	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
[1]	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
[2]	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
[3]	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6																
[0]	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
[1]	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
[2]	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
[3]	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7																
[0]	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
[1]	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
[2]	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
[3]	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8																
[0]	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
[1]	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
[2]	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
[3]	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

4. Permutação inversa: neste último passo os bits cifrados são reordenados de acordo com a Tabela 3 (IP^{-1}). $C_0=c_{40}c_8... c_{25}$.

4.2. ADVANCED ENCRYPTION STANDARD (AES)

O *Advanced Encryption Standard* (AES) também é um standard de criptografia simétrica mas, ao contrário do DES, cifra blocos de 128 bits de cada vez. O tamanho da chave pode ser 128, 192 ou 256 bits (de acordo com o nível de segurança pretendido), dando origem ao AES-128, AES-192 ou AES-256 respectivamente. A chave do AES baseada em 128 bits (pior caso) oferece uma segurança incomparavelmente superior ao DES ($3{,}4x10^{38}$ versus $7x10^{16}$ combinações de chave possiveis). O algoritmo orientado ao byte torna-se mais simples e tem tempo de encriptação inferior. Estas são as razões que levaram o DES a ser preterido pelo AES. Como em videovigilância os dados a decifrar são blocos comprimidos que, para além de necessitarem descompressão, só fazem sentido quando observados por humanos, qualquer forma de tentativa de descobrimento da chave como o ataque "força bruta", se torna muito lenta, logo o algoritmo AES-128 é claramente suficiente.

O algoritmo AES (baseada no algoritmo de *Rinjdael*) é aplicado a uma matriz de 4x4 bytes (blocos de 128 bits), logo considerando os valores em bytes, o número de colunas nos estados (matriz com os valores a cifrar) designada por *Nb* é quatro. Para uma chave de comprimento 128 bits, o *Nk* (número de colunas na matriz *Cipher key*) é também igual a quatro (4 bytes). O número de ciclos efectuados durante a execução do algoritmo (representados por *Nr*) depende directamente do tamanho da chave, para *Nk*=4 (128 bits) o número de ciclos é igual a dez.

Para cifrar, o algoritmo AES utiliza quatro transformações orientadas ao byte:

1. *SubBytes*: Faz substituições de bytes usando a tabela AES *S-Box* (Tabela 6). Trata-se de uma transformação não linear.

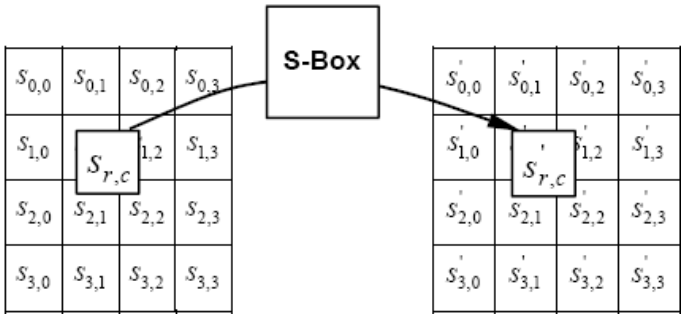


Figura 22 *SubByte2( )*[9]

Cada byte é substituído directamente da Tabela 6 desde 0x00 até 0xff conforme o seu valor.

Tabela 6 AES *S-Box* [9]

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

2. *ShiftRows*: Executa *shift* circulares de bytes para a esquerda na matriz de estado, variando em função da linha de 0 a 3 (Figura 23).

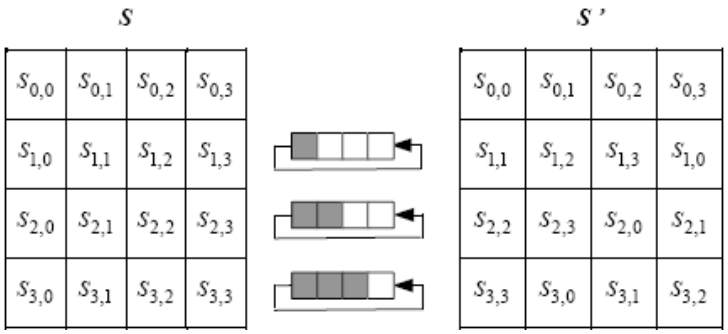


Figura 23 *ShiftRows()*[9]

3. *MixColumns*: Faz a multiplicação de cada coluna da matriz de estado pela matriz quadrada definida na Figura 24, em que a soma é convertida na operação XOR ao Byte

de cada matriz coluna do estado (c=0 até *Nb*) pela matriz quadrada de acordo com a figura seguinte.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \Leftrightarrow \begin{cases} s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}). \end{cases}$$

Figura 24 *MixColunns()*[9]

4. *AddRoundKey*: Realiza a soma (XOR) coluna à coluna entre a matriz de estado e a matriz *Round Key* (resulta da expansão da chave). Esta é a única transformação que envolve a chave de encriptação.

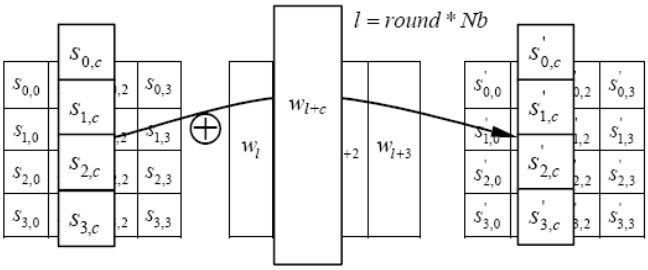


Figura 25 *AddRoundKey()*[9]

Para cada bloco de 128 bits, estas quatro funções são utilizadas de forma recursiva de acordo com o pseudocódigo disponível na publicação 197 do próprio standard AES [9].

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figura 26 Pseudocódigo *Cipher* [9]

A obtenção da expansão das chaves (através do procedimento *key expansion*) permite obter um total de *Nb* *(*Nr +1*) *bytes*, i.e. 11*4 *bytes* para o AES-128. Este é o primeiro passo na criptografia, uma vez que a chave expandida é utilizada na transformação *AddRoundKey*.

O procedimento *key expansion* está disponível através do pseudo código da figura seguinte. Apesar da aparente complexidade do seu algoritmo, a expansão da chave é obtida atravês de um algoritmo parecido com o da própria encriptação, i.e. a expansão recorre a transformações semelhantes a *ShiftRows* (primeiro ciclo *while*), *SubByte* (primeira condição *if* do segundo ciclo *while*).e *AddRoundKey* (segunda condição *if* do segundo ciclo *while*).

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end

Note that Nk=4, 6, and 8 do not all have to be implemented;
they are all included in the conditional statement above for
conciseness.   Specific implementation requirements for the
Cipher Key are presented in Sec. 6.1.
```

Figura 27 Pseudocódigo *KeyExpansion* [9]

Para decifrar o algoritmo AES utiliza quatro transformações que resultam da inversão das primeiras e são utilizadas em ordem inversa, de acordo o pseudocódigo da figura seguinte.

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                    // See Sec. 5.3.1
      InvSubBytes(state)                     // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                   // See Sec. 5.3.3
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Figura 28 Pseudocódigo *InvCipher* [9]

A única transformação que é utilizada para cifrar e decifrar é a *AddRoundKey*, todas as outras sofreram alteração:

1. *InvShiftRows*: Executa *shift* circulares de bytes na matriz de estado para a direita (ao contrário do *ShiftRows*) variando em função da linha de 0 a 3 (ver figura seguinte).

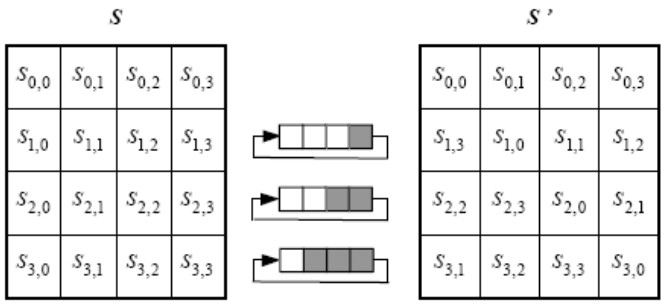


Figura 29 *InvShiftRows* [9]

2. *InvSubsBytes*: Faz substituições de bytes usando a tabela AES *InvS-Box.* Trata-se de uma transformação não linear que irá retornar o valor inicial (antes da transformação *SubsBytes*). Cada elemento da matriz estado é substituido por um elemento da tabela seguinte, em função do seu valor.

Tabela 7 AES *InvS-Box* [9]

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

3. *InvMixColumns*: Faz a multiplicação de cada coluna da matriz estado pela matriz da figura seguinte, em que a soma é convertida na operação XOR ao Byte de cada matriz coluna do estado (c=0 até *Nb*), pela seguinte matriz quadrada de acordo com a figura.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \Leftrightarrow \begin{cases} s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c}) \\ s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c}) \\ s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c}) \\ s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c}) \end{cases}$$

Figura 30 *InvMixColumns [9]*

5. FIRMWARE COMPRESSÃO SISTEMA VIDEOVIGI

O sistema VideoVigi, desenvolvido no Projecto de Curso da Licenciatura de Engenharia Electrotécnica – Electrónica e Computadores, é uma solução de videovigilância que permite digitalizar e comprimir sinais de vídeo de quatro câmaras analógicas e transmiti-las por uma rede IP. Como já foi referido este sistema de custo reduzido foi pensado e desenvolvido inicialmente para enviar vídeo comprimido MJPEG no formato CIF. A compressão é efectuada pelo *Digital Signal Processor* (DSP) ADSP2181 KS-133 da *Analog Device*® e dispõe de uma SRAM de 128KB.

O DSP executa instruções ao dobro da velocidade do seu "relógio" externo, i.e. 33 *Millions of Instructions Per Second* (MIPS). Apesar do DSP permitir, em algumas situações, executar múltiplas instruções em simultâneo (três no máximo), esta *performance*, foi escolhida para satisfazer as necessidades da compressão MJPEG, e não da compressão H.263.

Em relação à memória o problema é semelhante. O DSP para além de quatro segmentos de 8kB de memória interna (*data memory* e *program memory*), dispõe apenas de 16 segmentos externos de 8kB, i.e. 128kB de SRAM. Ela destina-se a guardar cada imagem

para ser comprimida em JPEG. Uma imagem completa no formato CIF tem 101376 píxeis (352x288). A preto e branco, cada píxel corresponde a um byte (256 tons de cinzento). A SRAM do ADSP2181 dispõe de capacidade para guardar apenas uma imagem em memória logo também insuficiente para o algoritmo H.263.

Para além disso, apesar de existir, no "hardware" desenvolvido, memória suficiente para o formato CIF (352x288 = 101376 bytes), pelo facto da memória externa estar dividida em 16 segmentos é importante, em termos de algoritmo que não haja blocos (8x8 píxeis) fraccionados em 2 segmentos. Sendo assim e como cada bloco possui 8kB (8192 bytes), o número máximo de blocos inteiros que se pode obter em cada segmento é 128 (8192/64), como horizontalmente existem 44 blocos (352/8), o número máximo de blocos inteiros verticais é 2 (128/44). O que torna a imagem ligeiramente menor na altura (2x8x16=256) mas simplifica muito o algoritmo. Cada imagem fica então segmentada de acordo com a Figura 31. Cada um destes segmento, que envolve 88 blocos, dá origem a um conjunto de blocos ao qual se chama-se *Group Of Blocks* (GOB)[16]. Este será o formato utilizado tanto na compressão MJPEG como H.263.

0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Figura 31 Segmentos de imagem VideoVigi

Os problemas atrás referidos vão também obrigar a grandes ajustes no algoritmo H.263 de forma a viabilizar uma compressão baseada na redundância espacial e temporal. Tratando-se de videovigilância, tendo em conta que as imagens recebidas, mesmo em câmaras móveis, apresentam-se durante uma grande parte do tempo em posições fixas, apenas uma parte da imagem apresenta variação em relação à anterior. Por esta razão, ao contrário do que acontece normalmente em cinema, em videovigilância o algoritmo *motion estimation* serve normalmente apenas para estimar o movimento de um objecto na imagem e não de

toda imagem. Dependendo do tamanho do objecto poderá haver apenas movimento em alguns GOB's e nesse caso a vantagem do algoritmo *motion estimation* poderá não ser significativa. Retirando este algoritmo melhora-se o problema de falta de processamento. Em relação à memória, o algoritmo H263 estima a diferença entre duas *frames* consecutivas de forma a comprimir apenas as diferenças. A *Random Access Memory* (RAM) mínima necessária deveria ser pelo menos suficiente para suportar duas *frames* completas, no entanto foi dimensionada para suportar apenas uma *frame*. A solução será guardar em memória apenas alguns dados que representem um bloco em vez de guardar todo o bloco. Como na compressão de uma imagem completa é sempre efectuada uma DCT e sabendo que os primeiros bytes da DCT representam a informação mais relevante do bloco, estes poderão ser guardados em memória para serem comparados com os da *frame* seguinte. Se considerarmos três bytes suficientes, para comparar uma imagem completa será apenas necessário 4224 bytes de SRAM (352x256/64 x3). Esta dimensão de RAM já não representa qualquer impedimento.

Todo o *firmware* desenvolvido para a compressão de vídeo, no DSP ADSP2181, será efectuado em linguagem *assembly* da *família* ADSP2100. O desenvolvimento de hardware e *firmware* do sistema VideoVigi que não esteja directamente relacionado com a compressão e criptografia não se enquadra no âmbito desta tese. Para qualquer esclarecimento adicional deve ser consultado o documento do referido projecto [1].

5.1. PROTOCOLO

De forma a enquadrar os desenvolvimentos das rotinas de compressão é necessário descrever o protocolo implementado no sistema VideoVigi.

Os pacotes que circulam entre o módulo VideoVigi e a aplicação do PC utilizam o protocolo de transporte UDP, uma vez que este implica um *overhead* menor (vs. TCP) e a retransmissão de pacotes não recebidos assim como o ordenamento não traz qualquer tipo de vantagem. Cada pacote de imagem, como será visto posteriormente, tem um byte que identifica a posição dentro da imagem, se dois pacotes seguirem caminhos diferentes, mesmo que o primeiro pacote chegue depois não afectará a sua posição na imagem. Portanto, não existe necessidade de utilizar um protocolo de transporte orientado às ligações, i.e. o *Transmission Connection Protocol* (TCP).

O porto de escuta na aplicação do *Personal Computer* (PC) é o 10008 assim como no módulo VideoVigi. Para não ocorrer a eventual recepção de um pacote que não seja da aplicação, cada pacote que é enviado pelo PC ou pelo módulo VideoVigi tem no 1º Byte o valor 0x88. Este *byte* serve de confirmação.

5.1.1. MENSAGENS DE CONFIGURAÇÃO

As mensagens de configuração são enviadas pelo PC para o módulo VideoVigi. Cada mensagem é um pacote com 3 *bytes* de dados, o primeiro byte é enviado com o valor fixo de 0x88, que serve de confirmação que o pacote vem da aplicação de videovigilância; o segundo *byte* indica o parâmetro que se pretende configurar e o terceiro *byte* o seu valor. Os parâmetros configuráveis estão representados na tabela seguinte.

Tabela 8 Mensagens de configuração[1]

Parâmetro	Hexadecimal	Comentário
Tráfego	0x01	Controlar o tráfego envido pelo módulo VideoVigi.
Compressão	0x02	Configurar sem compressão, MJPEG ou H.263
Câmara	0x03	Alterar as entradas de vídeo desde a entrada 1 à entrada 4.
Inicialização	0x04	Inicializar o módulo VideoVigi por defeito

A mensagem *tráfego* pode assumir quatro valores, que configuram o envio de pacotes de acordo com o valor da mensagem recebida:

Suspenso (0x00) – Cancela o envio de pacotes.

Envio (0x01) – Envio de pacotes apenas até completar uma imagem.

Continuo (0x02) – Envio contínuo de pacotes.

Lento (0x03) – Envio contínuo de pacotes mas com uma pausa.

Controlado (0x04) – Envio de um único pacote.

A mensagem *compressão* pode assumir três valores:

MJPEG (0x00) – Pacotes de imagens com MJPEG.

Sem compressão (0x01) – Pacotes sem compressão.

H.263. (0x02) – Pacotes de vídeo com H.263

A mensagem *câmara* pode assumir quatro valores:

Camara1 (0x01) até Camara4 (0x04)

A *inicialização* só pode assumir um único valor (0x00)

5.1.2. TRANSMISSÃO

O *Maximum Transmission Unit* (MTU) do pacote *Ethernet* é cerca de 1500 octetos (bytes). Para um pacote *User Datagram Protocol* (UDP), isto representa uma capacidade de 1458 bytes de dados. Para enviar uma imagem sem compressão, nos 1458 bytes de dados disponíveis, em cada pacote, pode-se enviar 4 linhas inteiras (1458/352). O que significa que cada imagem ficará completa ao fim de 64 pacotes (256/4). Cada pacote tem para além dos dados enviados mais 6 bytes de controlo (tabela seguinte).

Tabela 9 Bytes de controlo [1]

1º Byte	2º Byte	3º Byte	4º Byte	5 Byte		6º Byte
Identificação	Compressão	N.º segmento	Câmara	Quantificação	tamanhoMSB	tamanhoLSB
0x88	0x00 – H263 0x01 – S/ comp.	0x00 a 0x0F 0x00 a 0x3F	0x00 a 0x04	0x0_ a 0xF_	0x_0 a 0x_F	0x00 a 0xFF

Para indicar que o pacote é do sistema, o primeiro byte é enviado com o valor 0x88. O segundo byte indica o tipo de compressão da imagem. O valor 0x01 corresponde a imagem sem compressão, o valor 0x00 ao formato MJPEG e 0x02 à compressão H.263. O terceiro byte contem o valor da posição para garantir o correcto posicionamento dentro da imagem. A posição varia de zero a sessenta e três para o caso de uma imagem não comprimida e de zero a quinze para imagem comprimida. O quarto byte serve para indentificar a câmara, varia de zero a quatro. O quinto byte indica nos quatro bits mais significativos a matriz de quantificação, varia entre zero para a compressão mínima (melhor qualidade) e um para compressão máxima para cada um dos quatro GOB's. Os bits menos significativos contêm os quatro bits mais significativos do tamanho do pacote. O último byte contém os 8 bits menos significativos do tamanho do pacote. Sendo assim, no caso de pacotes sem compressão, o tamanho do pacote é fixo e tem o valor de 1456 *bytes* (4x352+6+42).

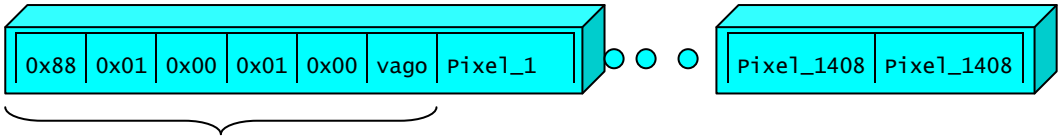


Figura 32 Pacote sem compressão [1]

Em pacotes com imagens comprimidas o tamanho irá variar. Como a compressão varia de acordo com a redundância espacial e temporal, se os dados de um GOB, que correspondem a um segmento comprimido ultrapassar um determinado limite, o baixo débito poderá ficar comprometido. Para corrigir esta situação é implementado um algoritmo de compressão dinâmica. Este algoritmo conduz a um aumento da compressão do próximo GOB, para que o pacote que contém quatro GOB's se mantenha dentro de um determinado tamanho. Este algoritmo será descrito posteriormente.

O carácter que identifica o fim de cada bloco é o valor negativo mais elevado no formato 8 bits com sinal, i.e. o valor 0x80 que corresponde a –128, e o valor que identificará o fim do segmento será o valor seguinte, i.e. o valor 0x81 que corresponde a –127.

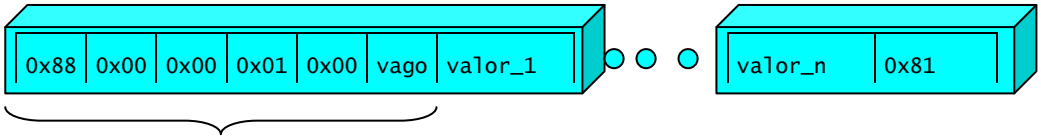


Figura 33 Pacote com compressão [1]

5.1.3. MODOS DE TRANSMISSÃO

A transmissão de pacotes de imagem pelo módulo VideoVigi é efectuada em quatro modos diferentes. O primeiro modo (modo *fast*) foi pensado para ser implementado numa rede onde se sabe que a largura de banda é superior ao tráfego máximo enviado pelo módulo com por exemplo numa LAN (*Local Area Network*). Para configurar neste modo, a aplicação do PC envia um pacote com a mensagem de configuração de tráfego com o valor *continuo* (0x02). Neste modo de funcionamento, os pacotes das imagens comprimidas são enviados à medida que são disponibilizados na sua máxima capacidade de transmissão.

O segundo modo de funcionamento (modo *slow*) é semelhante ao anterior, no entanto foi pensado para ser utilizado em situações de videovigilância não críticas em que se pretende

que o tráfego seja muito baixo de forma a utilizar poucos recursos de rede. Para configurar o sistema VideoVigi neste modo, a aplicação do PC envia um pacote com a mensagem de configuração de tráfego com o valor "lento" (0x03). Neste modo de funcionamento é colocada uma pausa entre a transmissão de pacotes sucessivos.

O terceiro modo de funcionamento (modo *frame*), foi pensado para situações de videovigilância ainda menos críticas que a anterior. Neste caso a transmissão de imagens é realizada a "pedido", i.e. sempre que for recebido um pedido o módulo VideoVigi envia pacotes até completar uma imagem. Para configurar neste modo, a aplicação do PC envia um pacote com a mensagem de configuração de tráfego com o valor "envio" (0x01). No caso de imagens não comprimidas enviará 64 pacotes, para imagens MJPEG serão enviados apenas 16.

O quarto modo de funcionamento (modo *run*) é utilizado sempre que se desconhece o percurso dos pacotes. Este modo de funcionamento adapta-se à rede, i.e. o pacote seguinte só é enviado após a recepção do anterior. Para configurar neste modo, a aplicação do PC envia um pacote com a mensagem de configuração de tráfego com o valor "controlado" (0x04). O módulo VideoVigi responde com um pacote de imagem com ou sem compressão de acordo com o configurado. Quando a aplicação recebe o pacote, verifica se pertence ao módulo VideoVigi, se pertence envia nova mensagem a solicitar novo pacote, antes mesmo da descompressão. O módulo VideoVigi, por sua vez responde novamente com um pacote de imagem e assim sucessivamente. A figura seguinte ilustra este modo de funcionamento.

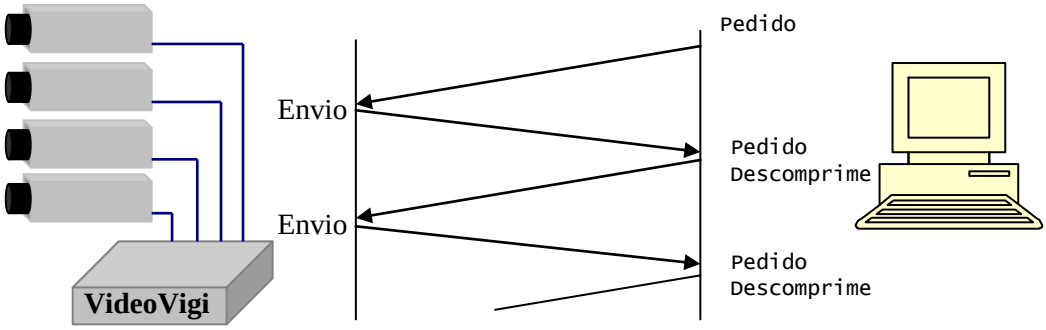


Figura 34 Modo de transmissão controlado [1]

5.2. H.263 (ADAPTADO)

O algoritmo implementado para a compressão completa pode ser descrito pelo fluxograma seguinte.

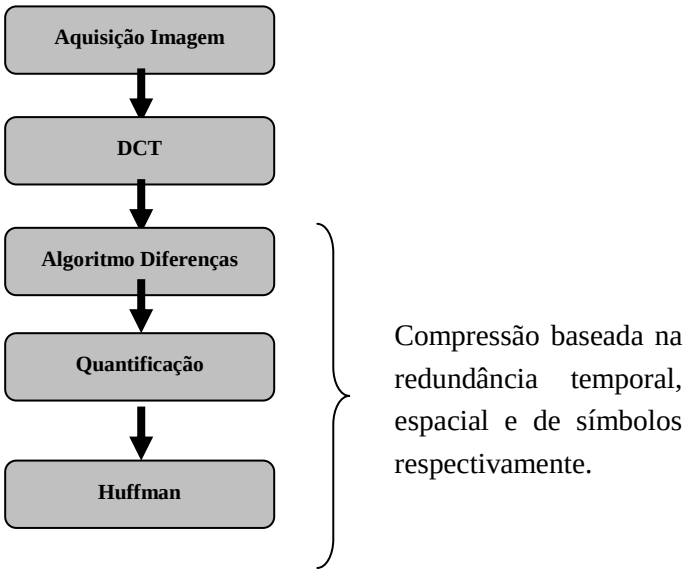


Figura 35 Fluxograma de compressão

No primeiro estado é efectuada a aquisição de uma imagem completa no formato CIF, no segundo estado, a imagem é convertida para o domínio das frequências, no terceiro estado, o algoritmo das diferenças estima os blocos de imagem (8x8) que devem ser actualizados (redução de redundância temporal), no quarto estado, a quantificação vai eliminar a informação de alta frequência da imagem pouco relevante (redução de redundância espacial) e no quinto o algoritmo Huffman reduz a redundância de símbolos.

As Rotinas de compressão desenvolvidas, devidamente comentadas, podem ser consultadas no Anexo B.

5.2.1. AQUISIÇÃO IMAGEM

A aquisição de imagem é efectuada pelo *video decoder* (ADV7183) da *Analoge Device*®. Como na aplicação específica se pretende uma utilização no formato CIF e o *decoder* apresenta uma resolução de 720 *píxels* por linha, apenas um quarto dos píxeis são considerados, i.e. metade dos píxeis por linha e em apenas um *field*. Para transferir os dados, que são tratados pelo DSP, é utilizado uma memória intermédia do tipo *First In First Out* (FIFO). No "bus" de 8 bits, numa imagem a P/B, cada píxel é disponibilizado no *bus* à cadência de 13,5 MHz (LLC2 – Figura 36).

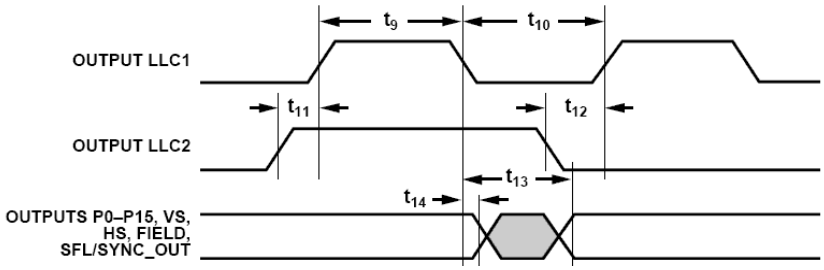


Figura 36 Sincronismo dados no *video decoder* [22]

A aquisição da imagem tem que ser realizada de forma síncrona, i.e. como se vai utilizar apenas um *field*, deve-se garantir que o início da imagem coincide com o início de um *field* e sempre do mesmo, para não ocorrer nenhum o efeito de oscilação de uma linha. Para isso é realizado um teste à *flag* PF3, e assim garantir que a aquisição é efectuada a partir do início do *field*1. Se estiver a meio do *field*1 deve esperar pelo seu fim e também pelo fim do *field*2 para iniciar a aquisição no início do primeiro. O teste da *flag* PF3 é realizado através de uma operação AND entre a *word* das *flags* e a respectiva máscara. O código envolvido no sincronismo de field é apresentado a seguir (Tabela 10).

Tabela 10 Sincronismo de *field* (sincronismo vertical)

```
{ESPERA PELO INICIO DO FIELD1}
       AY0=PMASC_PF3;                { o FIELD e' recebido na DSP_AUX (PF3)              }
       eimpar:                       {espera por fim de FIELD1 para comecar no seu inicio }
              AX0=DM(PFDATA);
              AR=AX0 AND AY0;
       if EQ JUMP eimpar;            {se for zero, esta a meio do FIELD1, deve aguardar  }
       AY0=PMASC_PF3;                {o FIELD e' recebido na DSP_AUX (PF3)               }
       epar:                         {espera por fim de FIELD2 para activar PIC_START    }
              AX0=DM(PFDATA);
              AR=AX0 AND AY0;
       if NE JUMP epar;              {se nao for 0, ainda esta no FIELD2, deve aguardar  }
```

O DSP, no início de uma *frame*, activa o /OE do *video decoder* de forma a este poder transferir os dados de cada píxel para a memória FIFO. Em simultâneo, à medida que os dados estão a ser escritos no FIFO, o DSP vai os lendo e guardando na *data memory* (DM) externa, de forma a não exceder a capacidade do FIFO. Como foi activada a interrupção IRQ2 do DSP, que por sua vez está ligada à /EF (*Empty Flag*); se o FIFO ficar vazio a leitura, através deste vector de interrupção, é cancelada durante dez ciclos de relógio, caso contrário o DSP faria leituras erradas do FIFO. Estes dez ciclos de relógio são suficientes para o *vídeo decoder* colocar novamente dados no FIFO. Assim que a imagem estiver completa, o DSP inibe a escrita para o FIFO, desactivando as saídas do *video decoder*, e inicia-se o tratamento de imagem.

A escrita no FIFO, como já foi referido é realizada à frequência de 13,5 MHz, como interessa guardar apenas metade dos píxeis por linha (para o formato CIF), em cada dois valores lidos, um é "descartado" como se pode ver na rotina "leSegmento" (Tabela 11).

Tabela 11 Leitura do FIFO para SRAM

```
leSegmento:
       CNTR=6912;     {cada segmento de memoria tem 8192B, 8192/360=22,8 Linhas, para     }
                                  {conter 1 Nº inteiro de linhas e de blocos(16 linhas)}
                                  {no primeiro segmento a 1ª tem erros (píxeis pretos) }
                                  {(80 bytes + 352 pixeis por linha) x 16 +2 tolerancia}
       DO lepíxel UNTIL CE;       {sem o valor 2 faltavam píxeis no fim do segmento    }
                      AX0=IO(0);
                      PM(i4,m4)=AX0; {Precisa desta instrucao p/ realisar leituras àfreq. }
                      AX1=IO(0);     {de 16 MHz, s/ ela as leituras eram feitas a 11 MHz  }
              lepíxel:DM(i0,m0)=AX1; {le 2 valores do FIFO mas só guarda 1(posicoes pares)}
rts;
```

Se o módulo VideoVigi estiver configurado para enviar imagens sem compressão, o DSP inicia a leitura de quatro linhas dos dados de um segmento de imagem guardada na DM externa e copia linha a linha para um *buffer* da DM interna fazendo o sincronismo por software, i.e. os dados só devem ser guardados quando forem os píxeis da linha e devem respeitar correctamente o seu início. Para isso, os primeiros dados de cada linha devem ser lidos de forma a detectar o inicio dos píxeis. Quando as quatro linhas estiverem completas o DSP sinaliza na respectiva *flag* e aguarda que o microcontrolador copie os dados para enviar o pacote. Assim que os dados estiverem copiados inicia-se novamente um ciclo para as quatro linhas seguintes até completar os 64 grupos de quatro linhas dos16 segmentos.

Se o módulo "VideoVigi" estiver configurado para enviar imagens com compressão, o DSP inicia a leitura dos dados de um segmento de imagem completo, guardado na DM externa, e copia-o linha a linha para um *buffer* da DM interna fazendo o sincronismo por software. De seguida copia para outro *buffer* da DM interna reordenando os píxeis na ordem dos blocos, i.e. são copiados bloco a bloco em grupos de 8 linhas. Quando os 88 blocos (352/8 x 16/8) estiverem todos ordenados o GOB estão prontos para serem comprimidos. Inicia-se então a compressão individual de cada bloco. No fim da compressão de todos os blocos do GOB corrente, lê e comprime os próximos 3 GOB de forma a preparar um pacote para ser enviado. Para isso, o DSP sinaliza na respectiva *flag* a disponibilidade de 4 GOB já comprimidos e aguarda que o microcontrolador os copie para enviar o pacote. Assim que os dados estiverem copiados inicia-se novamente um ciclo para os restantes segmentos (GOBs).

O DSP, em cada pacote, coloca nos *bytes* de controlo o posicionamento na imagem, o tipo de compressão e o nível de compressão. No final de uma imagem, o DSP reinicia a captura de uma nova imagem no *video decoder.*

As Rotinas de aquisição desenvolvidas, devidamente comentadas, podem ser consultadas no Anexo A

5.2.2. TRANSFORMADA DE COSENOS DISCRETA

O algoritmo foi implementado nos seguintes passos ao que correspondem as seguintes rotinas desenvolvidas em assembly:

- *copiaSegmento* – Copia os píxeis correspondentes a cada GOB para PM interna a partir dos dados não tratados de cada *frame* existente na SRAM externa, fazendo o sincronismo horizontal por *software.*

- *copiaBlocos* – Copia os píxeis para um buffer existente na DM interna, reordenando os píxeis por blocos (linhas de 8 *píxels*).

- *comprimeBloco* – Faz as operações matemáticas associadas à transformada de coseno (é utilizada de forma recursiva na rotina *comprimeBlocos*).

A rotina *copiaSegmento*, que é responsável por copiar os píxeis de cada linha da DM externa para a DM interna deve assegurar o sincronismo horizontal. É muito importante que a cópia respeite o inicio de cada linha sem falhas. Em cada linha da DM externa (copiada do *video decoder*), existem 352 *bytes* que são píxeis da linha e 80 *bytes* são dados. Dos 80 *bytes* de dados, os 70 primeiros *bytes* da linha têm o valor 0x01 seguidos do valor 0x80 ou, alternadamente, 0x00. Como existem pequenas flutuações nos valores lidos do *decoder*, na rotina *copiaSegmento*, é necessário um sincronismo horizontal, este é feito por software. O sincronismo é conseguido através da detecção do valor 0x80 ou 0x00 a partir da posição 53. Estes são os valores escritos pelo *video decoder*. Quando um destes valores é encontrado são verificados os dois anteriores para confirmar que ambos têm o valor 0x01. Se houver confirmação destes três valores então sabe-se que a linha inicia os seus píxeis três posições à frente.

Tabela 12 Sincronismo de linha (sincronismo horizontal)

```
procura1:
        AR=0;
        DMOVLAY=DM(valorDMOVLAY);
        AYbv0=DM(i0,m0);
        DMOVLAY=0;
        AR=AR+AY0;              {vai a procura do sincronismo horizontal 00h ou 80h  }
        if EQ jump confirmar3;
        AR=-0x0080;
        AR=AR+AY0;              {vai a procura do sincronismo horizontal 00h ou 80h  }
        if EQ jump confirmar3;
        AX1=AX0;
        AX0=AY0;
        jump procura1;
confirmar3:
        AR=AX0-0x0010;          {vai confirmar o ultimo valor  (01h)          }
        if EQ jump confirmar4;
        jump procura1;
confirmar4:
        AR=AX1-0x0010;          {vai confirmar o penultimo anterior (01h)     }
        if EQ jump encontrado1;
        jump procura1;
encontrado1:
```

A partir desta posição são lidos 352 bytes que correspondem a 352 píxeis da linha e copiados para um *buffer* interno (PM) chamado "entrada". Ao fim de 16 linhas o *buffer* contem os píxeis de todo o segmento ordenado por linhas.

A rotina *copiaBlocos* copia o segmento para outro *buffer* (*blocos*) existente na DM na ordem dos blocos (linhas de oito píxeis), i.e. reordena o segmento de forma a colocar todos os píxeis pertencentes ao mesmo bloco seguidos; desta forma o segmento deixa de estar ordenado por linhas de 352 píxeis e passa a estar ordenado por linhas de 8 píxeis, i.e. o segmento de imagem fica ordenado como um GOB. A rotina *copiaBlocos* invoca recursivamente a rotina *copiaBloco* para cada grupo de dois blocos situados verticalmente na mesma coluna (tabela 11).

Tabela 13 Reordenação por blocos

```
copiaBloco:
        CNTR=15;                            {le 16 linhas de dois blocos de cada vez     }
        DO linhabloco1 UNTIL CE;            {cada segmento tera 88 blocos (44x2)         }
                CNTR=8;               {para facilitar, a leitura e' feita por colunas de 2 }
                DO umalinha1 UNTIL CE;
                        AX0=PM(i4,m4);
                umalinha1:DM(i1,m1)=AX0;    {Guarda na DM interna na ordem dos blocos    }

                m4=344;
                MODIFY(i4,m4);              {para avancar uma linha completa (352-8)     }
                m4=1;
        linhabloco1:nop;
                CNTR=8;                     {ultima linha destes blocos                  }
                DO ultimalinha1 UNTIL CE;
                        AX0=PM(i4,m4);
                ultimalinha1:DM(i1,m1)=AX0; {Guarda na DM interna na ordem dos blocos    }
rts;
```

A rotina *comprimeBlocos* comprime todos os blocos sequencialmente. Para isso executa de forma recursiva a rotina *comprimeBloco* até ao final do GOB (88 blocos). O algoritmo efectuado pela rotina *comprimeBloco* corresponde directamente à implementação do *standard* JPEG com algumas adaptações para minimizar a utilização de recursos. Inicialmente, a cada coeficiente do bloco a comprimir é subtraído 128 de acordo com a expressão (1), para evitar a saturação do registo da unidade *Multiply Accumulate* (MAC) do DSP e tornar os valor da componentes DC próximos de zero (vantajoso para compressão do algoritmo Huffman). Em simultâneo o bloco é guardado no *buffer X_dm*.

$$X\_dm(i,j) = B(i,j) - 128 . \quad (\mathbf{1})$$

$$DCT\_dm = 1/4 . M_{cos} \times X\_dm \times M_{cos}^{T} . \quad (\mathbf{2})$$

De seguida é efectuada a DCT propriamente dita; esta é obtida através da multiplicação da matriz dos cosenos (Tabela 14) pela matriz bloco da imagem e a matriz resultante (guardada no *buffer* Y_dm) é novamente multiplicada pela transposta da matriz dos cosenos, expressão (2).

Tabela 14 Matriz dos cosenos [11]

1/√2	1/√2	1/√2	1/√2	1/√2	1/√2	1/√2	1/√2
cos(pi/16)	cos(3pi/16)	cos(5pi/16)	cos(7pi/16)	-cos(7pi/16)	-cos(5pi/16)	-cos(3pi/16)	-cos(pi/16)
cos(2pi/16)	cos(6pi/16)	cos(10pi/16)	cos(14pi/16)	cos(14pi/16)	cos(10pi/16)	cos(6pi/16)	cos(2pi/16)
cos(3pi/16)	cos(9pi/16)	cos(15pi/16)	cos(21pi/16)	-cos(21pi/16)	-cos(15pi/16)	-cos(9pi/16)	-cos(3pi/16)
cos(4pi/16)	cos(12pi/16)	cos(20pi/16)	cos(28pi/16)	cos(28pi/16)	cos(20pi/16)	cos(12pi/16)	cos(4pi/16)
cos(5pi/16)	cos(15pi/16)	cos(25pi/16)	cos(35pi/16)	-cos(35pi/16)	-cos(25pi/16)	-cos(15pi/16)	-cos(5pi/16)
cos(6pi/16)	cos(18pi/16)	cos(30pi/16)	cos(42pi/16)	cos(42pi/16)	cos(30pi/16)	cos(18pi/16)	cos(6pi/16)
cos(7pi/16)	cos(21pi/16)	cos(35pi/16)	cos(49pi/16)	-cos(49pi/16)	-cos(35pi/16)	-cos(21pi/16)	-cos(7pi/16)

A multiplicação de matrizes é a soma dos produtos de cada linha da primeira matriz por todas as colunas da segunda matriz. O primeiro produto ($Y\_dm = M_{cos} \times X\_dm$) é efectuado de acordo com a definição de produto de matrizes. No segundo produto ($Y\_dm \times M_{cos}^{T}$), em vez de se realizar a transposta da matriz dos cosenos é feita uma multiplicação "linha por linha" simplificando desta forma o algoritmo e consequentemente "poupando" alguns ciclos de relógio (Tabela 15). O resultado é guardado no *buffer DCT_dm.*

Tabela 15 Multiplicação linha por linhas (2º produto da DCT)

```
DO linha loop2 UNTIL CE;     {Y dm.A dm´,(1ª lin. Y pm c/ todas lin. de A dm e sucessiva/)}
       i5=i6;                        {I5 = INICIO DE A_dm (inicio da 1ª linha)           }
       CNTR=m2;
       DO coluna loop2 UNTIL CE;
              i0=i1;                 {coloca I0 na linha actual (Y dm -> 1ª matriz)      }
              i7=i5;                 {coloca I7 na linha actual (A pm -> 2ª matriz)      }
              CNTR=M2;     {contador igual ao numero de elementos da linha da matriz Y dm}
              MR=0, MX0=DM(I0,M0), MY0=PM(I7,M7);{inicializa MR e le elementos das matriz}
              DO elemento_loop2 UNTIL CE;
              elemento loop2: MR=MR+MX0*MY0 (SS), MX0=DM(I0,M0),MY0=PM(I7,M7);
                              {soma produtos da linha da 1ª mat. c/linha da 2ª mat.}
              SR=ASHIFT MR1 (HI),    {Shift e Actualiza I5 no mesmo ciclo relógio        }
              MY0=DM(I5,M5);         {Actualiza I5 para o inicio proxima linha de A_dm   }
              SR=SR OR LSHIFT MR0 (LO);{finaliza shift, resultado em sr1 formato 16.0    }
       coluna loop2: DM(I3,M3)=MR1;  {guarda o resultado na matriz DCT                    }
linha loop2: MODIFY(I1,M1);          {actualiza I1 (matriz Y pm) para a proxima linha    }
```

Como o coseno varia entre –1 e 1 o formato mais adequado para a matriz dos cosenos é o formato "1.15" (um bit inteiro e 15 bits fracionários), o valor mínimo será 0x1000 (com sinal) que corresponde a –1, o valor máximo é 0x7fff a que corresponde 0,999969 (32767/2^15) . Cada valor da matriz será obtido através da seguinte expressão (3).

$$valor_{(1.15)} = valor*2\text{^}15. \qquad (3)$$

Como a matriz dos cosenos vai ser guardada na memória de programa, após a conversão para o formato hexadecimal são acrescentado dois zeros para tornar o valor compatível com os 24 bits disponíveis na PM e permitir a sua correcta leitura. A multiplicação no formato "1.15" da matriz dos cosenos pela matriz do bloco de imagem no formato "16.0", origina o resultado no formato 17.15 (1+16,15+0) o que obrigaria à realização de um *shift* de 1 bit para obter o formato 16.0 na *word* MR1 (MR1 tem os bit 16 a 31 do registo MR) do DSP, a multiplicação seguinte obrigaria a um segundo *shift*. No entanto o cálculo da DCT é um quarto destes produtos logo ao ignorar os dois *shifts* o resultado irá ser um quarto dos produtos, como se pretende (expressão 2). Desta forma consegue-se realizar os dois produtos em "simultâneo" com o produto por um quatro, poupando-se novamente instruções.

Após a obtenção da DCT para o bloco de imagem actual, a imagem desse bloco está neste momento no domínio das freqências. Antes de passar aos blocos seguintes é necessário efectuar as próximas operações associadas directamente à compressão.

5.2.3. ALGORITMO DAS DIFERENÇAS

O algoritmo das diferenças foi implementado numa única rotina desenvolvida em assembly:

- *estimaDiferencas* – Compara o bloco actual com o bloco na mesma posição da imagem anterior, de forma a determinar se o bloco deve ou não ser actualizado de acordo com as diferenças determinadas (primeiro nível de compressão baseado na redundância temporal).

A rotina *estimaDiferencas* efectua uma comparação de três coeficientes do bloco corrente com os do mesmo bloco da DCT guardada da *frame* anterior, para determinar se a diferença é superior a um determinado valor. Os coeficientes da DCT que irão ser comparados são o valor da componente DC que representa o nível da luminosidade do bloco, o primeiro e o sexto valor da componente AC. Como a seguir à DCT será efectuado a quantificação que corresponde a uma divisão pelos valores da Tabela 1, a tolerância deverá ser ponderada com esta tabela. Os resultados práticos revelaram um bom compromisso para uma variação máxima de duas unidades na compressão final, à qual corresponde o dobro do valor da tabela de quantificação, i.e. uma variação de 32 (16x2) para a componente DC, 22 (11x2) para o primeiro valor AC e 24 (12x2) para o sexto valor AC. Para determinar esta diferença (intervalo) e devido às limitações da linguagem assembly, a cada um destes três valores da actual DCT é subtraído o respectivo valor do mesmo bloco da DCT guardada da *frame* anterior. Ao resultado intermédio é subtraído e adicionado o valor da diferença, se o primeiro for negativo e o segundo for positivo então o valor tem um desvio inferior à diferença. Na tabela seguinte pode-se ver um enxerto de código onde é efectuada a comparação entre os primeiros valores do bloco (componente DC).

Tabela 16 Comparação das componentes DC

```
condicao1:
       AY0=PM(I4,M4),AX0=DM(I3,M3);  {AY0 e AX0 tem o valor das componentes DC (1° Bytes) }
       AR=AX0-AY0;                   {vai encontrar a diferenca entre os 2 valores       }
       AY1=32;                       {o valor pode ser negativo logo                     }
       AR=AR-AY1;                    {vai ver se o erro e' inferior a diferenca (negativo)}
       if LE JUMP condicao2;
       jump diferente;
condicao2:                          {o valor pode ser positivo logo                     }
       AR=AX0-AY0;                   {vai encontrar a diferenca entre os 2 valores       }
       AR=AR+AY1;                    {vai ver se o erro e' inferior a diferenca (positivo)}
       if GE JUMP condicao3;
        jump diferente;
```

Se cada um dos três valores tem um desvio inferior à diferença estamos perante um bloco semelhante ao anterior. Neste caso o bloco será inteiramente substituído pelo carácter de fim de matriz, i.e. o valor 0x80, e a compressão do bloco termina aqui; na descompressão este valor significará que não houve actualização neste bloco. Se uma das três condições não se verificar o bloco não é semelhante, neste caso os três bytes são actualizados no buffer existente na *Program Memory* (PM) interna para este efeito (*oldFrame*) e a compressão do bloco persegue normalmente (quantificação).

5.2.4. QUANTIFICAÇÃO

O algoritmo foi implementado nos seguintes passos que correspondem rotinas com o mesmo nome desenvolvidas em *assembly*:

- *buscaTabelQ* – Lê o valor *nívelCompressão* para indexar a tabela de quantificação correspondente.

- *quantificacao* – Faz a operação de quantificação da matriz bloco-DCT a qual actua como um filtro na imagem no domínio das frequências (segundo nível de compressão baseada na redundância espacial).

- *actualizaNivelQ* – Actualiza o nível de compressão de acordo com a necessidade de forma a garantir um débito inferior a aproximadamente 64 kbps.

A rotina *buscaTabelQ* faz parte de um algoritmo desenvolvido, que ajusta a compressão de forma dinâmica. Este algoritmo pretende compensar a diminuição de compressão em imagens com elevados contrastes e/ou com muito movimento, de forma a garantir um baixo débito praticamente constante. Nesta primeira rotina executada no ínicio da compressão de cada GOB (grupo de 88 blocos) é efectuada a leitura do *nívelCompressão* que irá indicar a tabela de compressão a utilizar na quantificação, assim como as tabelas referentes ao algoritmo *Huffman*, descrito posteriormente. Este valor é actualizado pela rotina *actualizaNivelQ* que irá indexar na quantificação a tabela referida pela norma para a luminância (compressão mínima) ou a tabela referida para a crominância. Em resultados práticos a utilização da tabela da crominância, em relação à da luminância, representa no que se refer à quantificação um aumento da compressão em cerca de 50%, claro que também representa um aumento da degradação da imagem, mas este é o "preço" a pagar.

A rotina *quantificacao* é responsável pela eliminação das componentes de alta frequência de valor reduzido, através da divisão elemento a elemento da DCT com uma matriz de quantificação (expressão 3). Apesar de se poderem utilizar matrizes de quantificação com diversos coeficientes de acordo com a compressão pretendida. A matriz da Tabela 1 é definida na recomendação T.81 do ITU para a luminância ou imagens a preto e branco, esta corresponde a um bom compromisso entre uma compressão relativamente elevada e perdas quase imperceptíveis na imagem, pelo que será utilizada na compressão normal. No entanto esta matriz será ajustada dinamicamente como já foi referido.

$$Y(i,j) = DCT\_dm\ (i,j) \times 1/Q\ (i,j)\ . \qquad (4)$$

Como o DSP permite executar instruções de produto, leitura da memória de programa e leitura da memória de dados simultaneamente, é mais eficiente a execução de produtos em vez de quocientes. Por este motivo a matriz Q (quantificação) foi convertida para uma matriz de inversos. A atribuição do formato para a matriz dos inversos dos coeficientes de quantificação também é "1.15", uma vez que o máximo é 1/16=0,0625 e o mínimo é 1/121=0,00826. A matriz dos inversos dos coeficientes Q (matriz de quantificação) é também guardada na memória de programa.

A matriz resultante é por fim ordenada em ziguezague (Figura 10) de forma a juntar no fim o maior número possível de zeros. A operação de ordenamento em ziguezague e a respectiva multiplicação pela matriz dos inversos é realizada em simultâneo de forma a "poupar" ciclos de relógio. Para isso e para facilitar esta operação, a matriz dos inversos é guardada na ordem ziguezague e é criada na memória de programa uma terceira matriz contendo essa mesma ordem, que irá servir para indexar o acesso à DCT calculada anteriormente.

Por fim é procurado o primeiro zero do grupo de zeros, no final do bloco, no qual será colocado um carácter de fim de bloco. O resultado fica guardado no *buffer* "pacote". A matriz inicial no domínio do espaço está agora no domínio das frequências e limitada ao tamanho indicado pelo carácter de fim de bloco, resultando numa compressão que poderá ir até 90%. Esta última operação, para utilizar poucos recursos é realizada em sentido inverso, isto é do fim para o início. Então os valores são lidos, em sentido inverso, até encontrar um valor diferente de zero, neste primeiro zero é colocado o carácter de fim de matriz. Como o resultado da quantificação impossibilita a existência de valores altos, não poderá existir o valor 0x0080, pelo que este valor pode ser usado para marcar o fim do

bloco (EOB). O próximo bloco comprimido será guardado no mesmo *buffer pacote* imediatamente a seguir ao valor 0x0080. Na descompressão, ao encontrar estes valores, sabe-se que eles correspondem a zero, assim como todos os restantes elementos da mesma matriz.

Uma das imposições iniciais para esta Tese era garantir um débito baixo, o valor de 64 kbps corresponde ao débito existente num intervalo de tempo associado a uma linha telefónica. Este é portanto um bom valor de referência. Para um débito máximo de 64 kbps (65536 bps) teremos um débito disponível de 8192 Bps. Se considerarmos o valor de 4 *frame* por segundo como o mínimo aceitável para videovigilância (confirmado pela prática), teremos 2048 B/*frame*. Para baixar o *overhead* IP garantindo no entanto a existência de espaço suficiente para enviar toda a informação necessária (para os piores casos), os testes práticos revelaram o valor de quatro GOBs por pacote ser um bom compromisso, 8 GOB poderia em situações excepcionais exceder os cerca de 1500 bytes do MTU. Logo o pacote terá de ser enviado em 512 bytes, ou seja 128 bytes por GOB. Como esta comparação é efectuada na rotina *actualizaNivelQ* para cada GOB pelo último endereço colocado em *enderecoBloco* antes da compressão Huffman e sabendo que esta oferece um nível de compressão adicional mínimo em cerca de 50% então o valor disponível para cada GOB (antes de Huffman) é cerca de 256 bytes. Assim, no fim da compressão (sem huffman) de todos os blocos do GOB efectua-se a comparação do tamanho do GOB com o valor 256 (Tabela 17), se este for superior actualiza o *buffer nivelCompressao* com o valor 1 (compressão máxima) para todos os GOB seguintes de forma a compensar o aumento do GOB actual. A compressão máxima corresponde à tabela de quantificação definida na norma para a crominância, enquanto que a compressão mínima utiliza a tabela de quantificação definida na norma para a luminância. Se for então inferior a 111 bytes, i.e. ligeiramente inferior a 50% para não haver actualizações sucessivas (valor ajustado na prática), actualiza o *buffer nivelCompressao* com o valor 0 (compressão mínima). Se o valor estiver compreendido entre os dois mantêm a compressão actual. Todas as alterações na tabela são efectuadas isoladamente apenas nos GOB's onde ocorrem as alterações.

Tabela 17 Verificação do tamanho do GOB (sem huffman)

```
AY0=^pacote;                          {1º endereco o pacote                          }
AX0=DM(endBloco);                     {ultimo endereco o pacote (sem huffman)        }
AR=AX0-AY0;                           {AR tem a diferenca entre os dois enderecos    }
AX0=AR;
```

```
AY0=256;                              {p/ debito 65536bps->8192Bps/4(min frames aceitavel)}
                                      {2048B/F=>512B/pacote=>128B/GOB sem huffman =>256 B  }
AR=AX0-AY0;                           {verificar se e' superior a 256                      }
if GT JUMP maximo;                    {se for superior actualiza  o nivel para comp.max.   }
jump inferior
```

5.2.5. HUFFMAN

O algoritmo Huffman também sofreu alterações significativas que foram motivadas pelas alterações anteriores, mas todas elas foram apenas efectuadas ao nível das tabelas. Como no algoritmo das diferenças sempre que um bloco é considerado semelhante ao mesmo bloco da imagem anterior é substituído inteiramente pelo carácter de fim de bloco[3], numa situação de relativamente pouco movimento vai naturalmente haver uma predominância do carácter fim de bloco (EOB). Por esta razão, as tabelas da norma referidas para a crominância apresentam melhores resultados mesmo quando a matriz de quantificação utilizada é a da luminância, um vez que o carácter fim de bloco (EOB) da crominância usa apenas dois bits (00), enquanto o da luminância usa quatro (1010). Os resultados práticos mostram um acréscimo de compressão embora variável com o movimento em cerca de 25% com as tabelas Huffman da crominância, esta é a razão foram utilizadas estas tabelas. Em relação à componente DC surge uma outra necessidade de mudança. Como o carácter fim de bloco irá aparecer em posições onde é esperada uma componente DC, o valor de dois bits "00" correspondente à primeira linha da componente DC não pode ser usado para não ser confundido com o EOB; logo, o valor da primeira linha foi substituído por "010" e a segunda linha passa de "01" para "011" uma vez que nenhum código pode conter outro no seu começo (seriam confundidos). Apesar de haver uma perda instantânea de compressão quando estes valores são utilizados, uma vez que passaram de dois para três bits, o ganho continua a ser significativo. A razão para este facto é que todos os blocos têm um carácter fim de bloco (EOB) e, em muitos casos, têm apenas este carácter, no entanto raramente os blocos têm uma componente DC com o valor 0 ou 1; para além disso o ganho é de dois bits enquanto a perda é de apenas um bit. Com esta alteração, as tabelas Huffman para a componente DC usadas são as seguintes (o ultimo byte de cada valor não tem significado).

[3] A informação dos blocos actualizados poderia ser efectuada por 88 bits (11 bytes) nos bytes de controlo. Mas apesar de ser eficiente na situação de pouco movimento tornava o *over head* desnecessáriamente "pesado" quando houvesse muito movimento. E a situação de elevado movimento é aquela que mais importa resolver.

Tabela 18 Tabelas Huffman componente DC

```
.init DC CoefC:        0x000200,0x000300,0x000400,0x000500,0x000600,
               0x000E00,0x001E00,0x003E00,0x007E00,0x00FE00,0X01FE00;
.init DC_LengC:        0x000300,0x000300,0x000300,0x000300,0x000300,
               0x000400,0x000500,0x000600,0x000700,0x000800,0x000900;
```

O algoritmo Huffman foi implementado nos seguintes passos, aos quais correspondem *labels* no código principal:

- *trataPrimeiraComponenteDC* (Figura 37) – Verifica se é o carácter fim de bloco (EOB) para caso de não ter havido actualização, se sim prossegue no *label trataFimBloco*, se não indexa a tabela de "coeficientes DC" e a tabela de "comprimentos DC". De seguida chama a rotina *procuraValor* para localizar a linha e a posição do valor e depois chama a rotina *actualizaValor* para colocar os bits correspondente ao valor no *buffer pacoteFinal.*

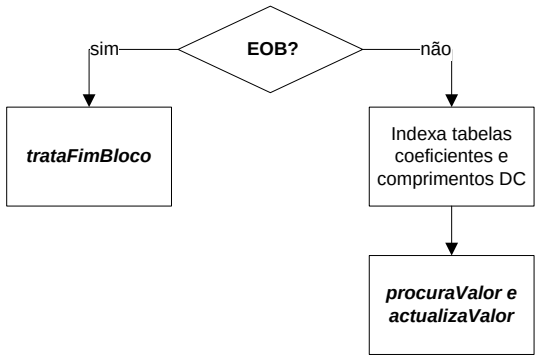


Figura 37 *trataPrimeiraComponenteDC*

- *verificaValor* (Figura 38) – Verifica se atingiu fim de bloco, se sim vai para o *label trataFimbloco*, se não verifica se é um zero, se sim contabiliza-o se não vai para o *label trataComponenteAC.*

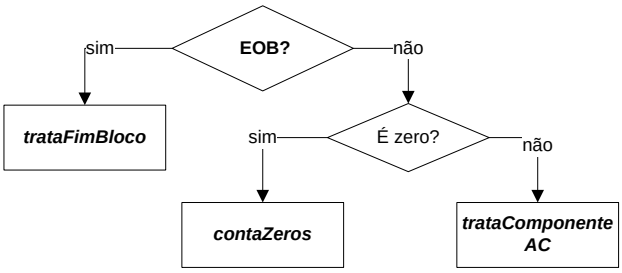


Figura 38 *verificaValor*

- *contaZeros* – Conta os zeros, actualiza o *buffer numeroZeros* e verifica se atingiu o número máximo se sim vai para o *label maximoZeros.*

- *maximoZeros* – Como o algoritmo Huffman não permite associar à componente AC um valor superior a 15 zeros, se este valor for ultrapassado actualiza o buffer de saída com o valor 0x3FA (12 bits) e reinicializa o número de zeros. O valor 0x3FA (12 bits) significará 16 zeros sucessivos (15 zeros antes da componente AC que também é zero). Para simplificar o algoritmo e "poupar" ciclos de relógio, este valor foi deslocado para o fim da tabela.

- *trataComponenteAC* – Vai buscar o número de zeros e indexa a tabela de "coeficientes AC" e a tabela de "comprimentos AC" desviadas o número de zeros multiplicado por dez mais um; uma vez que para cada número de zeros existem dez linhas e não existe a linha zero (porque também não existe a componente AC zero). De seguida chama a rotina *procuraValor* para localizar a linha e a posição do valor e depois chama a rotina *actualizaValor* para colocar os bits correspondentes ao valor no *buffer pacoteFinal.*

- *trataFimBloco* (Figura 39) – Indexa o primeiro valor da tabela de coeficientes AC e a tabela de comprimentos AC para ter o valor do EOB, chama a rotina *actualizaValor* para colocar os bits correspondente ao valor no *buffer pacoteFinal.* De seguida, verifica se o próximo valor é fim do GOB, se sim vai para o *label fimHuffman*, se não verifica se é um novo EOB, se sim vai novamente para o *label trataFimBloco*, se não é uma componente DC. Indexa a tabela de "coeficientes DC" e a tabela de "comprimentos DC". De seguida chama a rotina *procuraValor* para localizar a linha e a posição do valor e depois chama a rotina *actualizaValor* para colocar os bits correspondentes ao valor no *buffer pacoteFinal.* No final vai para o *label verificaValor.*

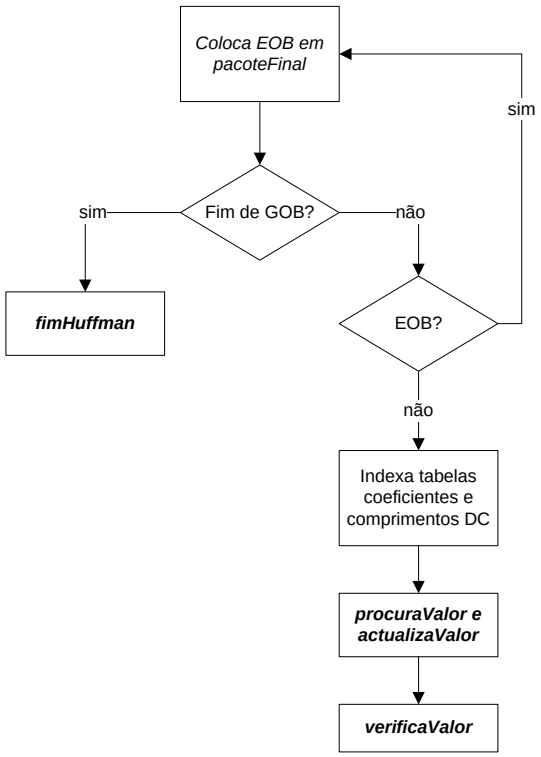


Figura 39 *trataFimBoloco*

- *fimHuffman* – Vai buscar os valores temporários que ainda não tinham sido colocados no *buffer pacoteFinal* e actualiza-os neste *buffer*. Faz a diferença entre o último endereço e o início do buffer para guardar o tamanho actual em *tamanhoPacote*. Este valor servirá para o próximo GOB e será enviado nos bytes de controlo (5º e 6º byte), tanto na leitura do bus *Internal Data Memory Access* (IDMA), como no envio do pacote para se saber onde acaba a informação.

Nos passos anteriores são usadas recursivamente as seguintes rotinas desenvolvidas em *assembly*:

- *procuraValor* – Determina a linha e posição do valor na Tabela Huffman (Figura 14).

- *actualizaValor* – Coloca os bits correspondente a cada valor no *buffer pacoteFinal* (terceiro nível de compressão baseado na redundância de símbolos).

A rotina *procuraValor* irá determinar a linha e posição de cada valor e "retornar" os bits que deverão ser actualizados no buffer de saída. Para isso, antes de ser invocada, para o caso da componente DC, são indexadas as tabelas de "coeficientes DC" e "comprimentos DC"; para o caso da componente AC são indexadas as tabelas de "coeficientes AC" e

"comprimentos AC" desviadas o número de zeros multiplicado por dez mais um. Para determinar a linha da tabela Huffman (Figura 14) basta efectuar comparações sucessivas para determinar o intervalo onde se encontra o valor. Cada comparação, correspondente a uma linha, pode ser definida matematicamente em relação ao intervalo anterior pela expressão seguinte.

$$2 \times \text{intervalo anterior} + 1 \leq \text{valor a testar} \leq 2 \times \text{intervalo anterior} + 1 \ . \qquad (\mathbf{5})$$

Em que o primeiro intervalo é zero para a componente DC (linha 0) e 1 para a componente AC (linha 1). À medida que é testada uma nova linha os apontadores que indexam as tabelas de coeficientes e comprimentos também são incrementados. No final os apontadores estão situados na linha onde se encontra o valor. O valor da linha é lido directamente da tabela dos coeficientes em que a tabela dos comprimentos determina o número de bits utilizados (varia de linha para linha[4]). O número de bits utilizados para a posição é equivalente à linha, i.e. a linha zero não utiliza qualquer bit para a posição (valor único zero), a linha um utiliza apenas um bit para a posição e assim sucessivamente. Analisando o código Huffman verifica-se que para determinar a posição e consequente número binário basta efectuar uma operação simples de acordo com a expressão 6.

Se o valor for negativo => posição = intervalo actual + valor . (6)

Se o valor for positivo => posição = valor .

A rotina *actualizaValor* coloca os bits correspondentes a cada valor no *buffer pacoteFinal*, para isso recebe em cada actualização, para além dos coeficientes, os comprimentos de cada código. Esta operação é realizada através do *shifter* do DSP à medida que vai recebendo valores, faz um *shift* à esquerda nos registos SR igual ao comprimento do próximo valor e coloca os bits do próximo valor encostados à direita; simultaneamente, incrementa os *bitsShiftados*. Quando o número de bits deslocados (*bitsShiftados*) ultrapassa o valor oito (byte completo) faz um deslocamento em sentido igual ao número de bits ainda não copiados para o *buffer pacoteFinal* menos oito e copia os oito bits menos significativos para o este *buffer*.

[4] O código Huffman como já foi referido tem um tamanho de dados variável (VLC) de acordo com a probabilidade de ocorrência.

Tabela 19 Cópia de bits para o *buffer pacoteFinal*

```
AR=DM(bitsShiftados);          {vai ver se ultrapassou oito (byte completo)         }
AR=AR-8;
if GE jump copiaValor1;        {AR tem o numero de bits shiftados menos 8           }
jump fimActualizaValor;
copiaValor1:
       DM(bitsShiftados)=AR;          {Actualiza bitShiftados que ainda nao foram guardados}
       MR0=DM(sr0Temporario);         {vai buscar os valores guardados de SR               }
       MR1=DM(sr1Temporario);
       if EQ jump copiarJa1;
       AR=-AR;
       SE=AR;                         {vai fazer shift contrario para repor byte mais      }
                                      {significativo em sr0                                }
       SR= ASHIFT MR1(HI);            {sr0 tem o valor parte do...                         }
       SR=SR OR LSHIFT MR0(LO);       {sr0 tem o valor completo                            }
       MR0=SR0;
copiarJa1:
       DM(i7,m7)=MR0;                 {copia para o buffer pacoteFinal                     }
```

Se ainda assim o número de bits não copiados ultrapassa oito (uma vez que existem valores com 16 bits) repete-se o processo. No final teremos uma sequência de bits no *buffer pacoteFinal* com uma compressão que ultrapassa 50 %. A fim de testar individualmente a rotina huffman foi realizado um teste prático com o seguinte pacote equivalente a 10 blocos, dos quais 9 não foram actualizados:

6,12,3,0,-4,-9,3,2,0,-1,0,0,0,-1,-1,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x80,0x81

O resultado obtido para as tabelas da luminância e crominância foram respectivamente:

0x9A 0xF1 0xFC 0xBB 0x67 0x6C 0x74 0x15 0x55 0x55 0x55 0x55 0x40

0xDB 0x19 0x3F 0x67 0x86 0x9C 0xAD 0xB2 0x00 0x00 0x00

Os dados originais eram constituídos 26 bytes, com as tabelas da luminância este número foi reduzido para 13 bytes (compressão de 50%), com as tabelas da crominância foi reduzido para 11 bytes (compressão de 58 %). .

.

6. FIRMWARE ENCRIPTAÇÃO SISTEMA VIDEOVIGI

O processamento principal do sistema VideoVigi e ligações de rede é suportado pelo microcontrolador de rede DS80C400 da *Dallas*®. Todo *firmware* desenvolvido para encriptação, que irá ser executado neste micro controlador, será efectuado em linguagem C. O sistema VideoVigi, como já foi referido envolveu desenvolvimentos de *hardware*, *firmware* e *software*. O *firmware* de compressão para o DSP foi totalmente desenvolvido em assembly, o *firmware* do microcontrolador DS80C400 teve desenvolvimentos em C e em assembly da família 80C51. O firmware de aplicação apesar de ter tido como ponto de partida a biblioteca UDP do microncontrolador, foi todo redesenhado para satisfazer as necessidades da aplicação do sistema VideoVigi. Foram também desenvolvidas rotinas de baixo nível (80C51) para configuração de todo *hardware*. Em relação ao firmware de encriptação, este foi apenas adaptado[5] a partir de um exemplo da Hoozi Resources (www.hoozi.com), de forma a poder ser executado no microcontrolador DS80C400 e também para óptimizar os recursos da arquitectura desenvolvida.

[5] O software de encriptação/ desencriptação e um método de refrescamento de imagens bmp em Java (descrito posteriormente) são as únicas porções de código que foram adaptadas (código inicial de outro autor).

6.1. AES

A encriptação AES-128 é executada de forma sequencial em grupos de 16 bytes (matriz de quatro por quatro – matriz *State*), utilizando uma chave também de 16 bytes (matriz *Key*), i.e. 128 bits. Considerando *Nb* o número de colunas na matriz *State* (igual a quatro), *Nk* o número de colunas na matriz *Key* (para uma chave de comprimento 128 bits é também igual a quatro) e *Nr* o número de ciclos efectuados durante a execução do algoritmo (este depende directamente do tamanho da chave, para *Nk*=4 (128 bits) o número de ciclos é igual a dez). O algoritmo recorre a quatro transformações orientadas ao *byte* (*SubBytes, ShiftRows, MixColumns, AddRoundKey*) de acordo com o fluxograma da figura seguinte, no entanto a chave expandida é utilizada exclusivamente na função *AddRoundKey*. De acordo com a volta (*Round*) é utilizada uma submatriz da matriz expandida desde as quatro primeiras linhas até às quatro últimas.

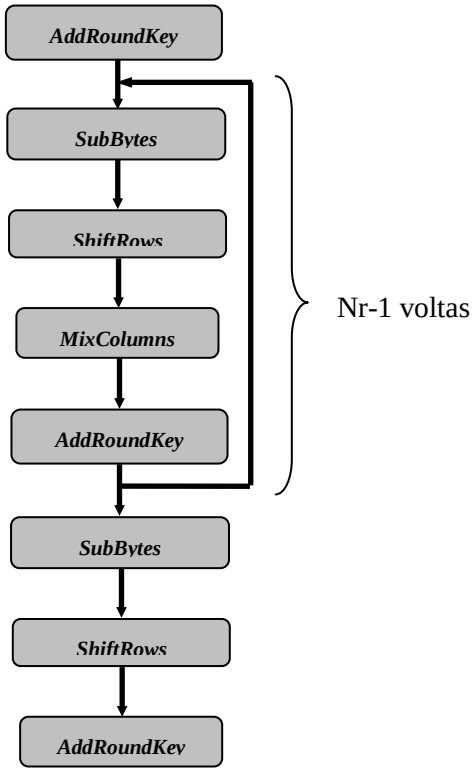


Figura 40 Fluxograma do algoritmo AES

6.1.1. KEYEXPANSION

Apesar da chave ser apenas de 16 byte, esta sofre uma expansão de forma a serem utilizados valores diferentes em cada uma das *Nr* volta (Figura 40). O primeiro passo na criptografia é portanto a função *KeyExpansion.* Para o AES-128 esta função irá expandir a chave de encriptação em 44 *words* (Nb*(Nr+4)) que irão encriptar directamente a matriz

de estado (*state*). A expansão é obtida através de um algoritmo semelhante ao da própria encriptação.

A ultima linha da matriz *Key* (quarta linha) é copiada para uma matriz linear de quatro colunas e sofre uma rotação de um byte para a esquerda. De seguida cada elemento da matriz temporária, conforme o seu valor, é substiuido por um elemento da matriz (*sBox*) numa transformação semelhante à que será descrita posteriormente na operação *SubsBytes*. As primeiras quatro linhas da matriz expandida (*RoundKey*) são as quatro linhas da matriz chave (*key*). A linha seguinte (quinta linha) é o resultado da operação XOR (Tabela 20), bit a bit, entre a matriz temporária, a primeira linha da matriz expandida e a matriz *Rcon*(4) (0x01;0x00;0x00;0x00). A sexta linha é o resultado da operação XOR, bit a bit, entre a segunda e a quinta linha da matriz expandida. A sétima linha é o resultado da operação XOR, bit a bit, entre a terceira e a sexta linha da matriz expandida. A oitava linha é o resultado da operação XOR, bit a bit, entre a quarta e a sétima linha da matriz expandida.

Para as quatro linhas seguintes repetem-se os passos anteriores tendo em conta que a nova matriz de entrada é aquela que se obteve anteriormente (quinta linha até à oitava). A primeira linha também terá um tratamento diferente, i.e. também terá um XOR com a matriz *Rcon*(8) (0x02,0x00,0x00,0x00). Estes passos repetem-se novamente para as quatro linhas seguintes até complementar 11 grupos de quatro linhas, i.e 44 linhas da matriz de chave expandida (*RoundKey*).

Tabela 20 Expansão – XOR entre linhas

```
RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
```

6.1.2. SUBBYTES

Na matriz estado (*State*), a função *void SubBytes()* substiui cada elemento por outro da matriz *sBox*, conforme o seu valor, i.e. o valor do elemento da matriz estado (*State*) indica a posição na matriz (*sBox*) de acordo com a tabela seguinte.

Tabela 21 *SubBytes*

```
void SubBytes()
{
    int i;
    for(i=0;i<16;i++)
    {
        state[i] = sBox[state[i]];
    }
}
```

6.1.3. SHIFTROWS

A função *void ShiftRows()* realiza uma rotação de um byte para a esquerda na segunda linha matriz estado (*State*), uma rotação de dois bytes na terceira linha e uma rotação de três bytes na quarta linha.

6.1.4. MIXCOLUMNS

A função *void MixColumns()* realiza produtos de matrizes de cada coluna da matriz estado pela matriz quadrada definida na Figura 24. Esta é de todas as funções, a que consome mais recursos.

Para a primeira coluna, o primeiro elemento da matriz resulta de um XOR entre todos os elementos da primeira coluna da matriz estado. No entanto, como a matriz auxiliar (da função *MixColumn* tem os dois primeiro elementos (primeira linha) diferentes da unidade, a macro *xtime* que está definida no standard AES (secção 4.2) complementa o produto da coluna pela linha para os dois primeiros valores em binário (Tabela 22). Para as restantes linhas da matriz estado, os passos são análogos.

Tabela 22 *MixColumns* – primeiro produto

```
  Tmp = state[i] ^ state[4+i] ^ state[8+i] ^ state[12+i] ;
  Tm = state[i] ^ state[4+i] ; Tm = xtime(Tm); state[i] ^= Tm ^ Tmp ;
```

6.1.5. ADDROUNDKEY

A função *void AddRoundKey(int round)* apesar de ser uma função muito simples assume toda a importancia pelo facto de ser a única função do algoritmo que utiliza a chave expandida. É nesta função que é introduzida codificação pela chave. Conforme a "volta" (*round*) onde se encontra o código, a matriz de estado sofrerá uma operação XOR elemento a elemento entre uma posição da matriz "chave expandida" (*RoundKey*) e a própria matriz de estado.

Tabela 23 *AddRoundKey* – XOR elemento a elemento

```
void AddRoundKey(int round)
{
    int i;
        for(i=0;i<16;i++)
    {
        state[i] ^= RoundKey[round * 16 + i];

    }
}
```

As Rotinas de encriptação, devidamente comentadas, podem ser consultadas no Anexo C.

7. SOFTWARE CLIENTE

A aplicação cliente foi desenvolvida em Java. A linguagem Java, para além de "oferecer" facilidades de programação em aplicações gráficas, tem a vantagem de se poder executar o código em qualquer máquina, através do conceito de "máquina virtual". O código quando compilado produz os *bytes codes* (bytes formatados para uma máquina virtual), estes são por sua vez interpretados pelo Java gerando-se o código para o *Central Processing Unit* (CPU) específico. Esta grande vantagem poderá permitir executar a aplicação numa máquina qualquer como por exemplo um *Personal Digital Assistant* (PDA). A constituição da aplicação assenta em componentes *swing* e gestão de eventos 1.1. O módulo de software irá desempenhar o ciclo inverso do módulo VideoVigi. Após desencriptação cada GOB de cada pacote recebido será descomprimido de forma a obter os blocos originais. Estes serão reordenados por linhas para reconstruir a imagem. Esta aplicação é a segunda componente do sistema VideVigi que permite remotamente controlar e visualizar as imagens adquiridas pelas câmaras. Para isso a aplicação, desencripta, descomprime e apresenta as quatro imagens das camâras disponíveis. Também disponibiliza toda a interface gráfica com o utilizador (ver figura seguinte).

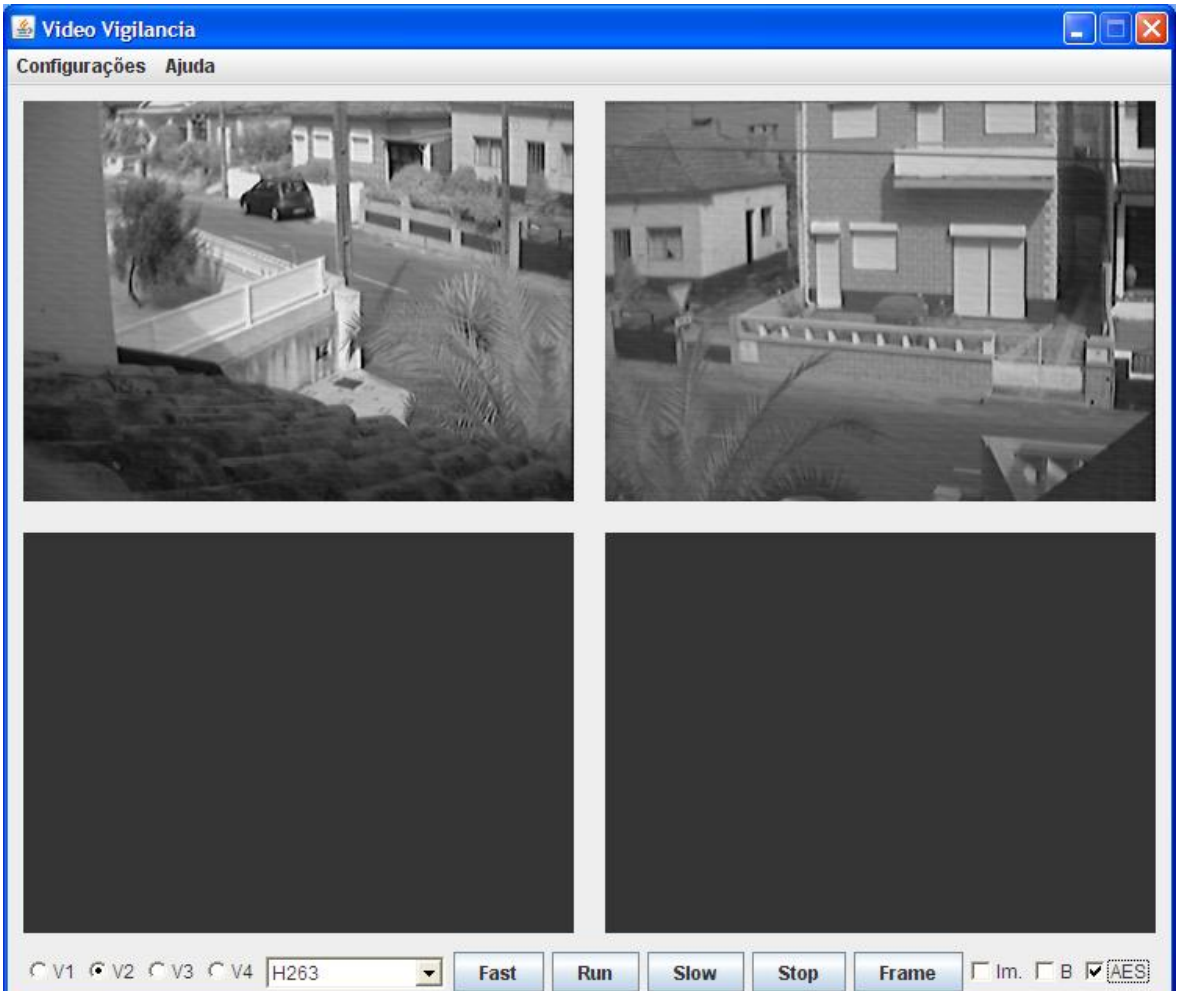


Figura 41 Aplicação VideVigi

O código foi desenvolvido num único ficheiro *VideoVigi.java*. Este pode ser executado como *applet*, a partir de uma página HTML ou *stand-alone*. Se for executado em *stand-alone*, o código começa pelo método *main(String[]args)*, neste caso instancia um objecto da *class* VideoVigi, coloca o campo deste objecto *TestaApplet* a *false* e executa o método *init()*. Este método vai instanciar um objecto da *class* JanelaVideoVigi passando como argumento o campo *TestaApplet*, que neste caso terá o valor *false*, de seguida executa o método *start()*. Se for executado como *applet*, o código começa a "correr" pelo método *init()*. Este método vai instanciar um objecto da *class* JanelaVideoVigi passando como argumento o campo *TestaApplet*, que agora terá o valor *true*, uma vez que foi neste caso, inicializado a *true*, de seguida executa o método *start()* (Tabela 24).

Tabela 24 Inicialização da aplicação

```
public class VideoVigi extends JApplet {
        boolean TestaApplet = true;
        JanelaVideoVigi janela = null;

        public void init(){
                janela = new JanelaVideoVigi(TestaApplet);
                janela.start();
        }

        public static void main(String[] args) {
                VideoVigi janelinha = new VideoVigi();
                janelinha.TestaApplet = false;
                janelinha.init();

        }
}
```

A variável *boolean* TestaApplet do campo de dados da classe principal, é utilizada para evitar que ocorra uma chamada ao sistema operativo (*EXIT_ON_CLOSE*) quando a janela é fechada, sempre que a aplicação for iniciada como *applet*, e no entanto permiti-lo se esta for iniciada em *stand-alone* (Tabela 25).

Tabela 25 Encerramento da Aplicação

```
if (TApplet) dispose();
else setDefaultCloseOperation(JFrame.EXIT ON CLOSE);
```

A classe *JanelaVideoVigi* é uma subclasse de *JFrame*. É esta classe que suporta toda a interface gráfica com o utilizador, ela tem dois menus (*Configurações e Ajuda*); um *JPanel* (*painel1*) com um gestor de posicionamento *GidLayout* (2x2), que recebe as quatro imagens; um *JPanel* (*painel2*) para todos botões (quatro *Radio Button* para a selecção de entrada de vídeo, um *Choice* para o tipo de compressão e os botões de controlo de tráfego). A classe *JanelaVideoVigi* implementa a interface *Runnable* para poder iniciar um *Thread.* O método *Start()* vai criar um *Thread* do objecto do objecto corrente (instancia de *JanelaVideoVigi)* e inicia a sua execução.

De forma a não perder o posicionamento de cada imagem dentro do *jpanel* principal (*jpanel1*), este tem em cada posição quatro novos *jpanel's* (*jpanel11, jpanel12, jpanel13* e *jpanel14*) que irão por sua vez receber os jpanel's que suportam as imagens.

Os quatro métodos *processImagem1/4* (Tabela 26) vão, não só colocar as imagens disponíveis no seu respectivo *jpanel* como também irão permitir as suas actualizações durante o funcionamento.

Tabela 26 ***processaImagem1()***

```
public void processaImagem1(){
        if (!inicio)painel11.remove(painel21);
        painel21 = new JPanel();
        painel21.setSize(336,256);
        compressao1 = LoadPanelBitmap(painel21,"image/imagem1.bmp");
         imagem1 = new JLabel(new ImageIcon(compressao1));
        painel21.add(imagem1);
        painel11.add(painel21);
        setVisible(true);
}
```

O método *loadPanelBitmap(jpanel, string)* invocado pelos quatro métodos *processaImagem*, permite visualizar ficheiros BMP num *jpanel*, este método foi encontrado num forum de *Java*. Após ter sido testado com sucesso, foi integrado na aplicação.

O modelo de eventos "1.1" foi utilizado na captura dos eventos. Os *radio button* que permitem escolher a entrada de vídeo também mostram a entrada actual, uma vez que estes são actualizados sempre que é recebida uma imagem. São constituídos por quatro objectos da classe *Checkbox* (*camara1, camara2, camara3 e camara4*) que fazem parte de um grupo (*CheckboxGroup*) chamado *camara*. Sempre que é pressionado um destes *Checkbox* é instanciado um objecto da classe TrataCamarax() (Tabela 27), de acordo com a opção escolhida. Como esta *class* implementa a interface *ItemListener*, o método *public void itemStateChanged(ItemEvent e)* vai verificar o estado do *Checkbox.* Se for *true*, envia a mensagem de configuração da entrada de vídeo.

Tabela 27 ***TrataCamara1()***

```
class TrataCamara1 implements ItemListener{
        byte camara=(byte)3, camara1=(byte)1, camara2=(byte)2, camara3=(byte)3,
        camara4=(byte)4;
        public void itemStateChanged(ItemEvent e){
                Boolean recebeCamara;
                if (e.getSource() instanceof Checkbox)
                {
                        Checkbox c = (Checkbox)e.getSource();
                        recebeCamara=c.getState();
                        if(recebeCamara){
                                {try    {enviaPacote(camara,camara1);
                                }catch (Exception e1){ e1.printStackTrace();}}
                        }
                }
        }
}
```

O *Choice choiceNivel* recebe as opções de compressão do sistema. O método *itemStateChanged(ItemEvent e)* trata qualquer alteração. Conforme a escolha do *Choice* (*H263 e Sem compressão*) é enviada uma mensagem de configuração de *compressao* com o valor correspondente a H263 ou sem compressão.

Os eventos dos cinco botões e dos *menuItem* são todos associados a objectos da classe *ProcessaBotao* que implementa o *ActionListener*, o objecto criado recebe um inteiro (de 0 a 7) que no método *actionPerformed(ActionEvent e)* vai permitir distingir o evento gerado (botão premido) e desta forma processar individualmente cada "instrução" (*Fast, Run, Slow, Stop, Frame, Inicializar* e *IP*).

O fluxograma da figura seguinte pretende ilustrar os passos desde a chegada de um datagrama UDP até à visualização da imagem.

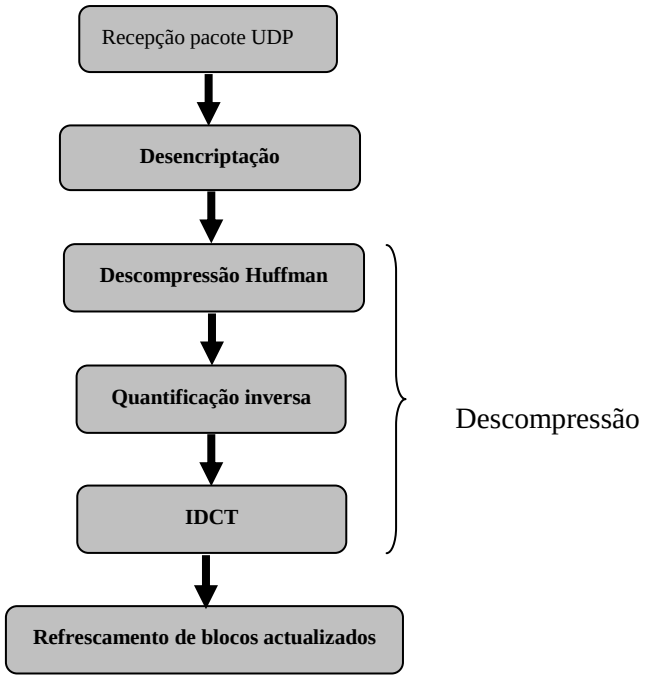


Figura 42 Fluxograma da descompressão

A aplicação cria um *Thread* (Tabela 28) que fica responsável pela recepção de pacotes.

Tabela 28 *Thread*

```
public void run(){
        while(!fim){
                try     {
                        recebePacote ();
                }
                catch (Exception e){   e.printStackTrace();}
        }
}
```

Quando é recebido um pacote é gerado um vector de 1458 *bytes* cujo conteúdo é copiado (após o *cast*) directamente do pacote recebido. Após a desencriptação, o método *actualiza(byte[])* vai analisar o pacote recebido; primeiro vai ver se pacote tem origem no módulo VideoVigi remoto (1º *byte* =0x88); se sim e se o *tráfego* for do tipo *controlado* envia um pacote para pedir ao módulo VideoVigi o envio de outro pacote. Depois vê a

entrada de vídeo a que diz respeito para actualizar o *Radio Button*, instancia o objecto *RandomAccessFile* para escrever no ficheiro correspondente à respectiva imagem. De seguida analisa a compressão, se for um pacote de um segmento de imagem sem compressão, actualiza na posição que diz respeito. Quando a imagem tiver completa invoca o método *processaImagemx()*, da imagem em causa; se for um pacote de um segmento de imagem com compressão, após a descompressão, por cada GOB retorna um *array* de 5376 bytes referentes às 16 linhas do segmento de imagem. O restante algoritmo é semelhante ao das imagens descomprimidas.

7.1. DESENCRIPTAÇÃO

À semelhança da encriptação, a desencriptaçao AES-128 também é executada de forma sequencial em grupos de 16 bytes (matriz de quatro por quatro – matriz *State*), utilizando a mesma chave da encriptação (criptografia simétrica). Considerando *Nb* o número de colunas na matriz *State* (igual a quatro), *Nk* o número de colunas na matriz *Key* (para uma chave de comprimento 128 bits é também igual a quatro) e *Nr* o número de ciclos efectuados durante a execução do algoritmo. O algoritmo recorre a quatro transformações orientadas ao *byte* (*InvSubBytes*, *InvShiftRows*, *InvMixColumns*, *AddRoundKey*) de acordo com o fluxograma da figura seguinte (ordem diferente da encriptação). Na desencriptação a chave expandida também é utilizada exclusivamente na função *AddRoundKey*. De acordo com a volta (*Round*) é utilizada uma submatriz da matriz expandida (*RoundKey*) desde as quatro primeiras linhas até às quatro últimas.

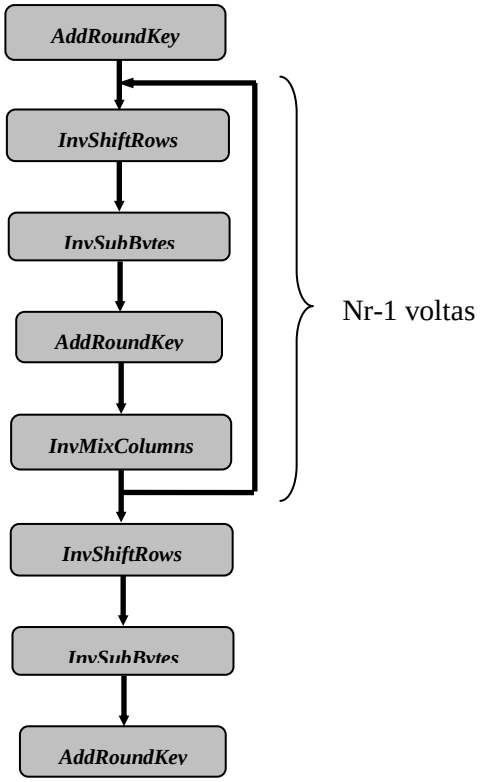


Figura 43 Fluxograma da desencriptação AES

7.1.1. KEYEXPANSION

Este método é igual à função da encriptação com o mesmo nome, uma vez que a chave expandida deve ser igual à da encriptação. A chave sofre uma expansão de forma a serem utilizados valores diferentes em cada uma das *Nr* volta (Figura 43). O primeiro passo na criptografia é portanto a função *KeyExpansion.* Para o AES-128 (utilizado) esta função irá expandir a chave de encriptação em 44 *words* (Nb*(Nr+4)) que irão desencriptar directamente a matriz de estado (*State*). Esta chave expandida será igual a da encriptação.

A ultima linha da matriz *Key* (quarta linha) é copiada para uma matriz linear de quatro colunas e sofre uma rotação de um byte para a esquerda. De seguida cada elemento da matriz temporária é substiuido, conforme o seu valor, por um elemento da matriz (*sBox*) numa transformação semelhante à descrita anteriormente na operação *SubsBytes*. As primeiras quatro linhas da matriz expandida (*RoundKey*) são as quatro linhas da matriz chave (*key*). A linha seguinte (quinta linha) é o resultado da operação XOR bit a bit entre a matriz temporária, a primeira linha da matriz expandida e a matriz *Rcon*(4) (0x01;0x00;0x00;0x00). A sexta linha é o resultado da operação XOR, bit a bit, entre a segunda e a quinta linha da matriz expandida. A sétima linha é o resultado da operação XOR, bit a bit, entre a terceira e a sexta linha da matriz expandida. A oitava linha é o resultado da operação XOR, bit a bit, entre a quarta e a sétima linha da matriz expandida.

Para as quatro linhas seguintes repetem-se os passos anteriores, tendo em conta que a nova matriz de entrada é aquela que se obteve anteriormente (quinta linha até à oitava). A primeira linha também terá um tratamento diferente, i.e. também terá um XOR com a matriz *Rcon*(8) (0x02,0x00,0x00,0x00). Estes passos repetem-se novamente para as quatro linhas seguintes até complementar 11 grupos de quatro linhas, i . e. as 44 linhas da matriz de chave expandida (*RoundKey*).

7.1.2. INVSUBBYTES

Na matriz estado (*State*), o método *void InvSubBytes()* substiui cada elemento por outro da matriz *sBoxInv*, conforme o seu valor, i.e. o valor do elemento da matriz estado (*State*) indica a posição na matriz (*sBoxInv*). Para esta operação inversa da *SubsBytes*, a matriz *sBoxInv* possui para cada valor a posição da matriz *sBox*. Como os elementos da matriz *State* são do tipo *byte* variam de –128 a 127, sempre que o valor seja negativo (primeiro bit a um) é necessário somar 256 para anular o sinal do elemento (Tabela 29).

Tabela 29 *InvSubBytes*

```
void InvSubBytes()
{
        int i,j;
        for(i=0;i<4;i++)
        {
                for(j=0;j<4;j++)
                {
                        if ((state[i][j])<0) state[i][j] =
(byte)(sBoxInv[(int)(state[i][j]+256)]);
                        else state[i][j] = (byte)(sBoxInv[(int)(state[i][j])]);
                }
        }
}
```

7.1.3. INVSHIFTROWS

O método *void InvShiftRows()* realiza simplesmente uma rotação contrária à função *ShiftRows* da encriptação, i.e. uma rotação de um byte para a direita na segunda linha matriz estado (*State*), uma rotação para a direita de dois bytes na terceira linha e uma rotação para a direita de três bytes na quarta linha.

7.1.4. INVMIXCOLUMNS

O método *void InvMixColumns()* realiza produtos de matrizes de cada coluna da matriz estado pela matriz quadrada definida na tabela seguinte. Os métodos *public int xtime(int x)* e *public int Multiply(int x, int y)* estão definidos no standard AES (na secção 4.2 e 4.3) [9].

Tabela 30 *InvMixColumns*

```
public void InvMixColumns(){

        int i;
        int a,b,c,d;
        for(i=0;i<4;i++){

                a = state[0][i];
                b = state[1][i];
                c = state[2][i];
                d = state[3][i];

                state[0][i] = (byte)(Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c,
0x0d) ^ Multiply(d, 0x09));
                state[1][i] = (byte)(Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c,
0x0b) ^ Multiply(d, 0x0d));
                state[2][i] = (byte)(Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c,
0x0e) ^ Multiply(d, 0x0b));
                state[3][i] = (byte)(Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c,
0x09) ^ Multiply(d, 0x0e));
        }
}
```

7.1.5. ADDROUNDKEY

No método *void AddRoundKey(int round)*[6] conforme o *round* onde se encontra o código, a matriz de estado sofrerá uma operação XOR elemento a elemento entre a matriz da chave expandida (*RoundKey*) e a própria matriz de estado (Tabela 31).

Tabela 31 *AddRoundKey* – XOR elemento a elemento

```
public void AddRoundKey(int round){
        int i,j;
        for(i=0;i<4;i++)
        {
                for(j=0;j<4;j++)
                {
                        state[i][j] ^= RoundKey[(int)(round * Nb * 4 + i * Nb + j)];
                }
        }
}
```

[6] Este método é igual a função com o mesmo nome na encriptação.

7.2. DESCOMPRESSÃO

A descompressão obedece a algoritmos semelhantes à compressão, mas com alterações tanto na ordem em que os algoritmos são utilizados como na forma em que são implementados.

7.2.1. DESCOMPRESSÃO HUFFMAN

O primeiro passo da descompressão é referente ao algoritmo Huffman que é executado pelo método Huffman para os quatro GOBs de cada pacote. Este método, à semelhança dos outros desenvovidos de raiz, recebe um *array* de *bytes* comprimidos com este algoritmo VLC e retorna um outro *array* com os *bytes* correspondentes descomprimidos. À semelhança da compressão o código necessita das tabelas dos coeficientes AC e DC mas, no entanto, não precisa das tabelas de comprimento. Enquanto na compressão é necessário conhecer o comprimento do código para realizar um *shift* correspondente, na descompressão basta efectuar uma leitura bit a bit comparando progressivamente o resultado com o valor da tabela correspondente (AC ou DC). Por outro lado, em termos de algoritmo, ao contrário da compressão, a determinação do valor na descompressão é mais fácil de executar lendo o valor na tabela de valores pelo que houve necessidade de criar uma tabela de valores Huffman bidimensional em que a primeira dimensão representa a linha e a segunda a posição. A leitura de cada bit e respectiva colocação em série por forma a tornar possível a comparação do valor actual com os das tabelas referidas é efectuada pelo método *leBit(byte insuflar[])* (Tabela 32).

Tabela 32 Leitura de bit

```
public void leBit (byte inBuffer []){
        byte valorBit=0, Mascara_1=1;
        if (shift > 0){
                valorBit = (byte)(inBuffer[in]>>shift) ;
                valorBit = (byte)(valorBit & Mascara 1);
                shift=shift-1;
        }
        else {
                valorBit = (byte)(inBuffer[in]& Mascara_1);//bit menos signif.(ultimo bit)
                shift=7;in=in+1;                           //proximo bit esta no proximo byte
        }
        valorTestar = (short)(valorTestar<<1) ;
        valorTestar = (short)(valorBit | valorTestar);
}
```

Após o conhecimento do coeficiente, sabe-se a linha e, consequentemente, também o comprimento dos bits para a posição. A leitura dos bits seguintes, igual ao número da

linha, dá a posição. Conhecendo-se a linha e posição, para determinar o valor descomprimido basta lê-lo na tabela Huffman (Tabela 33).

Tabela 33 Leitura da tabela Huffman

```
for (n=0;n<linha;n++) leBit(inBuffer);//vai entao ler a posicao que fica em valorTestar
outBuffer[out]=tabelaValores[linha][valorTestar];
```

O algoritmo desenvolvido faz uma comparação directa de valores apesar de não haver qualquer ambiguidade nas leituras, existem duas excepções que começam com o valor zero (010 e 011) que podem ser confundidos com 10 e 11 respectivamente. Estes dois valores da componente DC assim como o EOB são, por essa razão, tratados de forma diferente, i.e. no ínicio da leitura de um novo valor se o primeiro bit e o segundo forem zeros é um EOB, se o segundo bit for um e o terceiro bit zero é a primeira linha da componente DC se o último bit for um é a segunda linha. Em relação à componente AC após a determinação do valor (entre 161 valores possíveis) para cada grupo de 10 é incrementado o número de zeros (de 0 até 15). Esta quantidade de zeros serão precedidos do valor da componente AC cuja linha será determinada após retirar um múltiplo (inteiro) de 10 do valor encontrado.

7.2.2. QUANTIFICAÇÃO INVERSA

O *array* retornado pelo método Huffman possuiu os *bytes* descomprimidos os quais representam 352 blocos dos quatros GOBs de cada pacote (88x4). A quantificação inversa e a transformada de cosenos discreta inversa é realizada pelo método *descomprime(byte[])*, esta, como já foi referido, tem um algoritmo semelhante à compressão. A descompressão, da mesma forma que a compressão vai ser aplicada individualmente a cada bloco. Para a quantificação inversa, cada matriz bloco comprimida é multiplicada elemento a elemento pela matriz de quantificação utilizada na compressão. Como existem duas possíveis matrizes de quantificação que no protocolo definem o nível de compressão, o byte *pMasc* é actualizado de acordo com o GOB a descomprimir de forma a permitir ler no *byte* 4 (5º *byte*) do pacote o bit referente à matriz de quantificação utilizada para o GOB em causa (Tabela 34).

Tabela 34 Produto pela matriz de quantificação

```
while((int)msg[in]!= -128){
        if ((msg[4]& (byte)pMASC)==0)
                MatCompleta[i] = ((int)msg[in])*MatQ[i];
        else
                MatCompleta[i] = ((int)msg[in])*MatQ3[i];
```

```
        i++;in++;
}
```

Após o carácter EOB (-128) a matriz é complementada com zeros até perfazer os 64 elementos. Finalmente a matriz completa na ordem ziguezague é reordenada na ordem "normal" (sequencial) recorrendo-se para isso a uma matriz que contém a ordem ziguezague (Tabela 35).

Tabela 35 Finalização da quantificação

```
for (n=i;n<64;n++){
        MatCompleta[n]=0;                              //matriz completa na ordem ziguezague;
}
for (n=0;n<64;n++){
        MatNormal[n]=MatCompleta[MatZigzag[n]];      //matriz na ordem normal;
}
```

Os blocos que na compressão não foram actualizados ficam inteiramente completos com 64 zeros.

7.2.3. IDCT

Cada bloco de imagem encontra-se agora completo nas suas componentes espectrais. Através da IDCT serão transformados em blocos de imagem no domínio espacial de acordo com a segunda expressão da Figura 9. A expressão usada no algoritmo para este efeito (7) é semelhante à da compressão, mas, a matriz dos cosenos é a transposta da matriz dos cosenos da compressão.

$$IDCT = 1/4 \,.\, M_{cos}{}^{T} \times B \times M_{cos} \,. \qquad (7)$$

$$B(i,j) = B(i,j) + 128 \,. \qquad (8)$$

Para cada bloco a IDCT é um quarto do produto da matriz dos cosenos transposta pela matriz bloco (Tabela 36), e o produto da matriz resultante pela matriz dos cosenos. À semelhança da compressão, como existe apenas uma matriz dos cosenos, o produto pela transposta foi substituido por um produto "linha por linhas". No final dos produtos a matriz é então dividida por quatro e soma-se 128 para voltar ao formato 0-255 (expressão 8).

Tabela 36 Produto da matriz dos cosenos transposta pela matriz bloco (2º produto)

```
for (linha=0;linha<8 ;linha++ ){
    for (coluna=0;coluna<8;coluna++){
        for (j=0;j<8;j++){                              //todos os elementos da linha
           Temp = MatTemp1[j+linha*8]*MatA[j+coluna*8];//linha por linha por ser transposta
           MatTemp2[coluna+linha*8] = MatTemp2[coluna+linha*8] + Temp;
        }
   }
```

```
}
```

Para completar a IDCT o *buffer* que contém as matrizes bloco do GOB deve ser reordenado por linhas para a actualização da imagem. Para isso, para cada uma das 16 linhas referentes a dois blocos, a matriz é lida por linhas; são lidos 8 píxeis de cada vez, de cada uma das 44 colunas e reescrita na ordem normal (Tabela 37).

Tabela 37 Reescrita do GOB na ordem normal

```
for (linha=0;linha<16;linha++){              //para todas as linhas
   for (coluna=0;coluna<44;coluna++){       //para todas as colunas referentes a cada bloco
        for (i=0;i<8;i++){                   //para todos os píxeis da linha de cada bloco
             MatDescomprimida[out]=MatDescompBlocos[i+coluna*128+linha*8];
             out++;
        }
   }
}
```

7.2.4. REFRESCAMENTO DE BLOCOS ACTUALIZADOS

À medida que as 16 linhas de um GOB completo são descomprimidas, ou as quatro linhas, da imagem não comprimida são recebidas, os píxeis são actualizados da direita para a esquerda, de baixo para cima a partir da posição recebida. Quando for actualizado o último segmento ou grupo de quatro linhas (imagem não comprimida), a imagem BMP é novamente "carregada".

Como os píxeis referentes aos blocos que na compressão não foram actualizados ficaram com o valor zero. Estes numa situação normal não são actualizados na imagem. No entanto, para efeitos de teste foi disponibilizado uma *Checkbox* a qual actua directamente no campo *testeBlocos* de forma a permitir visualizar os blocos não actualizados (ficam a negro).

Tabela 38 Refrescamento de blocos

```
if (((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
```

O software cliente desenvolvido, devidamente comentado, pode ser consultadas no Anexo D.

8. DESEMPENHO DO SISTEMA

O estudo desta tese teve como suporte de desenvolvimento o VideoVigi. Trata-se de um sistema de baixo custo vocacionado para redes de baixo débito. Apesar de todo o código desenvolvido ter por base este sistema, também ele desenvolvido de raiz, é importante perceber que algumas destas adaptações poderão facilmente ser utilizadas em novos sistemas de videovigilância. No que concerne o enquadramento, entenda-se por sistemas de baixo custo como sendo aqueles que utilizam um DSP com uma capacidade de processamento, na ordem das 30 MIPS. Em relação às redes de baixo débito consideram-se com valores até 64 kbps. A avaliação de desempenho dos algoritmos vão portanto ser realizados neste sistema de baixo custo sobre uma rede de 64 kbps de débito.

8.1. ANÁLISE SISTEMA

No desenvolvimento de um sistema para funcionar em tempo real, a primeira preocupação a ter em conta é o número de milhões de operações por segundo (MIPS) do DSP. Esta frequência vai impor um determinado tempo de compressão e de encriptação. Mas a frequência pode não ser a única responsável pelo aumento ou diminuição desse tempo. O algoritmo também o pode influenciar fortemente. Além disso também é importante analisar

a qualidade de imagem, tendo em conta a aplicação pretendida, e o nível de compressão de forma a viabilizar a rede utilizada. Então são três as variáveis que podem determinar a viabilidade de um sistema de videovigilância.

Como a compressão depende da quantidade de movimento na imagem, para a análise da compressão e da qualidade, a imagem de referência utilizada é o movimento dos ponteiros de um relógio. A primeira imagem (figura seguinte) foi adquirida sem compressão, a segunda com compressão, a terceira com encriptação e a última, embora também tenha sido enviada com encriptação, foi activada a desencriptação na aplicação cliente.

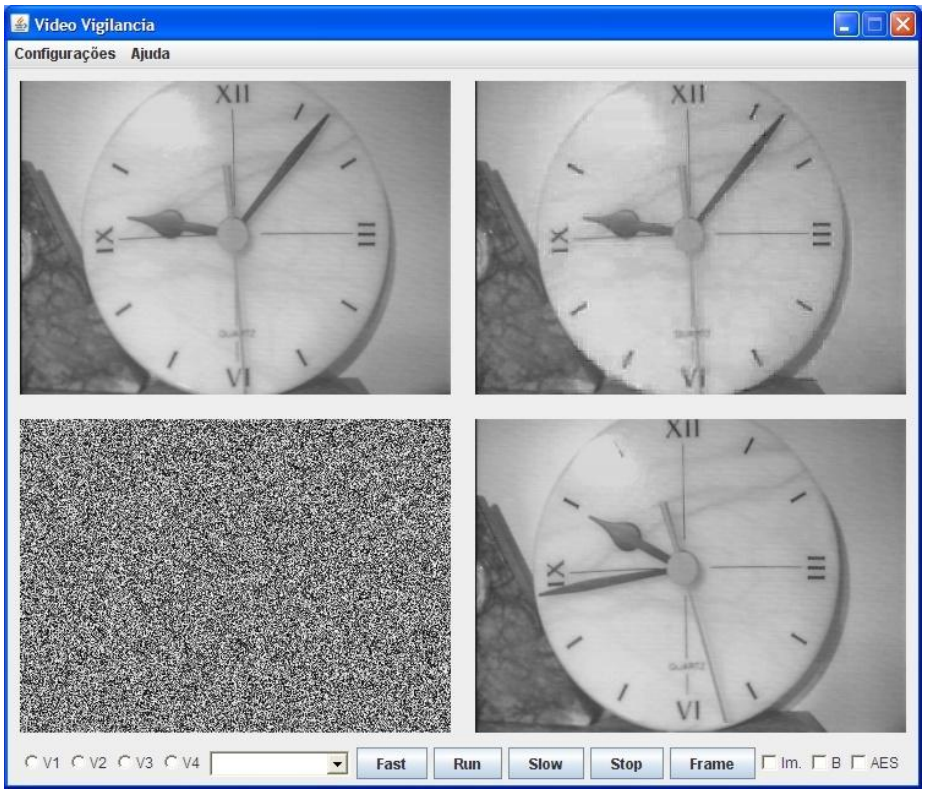


Figura 44 VideoVigi 1) sem compressão 2) H263 3) AES 4) Decifrada

Para analisar os tempos do sistema *VideoVigi*, recorreu-se a um osciloscópio Tektronix® TDS 3054B *Digital Phosphor Oscilloscope* (DPO). Este osciloscópio de 4 canais possui uma frequência de amostragem de 5GS/s e está especificado para sinais até 500 MHz.

8.1.1. TEMPOS DE COMPRESSÃO

Os tempos de aquisição e compressão de imagens dizem respeito ao DSP (ADSP2181-33 MIPS), uma vez que é este componente que suporta todo o processamento relacionado com a imagem. Os testes iniciais dizem respeito apenas a compressão da redundância espacial

(MJPEG sem Huffman). Na Figura 45 o canal 1 sinaliza a compressão ao nível lógico 1, o canal 2 sinaliza a captura, o canal 3 é o sinal de vídeo e o canal 4 é o sinal /W (*write*) do FIFO. O tempo em que o canal 1 está a um corresponde ao tempo de transferência da imagem para a memória interna do DSP (na ordem dos blocos) e ao respectivo tempo de compressão. Este tempo é de 116 ms, no formato 352x256, e o tempo total de aquisição, transferência e compressão é 160 ms.

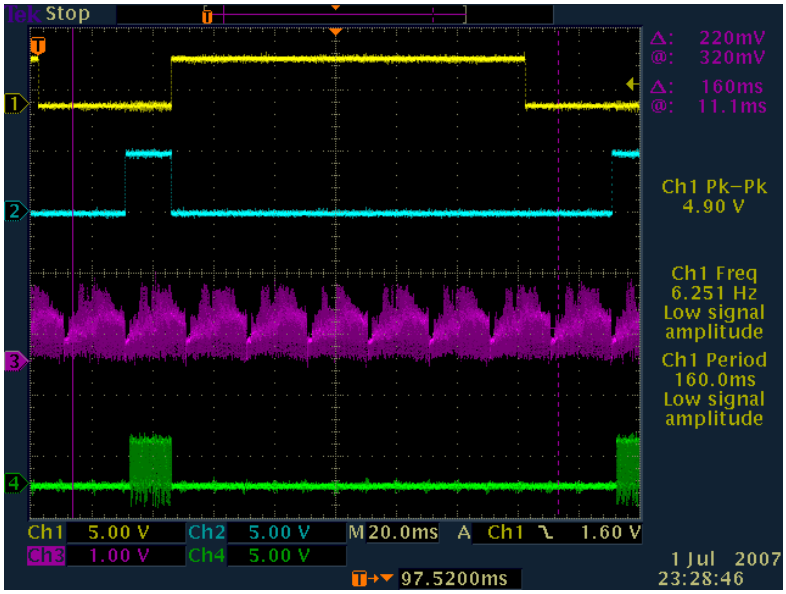


Figura 45 Aquisição e Compressão MJPEG

Contando com todo o tempo, desde a aquisição até à compressão, o DSP pode comprimir em MJPEG 6,25 imagens por segundo (1/0,160). Este resultado apesar de parecer satisfatório no que se refere ao tempo, não o é em relação ao nível de compressão (cerca de 80 % - imagem cinco vezes menor). Para uma rede de 64 kbps uma imagem completa levaria cerca de 2 segundos a ser actualizada.

Os próximos testes (Tabela 39) correspondem às três situações que interessam comparar (sem compressão, compressão parcial e compressão completa). Como a imagem dispõe de apenas 256 linhas, para a centralizar, na aquisição são lidas 272 linhas, as primeiras 16 são desprezadas; o que fez aumentar ligeiramente o tempo de aquisição para 49 ms. Em relação à compressão foi implementada a ordenação em blocos (desprezada nos testes iniciais) e o algoritmo de "compressão dinâmica", i.e. alteração da matriz de quantificação conforme a necessidade de maior ou menor compressão. No que diz respeito ao algoritmo completo (H263 adaptado) foi acrescentado o "algoritmo das diferenças" e o algoritmo Huffman em relação à compressão parcial. Os dados obtidos estão na tabela seguinte.

Tabela 39 Tempos compressão por imagem

	Sem Compressão	Compressão parcial (MJPEG)	Compressão completa
Aquisição Síncrona (ms)	40	49	32
Compressão e Preparação imagem (ms)	13	123	117
Leitura IDMA (ms)	1050	189	7
Tempo total (ms)	1103	361	156

Comparando os valores para cada uma das três situações notam-se algumas diferenças esperadas e outras surpreendentes.

A aquisição da imagem varia em cada uma das três situações. A razão para este facto deve-se à necessidade de realizar a aquisição com sincronismo (sempre no início do mesmo *field*). Logo conforme o instante em que deve ser iniciada a aquisição será necessário esperar pelo início do *field* escolhido (*field* ímpar). Como a aquisição das primeiras 272 linhas de cada *field* tem uma duração próxima de 20 ms, esta terá naturalmente um tempo variável entre cerca de 20 ms (melhor caso) e 60 ms (pior caso).

O tempo de compressão e preparação da imagem para o caso "sem compressão" não é aproximadamente nulo porque a imagem deve ser toda re-escrita fazendo sincronismo linha a linha, por software. A compressão completa é mais rápida que a compressão parcial. Apesar de a compressão completa ter o "algoritmo das diferenças" e a compressão Huffman inexistentes na compressão parcial, para uma imagem com pouco movimento (como a imagem de referência) o "algoritmo das diferenças" vai anular a quantificação referente aos blocos sem movimento. Como o algoritmo Huffman requer relativamente poucas instruções o resultado final pode conduzir a uma diminuição do tempo de compressão. No pior caso (imagem com movimento em todos blocos) o tempo será ligeiramente superior.

A leitura do *bus* IDMA por parte do microcontrolador serve para transferência das imagens do DSP para o microcontrolador. Esta depende do tamanho da informação a ser lida, é neste passo que se começa a sentir a vantagem da compressão completa mesmo em relação à parcial. O reduzido tempo de leitura da imagem comprimida (7 ms) mostra claramente a elevada compressão que se atingiu.

8.1.2. TEMPOS ENCRIPTAÇÃO

Apesar de não se terem efectuados testes na prática com o algoritmo DES, vários autores referem que pelo facto do AES ter um algoritmo baseado em transformações orientadas ao *byte*[7] este torna-se mais rápido. Além disso oferece uma segurança incomparavelmente superior (2^{128} vs. 2^{56} combinações de chave). Esta última razão é por si só suficiente para eliminar a necessidade da realização de qualquer teste em DES.

Os tempos de encriptação do AES dizem respeito ao microcontrolador (DS80C400-37,5MHz) uma vez que é este componente que processa a encriptação de imagens.

Tabela 40 Tempos encriptação por imagem

	Sem Compressão	Compressão completa
Tempo encriptação AES (ms)	50240	178

Os tempos de encriptação são muito elevados, essencialmente no caso de imagens não comprimidas. Este facto deve-se por um lado à elevada complexidade do algoritmo AES e por outro ao baixo número MIPS do microcontrolador (variável de 2 a 9). Mesmo desprezando o tempo de envio de uma imagem por uma rede de baixo débito, sem compressão uma imagem completa demora perto de um minuto a ser encriptada.

8.1.3. QUALIDADE IMAGEM

Desprezando o ruído analógico devido a interferências, o ruído nas imagens tem duas componentes, o ruído por quantificação e o ruido devido à compressão com perdas. Embora o sistema tenha três tipos diferentes de compressão, apenas a compressão da redundância espacial e temporal apresentam perdas.

Tomando a imagem sem compressão como referência para avaliar a qualidade da compressão, pode-se estimar a qualidade da compressão calculando o *Peak Signal-to-Noise Ratio* (PSNR). Por definição o PSNR para uma imagem a 8 bits é dado pela expressão seguinte [4].

[7] O algoritmo DES faz operações ao *bit*

$$PSNR = 10 \log \left\{ \frac{255^2}{1/NM \sum_{n_1=1}^{N} \sum_{n_2=1}^{M} [f(n_1,n_2) - \hat{f}(n_1,n_2)]^2} \right\} \quad (9)$$

Onde $f(n_1,n_2)$ e $\hat{f}(n_1,n_2)$,$1 \leq n_l \leq N$, $1 \leq n_2 \leq M$, são os píxeis da imagem original e da imagem comprimida com o tamanho N x M respectivamente. O segundo termo por baixo da fracção é conhecido por *Mean Squared Error* (MSE). Uma imagem de boa qualidade apresenta um PSNR que varia entre 30 e 40 dB [4], mas quanto maior o valor melhor será a qualidade.

Tendo o cuidado de capturar as duas primeiras imagens na mesma posição do relógio; para calcular o erro quadrático médio (MSE) bastaria calcular as diferenças entre cada píxel, somar todas as diferenças e elevar o resultado ao quadrado. Substituindo na expressão anterior, obtinha-se o PSNR. Apesar da baixa resolução das imagens, ainda assim seria necessário efectuar 90112 cálculos (352x256). Para calcular o PSNR foi então utilizado uma aplicação *freeware* (MSU-*Video Quality Measurement Tool* - versão 1.52). Como foi analisada apenas uma *frame* o resultado é fixo e tem o valor de 35,82 dB (figura seguinte). O que significa que a imagem tem uma qualidade bastante aceitável.

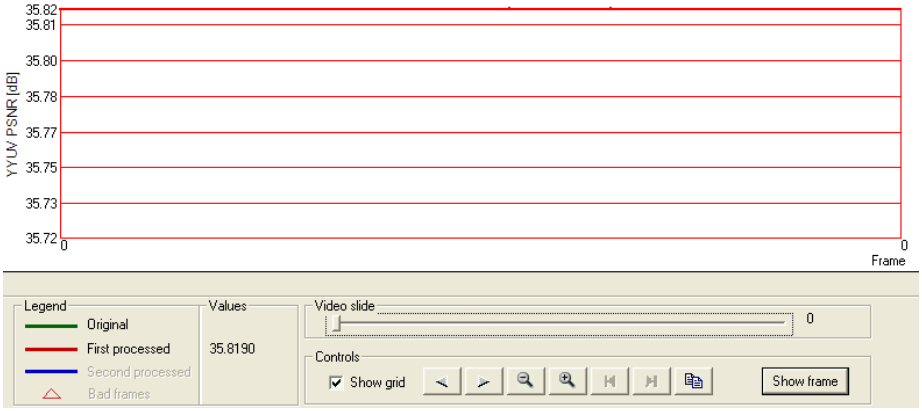


Figura 46 PSNR da imagem de referência

8.2. DESEMPENHO EM AMBIENTE SEMI-REAL

Para testar o sistema numa rede de baixo débito recorreu-se a um *pDSLAM8*[8] da PTInovação. Dado que os débitos na Internet variam conforme o número de utilizadores, para não haver ambiguidade, os testes serão efectuados numa rede isolada, i.e. sem ligação ao *Internet Service Provider* (ISP).

Para analisar o tráfego recorreu-se a uma aplicação *freeware* o *Packetyzer* versão 5.0.0. O sistema *VideoVigi* foi ligado a um *router* ADSL da primeira interface do *pDSLAM8* e o PC com o *sofware* cliente à segunda interface do *pDSLAM8* de acordo com a figura seguinte.

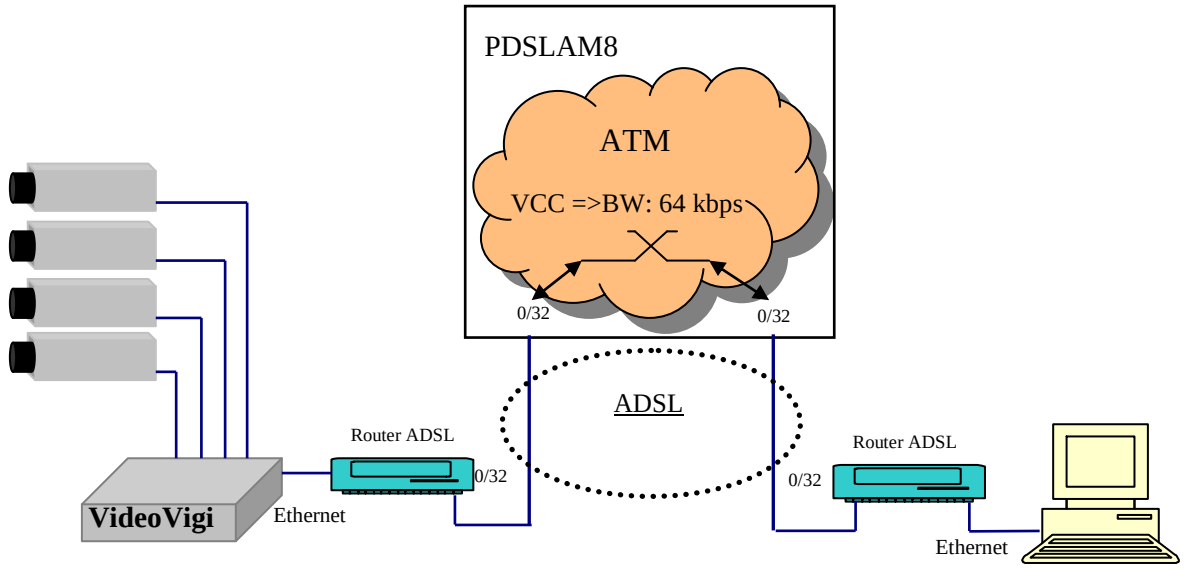


Figura 47 Diagrama de ensaio

Na configuração do pDSLAM8 utilizou-se um descritor de tráfego ATM com um *Peak Cell Rate* (PCR) igual a 175. Como cada célula ATM tem cerca de 47 *Bytes* de dados, teremos um *bandwidth* (BW) de 8178 Bps (174*47), i.e. cerca de 64 kbps. A configuração do pDSLAM8 e dos routers ADSL pode ser consultada no Anexo E.

8.2.1. TRÁFEGO

Os valores obtidos na primeira situação (Figura 48) referem-se ao caso "sem compressão" e "sem encriptação". A transmissão no modo controlado (*run*) envolve a emissão de um

[8] O pDSLAM8 é o sistema DSLAM de menor capacidade desenvolvido pela PTInovação que disponibiliza oito interfaces ADSL aos clientes e cuja interface *uplink* são dois E1 (PDH) ou uma interface Ethernet.

pacote sempre que a aplicação recebe um pacote de imagem (ver Figura 34), como cada imagem sem compressão é constituída por 64 pacotes, contando com o pacote de resposta serão um total de 128 pacotes por imagem. Consultando os valores obtidos (primeiro caso da Figura 48), durante 64 segundos foram transferidos 507 pacotes, o que prefaz uma taxa de 0,0619 imagens por segundo ((507/128) /64), i.e. uma imagem demora cerca de 16 segundos a ser transmitida.

Na situação seguinte (sem compressão mas com encriptação) os testes foram efectuados em modo *fast*. O número total de pacotes transaccionados por imagem é, neste caso, 64. Consultando os valores obtidos (segundo caso da Figura 48), durante 62 segundos foram transferidos 73 pacotes, o que prefaz uma taxa de 0,0184 imagens por segundo ((73/64) /62), i.e. uma imagem demora cerca de 54 segundos a ser transmitida.

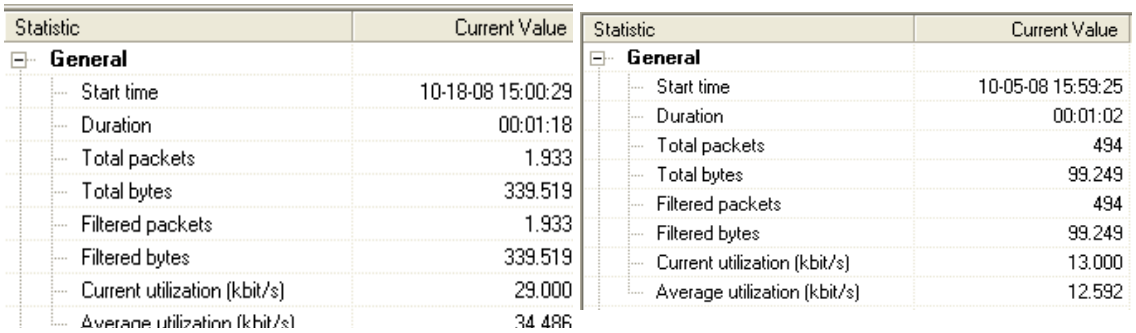
Statistic	Current Value
General	
Start time	10-05-08 15:59:25
Duration	00:01:04
Total packets	507
Total bytes	378.378
Filtered packets	507
Filtered bytes	378.378
Current utilization (kbit/s)	48.000
Average utilization (kbit/s)	46.922

Statistic	Current Value
General	
Start time	10-05-08 15:59:25
Duration	00:01:02
Total packets	73
Total bytes	106.288
Filtered packets	73
Filtered bytes	106.288
Current utilization (kbit/s)	11.000
Average utilization (kbit/s)	13.715

Figura 48 Tráfego sem compressão (sem e com AES respectivamente).

Em relação a situação "com compressão" (primeira situação da figura seguinte), em modo *fast* o número total de pacotes transaccionados por imagem é 4. Como a compressão depende da quantidade de movimento da imagem foi utilizado, como já foi referido, o movimento dos ponteiros de um relógio (Figura 44). O resultado é 6,20 imagens por segundo ((1933/4) /78).

Com compressão e encriptação (situação seguinte), também em modo *fast*, o número total de pacotes transaccionados por imagem é 4. O que dá 2,00 imagens por segundo ((494/4) /62).



Statistic	Current Value
General	
Start time	10-18-08 15:00:29
Duration	00:01:18
Total packets	1.933
Total bytes	339.519
Filtered packets	1.933
Filtered bytes	339.519
Current utilization (kbit/s)	29.000
Average utilization (kbit/s)	34.486

Statistic	Current Value
General	
Start time	10-05-08 15:59:25
Duration	00:01:02
Total packets	494
Total bytes	99.249
Filtered packets	494
Filtered bytes	99.249
Current utilization (kbit/s)	13.000
Average utilization (kbit/s)	12.592

Figura 49 Tráfego com compressão (sem e com AES respectivamente).

Comparando as quatro situações (Tabela 41), pode-se tirar uma conclusão de imediato. A encriptação não influencia a compressão mas o seu algoritmo é pesado para ser realizado por um microcontrolador com uma frequência de relógio interna de 37,5 MHz; o que origina um atraso excessivo. Este atraso baixou o débito e consequentemente o número de imagens por segundo em cerca de 66%. Em relação à compressão, para a imagem do relógio (imagem suave com pouco movimento) a compressão atingiu um valor muito elevado; o tamanho dos dados diminuiu 135 vezes, o que significa que a compressão atingiu o valor de 99,3 % (100 - 0,70/94,75x100).

Tabela 41 Ensaios Sistema VideoVigi

	c/ AES s/ H263	s/ AES s/ H263	c/ AES c/ H263	s/ AES c/ H263	c/ AES c/ H263 (após alteração)
Imagens/s	0,02	0,06	2,00	6.20	2.91
Tráfego (kbps)	13,72	46,92	12,59	34.49	17.78
Imagem (kbyte)	93,21	94,75	0,79	0,70	0.83

Apesar dos bons resultados nota-se que apenas numa situação houve uma utilização do débito próximo do máximo da rede, logo pode-se tentar melhorar os resultados resolvendo dois possíveis problemas, i.e. o atraso na leitura do *bus* IDMA e o elevado tempo encriptação dos dados.

Em relação ao *bus* IDMA, uma possível solução seria uma alteração de *hardware*. Usando uma *Dual Port Ram* (DPRAM) para transferência dos dados do DSP para o microcontrolador, com um tempo de acesso de 12 ns limitaria o tempo de transferência dos dados à velocidade máxima do microcontrolador. Como o microcontrolador executa cada instrução MOVX em oito ciclos de relógio cada imagem comprimida seria transferida do

DSP para o microcontrolador em cerca 0,15 ms (700x8/37,5 $x10^6$). O tempo de transferência IDMA seria praticamente anulado. Mas este representa apenas 4,4 % do tempo total, teria um "impacto" muito baixo. Em relação a imagens sem compressão, o débito está a ser limitado pela rede e não pela leitura do bus IDMA. O facto de não atingir o débito máximo da rede está relacionado com o *overhead* IP, os bytes de controlo e o tempo de espera dos pacotes no modo controlado. Então esta alteração para redes de baixo débito não trás vantagens significativas.

Em relação à encriptação, para melhorar o seu tempo poder-se-ia pensar em realizá-la pelo DSP em vez do microcontrolador. Mesmo sem considerar instruções múltiplas, o DSP executa 33 milhões de instruções por segundo, enquanto o microcontrolador executa apenas 2 a 9 milhões de instruções por segundo. Com esta alteração o tempo de encriptação diminuiria pelo menos 75%. Apesar de haver uma apreciável melhoria no tempo de encriptação, que seria muito interessante para imagens não comprimidas, perder-se-ia a possibilidade de realizar a encriptação em simultâneo com a compressão do DSP, o que trás uma vantagem muito significativa em imagens comprimidas. Como este sistema foi concebido para ser utilizado com compressão, porque de outra forma o refrescamento de imagens numa rede de baixo débito (mesmo sem encriptação) é desmasiado baixo (16 segundos), então esta é a alteração a implementar.

Após esta alteração foram efectuados testes com encriptação e sem compressão (primeira situação da figura seguinte). Como era esperado, nesta situação, o resultado é aproximadamente igual ao anterior 0,02 imagens por segundo ((71/64) /59). Na segunda situação (com encriptação e com compressão) obteve-se um resultado de 2,91 imagens por segundo ((687/4) /59).

Esta situação (com compressão) beneficia de facto muito da compressão e encriptação simultânea. Houve um acréscimo no número de *frames* por segundo em cerca de 50 %. Como o tempo de encriptação é ligeiramente superior ao tempo de compressão (178 ms vs. 156 ms), com esta alteração o tempo total passou a ser apenas o tempo de encriptação mais o tempo de transferência pela rede de baixo débito. Comparando o resultado com esta alteração ao resultado anterior, verifica-se que existe coerência entre o tempo de transferência pela rede de baixo débito na situação anterior, i.e. 166 ms (500-178-156) e o tempo actual 166 ms (344-178). Esta conclusão confirma o previsto, i.e. houve um "anulamento" do tempo de compressão. Claro que numa situação de muito movimento

poderá não existir anulamento no entanto o tempo ficará sempre reduzido do valor de 178 ms.

Statistic	Current Value
General	
Start time	10-18-08 15:00:29
Duration	00:00:59
Total packets	71
Total bytes	103.376
Filtered packets	71
Filtered bytes	103.376
Current utilization (kbit/s)	11.000
Average utilization (kbit/s)	13.820

Statistic	Current Value
General	
Start time	10-18-08 15:00:29
Duration	00:00:59
Total packets	687
Total bytes	132.651
Filtered packets	687
Filtered bytes	132.651
Current utilization (kbit/s)	16.000
Average utilization (kbit/s)	17.781

Figura 50 Tráfego com encriptação[9] (sem e com compressão respectivamente).

[9] Após a alteração que permite encriptar e comprimir em simultaneo.

9. CONCLUSÕES

Apesar da aparente oposição de compatibilidades existente no desenvolvimento de sistemas de videovigilância de baixo custo, para utilizar em redes de baixo débito, ficou demonstrado que, recorrendo a determinados mecanismos, não só é possível, satisfazer estas condições, como até se complementam. Quando se fala em redes de débito elevado também se exige uma elevada qualidade tanto no formato como no número de imagens por segundo. Pelo contrário em redes de baixo débito aceita-se com naturalidade um formato de vídeo mais reduzido, assim como um número mais baixo de *frames* por segundo. Por outro lado, um sistema de baixo custo não tem capacidade para fazer uso de uma rede de débito elevado, i.e. não tem processamento nem memória para tratar um formato de vídeo de elevada resolução mantendo um número de *frames* por segundo aceitável.

Sendo assim, no desenvolvimento de soluções de videovigilância de baixo custo é necessário definir o formato de vídeo e o número de *frames* por segundo, ponderando as necessidades concretas da aplicação, e, em função destas, optimizar os recursos e tomar decisões ao nível das especificações de hardware. A escolha destes dois parâmetros vai determinar duas necessidades, que são as MIPS e a RAM do DSP. Mas nestes sistemas, orientados para redes de baixo débito, o número de *frames* por segundo depende também da largura de banda (BW) e, de uma forma especial, do algoritmo de compressão implementado.

9.1. FORMATO VÍDEO

Para a maioria das aplicações de videovigilância, e tendo em conta o algoritmo desenvolvido, em imagens a preto e branco, o formato de vídeo CIF (352x288) é o mais adequado em sistemas de baixo custo; uma vez que o formato 4CIF (704x576) para além de exigir pelo menos 512 kbytes de RAM, necessita de quatro vez mais MIPS para manter o mesmo número de *frames* por segundo. No formato CIF, para este algoritmo, o DSP necessita de apenas 128 Mbytes de RAM e as 33 MIPS são suficientes para comprimir cerca de oito imagens por segundo (1/0,120), embora na prática este valor baixe devido a tempos de aquisição e transferências de dados (internos e externos).

Para imagens a cores são necessários 256 kbytes de RAM; para tratar a crominância do vídeo no formato *YCrCb*, numa sub-amostragem 4:2:2 ou 4:1:1[10], mantendo o mesmo algoritmo. A contrapartida seria uma redução no número de *frames* por segundo em 50% ou 25%. Como alternativa, e para aplicações onde a cor e número de *frames* por segundo é mais importante que a resolução, poder-se-ia utilizar o formato QCIF (176x144) o que neste caso traria um acréscimo de 100% ou 133% no número de *frames* por segundo, respectivamente.

Enquanto que a RAM e o número MIPS do DSP podem ser dimensionados durante o desenvolvimento, a rede não é uma variável, e neste sentido, considerando os 64 kbps da rede, o número de *frames* por segundo vai então depender de uma segunda condição, i.e. o nível de compressão.

9.2. NÍVEL COMPRESSÃO

Os testes práticos revelaram que a compressão da redundância espacial, apesar de oferecer um nível de compressão já elevado (cerca de 80% - imagem cinco vezes menor), a compressão da redundância temporal, embora muito variável com o movimento, elevou essa compressão até cerca de 97% (imagem cerca de 32 vezes menor), e a compressão da redundância de códigos Huffman tornou a compressão acima dos 99% (no exemplo com a imagem de referência esta ficou 135 vezes menor). A forma como o algoritmo de

[10] Estes formatos resultam de uma sub-amostragem de metade ou um quarto da resolução para a crominância respectivamente.

compressão de redundância temporal foi desenvolvido não trouxe atrasos de compressão, o que na prática se traduziu num aumento de imagens por segundo em relação à compressão apenas da redundância espacial. Utilizando uma rede de baixo débito (64 kbps) verificou-se que para um movimento em cerca de metade dos píxeis da imagem o número de *frames* por segundo se manteve acima de quatro. O que é suficiente para a videovigilância.

Para resolver o problema da redundância temporal foi desenvolvido e implementado o "algoritmo das diferenças" que é eficiente para videovigilância (imagens em câmaras de vídeo estáticas). Este algoritmo, executado a seguir à DCT, efectua uma comparação de três valores do bloco corrente com os do mesmo bloco da DCT guardada da *frame* anterior, para determinar se a diferença é superior a um determinado valor. Se cada um dos três valores tem um desvio inferior à diferença estabelecida, estamos perante um bloco semelhante ao anterior. Neste caso o bloco é inteiramente substituído pelo carácter de fim de matriz (EOB), e a compressão do bloco termina nessa altura. Na descompressão este valor significará que não houve actualização deste bloco. Se o bloco não for semelhante, são actualizados os três *bytes* do bloco corrente na PM e a compressão do bloco continua normalmente para a quantificação. Este algoritmo como se pode ver, além de ser eficiente torna ainda a compressão mais "leve" que apenas a DCT e quantificação, uma vez que os blocos semelhantes não necessitam de quantificação.

No sistema VideoVigi houve de facto uma melhoria de 123 para 117 ms da compressão parcial para a compressão completa (espacial, temporal e de símbolos – Huffman). Esta foi a maior razão para o sucesso deste sistema de baixo custo desenvolvido de raiz. A eficiência do "algoritmo das diferenças" pode de alguma forma ser traduzida pela figura seguinte (Figura 51). Esta figura mostra uma *frame* obtida directamente do sistema VideoVigi a partir da imagem de referência (relógio). Os blocos pretos são os blocos que não foram actualizados. Pode-se verificar que, à excepção de quatro blocos que sofreram alteração eventualmente devido a ruído na imagem, apenas os blocos à volta do ponteiro dos segundos sofreram actualizações. Os blocos visíveis na figura foram os únicos que passaram pelo processo de quantificação e compressão Huffman, i.e. são apenas estes que após compressão são enviados; é como se a imagem fosse muito mais pequena. No que se refere a ganho em processamento também houve algum, embora não tão significativo porque os outros blocos também foram sujeitos à DCT. Sendo esta rotina a que requer mais recursos na compressão da redundância espacial.

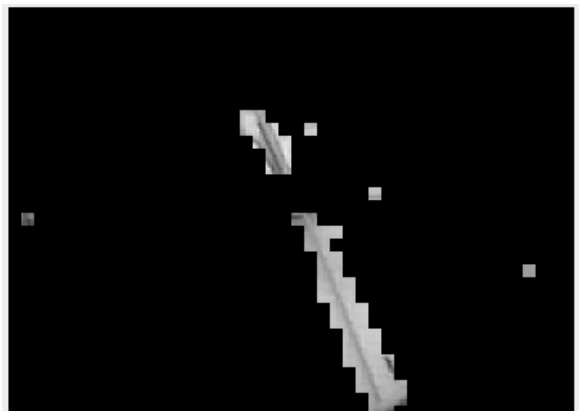


Figura 51 Blocos actualizados

Este "algoritmo das diferenças" conduziu à necessidade de efectuar duas alterações no algoritmo da compressão Huffman. Como numa situação de relativamente pouco movimento vai haver uma predominância do carácter fim de bloco (EOB), as tabelas da norma referidas para a crominância apresentam melhores resultados em cerca de 25%, mesmo quando a matriz de quantificação utilizada é a da luminância. Esta razão deve-se ao facto de o carácter fim de bloco (EOB) da crominância usar apenas dois *bits* (00), enquanto o da luminância usa quatro *bits* (1010). Sendo assim usam-se as tabelas Huffman da crominância. Em relação à componente DC surge outra necessidade de alteração. Como o carácter fim de bloco irá aparecer em posições onde é esperada uma componente DC, o valor de dois bits "00" que correspondente à primeira linha da componente DC não pode ser usado para não ser confundido com o EOB. Logo o valor da primeira linha deve ser substituído por "010" e a segunda linha passa de "01" para "011" uma vez que nenhum código pode conter outro no seu começo (seriam confundidos).

Como a compressão varia com o movimento, para atenuar o aumento do tamanho da imagem, que faria diminuir o número de imagens por segundo, foi desenvolvido um algoritmo de compressão dinâmica. Este algoritmo altera a tabela de quantificação definida na norma para a luminância pela da crominância o que faz duplicar aproximadamente a compressão de redundância espacial. Com esta alteração mantém-se cerca de quatro imagens por segundo (em 64 kbps) quando existe movimento em cerca de 50% da imagem. Apesar da vantagem referida, a qualidade de imagem, embora quase imperceptível, diminuiu.

Os resultados práticos referem-se aos níveis ajustados no sistema "VideoVigi" quer para o "algoritmo das diferenças" como para a "quantificação dinâmica", no entanto se houver

necessidade de melhor qualidade de imagem em detrimento do número de *frames* por segundo, podem reajustar-se os valores de decisão. Apesar de haver sempre alguma sobrejectividade em relação a estas decisões, na imagem de referência o ponteiro dos minutos sofre actualizações de dois em dois segundos. Bastam cerca de 5 segundos (ver figura seguinte) no desvio do ponteiro dos minutos (na posição 50), para ocorerem actualizações em todos os seus blocos. Este resultado mostra que apesar dos elevados níveis de compressão a imagem acompanha bem o movimento e é sensível a pequenas variações. A variação deste ponteiro, em 5 segundos é quase imperceptível a "olho nu".

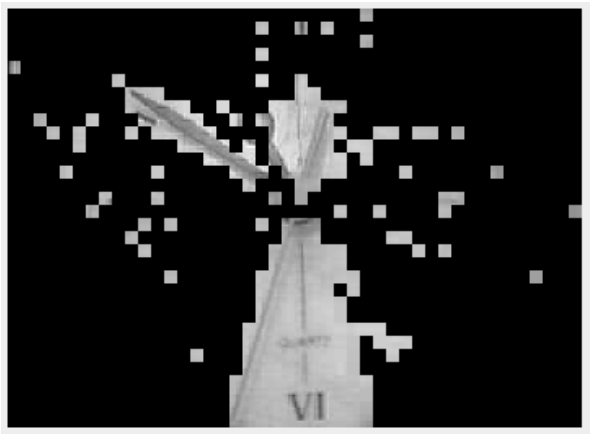


Figura 52 Fluidez da imagem

9.3. CRIPTOGRAFIA DE IMAGENS

Os testes de criptografia foram realizados com várias imagem com e sem luz. À semelhança daquela que foi obtida com o relógio (Figura 53), as imagens eram completamente imperceptíveis, não existindo sequer um relacionamento com a quantidade de luz da imagem. Também foi realizado um teste de tentativa de violação, procurando decifrar a imagem com uma chave que dispõe apenas de um bit trocado. O resultado foi exactamente igual à parte superior da Figura 53 (parte superior), a imagem ficou completamente imperceptível, como se não houvesse nenhuma tentativa de desencriptação.

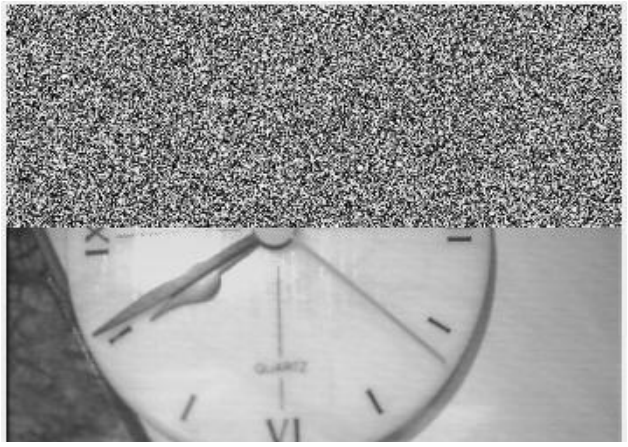


Figura 53 Imagem meia encriptada

A desencriptação com a chave correcta repõe a imagem exactamente igual àquela que foi cifrada (parte inferior da Figura 53). A segurança da chave é efectivamente de 3,40 x10^{38} combinações (2^{128}), i.e. a desencriptação da imagem só pode ser efectuada conhecendo os 128 bits da chave. Embora a criptografia não influencie o nível de compressão altera o tempo de disponibilização de imagens. Este tempo faz naturalmente baixar o número de *frames* por segundo.

9.4. COMENTÁRIOS FINAIS

Das alterações analisadas no capítulo anterior aquela que trouxe maior vantagem e facilidade de implementação foi a simples alteração de uma linha de código que permitiu realizar a compressão do GOB seguinte pelo DSP em simultâneo com a encriptação do GOB anterior pelo microcontrolador. No final do desenvolvimento da solução completa é sempre importante medir os tempos com o osciloscópio, e realizar ensaios em ambiente semi-real para, em conformidade com a arquitectura de *hardware* desenvolvida, tentar preceber onde poderá haver possíveis limitações que possam ser resolvidas por *software* ou, eventualmente, por *hardware*. Para evitar que haja a necessidade de redesenhar um novo *Printed Circuit Board* **(**PCB), para efectuar alterações de *hardware*, deve se interligar os principais barramento de dados, dos diversos componentes, da carta através de uma *Programmable Logic Device* (PLD). Assim uma grande parte das alterações poderão ser efectuadas nesta PLD.

Os ensaios em ambiente semi-real do sistema "VideoVigi" mostraram que realmente se pode desenvolver um sistema de baixo custo para utilização numa rede de baixo débito. O

desenvolvimento de um sistema similar, implementando o mesmo algoritmo, permitirá o envio de 3 *frames* com encriptação pela rede de baixo débito ou 6 sem encriptação. As seis *frames* garantem uma fluidez da imagem perfeita para videovigilância. Com encriptação a videovigilância torna-se segura em redes partilhadas (e.g. Internet); este factor pode ser crítico e portanto justificar as 3 *frames* por segundo que, ainda assim, podem ser suficientes.

Os mecanismos de optimização da compressão e encriptação implementados e respectivamente avaliados tornaram por um lado os dados da videovigilância suficientemente comprimidos para serem enviados por uma rede de baixo débito (BW= 64 kbps) e seguros para utilizar uma rede partilhada como a Internet. Por outro lado a simplificação destes mecanismos tornaram o seu algoritmo relactivamente simples e passível de ser implementado em sistemas com reduzida capacidade de processamento (30 MIPS). Desta forma, sistemas de baixo custo tornam a videovigilância de locais remotos viável através da Internet, recorrendo a redes de acesso de baixo débito (e.g. *General Packet Radio Service* – GPRS).

9.5. FUTUROS DESENVOLVIMENTOS

No desenvolvimento de qualquer projecto ocorrem sempre dificuldades em encontrar componentes compatíveis com os restantes já seleccionados. Estas dificuldades tornam-se mais evidentes em produtos de baixo custo. O sistema *VideoVigi*[1], desenvolvido no Projecto do Curso da Licenciatura de Engenharia Electrotécnica – Electrónica e Computadores, foi um enorme desafio que tinha como principal objectivo comprimir a redundância espacial (MJPEG sem Huffman) de imagens de videovigilância, utilizando para isso um DSP simples da *Analog Device* (ADSP2181[11]). Foi desenhado e implementado um sistema que envolveu a interligação de diversos componentes e respectivas memórias (ADV7183, ADSP2181, DS80C400 e LXT971A), numa arquitectura completamente original (não referida em *applications notes* nem em *datasheets*).

[11] Este DSP é um componente de 5 V, normalmente usado em aplicações de áudio, que dispõe apenas capacidade de endereçamento para 16 k de DM externa.

Para complementar o estudo da influência das variações de débito e não propriamente para melhorar o sistema *VideoVigi*, que foi muito além dos objectivos inicias, era necessário dispor no DSP 512 kbytes de DM (SRAM). No entanto as SRAM de 5 V superiores a 128 kbytes têm tempos de acesso que comprometem o seu funcionamento a 33 MHz. Por esta razão o novo sistema deveria utilizar um DSP de 3,3 V, como o DSP ADSP2183. Com esta alteração, além de resolver o problema da SRAM, também se simplifica o *hardware* na medida em que deixa de haver necessidade de interligar os diversos barramentos com *buffers* de adaptação de tensões. Um sistema com 512 kbytes de DM torna possível testar a solução a cores no formato CIF 4:2:2, assim como implementar e avaliar um possível melhoramento do "algoritmo das diferenças".

Em muitas aplicações de videovigilância a cor das imagens não é algo supérfluo mas uma necessidade. A imagem a cores no formato CIF 4:2:2, pelo facto de apresentar um tamanho correspondente ao dobro da versão monocromática terá uma possível redução no número de *frames* por segundo em 50 %. No entanto, considerando a situação de compressão e encriptação que são realizadas em simultâneo, e como a encriptação é mais lenta que a compressão, existe alguma disponibilidade de tempo em que o DSP pode melhorar a compressão. Como se pode ver na Figura 51 todos os blocos onde ocorreu movimento foram actualizados, porém em alguns píxeis desses blocos não ocorreu qualquer movimento. Na recomendação H.263, para a reconstrução da próxima imagem, para além do *Motion Estimation* (ME), da DCT e Q são realizadas as operações inversas (IDCT e Q^{-1}). Como já foi visto a recomendação H.263 é demasiada complexa para ser implementada num sistema de baixo custo tornava a compressão muito lenta. Mas pode ser interessante tentar aumentar ainda mais a compressão integrando mais "conceitos" do H263.

Mantendo um algoritmo semelhante ao desenvolvido, para os blocos considerados semelhantes não haveria actualização, para os blocos considerados diferentes seriam realizadas diferenças. As diferenças entre o bloco actual e o seu antecessor seriam alvo de um processo semelhante á compressão JPEG, i.e. ao resultado da diferença seria efectuada uma DCT, uma operação de quantificação e finalmente a compressão Huffman. Este algoritmo implica naturalmente espaço em memória suficiente para guradar a imagem anterior completa.

Referências Documentais

[1] CARVALHO, J. F. Oliveira—*Sistema Videovigilância*. Projecto de Licenciatura em Engenharia Electrotécnica—Electrónica e Computadores ISEP, 2007.

[2] POYTON, Charles—*Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann, 2002.

[3] GHANBARI, Mohammed—*Standard Codecs: Video Compression for Video Coding*. Institution of Electrical Engineers, 2003.

[4] UNIVERSITY OF TEXAS—*Handbook of Image and Video Processing*. Al Bovik, 1999.

[5] STALLINGS, Willam—*Data and Computer Communications.* Pearson Education International, 2004.

[6] AXIS COMMUNICATIONS—*Digital Image and Video for Surveillance Applications*. Axis White Paper, 2002.

[7] Menezes, Alfred; Oorschot, Paul; Vanstone Scott —Applied Cryptography. Massachusetts Institute of Technology june, 1996.

[8] FIPS PUB 46-3—Data Encryption Standard (DES). Federal Information Processing Standard Publication. October 1999.

[9] FIPS PUB 197—Advanced Encryption Standard (AES). Federal Information Processing Standard Publication. November 2001.

[10] Network Associates—Une Introduction à la Cryptographie. Novembre 1998.

[11] TEXAS INSTRUMENT—*Implementing JPEG With TMS320C2xx Assembly Language Software.* Application Report SPRA615, 2000.

[12] TEXAS INSTRUMENT—*H.261 Implementation on TMS320C80 DSP*. Application Report SPRA161, 1997.

[13] TEXAS INSTRUMENT—*On the Implementation of MPEG-4 Motion Compensation Using the TMS320C62x.* Application Report SPRA586, 1999.

[14] CCITT, Recommendation T.81—*Information Technology Digital Compression and Coding of Continuous-Tone Still Images-Requirement and Guidelines*, ITU, 1993.

[15] ITU-T, Recommendation H.261—*Video Codec for Audiovisual Service at Px64 kbits*. ITU-T, 1994.

[16] ITU-T, Recommendation H.263—*Video Coding for Low bit Rate Communication* ITU-T, 2005.

[17] ANALOG DEVICE—*ADSP2181 Programmer's Quick Reference* Analog Device.

[18] ANALOG DEVICE—*ADSP2181 EZ-KIT Lite Programmer's Quick Reference.* Analog Device

[19] DALLAS SEMICONDUCTOR—*High-Speed Microcontroller User's Guide.* Dallas Revision 021704.

[20] DALLAS SEMICONDUCTOR—*High-Speed Microcontroller User's Guide: Network Supplement.* Dallas Revision 080806.

[21] ANALOG DEVICE—*DSP Microcomputers.* ADSP-2181/ADSP2183, 1996.

[22] ANALOG DEVICE—*Multiformat SDTV Video Decoder.* ADV7183A, 2005.

[23] DALLAS SEMICONDUCTOR—*Network Microcontroller*. DS80C400, 2007.

Anexo A. Firmware de Aquisição

```
.module/ram/abs=0 principal;
.external  inicializa,DSP_FLG_ON,DSP_FLG_OFF,DSP_SIN_ON,DSP_SIN_OFF,PIC_START_ON,PIC_START_OFF,
                VD_OE_ON,VD_OE_OFF,FIFO_RST_ON,FIFO_RST_OFF,trataSegmento;
.include <reg2181.h>;

.var/pm/ram/circ vazio[2];                { Buf. p/ ler o FIFO s/ guardar  píxel que nao interessa}

jump inicio; nop; nop; nop;               { interrupcao RESET              (end. 0X0000)             }
jump paraleitura; nop; nop; nop;  { interrupcao IRQ2               (end. 0X0004)             }
  rti; nop; nop; nop;                     { interrupcao IRQL1              (end. 0X0008)             }
  rti; nop; nop; nop;                     { interrupcao IRQL0              (end. 0X000C)             }
  rti; nop; nop; nop;                     { interrupcao SPORT0 Tx          (end. 0X0010)             }
  rti; nop; nop; nop;                     { interrupcao SPORT0 Rx          (end. 0X0014)             }
  rti; nop; nop; nop;                     { interrupcao IRQE               (end. 0X0018)             }
  rti; nop; nop; nop;                     { interrupcao BDMA               (end. 0X001C)             }
  rti; nop; nop; nop;                     { interrupcao SPORT1 Tx ou IRQ1  (end. 0X0020)             }
  rti; nop; nop; nop;                     { interrupcao SPORT1 Rx ou IRQ0  (end. 0X0024)             }
  rti; nop; nop; nop;                     { interrupcao Timer              (end. 0X0028)             }
  rti; nop; nop; nop;                     { interrupcao Powerdown          (end. 0X002C)             }

paraleitura:                              {para durante cerca de 10 cilclos de relogio, como o DSP}
        CNTR=10;                  {escreve a 16MHz e o VD a 13,5MHz este val.e' suficiente}
        DO espera UNTIL CE;
        espera:NOP;
        rti;

leDescartaSegmento:
        CNTR=6912;

                                          { para centralizar a imagem o 1°segmento e' descartado      }
                                          {(80 bytes + 352 pixeis por linha) x 16 +3 tolerancia       }
        DO lepíxel0 UNTIL CE;             {sem o valor 2 faltavam píxeis no fim do segmento   }
                        AX0=IO(0);
                        PM(i4,m4)=AX0;    {Precisa desta instrucao para realisar leituras à freq. }
                        AX1=IO(0);        {de 16 MHz, s/ ela as leituras eram feitas a 11 MHz }
                lepíxel0:DM(i4,m4)=AX1;   {le 2 valores do FIFO e descarta ambos (neste caso) }
        rts;

leSegmento:
        CNTR=6912;                        { cada segmento tem 8192B, 8192/360=22,8 Linhas, para       }
                                          { conter 1 N° inteiro de linhas e de blocos (16 linhas)     }
                                          { no primeiro segmento a 1ª tem erros (píxeis pretos)       }
                                          { (80 bytes + 352 pixeis por linha) x 16 +2 tolerancia      }
        DO lepíxel UNTIL CE;              {sem o valor 2 faltavam píxeis no fim do segmento   }
                        AX0=IO(0);
                        PM(i4,m4)=AX0;    {Precisa desta instrucao para realisar leituras à freq. }
                        AX1=IO(0);        {de 16 MHz, s/ ela as leituras eram feitas a 11 MHz }
                lepíxel:DM(i0,m0)=AX1;    { le 2 valores do FIFO mas só guarda 1 (posicoes pares)     }
        rts;

inicio:
call inicializa;
imagem:
i4=^vazio;m4=1;l4=2;
{INICIALIZACAO FLAGS PARA LER PROXIMA IMAGEM}
        AR=0x00D0;                        { inicializacao das FLAGES (PFx)                    }
        DM(PFDATA)=AR;                    {                                                           }
        call DSP_SIN_OFF;         { activa o DSP_SIN                                  }
        call VD_OE_OFF;                   { activa o VD_OE                                    }
{ESPERA PELO INICIO DO FIELD1}
        AY0=PMASC_PF3;                    { o FIELD e' recebido na DSP_AUX (PF3)              }
        eimpar:                           { espera por fim de FIELD1 para comecar no seu inicio       }
                AX0=DM(PFDATA);
                AR=AX0 AND AY0;
        if EQ JUMP eimpar;              { se for zero, esta a meio do FIELD1, deve aguardar   }
        AY0=PMASC_PF3;                    { o FIELD e' recebido na DSP_AUX (PF3)              }
        epar:                             { espera por fim de FIELD2 para activar PIC_START   }
                AX0=DM(PFDATA);
                AR=AX0 AND AY0;
        if NE JUMP epar;                { se nao for zero, ainda esta no FIELD2, deve aguardar      }
{PREPARA LEITURA DO FIFO}
        call FIFO_RST_OFF;                { faz o reset ao fifo (PF7 a 0)                     }
        call FIFO_RST_ON;         { desfaz o reset ao fifo (PF7 a 1)                  }
        call PIC START ON;                { activa o PIC START                                }
        IMASK=b#0000001000000000;         { activa IRQ2 (FIFO EF)Nota: so depois do FIFO W activo     }
        ICNTL=b#00000;                    { configura interrupcao IRQ2 ao nivel               }
{LEITURA DO FIFO E ESCRITA NA DM EXTERNA (16 SEGMENTOS)}
        i0=0x0000;                        { i0 -> endereco inicial do segmento da DM externa  }
        l0=0;                             { Nao e' circular                                   }
```

```
        m0=1;                           { valor do incremento                                   }
        DMOVLAY=1;                      { 1-000(1° segmento)                                    }
        reset FL0;
        reset FL1;
        reset FL2;
        call leDescartaSegmento;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-000(2° segmento)                                    }
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=1;                      { 1-001(3° segmento)                                    }
        set FL0;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-001(4° segmento)                                    }
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=1;                      { 1-010(5° segmento)                                    }
        reset FL0;
        set FL1;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-010(6° segmento)                                    }
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=1;                      { 1-011(7° segmento)                                    }
        set FL0;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-011(8° segmento)                                    }
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=1;                      { 1-100(9° segmento)                                    }
        reset FL0;
        reset FL1;
        set FL2;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-100(10° segmento)                                   }
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=1;                      { 1-101(11° segmento)                                   }
        set FL0;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-101(12° segmento)                                   }
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=1;                      { 1-110(13° segmento)                                   }
        reset FL0;
        set FL1;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-110(14° segmento)                                   }
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=1;                      { 1-111(15° segmento)                                   }
        set FL0;
       call leSegmento;
        i0=0x0000;                      { i0 -> endereco inicial do segmento da DM externa  }
        DMOVLAY=2;                      { 2-111(16° segmento)                                   }
       call leSegmento;
{A IMAGEM ESTA NA DM EXTERNA, DESACTIVA CONDICOES DE AQUISICAO}
        DMOVLAY=0;
        IMASK=b#0000000000000000;       { desactiva as interrupcoes                             }
        call VD_OE_ON;                  { desactiva o VD_OE                                     }
        call PIC_START_OFF;             { desactiva o PIC_START                                 }
        call DSP_SIN_ON;          { desactiva o DSP_SIN                                 }
{LEITURA DA DM EXTERNA, DOS 1° 88 BLOCOS, COPIA PARA A DM ORDENADOS EM BLOCOS, COMPRESSAO AVISO MIC}
{#################  1°  S E G M E N T O  #################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        l0=0;                           { Nao e' circular                                       }
        m0=1;                           { valor do incremento                                   }
        SR0=1;                          { vai transportar a informacao DMOVLAY a outra rotina   }
        SR1=1;                          { vai transportar a informacao GOB a outra rotina   }
        MX0=0;                          {vai identificar o segmento na outra rotina             }
        reset FL0;
        reset FL1;
        reset FL2;
        call trataSegmento;
{#################  2°  S E G M E N T O  #################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}

        SR0=2;
        SR1=2;                          { vai transportar a informacao GOB a outra rotina   }
        MX0=1;
        call trataSegmento;
```

```
{################  3°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=1;
        SR1=3;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=2;
        set FL0;
        call trataSegmento;
{################  4°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=2;
        SR1=4;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=3;
        call trataSegmento;
{################  5°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=1;
        SR1=1;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=4;
        reset FL0;
        set FL1;
        call trataSegmento;
{################  6°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}

        SR0=2;
        SR1=2;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=5;
        call trataSegmento;
{################  7°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=1;
        SR1=3;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=6;
        set FL0;
        call trataSegmento;
{################  8°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}

        SR0=2;
        SR1=4;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=7;
        call trataSegmento;
{################  9°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=1;
        SR1=1;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=8;
        reset FL0;
        reset FL1;
        set FL2;
        call trataSegmento;
{################  10°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}

        SR0=2;
        SR1=2;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=9;
        call trataSegmento;
{################  11°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=1;
        SR1=3;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=10;
        set FL0;
        call trataSegmento;
{################  12°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=2;
        SR1=4;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=11;
        call trataSegmento;
{################  13°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}

        SR0=1;
        SR1=1;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=12;
        reset FL0;
        set FL1;
        call trataSegmento;
{################  14°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
        SR0=2;
        SR1=2;                          { vai transportar a informacao GOB a outra rotina  }
        MX0=13;
        call trataSegmento;
{################  15°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}
```

```
        SR0=1;
        SR1=3;                          { vai transportar a informacao GOB a outra rotina   }
        MX0=14;
        set FL0;
        call trataSegmento;
{################  16°  S E G M E N T O  ################}
        i0=0x0000;                      { i0 -> endereco inicial 1ª linha do segmento da DM ext.}

        SR0=2;
        SR1=4;                          { vai transportar a informacao GOB a outra rotina   }
        MX0=15;
        call trataSegmento;

jump imagem;
.endmod;
```

Anexo B. Firmware de Compressão

```
.module/RAM     H263COMP;
.external  inicializa,DSP_FLG_ON,DSP_FLG_OFF;
.include <reg2181.h>;
.entry  trataSegmento;
.const          m=8, delta=128; {###para teste era 128}             {Define uma constante (dimensao das matrizes e val
a subtr.)        }
.var/dm segmento;
.var/dm valorDMOVLAY;
.var/dm endRelOldBloco;
.var/dm endBloco;
.var/dm bitsShiftados;
.var/dm sr0Temporario;
.var/dm sr1Temporario;
.var/dm linhaCod;
.var/dm linhaLeng;
.var/dm posicaoCod;
.var/dm posicaoLeng;
.var/dm codigoCod;
.var/dm codigoLeng;
.var/dm numeroZeros;
.var/dm DC_Coef;
.var/dm DC_Leng;
.var/dm AC_Coef;
.var/dm AC Leng;
.var/dm tamanhoPacote;
.var/dm nivelCompressao;
.var/dm pMASC;
.var/dm nMASC;
.var/dm GOB;

.var/pm/ram Q pm[m*m];                    {Declara 1 buffer p/ inversos da quantificao na ordem zigzag        }
.var/pm/ram Q1_pm[m*m];                   {Declara 1 buffer p/ inversos da quantificao na ordem zigzag        }
.var/pm/ram Q2_pm[m*m];                   {Declara 1 buffer p/ inversos da quantificao na ordem zigzag        }
.var/pm/ram Q3_pm[m*m];                   {Declara 1 buffer p/ inversos da quantificao na ordem zigzag        }
.var/pm/ram zigzag_pm[m*m];               {Declara 1 buffer na pm de m*m para as posicoes de zigzag   }
.var/dm/ram X_dm[m*m];                    {Declara 1 buffer na dm de m*m para receber os blocos da imagem     }
.var/dm/ram Y dm[m*m];                    {Decl. 1 buffer na dm p/ valores interme'dios e Resultado Final     }
.var/dm/ram DCT_dm[m*m];                  {Declara um buffer na dm de m*m para resultados da DCT              }
.var/pm/ram A_pm[m*m];                    {Declara 1 buffer na pm de m*m para a matriz dos coef. cosenos      }
.var/pm/ram entrada[5640];                        {Buffer na pm para cada seg. na ordem normal (linhas tratadas)
.var/dm/ram pacote[2300];                 {Buffer na dm para guardar um pacote (4 linhas sem compressao)      }
                                                  {ou 88 blocos comprimidos. Como apenas 1452 sao p/ pixeis   }
                                                  {a comp minima devera ser aproximadamente 75%. o buff e' maior
                                                  {os dados deverao ser <1500B p/ enviar em pacotes IP
.var/pm/ram oldFrame[4224];                       {Buffer na pm para guardar 3 B de cada blocos da imagem completa}
.var/dm/ram blocos[5640];                 {Buffer na dm para guardar 88 blocos de cada seg. (ordem blocos)}
.var/pm/ram DC_CoefL[12];                 {Coefficientes DC para a compressao Huffman                         }
.var/pm/ram DC_LengL[12];                 {Comprimento dos coefficientes DC para a compressao Huffman }
.var/pm/ram DC CoefC[12];                 {Coefficientes DC para a compressao Huffman                         }
.var/pm/ram DC LengC[12];                 {Comprimento dos coefficientes DC para a compressao Huffman }
.var/pm/ram AC_CoefL[162];                        {Coefficientes AC para a compressao Huffman (com zeros)
.var/pm/ram AC_LengL[162];                        {Comprimento dos coefficientes AC para a compressao Huffman }
.var/pm/ram AC_CoefC[162];                        {Coefficientes AC para a compressao Huffman (com zeros)
.var/pm/ram AC_LengC[162];                        {Comprimento dos coefficientes AC para a compressao Huffman }

.var/dm/ram pacoteFinal[2000];                    {Buffer na dm para guardar um pacote com compressao huffman }
.var/dm/ram TESTE[178];

.init TESTE:     0x0088,0,8,2,0,8,0x0080,8,0x0080,1,12,3,0,-4,-9,3,2,0,-1,
                 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,-1,-1,0x0080,-80,30,-80,0x0080,120,3,0x0080,
                 0x0080,120,0,0,0,0,-125,100,0x0080,-100,0,0,1,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,
                 0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,
                 0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,
                 0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,
                 0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,
                 0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,
                 -30,9,8,7,6,5,4,3,2,1,0x0080,0x0080,0x0080,0x0080,77,-83,0x0080,0x0080,0x0080,
                 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,120,-120,
                 0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0080,0x0081;

.init DC_CoefC:  0x000200,0x000300,0x000400,0x000500,0x000600,
                 0x000E00,0x001E00,0x003E00,0x007E00,0x00FE00,0X01FE00;
.init DC_LengC:  0x000300,0x000300,0x000300,0x000300,0x000300,
                 0x000400,0x000500,0x000600,0x000700,0x000800,0x000900;

{.init DC_CoefC:0x000000,0x000100,0x000200,0x000600,0x000E00,0x001E00,       }
```

```
{                0x003E00,0x007E00,0x00FE00,0x01FE00,0x03FE00,0X07FE00;      }
{.init DC LengC:0x000200,0x000200,0x000200,0x000300,0x000400,0x000500,       }
{                0x000600,0x000700,0x000800,0x000900,0x000A00,0x000B00;      }

.init AC_CoefL:  0x000A00,
                 0x000000,0x000100,0x000400,0x000B00,0x001A00,0x007800,0x00F800,0x02F600,0xFF8200,0xFF8300,
                 0x000C00,0x001B00,0x007900,0x01F600,0X07F600,0xFF8400,0xFF8500,0xFF8600,0xFF8700,0xFF8800,
                 0x001C00,0x00F900,0x02F700,0x0FF400,0XFF8900,0xFF8A00,0xFF8B00,0xFF8C00,0xFF8D00,0xFF8E00,
                 0x003A00,0x01F700,0x0FF500,0xFF8F00,0XFF9000,0xFF9100,0xFF9200,0xFF9300,0xFF9400,0xFF9500,
                 0x003B00,0x03F800,0xFF9600,0xFF9700,0XFF9800,0xFF9900,0xFF9A00,0xFF9B00,0xFF9C00,0xFF9D00,
                 0x007A00,0x07F700,0xFF9E00,0xFF9F00,0XFFA000,0xFFA100,0xFFA200,0xFFA300,0xFFA400,0xFFA500,
                 0x007B00,0x0FF600,0xFFA600,0xFFA700,0XFFA800,0xFFA900,0xFFAA00,0xFFAB00,0xFFAC00,0xFFAD00,
                 0x00FA00,0x0FF700,0xFFAE00,0xFFAF00,0XFFB000,0xFFB100,0xFFB200,0xFFB300,0xFFB400,0xFFB500,
                 0x01F800,0x7FC000,0xFFB600,0xFFB700,0XFFB800,0xFFB900,0xFFBA00,0xFFBB00,0xFFBC00,0xFFBD00,
                 0x01F900,0xFFBE00,0xFFBF00,0XFFC000,0xFFC100,0xFFC200,0xFFC300,0xFFC400,0xFFC500,0xFFC600,
                 0x01FA00,0xFFC700,0xFFC800,0xFFC900,0XFFCA00,0xFFCB00,0xFFCC00,0xFFCD00,0xFFCE00,0xFFCF00,
                 0x03F900,0xFFD000,0xFFD100,0xFFD200,0XFFD300,0xFFD400,0xFFD500,0xFFD600,0xFFD700,0xFFD800,
                 0x03FA00,0xFFD900,0xFFDA00,0xFFDB00,0XFFDC00,0xFFDD00,0xFFDE00,0xFFDF00,0xFFE000,0xFFE100,
                 0x07F800,0xFFE200,0xFFE300,0xFFE400,0XFFE500,0xFFE600,0xFFE700,0xFFE800,0xFFE900,0xFFEA00,
                 0xFFEB00,0xFFEC00,0xFFED00,0xFFEE00,0XFFEF00,0xFFF000,0xFFF100,0xFFF200,0xFFF300,0xFFF400,
                 0xFFF500,0xFFF600,0xFFF700,0XFFF800,0xFFF900,0xFFFA00,0xFFFB00,0xFFFC00,0xFFFD00,0xFFFE00,
                 0x07F900;
.init AC_LengL:  0x000400,
                 0x000200,0x000200,0x000300,0x000400,0x000500,0x000700,0x000800,0x000A00,0x001000,0x001000,
                 0x000400,0x000500,0x000700,0x000900,0x000B00,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000500,0x000800,0x000A00,0x000C00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000600,0x000900,0x000C00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000600,0x000A00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000700,0x000B00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000700,0x000C00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000800,0x000C00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000900,0x000F00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000900,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000900,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000A00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000A00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000B00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000B00;
.init AC CoefC:  0x000000,
                 0x000100,0x000400,0x000A00,0x001800,0x001900,0x003800,0x007800,0x01F400,0x03F600,0x0FF400,
                 0x000B00,0x003900,0x00F600,0x01F500,0X07F600,0x0FF500,0xFF8800,0xFF8900,0xFF8A00,0xFF8B00,
                 0x001A00,0x00F700,0x03F700,0x0FF600,0X7FC200,0xFF8C00,0xFF8D00,0xFF8E00,0xFF8F00,0xFF9000,
                 0x001B00,0x00F800,0x03F800,0x0FF700,0XFF9100,0xFF9200,0xFF9300,0xFF9400,0xFF9500,0xFF9600,
                 0x003A00,0x01F600,0xFF9700,0xFF9800,0XFF9900,0xFF9A00,0xFF9B00,0xFF9C00,0xFF9D00,0xFF9E00,
                 0x003B00,0x03F900,0xFF9F00,0xFFA000,0XFFA100,0xFFA200,0xFFA300,0xFFA400,0xFFA500,0xFFA600,
                 0x007900,0x07F700,0xFFA700,0xFFA800,0XFFA900,0xFFAA00,0xFFAB00,0xFFAC00,0xFFAD00,0xFFAE00,
                 0x007A00,0x07F800,0xFFAF00,0xFFB000,0XFFB100,0xFFB200,0xFFB300,0xFFB400,0xFFB500,0xFFB600,
                 0x00F900,0xFFB700,0xFFB800,0xFFB900,0XFFBA00,0xFFBB00,0xFFBC00,0xFFBD00,0xFFBE00,0xFFBF00,
                 0x01F700,0xFFC000,0xFFC100,0xFFC200,0XFFC300,0xFFC400,0xFFC500,0xFFC600,0xFFC700,0xFFC800,
                 0x01F800,0xFFC900,0xFFCA00,0xFFCB00,0XFFCC00,0xFFCD00,0xFFCE00,0xFFCF00,0xFFD000,0xFFD100,
                 0x01F900,0xFFD200,0xFFD300,0xFFD400,0XFFD500,0xFFD600,0xFFD700,0xFFD800,0xFFD900,0xFFDA00,
                 0x01FA00,0xFFDB00,0xFFDC00,0xFFDD00,0XFFDE00,0xFFDF00,0xFFE000,0xFFE100,0xFFE200,0xFFE300,
                 0x07F900,0xFFE400,0xFFE500,0xFFE600,0XFFE700,0xFFE800,0xFFE900,0xFFEA00,0xFFEB00,0xFFEC00,
                 0x3FE000,0xFFED00,0xFFEE00,0xFFEF00,0XFFF000,0xFFF100,0xFFF200,0xFFF300,0xFFF400,0xFFF500,
                 0x7FC300,0xFFF600,0xFFF700,0XFFF800,0xFFF900,0xFFFA00,0xFFFB00,0xFFFC00,0xFFFD00,0xFFFE00,
                 0x03FA00;
.init AC LengC:  0x000200,
                 0x000200,0x000300,0x000400,0x000500,0x000500,0x000600,0x000700,0x000900,0x000A00,0x000C00,
                 0x000400,0x000600,0x000800,0x000900,0x000B00,0x000C00,0x001000,0x001000,0x001000,0x001000,
                 0x000500,0x000800,0x000A00,0x000C00,0x000F00,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000500,0x000800,0x000A00,0x000C00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000600,0x000900,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000600,0x000A00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000700,0x000B00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000700,0x000B00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000800,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000900,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000900,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000900,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000900,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000B00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000E00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000F00,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,0x001000,
                 0x000A00;


.init A pm:0x5A8200,0x5A8200,0x5A8200,0x5A8200,0x5A8200,0x5A8200,0x5A8200,0x5A8200,
           0x7D8A00,0x6A6E00,0x471D00,0x18F900,0xE70700,0xB8E300,0x959200,0x827600,
           0x764200,0x30FC00,0xCF0400,0x89BE00,0x89BE00,0xCF0400,0x30FC00,0x764200,
           0x6A6E00,0xE70700,0x827600,0xB8E300,0x471D00,0x7D8A00,0x18F900,0x959200,
           0x5A8200,0xA57E00,0xA57E00,0x5A8200,0x5A8200,0xA57E00,0xA57E00,0x5A8200,
           0x471D00,0x827600,0x18F900,0x6A6E00,0x959200,0xE70700,0x7D8A00,0xB8E300,
           0x30FC00,0x89BE00,0x764200,0xCF0400,0xCF0400,0x764200,0x89BE00,0x30FC00,
           0x18F900,0xB8E300,0x6A6E00,0x827600,0x7D8A00,0x959200,0x471D00,0xE70700;

.init Q_pm:0x080000,0x0BA300,0x0AAB00,0x092500,0x0AAB00,0x0CCD00,0x080000,0x092500,
```

```
           0x09D900,0x092500,0x071C00,0x078800,0x080000,0x06BD00,0x055500,0x033300,
           0x04EC00,0x055500,0x05D100,0x05D100,0x055500,0x029D00,0x03A800,0x037600,
           0x046A00,0x033300,0x023500,0x028300,0x021900,0x022200,0x023F00,0x028300,
           0x024900,0x025400,0x020000,0x01C700,0x016400,0x01A400,0x020000,0x01E200,
           0x017900,0x01DB00,0x025400,0x024900,0x019A00,0x012D00,0x019500,0x017900,
           0x015900,0x014E00,0x013E00,0x013B00,0x013E00,0x021100,0x01AA00,0x012200,
           0x010F00,0x012500,0x014800,0x011100,0x016400,0x014400,0x013E00,0x014B00;

{Nota: para melhorar a compressao para garantir um pacote por segmento os valores sofreram um aumento de 10%}
{Q = 1/q x 2^15  passou para Q = 1/(1,1 x q) x 2^15 }
{Aumento de Q em 25% (parede lisa-media pacote 1000Bytes, satura em imagens com muitas transicoes)}
{ORDEM zigzag}

.init Q1 pm:0x066600,0x094F00,0x088800,0x075000,0x088900,0x0A3D00,0x066600,0x075000,
           0x07E000,0x075000,0x05B000,0x060600,0x066600,0x056400,0x044400,0x028F00,
           0x03F000,0x044400,0x04A800,0x04A800,0x044400,0x021700,0x02ED00,0x02C400,
           0x038800,0x028F00,0x01C400,0x020200,0x01AE00,0x01B500,0x01CC00,0x020200,
           0x01D400,0x01DD00,0x019A00,0x016C00,0x011D00,0x015000,0x019A00,0x018200,
           0x012D00,0x017C00,0x01DD00,0x01D400,0x014800,0x00F000,0x014300,0x012D00,
           0x011400,0x010B00,0x00FF00,0x00FC00,0x00FF00,0x01A700,0x015400,0x00E800,
          0x00D900,0x00EA00,0x010600,0x00DA00,0x011D00,0x010400,0x00FF00,0x010900;

 {Aumento de Q em 50% (parede lisa-media pacote 850Bytes)}
.init Q2_pm:0x055500,0x07C200,0x071C00,0x061800,0x071C00,0x088900,0x055500,0x061800,
        0x069000,0x061800,0x04BE00,0x050500,0x055500,0x047E00,0x038E00,0x022200,
        0x034800,0x038E00,0x03E100,0x03E100,0x038E00,0x01BE00,0x027000,0x024E00,
        0x02F100,0x022200,0x017900,0x01AC00,0x016600,0x016C00,0x017F00,0x01AC00,
        0x018600,0x018D00,0x015500,0x012F00,0x00ED00,0x011800,0x015500,0x014100,
        0x00FB00,0x013D00,0x018D00,0x018600,0x011100,0x00C800,0x010E00,0x00FB00,
        0x00E600,0x00DF00,0x00D400,0x00D200,0x00D400,0x016000,0x011C00,0x00C100,
        0x00B500,0x00C300,0x00DA00,0x00B600,0x00ED00,0x00D800,0x00D400,0x00DD00;

{Q usada na crominancia(fraca resolucao)}
.init Q3 pm:0x078800,0x071C00,0x071C00,0x055500,0x061800,0x055500,0x02B900,0x04EC00,
          0x04EC00,0x02B900,0x014B00,0x01F000,0x024900,0x01F000,0x014B00,0x014B00,
          0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,
          0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,
           0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,
          0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,
          0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,
          0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00,0x014B00;

.init nivelCompressao:0x0000;

.init zigzag pm:0x000000,0x000100,0x000800,0x001000,0x000900,0x000200,0x000300,0x000A00,
                0x001100,0x001800,0x002000,0x001900,0x001200,0x000B00,0x000400,0x000500,
                0x000C00,0x001300,0x001A00,0x002100,0x002800,0x003000,0x002900,0x002200,
                0x001B00,0x001400,0x000D00,0x000600,0x000700,0x000E00,0x001500,0x001C00,
                0x002300,0x002A00,0x003100,0x003800,0x003900,0x003200,0X002B00,0x002400,
                0x001D00,0x001600,0x000F00,0x001700,0x001E00,0x002500,0x002C00,0x003300,
                0x003A00,0X003B00,0x003400,0x002D00,0x002600,0x001F00,0x002700,0x002E00,
                0x003500,0x003C00,0x003D00,0x003600,0x002F00,0x003700,0x003E00,0x003F00;


copiaBloco:
        CNTR=15;                        { le 16 linhas de dois blocos de cada vez                   }
        DO linhabloco1 UNTIL CE;        { cada segmento tera 84 blocos (42x2)                               }
                CNTR=8;                         { para facilitar, a leitura sera feita por colunas de 2
                DO umalinha1 UNTIL CE;
                        AX0=PM(i4,m4);
                umalinha1:DM(i1,m1)=AX0;  { Guarda na DM interna na ordem dos blocos                }
                m4=344;
                MODIFY(i4,m4);                  { para avancar uma linha completa (352-8)                   }
                m4=1;
        linhabloco1:nop;
                CNTR=8;                         { ultima linha destes blocos
                DO ultimalinha1 UNTIL CE;
                        AX0=PM(i4,m4);
                ultimalinha1:DM(i1,m1)=AX0;       { Guarda na DM interna na ordem dos blocos                }
rts;
copiaBlocos:                                    { reordena o segmento para a DM na ordem dos blocos         }
        i4=^entrada;m4=1;l4=0;                  { inicio 1º Bloco                                           }
        i1=^blocos;m1=1;l1=0;
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
        i4=^entrada+8;                          { inicio 3º Bloco                                           }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
        i4=^entrada+16;                         { inicio 5º Bloco                                           }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
        i4=^entrada+24;                         { inicio 7º Bloco                                           }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
        i4=^entrada+32;                         { inicio 9º Bloco                                           }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
        i4=^entrada+40;                         { inicio 11º Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
        i4=^entrada+48;                         { inicio 13º Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
        i4=^entrada+56;                         { inicio 15º Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez                   }
```

```
        i4=^entrada+64;                         { inicio 17° Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+72;                         { inicio 19° Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+80;                         { inicio 21° Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+88;                         { inicio 23° Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+96;                         { inicio 25° Bloco
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+104;                { inicio 27° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+112;                { inicio 29° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+120;                { inicio 31° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+128;                { inicio 33° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+136;                { inicio 35° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+144;                { inicio 37° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+152;                { inicio 39° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+160;                { inicio 41° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+168;                { inicio 43° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+176;                { inicio 45° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+184;                { inicio 47° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+192;                { inicio 49° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+200;                { inicio 51° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+208;                { inicio 53° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+216;                { inicio 55° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+224;                { inicio 57° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+232;                { inicio 59° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+240;                { inicio 61° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+248;                { inicio 63° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+256;                { inicio 65° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+264;                { inicio 67° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+272;                { inicio 69° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+280;                { inicio 71° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+288;                { inicio 73° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+296;                { inicio 75° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+304;                { inicio 77° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+312;                { inicio 79° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+320;                { inicio 81° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+328;                { inicio 83° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+336;                { inicio 85° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
        i4=^entrada+344;                { inicio 87° Bloco                                              }
        call copiaBloco;                { le 16 linhas de dois blocos de cada vez               }
rts;

comprimebloco:
{SUBTRAI 128 AOS ELEMENTOS DA MARTRIZ DO BLOCO DE IMAGEM, GUARDA EM X_dm}
        dis m_mode;
        i2=^X_dm;
        l2=0;
        m2=1;
       AX0=DM(I4,M4);                           {I4 esta a enderecar o bloco (buffer blocos)
       AY0=delta;
       cntr=m*m-1;
        DO loop_sub UNTIL CE;                   {Vai subtrair aos valores de entrada 128                }
               AR=AX0-AY0, AX0=DM(I4,M4);      {Faz subtraccao e vai ler a proxima posicao             }
        loop_sub: DM(I2,M2)=AR;         {guarda resultado da subtraccao na  matriz X_dm                 }
        AR=AX0-AY0;                       {Faz subtraccao do ultimo coeficiente                         }
        DM(I2,M2)=AR;
```

```
{MULTIPLICA MATRIZ DOS COS COM BLOCO DE IMAGEM, GUARDA EM Y_dm }
        i6=^A pm;                                   { i6 -> inicio A pm -> formato (1.15) com sinal
        l6=0;                                       { A pm e' buffer NAO circular
        m5=1;
        l5=0;
        i2=^X_dm;                                   { i2 -> inicio X_dm -> formato (15.1) sem sinal
        l2=0;                                       { X_dm e' buffer NAO circular
        m2=m;                                       { m2 servira para incrementar i1                            }
        i1=^Y_dm;                                   { i1 -> inicio Y_dm -> formato (16.0) com sinal
        l1=0;                                       { Y_dm e' buffer NAO circular
        m1=1;
        m0=m;
        l0=0;
        m6=m;
        m7=1;
        l7=0;

        ena m_mode;                                 {Nao faz shift automático (modo inteiro)                    }
        se=1;                                      {contem o valor do shift 1 (nao está a ser usado)            }
        cntr=m;                                     { carrega CNTR com o nr. de linhas da matriz A pm           }
        DO linha_loop UNTIL CE;                     {A pm.X dm, (1ª linha A pm c/ todas col. de X dm e sucessiva/)
                i5=i2;                              {I5 = INICIO DE X_dm (inicio da 1ª coluna)                  }
                CNTR=m2;
                DO coluna_loop UNTIL CE;
                        i7=i6;                      {coloca I7 na linha actual (A_pm -> 1ª matriz)
                        i0=i5;                      {coloca I0 na coluna actual (X_dm -> 2ª matriz)
                        CNTR=M2;                   {contador igual ao numero de elementos da linha da matriz A   }
                         MR=0, MX0=DM(I0,M0), MY0=PM(I7,M7);   {inicializa o MR e le' os elementos de cada matriz
                         DO elemento_loop UNTIL CE;
                         elemento_loop:  MR=MR+MX0*MY0 (SS), MX0=DM(I0,M0),MY0=PM(I7,M7);
                                         {soma todos produtos da linha da 1ª matriz com a col. da 2ª mat.}
                         SR=ASHIFT MR1 (HI),        {Shift e Actualiza I5 para o inicio proxima linha de A dm   }
                         MY0=DM(I5,M5);             {Actualiza I5 para o inicio proxima coluna de X dm          }
                         SR=SR OR LSHIFT MR0 (LO);{finaliza shift e coloca o resultado em sr1 no formato 16.0  }
               coluna_loop: DM(I1,M1)=MR1;     {guarda o resltado na matriz Y_dm                               }
        linha_loop: MODIFY(I6,M6);                  {actualiza I6 (matriz A_pm) para a proxima linha            }

{MULTIPLICA MATRIZ RESULTADO COM MATRIZ DOS COS TRANSPOSTA (DCT), GUARDA EM DCT_dm}
        i1=^Y_dm;                                   { i1 -> inicio Y_dm -> formato (16.0) com sinal
        i6=^A pm;                                   { i1 -> inicio A dm -> formato (1.15) com sinal
        i3=^DCT dm;                                 { i3 -> inicio DCT dm -> formato (16.0) sem sinal           }
        l3=0;                                       { DCT_dm e' buffer NAO circular                             }
        m3=1;
        m5=m;
        m0=1;
        m1=m;

        cntr=m;
        DO linha_loop2 UNTIL CE;          {Y_dm . A_dm´, (1ª lin. Y_pm c/ todas lin. de A_dm e sucessiva/)}
                i5=i6;                              {I5 = INICIO DE A_dm (inicio da 1ª linha)                   }
                CNTR=m2;
                DO coluna_loop2 UNTIL CE;
                        i0=i1;                      {coloca I0 na linha actual (Y dm -> 1ª matriz)
                        i7=i5;                      {coloca I7 na linha actual (A pm -> 2ª matriz)
                        CNTR=M2;                {contador igual ao numero de elementos da linha da matriz Y_dm
                        MR=0, MX0=DM(I0,M0), MY0=PM(I7,M7);     {inicializa MR e le' os elementos de cada matriz
                        DO elemento_loop2 UNTIL CE;
                        elemento loop2: MR=MR+MX0*MY0 (SS), MX0=DM(I0,M0),MY0=PM(I7,M7);
                                         {soma todos produtos da linha da 1ª matriz c/ linha da 2ª mat.
                         SR=ASHIFT MR1 (HI),        {Shift e Actualiza I5 para o inicio proxima linha de A_dm   }
                         MY0=DM(I5,M5);             {Actualiza I5 para o inicio proxima linha de A_dm           }
                         SR=SR OR LSHIFT MR0 (LO);{finaliza shift e coloca o resultado em sr1 no formato 16.0 }
              coluna_loop2: DM(I3,M3)=MR1;    {guarda o resultado na matriz DCT                               }
        linha_loop2: MODIFY(I1,M1);             {actualiza I1 (matriz Y_pm) para a proxima linha             }
rts;

estimaDiferencas:
{COMPARACAO COM BLOCO da FRAME ANTERIOR PARA COMPRIMIR APENAS OS DIFERENTES}
        m1=1;l1=0;
        MX0=DM(segmento);                {lê o numero do segmento (i.e. o GOB actual)                           }
        MY0=264;                         {corresponde aos 3 B de cada bloco p/ GOB da old frame (88x3)          }
        MR=MX0*MY0 (UU);                 {MR fica com o valor relativo Nº GOB x 88 x 3 ->(GOB x 264) }
        AY1=DM(endRelOldBloco);                  {AY1 indexa o bloco da OLd frame (contando com as 3 posicoes)
        AR=MR0+AY1;                              {AR tem o endereco relativo do bloco actual da old frame    }
        AY1=^oldFrame;
        AR=AR+AY1;                               {AR tem o endereco absoluto do bloco actual da old frame    }
        sr0=AR;                                  {fica guardado em sr0
        i4=AR;m4=1;l4=0;                 {I4 indexa a oldFrame na posicao do bloco actual           }
        i3=^DCT dm;m3=1;l3=0;                    {I3 indexa a DCT do bloco actual                            }
        {jump diferente;}{##### SO PARA TESTES #####}
        condicao1:
                AY0=PM(I4,M4),AX0=DM(I3,M3);     {AY0 e AX0 tem o valor das componentes DC (primeiros Bytes)
                AR=AX0-AY0;                      {vai encontrar a diferenca entre os 2 valores
                AY1=32;
                AR=AR-AY1;                       {vai ver se o erro e' inferior a -diferenca
                if LE JUMP condicao2;
                jump diferente;
        condicao2:
```

```
                AR=AX0-AY0;                     {vai encontrar a diferenca entre os 2 valores
                AR=AR+AY1;                      {vai ver se o erro e' inferior a +diferenca
                if GE JUMP condicao3;
                jump diferente;
        condicao3:
                AY0=PM(I4,M4),AX0=DM(I3,M3);    {AY0 e AX0 tem o valor da 1 componentes AC              }
                AR=AX0-AY0;                     {vai encontrar a diferenca entre os 2 valores
                AY1=22;
                AR=AR-AY1;                      {vai ver se o erro e' inferior a -diferenca
                if LE JUMP condicao4;
                jump diferente;
        condicao4:
                AR=AX0-AY0;                     {vai encontrar a diferenca entre os 2 valores
                AR=AR+AY1;                      {vai ver se o erro e' inferior a +diferenca
                if GE JUMP condicao5;
                jump diferente;
        condicao5:
                i3=^DCT_dm+8;
                AY0=PM(I4,M4),AX0=DM(I3,M3);    {AY0 e AX0 tem o valor da 2 componentes AC              }
                AR=AX0-AY0;                     {vai encontrar a diferenca entre os 2 valores
                AY1=24;
                AR=AR-AY1;                      {vai ver se o erro e' inferior a -diferenca
                if LE JUMP condicao6;
                jump diferente;
        condicao6:
                AR=AX0-AY0;                     {vai encontrar a diferenca entre os 2 valores
                AR=AR+AY1;                      {vai ver se o erro e' inferior a +diferenca
                if GE JUMP igual;
                jump diferente;
        igual:
                i1=DM(endBloco);l1=0;m1=1;
        DM(I1,M1)=0x0080;         {altera 1º Byte do Bloco p/ o valor de fim de bloco (80)    }
                jump fimCompressaoBloco;  {a compressao do bloco termina aqui                           }
        diferente:
                i4=sr0;m4=1;l4=0;         {I4 esta no 1º Byte da oldFrame do actual bloco    (estava em SR0)  }
                i3=^DCT_dm;                     {vai actualizar os 3 Bytes do respectivo bloco
                AX0=DM(I3,M3);
                PM(I4,M4)=AX0;
                AX0=DM(I3,M3);
                i3=^DCT dm+8;
                PM(I4,M4)=AX0;
                AX0=DM(I3,M3);
                PM(I4,M4)=AX0;
                call quantificacao;
        fimCompressaoBloco:                     {vai actualizar end para o bloco seguinte da Old frame e actual
                AX0=DM(endRelOldBloco);AY0=3;
                AR=AX0+AY0;
                DM(endRelOldBloco)=AR;          {guarda end para o bloco seguinte da Old frame
                DM(endBloco)=i1;          {guarda ultimo endereco escrito no pacote comprimido          }
rts;

quantificacao:
{QUANTIFICACAO ORDEM ZIGZAG}
        i1=DM(endBloco);l1=0;m1=1;              {vai buscar o ultimo endereco do pacote guardado        }
        dis m_mode;
        i3=^DCT_dm;m3=0;
        i7=MY1;l7=0;m7=1;               { i7 -> inicio Q_pm a usar -> formato (1.15) com sinal          }
        i6=^zigzag pm+1;                     { i6 -> inicio zigzag pm (enderecos das posicoes zigzag
        l6=0;
        m6=1;
        AX0=i3;
        AY0=PM(I6,M6);
        MR=0, MX0=DM(I3,M3), MY0=PM(I7,M7);     {inicializa MR e le' os 1º valores da DCT e quantificacao inversa}
        cntr=m*m-2;                             {efectua apenas 62 ciclos os 2 ultimos sao feitos individual/
        DO loop3 UNTIL CE;                   {DCT dm x 1/Q dm, (quantificacao)                          }
               AR=AX0+AY0, AY0=PM(I6,M6);      {vai ler a proxima posicao zigzag para a DCT               }
               I3=AR;                          {guarda em I3 para ler a DCT no proximo ciclo              }
               MR=MX0*MY0 (RND), MX0=DM(I3,M3),MY0=PM(I7,M7);  {produto de cada elemento das 2 matrizes  }
        loop3: DM(I1,M1)=MR1;           {guarda o resultado Y_dm na ordem zigzag                    }
        MR=MX0*MY0(RND);                {63º produto                                                   }
        DM(I1,M1)=MR1;                          {guarda o resultado Y dm na ordem zigzag               }
        AR=AX0+AY0;
        I3=AR;                                  {Proxima posicao zigzag
        MX0=DM(I3,M3),MY0=PM(I7,M7);            {Leitura dos ultimos valores
        MR=MX0*MY0 (RND);               {63º produto                                                   }
        DM(I1,M1)=MR1;                          {guarda o resultado Y_dm na ordem zigzag               }
{PROCURA ULTIMO ZERO PARA COLOCAR CARACTER DE FIM BLOCO (00FF)}
        M1=-1;                                  {M1 negativo para decrementar o endereco               }
        modify(I1,M1);                       {Repoe I1 no ultimo valor Y dm
       AX0=DM(I1,M1);                           {Le ultimo valor                                       }
        ezero:
            AR=PASS AX0, AX0=DM(I1,M1);   {coloca-o em AR e le valor seguinte                           }
        if EQ JUMP ezero;               {se for zero vai para o label ezero iniciando um novo ciclo }
        M1=3;                           {altera M1 para voltar a posicao do 1º zero                    }
        MODIFY(I1,M1);
                M1=1;
        DM(I1,M1)=0x0080;         {altera 1º zero do ultimo grupo p/ o valor de fim de bloco (80)      }
                                                {nota este e' o maximo absoluto em 8 bits com sinal (-128)  }
```

```
rts;

buscaTabelaQ:
{vai procurar a tabela Q a usar}
        AX0=DM(nivelCompressao);
        AR=PASS AX0;                            {AR tem o nivel da tabela q de compressao (quantificacao)   }
        if EQ JUMP nivelzero;                   {se for zero vai para o label nivelzero compressao minima   }
                MY1=^Q3 pm;                     { i7 -> inicio Q3 pm a usar -> formato (1.15) com sinal
                AX0=^DC_CoefC;   DM(DC_Coef)=AX0;
                AX0=^DC_LengC;   DM(DC_Leng)=AX0;
                AX0=^AC_CoefC;   DM(AC_Coef)=AX0;
                AX0=^AC_LengC;   DM(AC_Leng)=AX0;
                jump fim1;
        nivelzero:
               MY1=^Q pm;                       { i7 -> inicio Q pm a usar -> formato (1.15) com sinal
                AX0=^DC CoefC;   DM(DC Coef)=AX0;
                AX0=^DC_LengC;   DM(DC_Leng)=AX0;
                AX0=^AC_CoefC;   DM(AC_Coef)=AX0;
                AX0=^AC_LengC;   DM(AC_Leng)=AX0;
        fim1:
rts;
actualizaNivelQ:
        {jump fimActualizaQ;}{######PARA TESTES#######}
        i7=^pacoteFinal+4;l7=0;m7=0;
       AY0=^pacote;                                     {1° endereco o pacote
        AX0=DM(endBloco);               {ultimo endereco o pacote (sem huffman)                             }
        AR=AX0-AY0;                             {AR tem a diferenca entre os dois enderecos
        AX0=AR;
        AY0=256;                        {para um debito de 65536bps->8192Bps/4(n° min frames aceitavel)     }
                                                {2048B/F=>512B/pacote=>128B/GOB sem huffman =>256 B         }
        AR=AX0-AY0;                             {verificar se e' suprior 256
        if GT JUMP maximo;                      {se for superior actualiza  o nivel para comp.max.          }
        jump inferior;
        maximo:
                AX0=1;
                DM(nivelCompressao)=AX0;  {Actuliza o nivel de compressao para o proximo GOB          }
                AX0=DM(i7,m7);                  {vai activar o bit no respectivo Byte de controlo           }
                AY0=DM(pMASC);
                AR=AX0 OR AY0;
                DM(i7,m7)=AR;
                jump fimActualizaQ;
        inferior:
                AY0=111;
                AR=AX0-AY0;                     {verificar se e' inferior a 111 (256/2,3)
        if LT JUMP minimo;                      {se for superior ou igual actualiza  o nivel para comp.minima.
        jump fimActualizaQ;
        minimo:
                AX0=0;
                DM(nivelCompressao)=AX0;  {Actuliza o nivel de compressao para o proximo GOB          }
                AX0=DM(i7,m7);                  {vai desactivar o bit no respectivo Byte de controlo
                AY0=DM(nMASC);
                AR=AX0 AND AY0;
                DM(i7,m7)=AR;
        fimActualizaQ:
rts;
comprimeBlocos:
        call buscaTabelaQ;
        sr0=0;DM(endRelOldBloco)=sr0;           { inicializa os end para a old frame e frame comprimida
        sr1=^pacote+6;l1=0;m1=1;DM(endBloco)=sr1;{ os 4 1° Bytes sao de controlo(o MIC escreve no 1°e 2° byte) }
        i4=^blocos;l4=0;m4=1;                   {#####################  1°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+64;l4=0;m4=1;        {#####################  2°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+128;l4=0;m4=1;       {#####################  3°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+192;l4=0;m4=1;       {#####################  4°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+256;l4=0;m4=1;       {#####################  5°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+320;l4=0;m4=1;       {#####################  6°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+384;l4=0;m4=1;       {#####################  7°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+448;l4=0;m4=1;       {#####################  8°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+512;l4=0;m4=1;       {#####################  9°  B L O C O   ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+576;l4=0;m4=1;       {#####################  10°  B L O C O  ########################}
        call comprimebloco;
```

```
        call estimaDiferencas;
        i4=^blocos+640;l4=0;m4=1;         {#####################  11°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+704;l4=0;m4=1;         {#####################  12°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+768;l4=0;m4=1;         {#####################  13°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+832;l4=0;m4=1;         {#####################  14°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+896;l4=0;m4=1;         {#####################  15°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+960;l4=0;m4=1;         {#####################  16°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1024;l4=0;m4=1;                {#####################  17°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1088;l4=0;m4=1;                {#####################  18°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1152;l4=0;m4=1;                {#####################  19°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1216;l4=0;m4=1;                {#####################  20°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1280;l4=0;m4=1;                {#####################  21°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1344;l4=0;m4=1;                {#####################  22°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1408;l4=0;m4=1;                {#####################  23°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1472;l4=0;m4=1;                {#####################  24°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1536;l4=0;m4=1;                {#####################  25°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1600;l4=0;m4=1;                {#####################  26°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1664;l4=0;m4=1;                {#####################  27°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1728;l4=0;m4=1;                {#####################  28°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1792;l4=0;m4=1;                {#####################  29°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1856;l4=0;m4=1;                {#####################  30°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1920;l4=0;m4=1;                {#####################  31°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+1984;l4=0;m4=1;                {#####################  32°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2048;l4=0;m4=1;                {#####################  33°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2112;l4=0;m4=1;                {#####################  34°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2176;l4=0;m4=1;                {#####################  35°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2240;l4=0;m4=1;                {#####################  36°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2304;l4=0;m4=1;                {#####################  37°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2368;l4=0;m4=1;                {#####################  38°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2432;l4=0;m4=1;                {#####################  39°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
```

```
        i4=^blocos+2496;l4=0;m4=1;                {#####################  40°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2560;l4=0;m4=1;                {#####################  41°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2624;l4=0;m4=1;                {#####################  42°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2688;l4=0;m4=1;                {#####################  43°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2752;l4=0;m4=1;                {#####################  44°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2816;l4=0;m4=1;                {#####################  45°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2880;l4=0;m4=1;                {#####################  46°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+2944;l4=0;m4=1;                {#####################  47°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3008;l4=0;m4=1;                {#####################  48°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3072;l4=0;m4=1;                {#####################  49°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3136;l4=0;m4=1;                {#####################  50°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3200;l4=0;m4=1;                {#####################  51°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3264;l4=0;m4=1;                {#####################  52°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3328;l4=0;m4=1;                {#####################  53°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3392;l4=0;m4=1;                {#####################  54°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3456;l4=0;m4=1;                {#####################  55°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3520;l4=0;m4=1;                {#####################  56°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3584;l4=0;m4=1;                {#####################  57°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3648;l4=0;m4=1;                {#####################  58°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3712;l4=0;m4=1;                {#####################  59°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3776;l4=0;m4=1;                {#####################  60°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3840;l4=0;m4=1;                {#####################  61°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3904;l4=0;m4=1;                {#####################  62°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+3968;l4=0;m4=1;                {#####################  63°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4032;l4=0;m4=1;                {#####################  64°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4096;l4=0;m4=1;                {#####################  65°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4160;l4=0;m4=1;                {#####################  66°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4224;l4=0;m4=1;                {#####################  67°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4288;l4=0;m4=1;                {#####################  68°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4352;l4=0;m4=1;                {#####################  69°  B L O C O  ########################}
```

```
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4416;l4=0;m4=1;              {#####################  70°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4480;l4=0;m4=1;              {#####################  71°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4544;l4=0;m4=1;              {#####################  72°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4608;l4=0;m4=1;              {#####################  73°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4672;l4=0;m4=1;              {#####################  74°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4736;l4=0;m4=1;              {#####################  75°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4800;l4=0;m4=1;              {#####################  76°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4864;l4=0;m4=1;              {#####################  77°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4928;l4=0;m4=1;              {#####################  78°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+4992;l4=0;m4=1;              {#####################  79°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5056;l4=0;m4=1;              {#####################  80°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5120;l4=0;m4=1;              {#####################  81°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5184;l4=0;m4=1;              {#####################  82°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5248;l4=0;m4=1;              {#####################  83°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5312;l4=0;m4=1;              {#####################  84°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5376;l4=0;m4=1;              {#####################  85°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5440;l4=0;m4=1;              {#####################  86°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5504;l4=0;m4=1;              {#####################  87°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
        i4=^blocos+5568;l4=0;m4=1;              {#####################  88°  B L O C O  ########################}
        call comprimebloco;
        call estimaDiferencas;
rts;

huffman:
        ena m_mode;                             {Nao faz shift automático (modo inteiro)                        }
        i1=^pacote+6;l1=0;m1=0;m2=1;            { os 6 1° Bytes sao de controlo(o MIC escreve no 1°e 4° byte)
        {i1=^TESTE+6;l1=0;m1=0;m2=1;}{####### SO PARA TESTE ######}
        AX0=DM(tamanhoPacote);AY0=^pacoteFinal;
        AR=AX0+AY0;
        i7=AR;l7=0;m7=1;
        {#########                    OS REGISTOS SR0 E SR1 VAO SER PERDIDOS                  #########
        {#########   OS REGISTOS AX1 VAI SER USADO PARA CONTAR OS BITs NECESSARIOS POR LINHA  #########

        sr0=0;sr1=0;                            {inicializacao de valores                                       }
        DM(bitsShiftados)=sr0;
        DM(numeroZeros)=sr0;
        DM(sr0Temporario)=sr0;
        DM(sr1Temporario)=sr0;
        trataPrimeiraComponenteDC:
                AX0=DM(I1,M1);                  {AX0 tem o primeiro valor a tratar                              }
                AY0=0x0080;
                AR=AX0-AY0;                     {vai verificar se e' o fim de bloco (ausencia de actualizacao)
                if EQ jump trataFimBloco;
                i5=DM(DC_Leng);m5=1;l5=0; {I5 indexa a tabela de comprimento dos coeficientes DC                }
                i6=DM(DC Coef);m6=0;l6=0; {I6 indexa a tabela de coeficientes DC                                }
                AY0=0;                          {inicializacao do valor do intervalo DC->0                      }
                AX1=0;                          {inicializacao do n° bits e' para a posicao DC->0               }
                call procuraValor;
                AX0=DM(linhaLeng);
                MR0=DM(linhaCod);
```

```
                DM(codigoLeng)=AX0;
                DM(codigoCod)=MR0;
                call actualizaValor;
                AX0=DM(posicaoLeng);
                MR0=DM(posicaoCod);
                DM(codigoLeng)=AX0;
                DM(codigoCod)=MR0;
                call actualizaValor;
        verificaValor:                          {vai verificar se chegou ao fim da matriz ou segmento
                modify (i1,m2);                 {incrementa i1 para o proximo valor
                AX0=DM(I1,M1);                  {AX0 tem o proximo valor a tratar                       }
                AY0=0x0080;
                AR=AX0-AY0;                     {vai verificar se e' o fim do bloco actual              }
                if EQ jump trataFimBloco;
                AY0=0x0000;
                AR=AX0-AY0;                     {vai verificar se e' zero, se sim contabiliza-o
                if EQ jump contaZero;
                jump trataComponenteAC;
        contaZero:
                AR=DM(numeroZeros);
                AR=AR+1;
                DM(numeroZeros)=AR;
                AR=AR-16;                       {vai verificar se utrapassou o max de 0s(15) antes do valor AC
                if EQ jump maximoZeros;
                jump VerificaValor;
        maximoZeros:
                AR=0;
                DM(numeroZeros)=AR;
                AR=10;
                DM(codigoLeng)=AR;
                AR=0x03FA;
                DM(codigoCod)=AR;
                call actualizaValor;
                jump VerificaValor;
        trataComponenteAC:
        {###### AY0 vai ficar com posicao relativa dentro do buffer AC_Coef,i.e. 1 + número de zeros x 10 #####
                MR0=1;MR1=0;MR2=0;
                MX0=10;
                MY0=DM(numeroZeros);
                MR=MR+MX0*MY0 (UU);
                AY0=MR0;
                AX0=DM(AC_Leng);m5=1;l5=0;      {I2 indexa a tabela de comprimento dos coeficientes DC
                AR=AX0+AY0;
                i5=AR;
                AX0=DM(AC Coef);m6=0;l6=0;      {I3 indexa a tabela de coeficientes DC
                AR=AX0+AY0;
                i6=AR;
                AY0=1;                          {inicializacao do valor do intervalo AC->1              }
                AX1=1;                          {inicializacao do n° bits e' para a posicao  AC->1      }
                call procuraValor;
                AX0=0;
                DM(numeroZeros)=AX0;            {inicializa o numero de zeros
                AX0=DM(linhaLeng);
                MR0=DM(linhaCod);
                DM(codigoLeng)=AX0;
                DM(codigoCod)=MR0;
                call actualizaValor;
                AX0=DM(posicaoLeng);
                MR0=DM(posicaoCod);
                DM(codigoLeng)=AX0;
                DM(codigoCod)=MR0;
                call actualizaValor;
                jump verificaValor;
        trataFimBloco:
                i6=DM(AC Leng);l6=0;m6=0;
                AX0=PM(i6,m6);                  {AX0 tem o comprimento do codigo de fim bloco (1° valor tabela)
                i6=DM(AC_Coef);l6=0;m6=0;
                MR0=PM(i6,m6);                  {mr0 tem o codigo de fim de bloco (1° valor da tabela)
                DM(codigoLeng)=AX0;
                DM(codigoCod)=MR0;
                call actualizaValor;
                modify (i1,m2);                 {incrementa i1 para o proximo valor
                AX0=DM(I1,M1);                  {AX0 tem o próximo valor a tratar                       }
                AY0=0x0081;
                AR=AX0-AY0;                     {vai verificar se proximo e' o fim do GOB               }
                if EQ jump fimHuffman;
                AY0=0x0080;
                AR=AX0-AY0;                     {vai verificar se e' novo fim de bloco se nao e' componenteDC
                if EQ jump trataFimBloco;
                i5=DM(DC_Leng);m5=1;l5=0; {I5 indexa a tabela de comprimento dos coeficientes DC        }
                i6=DM(DC_Coef);m6=0;l6=0; {I6 indexa a tabela de coeficientes DC                        }
                AY0=0;                          {inicializacao do valor do intervalo DC->0              }
                AX1=0;                          {inicializacao do n° bits e' para a posicao DC->0       }
                call procuraValor;
                AX0=DM(linhaLeng);
                MR0=DM(linhaCod);
                DM(codigoLeng)=AX0;
                DM(codigoCod)=MR0;
```

```
                call actualizaValor;
                AX0=DM(posicaoLeng);
                MR0=DM(posicaoCod);
                DM(codigoLeng)=AX0;
                DM(codigoCod)=MR0;
                call actualizaValor;
                jump verificaValor;
        fimHuffman:
                MR0=DM(sr0Temporario);          {vai buscar os ultimos valores de SR para guardar ultimo valor
                AR=DM(bitsShiftados);
                AR=8-AR;
                AR=-AR;  SE=AR;                 {faz shift para colocar ultimo valor em sr0 encostado a esquerda}

                SR= NORM MR0(LO);       {sr0 tem o valor da linha encostado a direita                   }
                DM(i7,m7)=sr0;AX0=i7;
                AY0=^pacoteFinal;
                AR=AX0-AY0;
                DM(tamanhoPacote)=AR;           {o tamanho do pacote (com huffman) e' guardado temporariamente
                                                {em memoria, e' necessario para proximo GOB e na descompressao
                                                {sera  indicado no 5º e 6º B de controlo                }
rts;

procuraValor:                                   {vai verificar as diferencas sucessivas para localizar a linha
                {AY0 (valor do intervalo e' inicializado na rotina huffman conforme seja para AC->1 ou DC->0  }
                {AX1 para o nº bits e' inicializado na rotina huffman conforme seja para AC->1 ou DC->0
                AY1=0;                          {AY1 guarda o valor anterior de AY0 (intervalo anterior
        testaLinha:
                AX0=DM(I1,M1);                  {AX0 tem o valor da (primeiros Bytes)
                AR=AX0-AY0;                     {vai encontrar a diferenca entre valor e intervalo      }
                if LE JUMP continuaTestar;      {se e' inferior ou igual a zero o valor pode ser negativo   }
                jump outraLinha;        {se e' maior que zero entao o valor e' superior ao intervalo    }
                continuaTestar:
                        AR=AX0+AY0;             {vai encontrar a soma entre os 2 valores                }
                        if GE JUMP procuraPosicao;{se e' superior ou igual a zero o valor esta no intervalo   }
                outraLinha:
                        MR=0;                   {reinicializa MR                                        }
                        MR0=1;                  {MR fica com o valor 1 para calculo de proximo intervalo    }
                        MX0=2;                  {para calculo de proximo intervalo                      }
                        MY0=AY0;        {para calculo de proximo intervalo                      }
                        AY1=AY0;        {vai guardar o valor anterior de AY0 em AY1 (intervalor anter.)     }
                        MR=MR+MX0*MY0 (SS);     {vai actualizar AY0 para a linha seguinte (valorAnterior x 2 +1)}
                        AY0=MR0;        {este novo intervalo vai validar a proxima linha        }
                        AR=AX1+1;               {vai actualizar AX1 para o nº bits da posicao da proxima linha
                        AX1=AR;
                        modify (i6,m5);         {incrementa o apontador de DC Coef para a proxima linha
                        modify (i5,m5);         {incrementa o apontador de DC Leng para a proxima linha
                        jump testaLinha;
        procuraPosicao:                         {A linha foi encontrada, vai procurar a posicao
                valorNegativo:
                        AX0=DM(I1,M1);          {AX0 tem o valor da componente DC nota m1=0
                        AR=AX0+AY0;             {da o valor da posicao se o valor for negativo
                        MR1=AR;                 {guarda a posicao temporaria em MR1
                        AR=AX0+AY1;             {se for negativo o valor sumado ao intervalo anterior e negativo}
                if LE JUMP valorEncontrado;     {se e' negativo MR1 tem a posicao correcta              }
                        AR=AX0;                 {se nao o valor e' positivo entao a posicao e' igual ao valor
                        MR1=AR;                 {MR1 tem o valor da posicao
        valorEncontrado:
                                                {ja temos toda a informacao para codificar o valor      }
                        AX0=pm(i5,m6);          {AX0 tem o comprimento do valor da linha (codigo)       }
                        DM(linhaLeng)=AX0;
                        MR0=pm(i6,m6);          {mr0 tem o codigo da linha
                        DM(linhaCod)=MR0;
                        DM(posicaoLeng)=AX1;
                        DM(posicaoCod)=MR1;
rts;

actualizaValor:
                sr0=DM(sr0Temporario);          {vai buscar os valores guardados SR (talvez ainda nao copiado)
                AX0=DM(codigoLeng);             {AX0 tem o comprimento do valor da linha (codigo)       }
                AY1=DM(codigoCod);              {mr0 tem o codigo da linha
                                                {Vai actualizar bits shitado e realizar novo shift      }
                AY0=DM(bitsShiftados);          {AY0 tem o valor dos bits anteriores shiftados
                AR=AX0+AY0;
                DM(bitsShiftados)=AR;           {actualiza o valor dos bits anteriores shiftados        }
                AR=-AX0;
                SE=AR;
                SR= NORM SR0(LO);       {sr0 tem espaco para o valor disponivel a direita       }
                AR=SR0 OR AY1;                  {ar tem o valor  encostado a direita
                DM(sr0Temporario)=AR;           {guarda os valores temporarios de SR
                DM(sr1Temporario)=sr1;
                AR=DM(bitsShiftados);           {vai ver se ultrapassou oito (byte completo)
                AR=AR-8;
                if GE jump copiaValor1;         {AR tem o numero de bits shiftados menos 8              }
                jump fimActualizaValor;
        copiaValor1:
                DM(bitsShiftados)=AR;           {Actualiza bitShiftados que ainda nao foram guardados
                MR0=DM(sr0Temporario);          {vai buscar os valores guardados de SR
                MR1=DM(sr1Temporario);
```

```
                if EQ jump copiarJa1;
                AR=-AR;
                SE=AR;                          {faz shift contrario para repor byte mais significativo em sr0
                SR= ASHIFT MR1(HI);             {sr0 tem o valor parte do...
                SR=SR OR LSHIFT MR0(LO);  {sr0 tem o valor completo                             }
                MR0=SR0;
                copiarJa1:
                DM(i7,m7)=MR0;
                AR=DM(bitsShiftados);           {o valor pode ser de 16 bits e haver entao 2 valores a copiar
                AR=AR-8;
                if GE jump copiaValor2;         {AR tem o numero de bits shiftados menos 8              }
                jump fimActualizaValor;
        copiaValor2:
                DM(bitsShiftados)=AR;           {Actualiza bitShiftados que ainda nao foram guardados
                MR0=DM(sr0Temporario);          {vai buscar os valores guardados de SR
                MR1=DM(sr1Temporario);
                if EQ jump copiarJa2;
                AR=-AR;
                SE=AR;                          {faz shift contrario para repor byte mais significativo em sr0

                SR= ASHIFT MR1(HI);             {sr0 tem o valor parte do...
                SR=SR OR LSHIFT MR0(LO);  {sr0 tem o valor completo                             }
                MR0=SR0;
                copiarJa2:
                DM(i7,m7)=MR0;
        fimActualizaValor:
rts;

copiaPacote:                                    { copia 4 linhas para construir pacote na DM na ordem normal
        i1=^pacote;m1=1;l1=0;                   { os 6 primeiros Bytes sao de controlo(o MIC escreve no 1º byte)}
        m1=6;
        MODIFY(I1,M1);
        m1=1;
        CNTR=4;                                 { le  4 linhas píxeis do segmento                       }
        DO linha00 UNTIL CE;
                m0=53;
                MODIFY(I0,M0);                  {altera m0 para ir para linha seguinte
        m0=1;
                procura0:
                        AR=0;
                        DMOVLAY=DM(valorDMOVLAY);
                        AY0=DM(i0,m0);
                        DMOVLAY=0;
                        AR=AR+AY0;              {vai a procura do sincronismo horizontal 00h ou 80h     }
                        if EQ jump confirmar1;
                        AR=-0x0080;
                        AR=AR+AY0;              {vai a procura do sincronismo horizontal 00h ou 80h     }
                        if EQ jump confirmar1;
                        AX1=AX0;
                        AX0=AY0;
                        jump procura0;
                confirmar1:
                        AR=AX0-0x0010;          {vai confirmar o ultimo valor  (01h)
                        if EQ jump confirmar2;
                        jump procura0;
                confirmar2:
                        AR=AX1-0x0010;          {vai confirmar o penultimo anterior (01h)               }
                        if EQ jump encontrado0;
                        jump procura0;
                encontrado0:
                m0=3;
                MODIFY(I0,M0);                  {altera m0 para ir 1º píxel da linha seguinte
        m0=1;
                CNTR=352;                       {numero de pixeis por linha}
                DO umalinha00 UNTIL CE;
                        DMOVLAY=DM(valorDMOVLAY);
                        AX0=DM(i0,m0);
                        DMOVLAY=0;
                umalinha00:DM(i1,m1)=AX0; { Guarda na DM interna na ordem normal                        }
        linha00:nop;
rts;
copiaSegmento:                                  { copia segmeno para a DM interna na ordem normal       }
        i4=^entrada;m4=1;l4=0;
        CNTR=16;                        { le  16 linhas píxeis do segmento                      }
        DO linha01 UNTIL CE;                    { cada segmento tera 84 blocos (42x2)
                m0=53;
                MODIFY(I0,M0);                  {altera m0 para ir para a linha seguinte                }
        m0=1;
                procura1:
                        AR=0;
                        DMOVLAY=DM(valorDMOVLAY);
                        AY0=DM(i0,m0);
                        DMOVLAY=0;
                        AR=AR+AY0;              {vai a procura do sincronismo horizontal 00h ou 80h     }
                        if EQ jump confirmar3;
                        AR=-0x0080;
                        AR=AR+AY0;              {vai a procura do sincronismo horizontal 00h ou 80h     }
                        if EQ jump confirmar3;
```

```
                        AX1=AX0;
                        AX0=AY0;
                        jump procura1;
                confirmar3:
                        AR=AX0-0x0010;          {vai confirmar o ultimo valor  (01h)
                        if EQ jump confirmar4;
                        jump procura1;
                confirmar4:
                        AR=AX1-0x0010;          {vai confirmar o penultimo anterior (01h)               }
                        if EQ jump encontrado1;
                        jump procura1;
                encontrado1:
                m0=3;
                MODIFY(I0,M0);                  {altera m0 para ir 1º píxel da linha seguinte
        m0=1;
                CNTR=352;                       {numero de pixeis por linha
                DO umalinha01 UNTIL CE;
                        DMOVLAY=DM(valorDMOVLAY);
                        AX0=DM(i0,m0);
                        DMOVLAY=0;
                umalinha01:PM(i4,m4)=AX0; { Guarda na DM interna na ordem normal                        }
        linha01:nop;
rts;
semCompressao:
        ena m_mode;                             {Nao faz shift automático (modo inteiro)                }
        call DSP_FLG_OFF;

        call copiaPacote;               {copia 1º pacote                                        }
        AX1=^pacote;                            {para o MIC ler a 1ª posicao dos dados o reg IDMA (3FE0) ja
        AR=AX1 or 0x4000;               {para aceder a DM                                       }
        DM(0x3fe0)= AR;                         {contem a primeira posicao do buffer de saida
        i1=^pacote+1;m1=1;l1=0;
        MY0=4;
        MR=MX0*MY0(UU);                         {numero do pacote                                       }
        DM(i1,m1)=1;                            {compressao do pacote
        DM(i1,m1)=MR0;

        call DSP_FLG_ON;                { aviso do DSP ao MIC que existe um pacote pronto       }
        AY0=PMASC_PF0;                          { espera pelo MIC p/ saber se ja leu pacote anterior
        naoenviado1:                            { a MIC FLG e' recebida na PF0
                AX0=DM(PFDATA);
                AR=AX0 AND AY0;
        if EQ JUMP naoenviado1;                 { se for zero, o MIC ainda nao leu o pacote
        call DSP_FLG_OFF;               { aviso do DSP ao MIC que nao ha pacote novo            }

        call copiaPacote;               {copia 2º pacote                                        }
        AX1=^pacote;                            {para o MIC ler a 1ª posicao dos dados o reg IDMA (3FE0) ja
        AR=AX1 or 0x4000;               {para aceder a DM                                       }
        DM(0x3fe0)= AR;                         {contem a primeira posicao do buffer de saida
        i1=^pacote+1;m1=1;l1=0;
        MY0=4;
        DM(i1,m1)=1;                            {compressao do pacote
        MR=MX0*MY0(UU);                         {numero do pacote                                       }
        AR=MR0+1;
        DM(i1,m1)=AR;

        call DSP_FLG_ON;                { aviso do DSP ao MIC que existe um pacote pronto       }
        AY0=PMASC PF0;                          { espera pelo MIC p/ saber se ja enviou pacote anterior
        naoenviado2:                            { a MIC FLG e' recebida na PF0
                AX0=DM(PFDATA);
                AR=AX0 AND AY0;
        if EQ JUMP naoenviado2;                 { se for zero, o MIC ainda nao enviou o pacote
        call DSP_FLG_OFF;               { aviso do DSP ao MIC que nao ha pacote novo            }

        call copiaPacote;               {copia 3º pacote                                        }
        AX1=^pacote;                            {para o MIC ler a 1ª posicao dos dados o reg IDMA (3FE0) ja
        AR=AX1 or 0x4000;               {para aceder a DM                                       }
        DM(0x3fe0)= AR;                         {contem a primeira posicao do buffer de saida
        i1=^pacote+1;m1=1;l1=0;
        MY0=4;
        DM(i1,m1)=1;                            {compressao do pacote
        MR=MX0*MY0(UU);                         {numero do pacote                                       }
        AR=MR0+2;
        DM(i1,m1)=AR;

        call DSP_FLG_ON;                { aviso do DSP ao MIC que existe um pacote pronto       }
        AY0=PMASC_PF0;                          { espera pelo MIC p/ saber se ja enviou pacote anterior
        naoenviado3:                            { a MIC FLG e' recebida na PF0
                AX0=DM(PFDATA);
                AR=AX0 AND AY0;
        if EQ JUMP naoenviado3;                 { se for zero, o MIC ainda nao enviou o pacote
        call DSP_FLG_OFF;               {aviso do DSP ao MIC que nao ha pacote novo             }

        call copiaPacote;               {copia 4º pacote                                        }
        AX1=^pacote;                            {para o MIC ler a 1ª posicao dos dados o reg IDMA (3FE0) ja
        AR=AX1 or 0x4000;               {para aceder a DM                                       }
        DM(0x3fe0)= AR;                         {contem a primeira posicao do buffer de saida
        i1=^pacote+1;m1=1;l1=0;
```

```
        DM(i1,m1)=1;                            {compressao do pacote
        MR=MX0*MY0(UU);                         {numero do pacote                                       }
        AR=MR0+3;
        DM(i1,m1)=AR;

        call DSP_FLG_ON;                { aviso do DSP ao MIC que existe um pacote pronto          }
        AY0=PMASC_PF0;                          { espera pelo MIC p/ saber se ja enviou pacote anterior
        naoenviado4:                            { a MIC FLG e' recebida na PF0
                AX0=DM(PFDATA);
                AR=AX0 AND AY0;
        if EQ JUMP naoenviado4;                 { se for zero, o MIC ainda nao enviou o pacote
        call DSP_FLG_OFF;               { aviso do DSP ao MIC que nao ha pacote novo                    }

rts;
comCompressao:
        AY0=1;
        AX0=DM(GOB);
        AR=AX0-AY0;
                if EQ jump GOB1;
        AY0=2;
        AR=AX0-AY0;
                if EQ jump GOB2;
        AY0=3;
        AR=AX0-AY0;
                if EQ jump GOB3;
        AY0=4;
        AR=AX0-AY0;
                if EQ jump GOB4;
        GOB1:
                call DSP_FLG_OFF;
                call copiaSegmento;             { copia para a DM interna ordenada em 88 blocos
                i1=^pacoteFinal+1;m1=1;l1=0;
                DM(i1,m1)=0;                    {tipo de compressao do pacote
                AR=DM(segmento);        {numero do pacote (segmento)                                    }
                DM(i1,m1)=AR;
                AY0=0;
                AR=AR+AY0;                      {vai confirmar se esta no 1° GOB do primeiro segmento
                if EQ jump reiniciaQ;           {se sim reiniciaQ com o valor 0 para maxima qualidade
                jump naoReiniciaQ;
                reiniciaQ:
                        DM(nivelCompressao)=AY0;
                        i7=^pacoteFinal+4;l7=0;m7=0;
                        DM(i7,m7)=0;            {reinicializa a compressao para o valor minimo (compress. maxima}
                naoReiniciaQ:
                AY0=6;
                DM(tamanhoPacote)=AY0;          {devido aos Bytes de controlo depois sera o fim do GOB anterior
                {###################### 1° GOB do PACOTE ########################}
                call copiaBlocos;       { copia para a DM interna ordenada em 84 blocos                 }
                call comprimeBlocos;            { compressao dos 88 blocos e preparacao de pacote unico
                DM(I1,M1)=0x0081;       {o  Byte 81 sigifica fim de segmento (de pacote)        }
                call huffman;
                AR=PMASC_PF6;
                DM(pMASC)=AR;                   {para actualizar a matriz de quantificacao do proximo GOB   }
                AR=NMASC PF6;
                DM(nMASC)=AR;                   {para actualizar a matriz de quantificacao do proximo GOB   }
                call actualizaNivelQ;
                {jump testeGOB1;}{#############TESTES################}
                jump fimGOB;
        GOB2:
                call copiaSegmento;             { copia para a DM interna ordenada em 88 blocos
                {###################### 2° GOB do PACOTE ########################}
                call copiaBlocos;       { copia para a DM interna ordenada em 84 blocos                 }
                call comprimeBlocos;            { compressao dos 88 blocos e preparacao de pacote unico
                DM(I1,M1)=0x0081;       {o  Byte 81 sigifica fim de segmento (de pacote)        }
                call huffman;
                AR=PMASC PF5;
                DM(pMASC)=AR;                   {para actualizar a matriz de quantificacao do proximo GOB   }
                AR=NMASC_PF5;
                DM(nMASC)=AR;                   {para actualizar a matriz de quantificacao do proximo GOB   }
                call actualizaNivelQ;
                jump fimGOB;
        GOB3:
                call copiaSegmento;             { copia para a DM interna ordenada em 88 blocos
                {###################### 3° GOB do PACOTE ########################}
                call copiaBlocos;       { copia para a DM interna ordenada em 84 blocos                 }
                call comprimeBlocos;            { compressao dos 88 blocos e preparacao de pacote unico
                DM(I1,M1)=0x0081;       {o  Byte 81 sigifica fim de segmento (de pacote)        }
                call huffman;
                AR=PMASC PF4;
                DM(pMASC)=AR;                   {para actualizar a matriz de quantificacao do proximo GOB   }
                AR=NMASC_PF4;
                DM(nMASC)=AR;                   {para actualizar a matriz de quantificacao do proximo GOB   }
                call actualizaNivelQ;
                jump fimGOB;
        GOB4:
                call copiaSegmento;             { copia para a DM interna ordenada em 88 blocos
                {###################### 4° GOB do PACOTE ########################}
                call copiaBlocos;       { copia para a DM interna ordenada em 84 blocos                 }
```

```
                call comprimeBlocos;              { compressao dos 88 blocos e preparacao de pacote unico
                DM(I1,M1)=0x0081;        {o  Byte 81 sigifica fim de segmento (de pacote)           }
                call huffman;
        {######### vai colocar o tamanhoPacote no 5° e 6° byte #########}
        testeGOB1:
        i1=^pacoteFinal+5;m1=-1;l1=0;
        AR=DM(tamanhoPacote);
        {AR=1353;}{######PARA TESTES#########}
        DM(i1,m1)=AR;                             {os 8 bits LSB do tamanho de 4 GOBs (com huffman) vai no 6° Byte}
        SI=AR;
        SR=LSHIFT SI BY -8 (HI);          {vai fazer shift 8 bits para colocar os 4 bits MSB no 5 Byte        }
                                                  {SR0 tem os 4 Bytes MSB
        i7=^pacoteFinal+4;l7=0;m7=0;
        AY0=DM(i7,m7);                            {AY0 tem o valor da quantificacao nos 4 bits mais significativos}
        AR=SR1 OR AY0;
        DM(i1,m1)=AR;                             {os 4 bis MSB vao no 5° Byte junto com a quantificacao
        AX1=^pacoteFinal;                 {para o MIC ler a 1ª posicao dos dados o reg IDMA (3FE0) ja         }
        AR=AX1 or 0x4000;                 {para aceder a DM                                                   }
        DM(0x3fe0)= AR;                           {contem a primeira posicao do buffer de saida

        call DSP FLG ON;                  { aviso do DSP ao MIC que existe um pacote pronto          }
        AY0=PMASC_PF0;                            { espera pelo MIC p/ saber se ja enviou pacote anterior
        naoenviado:                               { a MIC_FLG e' recebida na PF0
                AX0=DM(PFDATA);
                AR=AX0 AND AY0;
        if EQ JUMP naoenviado;                    { se for zero, o MIC ainda nao enviou o pacote
        call DSP FLG OFF;                 { aviso do DSP ao MIC que nao ha pacote novo                        }

        AR=PMASC_PF7;
        DM(pMASC)=AR;                             {para actualizar a matriz de quantificacao do proximo GOB   }
        AR=NMASC_PF7;
        DM(nMASC)=AR;                             {para actualizar a matriz de quantificacao do proximo GOB   }
        call actualizaNivelQ;

        fimGOB:
rts;
trataSegmento:
        DM(segmento)=MX0;
        DM(valorDMOVLAY)=SR0;
        DM(GOB)=SR1;
        AY0=PMASC PF2;                            { vai ler o tipo de compressao desejada (sem / mjpeg)
        AX0=DM(PFDATA);
        AR=AX0 AND AY0;
        if EQ jump com;
        if NE jump sem;
        com:
                call comCompressao;               {0 compressao-copia para a DM 88 blocos e prepara 1 pacotes }
               jump fim;
        sem:
                call semCompressao;               {1 copia para a DM int. 4 linhas de cada vez e prep 4 pacotes
        fim:
rts;

.ENDMOD;
```

Anexo C. Firmware de Encriptação

```
/*
*****************************************************************
**       Advanced Encryption Standard implementation in C.      **
**       By Niyaz PK                                            **
**       E-mail: niyazlife@gmail.com                            **
**       Downloaded from Website: www.hoozi.com                 **
*****************************************************************
This is the source code for encryption using the latest AES algorithm.
AES algorithm is also called Rijndael algorithm. AES algorithm is
recommended for non-classified use by the National Institute of Standards
and Technology(NIST), USA. Now-a-days AES is being used for almost
all encryption applications all around the world.

THE MAIN FEATURE OF THIS AES ENCRYPTION PROGRAM IS NOT EFFICIENCY; IT
IS SIMPLICITY AND READABILITY. THIS SOURCE CODE IS PROVIDED FOR ALL
TO UNDERSTAND THE AES ALGORITHM.

Comments are provided as needed to understand the program. But the
user must read some AES documentation to understand the underlying
theory correctly.

It is not possible to describe the complete AES algorithm in detail
here. For the complete description of the algorithm, point your
browser to:
http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Find the Wikipedia page of AES at:
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
*****************************************************************
*/

// Include stdio.h for standard input/output.
// Used for giving output to the screen.
#include<stdio.h>

// The number of columns comprising a state in AES. This is a constant in AES. Value=4
#define Nb 4

// The number of rounds in AES Cipher. Valor 10 para AES-128.
int Nr=10;

// The number of 32 bit words in the key. Valor 4 para AES-128.
int Nk=4;

// in - it is the array that holds the plain text to be encrypted.
// out - it is the array that holds the output CipherText after encryption.
// state - the array that holds the intermediate results during encryption.
unsigned char in[16], out[16], state[16];

// The array that stores the round keys.
unsigned char RoundKey[240];

// The Key input to the AES Program
unsigned char Key[16] = {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};

int sBox[256] =   {
    //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E     F
    0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76, //0
    0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0, //1
    0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15, //2
    0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75, //3
    0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84, //4
    0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf, //5
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8, //6
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2, //7
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73, //8
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb, //9
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79, //A
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08, //B
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a, //C
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e, //D
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf, //E
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16 }; //F

// The round constant word array, Rcon[i], contains the values given by
// x to th e power (i-1) being powers of x (x is denoted as {02}) in the field GF(28)
```

```
// Note that i starts at 1, not 0).
int Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
    0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39,
    0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a,
    0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
    0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
    0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc,
    0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
    0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
    0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94,
    0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
    0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
    0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
    0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb  };

// This function produces Nb(Nr+1) round keys. The round keys are used in each round to encrypt the states.
void KeyExpansion()
{
    int i,j;
    unsigned char temp[4],k;

    // The first round key is the key itself.
    for(i=0;i<4;i++)
    {
        RoundKey[i*4]=Key[i*4];
        RoundKey[i*4+1]=Key[i*4+1];
        RoundKey[i*4+2]=Key[i*4+2];
        RoundKey[i*4+3]=Key[i*4+3];
    }

    // All other round keys are found from the previous round keys.
    while (i < (4 * (10+1)))
    {
        for(j=0;j<4;j++)
        {
            temp[j]=RoundKey[(i-1) * 4 + j];
        }
        if (i % 4 == 0)
        {
            // This function rotates the 4 bytes in a word to the left once.
            // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

            // Function RotWord()
            {
                k = temp[0];
                temp[0] = temp[1];
                temp[1] = temp[2];
                temp[2] = temp[3];
                temp[3] = k;
            }

            // SubWord() is a function that takes a four-byte input word and
            // applies the S-box to each of the four bytes to produce an output word.

            // Function Subword()
            {
                temp[0]=sBox[temp[0]];
                temp[1]=sBox[temp[1]];
                temp[2]=sBox[temp[2]];
                temp[3]=sBox[temp[3]];
            }

            temp[0] =  temp[0] ^ Rcon[i/Nk];
        }
        else if (4 > 6 && i % 4 == 4)
        {
            // Function Subword()
            {
                temp[0]=sBox[temp[0]];
                temp[1]=sBox[temp[1]];
                temp[2]=sBox[temp[2]];
                temp[3]=sBox[temp[3]];
            }
        }
        RoundKey[i*4+0] = RoundKey[(i-Nk)*4+0] ^ temp[0];
        RoundKey[i*4+1] = RoundKey[(i-Nk)*4+1] ^ temp[1];
        RoundKey[i*4+2] = RoundKey[(i-Nk)*4+2] ^ temp[2];
        RoundKey[i*4+3] = RoundKey[(i-Nk)*4+3] ^ temp[3];
        i++;
    }
}

// This function adds the round key to state.
// The round key is added to the state by an XOR function.
```

```
void AddRoundKey(int round)
{
    int i;
         for(i=0;i<16;i++)
    {
        state[i] ^= RoundKey[round * 16 + i];

    }
}

// The SubBytes Function Substitutes the values in the
// state matrix with values in an S-box.
void SubBytes()
{
    int i;
    for(i=0;i<16;i++)
    {
        state[i] = sBox[state[i]];
    }
}

// The ShiftRows() function shifts the rows in the state to the left.
// Each row is shifted with different offset.
// Offset = Row number. So the first row is not shifted.
void ShiftRows()
{
    unsigned char temp;

    // Rotate first row 1 columns to left
    temp=state[4];
    state[4]=state[5];
    state[5]=state[6];
    state[6]=state[7];
    state[7]=temp;

    // Rotate second row 2 columns to left
    temp=state[8];
    state[8]=state[10];
    state[10]=temp;

    temp=state[9];
    state[9]=state[11];
    state[11]=temp;

    // Rotate third row 3 columns to left
    temp=state[12];
    state[12]=state[15];
    state[15]=state[14];
    state[14]=state[13];
    state[13]=temp;
}

// xtime is a macro that finds the product of {02} and the argument to xtime modulo {1b}
#define xtime(x)   ((x<<1) ^ (((x>>7) & 1) * 0x1b))

// MixColumns function mixes the columns of the state matrix
// The method used may look complicated, but it is easy if you know the underlying theory.
// Refer the documents specified above.
void MixColumns()
{
    int i;
    unsigned char Tmp,Tm,t;
    for(i=0;i<4;i++)
    {
        t=state[i];
        Tmp = state[i] ^ state[4+i] ^ state[8+i] ^ state[12+i] ;
        Tm = state[i] ^ state[4+i] ; Tm = xtime(Tm); state[i] ^= Tm ^ Tmp ;
        Tm = state[4+i] ^ state[8+i] ; Tm = xtime(Tm); state[4+i] ^= Tm ^ Tmp ;
        Tm = state[8+i] ^ state[12+i] ; Tm = xtime(Tm); state[8+i] ^= Tm ^ Tmp ;
        Tm = state[12+i] ^ t ; Tm = xtime(Tm); state[12+i] ^= Tm ^ Tmp ;
    }
}

// Cipher is the main function that encrypts the PlainText.
void Cipher()
{
    int i,round=0;

         //Copy the input PlainText to state array.
    for(i=0;i<16;i++)
    {
        state[i] = in[i];
    }

    // Add the First round key to the state before starting the rounds.
    AddRoundKey(0);

    // There will be Nr rounds.
```

```
    // The first Nr-1 rounds are identical.
    // These Nr-1 rounds are executed in the loop below.
    for(round=1;round<10;round++)
    {
        SubBytes();
         ShiftRows();
         MixColumns();
                 AddRoundKey(round);
    }

    // The last round is given below.
    // The MixColumns function is not here in the last round.
    SubBytes();
    ShiftRows();
    AddRoundKey(Nr);

    // The encryption process is over.
    // Copy the state array to output array.
    for(i=0;i<16;i++)
         {
         out[i]=state[i];
    }
}
void writeBuffer (unsigned char valorEscrito, int j)
{
         in[j]=valorEscrito;
}

unsigned char readBuffer(int j)
{
         unsigned char valorLido;
         valorLido=out[j];
         return valorLido;
}
unsigned char readKey(int j){
         unsigned char valorLido;
         valorLido=Key[j];
         return valorLido;
}
unsigned char readRoundKey(int j){
    unsigned char valorLido;
         valorLido=RoundKey[j];
         return valorLido;
}
```

Anexo D. Software Cliente

```
import javax.swing.*;
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.*;
import javax.imageio.*;


public class VideoVigi extends JApplet {
        boolean TestaApplet = true;
        JanelaVideoVigi janela = null;

        public void init(){
                janela = new JanelaVideoVigi(TestaApplet);
                janela.start();
        }

        public static void main(String[] args) {
                VideoVigi janelinha = new VideoVigi();
                janelinha.TestaApplet = false;
                janelinha.init();

        }
}
class JanelaVideoVigi extends JFrame implements Runnable{
        int Nb=4;
        int Nr=10;
        int Nk=4;
        byte AES_in[]= new byte [16];
        byte AES_out[] = new byte [16];
        byte state[][] = new byte [4][4];
        byte RoundKey[] = new byte[240];
        byte Key[] =  {0x00  ,0x01  ,0x02  ,0x03  ,0x04  ,0x05  ,0x06  ,0x07  ,0x08  ,0x09  ,0x0a  ,0x0b
,0x0c  ,0x0d  ,0x0e  ,0x0f};
        int Rcon[] = {
                0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
0x9a,
                0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,
0x39,
                0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
0x3a,
                0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,
0xd8,
                0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
0xef,
                0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66,
0xcc,
                0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
0x1b,
                0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4,
0xb3,
                0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,
0x94,
                0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10,
0x20,
                0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97,
0x35,
                0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2,
0x9f,
                0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,
0x04,
                0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc,
0x63,
                0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3,
0xbd,
                0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb
};
        int sBoxInv[] = {
                0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7,
0xfb
                , 0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde,
0xe9, 0xcb
                , 0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa,
0xc3, 0x4e
```

```
                , 0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b,
0xd1, 0x25
                , 0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65,
0xb6, 0x92
                , 0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d,
0x9d, 0x84
                , 0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3,
0x45, 0x06
                , 0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13,
0x8a, 0x6b
                , 0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4,
0xe6, 0x73
                , 0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75,
0xdf, 0x6e
                , 0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18,
0xbe, 0x1b
                , 0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd,
0x5a, 0xf4
                , 0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80,
0xec, 0x5f
                , 0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9,
0x9c, 0xef
                , 0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53,
0x99, 0x61
                , 0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21,
0x0c, 0x7d };
        int sBox[] =  {
                //0     1    2      3     4    5     6     7      8    9     A      B    C     D     E
F
                0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab,
0x76,
                0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72,
0xc0,
                0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31,
0x15,
                0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2,
0x75,
                0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f,
0x84,
                0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58,
0xcf,
                0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f,
0xa8,
                0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3,
0xd2,
                0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19,
0x73,
                0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b,
0xdb,
                0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4,
0x79,
                0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae,
0x08,
                0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b,
0x8a,
                0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d,
0x9e,
                0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28,
0xdf,
                0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb,
0x16 };

        int i,j,n,in=0,out=0,shift=7;
        short valorTestar=0;
        boolean TApplet,fim=false, inicio=true, receber=false, trafegoControlado=false, testeBlocos=true,
encriptacao=false;
        JPanel painel1 = null, painel2 = null;
        JPanel painel11 = null, painel12 = null,painel13 = null,painel14 = null;
        JPanel painel21 = null, painel22 = null,painel23 = null,painel24 = null;
        JLabel imagem1, imagem2, imagem3, imagem4;
        BufferedImage compressao1, compressao2, compressao3, compressao4;
        Container c;
        Choice choiceNivel;
        CheckboxGroup camara;Checkbox camara1, camara2, camara3, camara4, Pacote, testeB,testeAES;
        ClassLoader cl;

        Thread inputThread;
        String stringtest,endereco="10.0.0.8";
        DatagramSocket sd;
        DatagramPacket pacoteEnviado, pacoteRecebido;
        String Nome;
        int PortoLocal=10008;
        InetAddress IPpar;
        int PortoPar=10008;

        public JanelaVideoVigi(boolean TestaApplet) {
                TApplet=TestaApplet;
                setTitle("Video Vigilancia");
```

```
                setSize(750, 640);
                setResizable(false);
                if (TApplet) dispose();
                else setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setLayout(new BorderLayout());

                addWindowListener(new JanelaActiva());
                c = getContentPane();

                JMenu opcoes = new JMenu("Configurações");
                JMenuItem rede = new JMenuItem("IP");
                opcoes.add(rede);
                ProcessaBotao prede = new ProcessaBotao(6);
                rede.addActionListener(prede);
                JMenuItem reinit = new JMenuItem("Reinicializar");
                opcoes.add(reinit);
                ProcessaBotao preinit = new ProcessaBotao(7);
                reinit.addActionListener(preinit);

                JMenu ajuda = new JMenu("Ajuda");
                JMenuItem acerca = new JMenuItem("Acerca");
                ajuda.add(acerca);
                ProcessaBotao pacerca = new ProcessaBotao(5);
                acerca.addActionListener(pacerca);

                JMenuBar mbar = new JMenuBar();
                mbar.add(opcoes);
                mbar.add(ajuda);
                setJMenuBar(mbar);

                painel1 = new JPanel();
                painel1.setLayout(new GridLayout(2,2));
                painel1.setSize(750,640);
                painel1.setBackground(Color.black);

                c.add(painel1);
                add("North",painel1);
                setVisible(true);

                painel2 = new JPanel();
                painel2.setLayout(new FlowLayout());
                c.add(painel2);
                camara= new CheckboxGroup();
                add(camara1 = new Checkbox("V1",camara,false));
                painel2.add(camara1);
                add(camara2 = new Checkbox("V2",camara,false));
                painel2.add(camara2);
                add(camara3 = new Checkbox("V3",camara,false));
                painel2.add(camara3);
                add(camara4 = new Checkbox("V4",camara,false));
                painel2.add(camara4);
                camara1.setEnabled(true);
                camara2.setEnabled(true);
                camara3.setEnabled(true);
                camara4.setEnabled(true);
                camara1.addItemListener(new TrataCamara1());
                camara2.addItemListener(new TrataCamara2());
                camara3.addItemListener(new TrataCamara3());
                camara4.addItemListener(new TrataCamara4());

                choiceNivel = new Choice();
                choiceNivel.add("");
                choiceNivel.add("S/ Compressão");
                choiceNivel.add("H263");
                choiceNivel.add("AES");
                choiceNivel.add("H263 e AES");
                painel2.add(choiceNivel);
                choiceNivel.addItemListener(new TrataFormato());
                JButton botaoMaximo = new JButton("Fast");
                painel2.add(botaoMaximo);
                ProcessaBotao pmaximo = new ProcessaBotao(0);
                botaoMaximo.addActionListener(pmaximo);
                JButton botaoIniciar = new JButton("Run");
                painel2.add(botaoIniciar);
                ProcessaBotao piniciar = new ProcessaBotao(1);
                botaoIniciar.addActionListener(piniciar);
                JButton botaoPausa = new JButton("Slow");
                painel2.add(botaoPausa);
                ProcessaBotao ppausa = new ProcessaBotao(2);
                botaoPausa.addActionListener(ppausa);
                JButton botaoDesligar = new JButton("Stop");
                painel2.add(botaoDesligar);
                ProcessaBotao pdesligar = new ProcessaBotao(3);
                botaoDesligar.addActionListener(pdesligar);
                JButton botaoFrame = new JButton("Frame");
                painel2.add(botaoFrame);
                ProcessaBotao pframe = new ProcessaBotao(4);
                botaoFrame.addActionListener(pframe);
```

```
                Pacote = new Checkbox("Im.",false);
                Pacote.addItemListener(new TrataPacote());
                painel2.add(Pacote);
                testeB = new Checkbox("B",false);
                testeB.addItemListener(new TrataTesteBlocos());
                painel2.add(testeB);
                testeAES = new Checkbox("AES",false);
                testeAES.addItemListener(new TrataTesteAES());
                painel2.add(testeAES);
                add("South",painel2);
                setVisible(true);

                painel11 = new JPanel();
                painel11.setSize(352,256);
                painel1.add(painel11);
                setVisible(true);

                painel12 = new JPanel();
                painel12.setSize(352,256);
                painel1.add(painel12);
                setVisible(true);

                painel13 = new JPanel();
                painel13.setSize(352,256);
                painel1.add(painel13);
                setVisible(true);

                painel14 = new JPanel();
                painel14.setSize(352,256);
                painel1.add(painel14);
                setVisible(true);

                processaImagem1();
                processaImagem2();
                processaImagem3();
                processaImagem4();
                inicio=false;

        }
        public void processaImagem1(){
                if (!inicio)painel11.remove(painel21);
                painel21 = new JPanel();
                painel21.setSize(352,256);
                compressao1 = LoadPanelBitmap(painel21,"image/imagem1.bmp");
        imagem1 = new JLabel(new ImageIcon(compressao1));
                painel21.add(imagem1);
                painel11.add(painel21);
                setVisible(true);
        }
        public void processaImagem2(){
                if (!inicio)painel12.remove(painel22);
                painel22 = new JPanel();
                painel22.setSize(352,256);
                compressao2 = LoadPanelBitmap(painel22,"image/imagem2.bmp");
        imagem2 = new JLabel(new ImageIcon(compressao2));
                painel22.add(imagem2);
                painel12.add(painel22);
                setVisible(true);
        }
        public void processaImagem3(){
                if (!inicio)painel13.remove(painel23);
                painel23 = new JPanel();
                painel23.setSize(352,256);
                compressao3 = LoadPanelBitmap(painel23,"image/imagem3.bmp");
        imagem3 = new JLabel(new ImageIcon(compressao3));
                painel23.add(imagem3);
                painel13.add(painel23);
                setVisible(true);
        }
        public void processaImagem4(){
                if (!inicio)painel14.remove(painel24);
                painel24 = new JPanel();
                painel24.setSize(352,256);
                compressao4 = LoadPanelBitmap(painel24,"image/imagem4.bmp");
        imagem4 = new JLabel(new ImageIcon(compressao4));
                painel24.add(imagem4);
                painel14.add(painel24);
                setVisible(true);
        }
        public void processaIP(){
                novaJanela janelaIP = new novaJanela("IP","IP do Sistema Vigilancia:   ",this);
        }
        public void start(){
                String porto=("10008");
                try{
                        IPpar = InetAddress.getByName(endereco);
                        sd = new DatagramSocket( PortoLocal );
                }
```

```
                catch (IOException e){
                        e.printStackTrace();
                }
                novaJanela janelaInfo = new novaJanela("Info","A aplicação encontra-se pronta a
receber!");
                inputThread = new Thread (this);
                inputThread.start();
                KeyExpansion();
                System.out.printf("\nA chave de encriptacao e':\n");
                for (int n=0;n<16 ;n++ ){
                        System.out.printf(":%d",(int)Key[n]);
                }
                System.out.printf("\nA chave de expandida e':\n");
                for (int n=0;n<240 ;n++ ){
                        System.out.printf(":%d",(int)RoundKey[n]);
                }
        }
        public void run(){
                while(!fim){
                        try     {
                                recebePacote ();
                        }
                        catch (Exception e){     e.printStackTrace();}
                //      try     {
                //              Thread.sleep(1);
                //      }
                //      catch (InterruptedException e){
                //                      e.printStackTrace();

                //      }
                }
        }
        class TrataFormato implements ItemListener{
                byte compressao=(byte)2, h263=(byte)0, normal=(byte)1, h263aes=(byte)2,aes=(byte)3;
                public void itemStateChanged(ItemEvent e){
                        String recebeNivel;
                        if (e.getSource() instanceof Choice)
                        {
                                Choice c = (Choice)e.getSource();
                                recebeNivel=c.getSelectedItem();
                                if(recebeNivel=="S/ Compressão")
                                        {try    {enviaPacote(compressao,normal);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                                if(recebeNivel=="H263")
                                        {try    {enviaPacote(compressao,h263);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                                if(recebeNivel=="H263 e AES")
                                        {try    {enviaPacote(compressao,h263aes);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                                if(recebeNivel=="AES")
                                        {try    {enviaPacote(compressao,aes);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                        }
                }
        }
        class TrataCamara1 implements ItemListener{
                byte camara=(byte)3, camara1=(byte)1, camara2=(byte)2, camara3=(byte)3, camara4=(byte)4;
                public void itemStateChanged(ItemEvent e){
                        Boolean recebeCamara;
                        if (e.getSource() instanceof Checkbox)
                        {
                                Checkbox c = (Checkbox)e.getSource();
                                recebeCamara=c.getState();
                                if(recebeCamara){
                                        {try    {enviaPacote(camara,camara1);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                                }
                        }
                }
        }
        class TrataCamara2 implements ItemListener{
                byte camara=(byte)3, camara1=(byte)1, camara2=(byte)2, camara3=(byte)3, camara4=(byte)4;
                public void itemStateChanged(ItemEvent e){
                        Boolean recebeCamara;
                        if (e.getSource() instanceof Checkbox)
                        {
                                Checkbox c = (Checkbox)e.getSource();
                                recebeCamara=c.getState();
                                if(recebeCamara){
                                        {try    {enviaPacote(camara,camara2);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                                }
                        }
                }
        }
        class TrataCamara3 implements ItemListener{
                byte camara=(byte)3, camara1=(byte)1, camara2=(byte)2, camara3=(byte)3, camara4=(byte)4;
                public void itemStateChanged(ItemEvent e){
```

```
                        Boolean recebeCamara;
                        if (e.getSource() instanceof Checkbox)
                        {
                                Checkbox c = (Checkbox)e.getSource();
                                recebeCamara=c.getState();
                                if(recebeCamara){
                                        {try    {enviaPacote(camara,camara3);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                                }
                        }
                }
        }
        class TrataCamara4 implements ItemListener{
                byte camara=(byte)3, camara1=(byte)1, camara2=(byte)2, camara3=(byte)3, camara4=(byte)4;
                public void itemStateChanged(ItemEvent e){
                        Boolean recebeCamara;
                        if (e.getSource() instanceof Checkbox)
                        {
                                Checkbox c = (Checkbox)e.getSource();
                                recebeCamara=c.getState();
                                if(recebeCamara){
                                        {try    {enviaPacote(camara,camara4);
                                        }catch (Exception e1){    e1.printStackTrace();}}
                                }
                        }
                }
        }
        class TrataPacote implements ItemListener{
                byte trafego=(byte)1, envio=(byte)1;
                public void itemStateChanged(ItemEvent e){
                        Boolean recebePacote;
                        if (e.getSource() instanceof Checkbox)
                        {
                                Checkbox c = (Checkbox)e.getSource();
                                recebePacote=c.getState();
                                if(recebePacote){
                                        {try    {enviaPacote(trafego,envio);
                                                }catch (Exception e1){    e1.printStackTrace();}}
                                }
                        }
                }
        }
        class TrataTesteBlocos implements ItemListener{
                public void itemStateChanged(ItemEvent e){
                        if (e.getSource() instanceof Checkbox){
                                testeBlocos = !testeBlocos;

                        }
                }
        }
        class TrataTesteAES implements ItemListener{
                public void itemStateChanged(ItemEvent e){
                        if (e.getSource() instanceof Checkbox){
                                encriptacao = !encriptacao;

                        }
                }
        }
        class JanelaActiva extends WindowAdapter{
                public void windowDeactivated(WindowEvent e){
                //      pausa=true;
                }
                public void windowActivated(WindowEvent e){
                //      pausa=false;
                }
        }

        class ProcessaBotao implements ActionListener {
                byte trafego=(byte)1, inicializa=(byte)4, envio=(byte)1, continuo=(byte)2,
                        lento=(byte)3, suspenso=(byte)0, controlado=(byte)4;
                int botao;
                ProcessaBotao(int botaopremido) {
                        botao=botaopremido;
                        }
                public void actionPerformed(ActionEvent e) {
                        if (botao==0){try{enviaPacote(trafego,continuo);
                                                                trafegoControlado=false;
                                                }catch (Exception e1){    e1.printStackTrace();}}
                        if (botao==1){try{enviaPacote(trafego,controlado);
                                                                trafegoControlado=true;
                                                }catch (Exception e1){    e1.printStackTrace();}}
                        if (botao==2){try{enviaPacote(trafego,lento);
                                                                trafegoControlado=false;
                                                }catch (Exception e1){    e1.printStackTrace();}}
                        if (botao==3){try{enviaPacote(trafego,suspenso);
                                                                trafegoControlado=false;
                                                }catch (Exception e1){    e1.printStackTrace();}}
                        if (botao==4){try{enviaPacote(trafego,envio);
```

```
                                                                        trafegoControlado=false;
                                                }catch (Exception e1){    e1.printStackTrace();}}
                        if (botao==5){
                                novaJanela janelaAcerca = new novaJanela("Acerca","Video Vigilancia ISEP
José Fernando Carvalho Nº1040919 16-07-2007");
                        }
                        if (botao==6){
                                processaIP();
                        }
                        if (botao==7){try{enviaPacote(inicializa,suspenso);
                                                }catch (Exception e1){    e1.printStackTrace();}}
                }
        }
        private boolean SavePanelBitmap(JPanel jp, String filename)
        {
                int width = jp.getWidth();
                int height = jp.getHeight();

                BufferedImage bi = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
                Graphics2D g2d = bi.createGraphics();

                jp.paint(g2d);

                try
                {
                        File file = new File(filename);
                        ImageIO.write(bi, "bmp", file);
                }
                catch (Exception e)
                {
                        return false;
                }

                return true;
        }
        private BufferedImage LoadPanelBitmap(JPanel jp, String filename)
        {
                int width = jp.getWidth();
                int height = jp.getHeight();

                BufferedImage bi = new BufferedImage(width, height, BufferedImage.TYPE INT RGB);
                Graphics2D g2d = bi.createGraphics();

                jp.paint(g2d);

                try
                {
                        File file = new File(filename);
                        bi=ImageIO.read( file);
                }
                catch (Exception e)
                {
                        return bi;
                }

                return bi;
        }
        public void recebePacote() throws Exception {
                byte buffer in[] = new byte[2000];
                pacoteRecebido  = new DatagramPacket (buffer_in, buffer_in.length );
                sd.receive( pacoteRecebido ); // bloqueia ate receber
                byte [] msg_in = new byte [1458];
                        msg_in = (byte [])pacoteRecebido.getData();
                                // System.out.printf(".");
                actualiza(msg in);
        }
        public void actualiza (byte[] msg) throws java.io.IOException{
                byte[] msgDescomp;
                int compressao, pacote, camara, verificacao,tamanhoLSB,tamanhoMSB,tamanho;
                RandomAccessFile fo = null;
                camara=(int)msg[3];
                compressao=(int)msg[1];
                pacote=(int)msg[2];
                verificacao=(int) msg[0];
        //      tamanhoPacote=(int)msg[5]*4+3;
                if (verificacao == -120){
        //corresponde a 88h de -127 a 128
                        if (camara==1) {
                                fo = new RandomAccessFile ("image/imagem1.bmp","rw");
                                if (pacote==0)camara1.setState(true);
                        }
                        if (camara==2) {
                                fo = new RandomAccessFile ("image/imagem2.bmp","rw");
                                if (pacote==0)camara2.setState(true);
                        }
                        if (camara==3) {fo = new RandomAccessFile ("image/imagem3.bmp","rw");
                                if (pacote==0)camara3.setState(true);
                        }
```

```
                        if (camara==4) {fo = new RandomAccessFile ("image/imagem4.bmp","rw");
                                if (pacote==0)camara4.setState(true);
                        }
                        if (compressao==01){
                //sem compressao
                                if (encriptacao){
                                        i=0;
                                        for (n=6; n<1414; n++){
                                                AES_in[i]= (msg[n]);
                                                //AES_in[i]=(byte)(0x63);
                                                //      System.out.printf(":%c",AES_out[j]);
                                                i++;
                                                if (i==16){
                                                        InvCipher();
                                                        i=0;
                                                        for (j=0;j<16;j++)
                                                                msg[n-15+j]=(AES_out[j]);

                                                }
                                        }
                                }
                                j=90838-(pacote*4*352);
        //86758-> inicio 1ª linha visto na imagem
                                for (int i=6;i<358 ;i++ ){

                                        fo.seek(j);
                                        fo.writeByte(msg[i]);
                                        j++;
                                }
                                j=90486-(pacote*4*352);
        // inicio 2ª linha visto na imagem
                                for (int i=358;i<710 ;i++ ){
                                        fo.seek(j);
                                        fo.writeByte(msg[i]);
                                        j++;
                                }
                                j=90134-(pacote*4*352);
        // inicio 3ª linha visto na imagem
                                for (int i=710;i<1062 ;i++ ){
                                        fo.seek(j);
                                        fo.writeByte(msg[i]);
                                        j++;
                                }
                                j=89782-(pacote*4*352);                                                 //
inicio 4ª linha visto na imagem
                                for (int i=1062;i<1414 ;i++ ){
                                        fo.seek(j);
                                        fo.writeByte(msg[i]);
                                        j++;
                                }
                                if (camara==1)//&&(pacote==63)) para teste
                                        {processaImagem1();Pacote.setState(!Pacote.getState());}
                                if
((camara==2)&&(pacote==63)){processaImagem2();Pacote.setState(!Pacote.getState());}
                                if
((camara==3)&&(pacote==63)){processaImagem3();Pacote.setState(!Pacote.getState());}
                                if
((camara==4)&&(pacote==63)){processaImagem4();Pacote.setState(!Pacote.getState());}
                        }
                        if (compressao==00){
                //com compressao

                                for (int nGOB=0;nGOB<4;nGOB++){           //n<1 para testar
                                        if (encriptacao){
                                                i=0;
                                                tamanhoMSB=msg[4]&0x000F;
                                                tamanhoLSB=msg[5];
                                                tamanho=tamanhoMSB*256+tamanhoLSB;
                                                for (n=6; n<tamanho-6; n++){

                                                        AES in[i]= (msg[n]);
                                                        //AES in[i]=(byte)(0x63);
                                                        //      System.out.printf(":%c",AES_out[j]);
                                                        i++;
                                                        if (i==16){
                                                                InvCipher();
                                                                i=0;
                                                                for (j=0;j<16;j++)
                                                                        msg[n-15+j]=(AES out[j]);

                                                        }
                                                }
                                        }
                                        //System.out.print("\nGOB:"+(pacote+nGOB)+"\n");
                                        msgDescomp=huffman(msg,nGOB);
                                        msgDescomp = descomprime(msgDescomp,nGOB);

                                        //huffman(msg,tamanhoPacote);
```

```
                                        j=90838-((pacote+nGOB)*16*352);
                        //86758-> inicio 1ª linha visto na imagem
                                        for (int i=0;i<352 ;i++ ){

                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=90486-((pacote+nGOB)*16*352);
                        // inicio 2ª linha visto na imagem
                                        for (int i=352;i<704 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=90134-((pacote+nGOB)*16*352);
                        // inicio 3ª linha visto na imagem
                                        for (int i=704;i<1056 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=89782-((pacote+nGOB)*16*352);
                                // inicio 4ª linha visto na imagem
                                        for (int i=1056;i<1408 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=89430-((pacote+nGOB)*16*352);
                         // inicio 5ª linha visto na imagem
                                        for (int i=1408;i<1760 ;i++ ){

                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=89078-((pacote+nGOB)*16*352);
                        // inicio 6ª linha visto na imagem
                                        for (int i=1760;i<2112 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=88726-((pacote+nGOB)*16*352);
                        // inicio 7ª linha visto na imagem
                                        for (int i=2112;i<2464 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=88374-((pacote+nGOB)*16*352);
                        // inicio 8ª linha visto na imagem
                                        for (int i=2464;i<2816 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=88022-((pacote+nGOB)*16*352);
                         //inicio 9ª linha visto na imagem
                                        for (int i=2816;i<3168 ;i++ ){

                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=87670-((pacote+nGOB)*16*352);
                        // inicio 10ª linha visto na imagem
                                        for (int i=3168;i<3520 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=87318-((pacote+nGOB)*16*352);
                        // inicio 11ª linha visto na imagem
                                        for (int i=3520;i<3872 ;i++ ){
                                                fo.seek(j);
                                                if
```

```
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=86966-((pacote+nGOB)*16*352);
                                // inicio 12ª linha visto na imagem
                                        for (int i=3872;i<4224 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=86614-((pacote+nGOB)*16*352);
                                 // inicio 13ª linha visto na imagem
                                        for (int i=4224;i<4576 ;i++ ){

                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=86262-((pacote+nGOB)*16*352);
                                // inicio 14ª linha visto na imagem
                                        for (int i=4576;i<4928 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=85910-((pacote+nGOB)*16*352);
                                // inicio 15ª linha visto na imagem
                                        for (int i=4928;i<5280 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        j=85558-((pacote+nGOB)*16*352);
                                // inicio 16ª linha visto na imagem
                                        for (int i=5280;i<5632 ;i++ ){
                                                fo.seek(j);
                                                if
(((int)msgDescomp[i]!=0)||(!testeBlocos)){fo.writeByte(msgDescomp[i]);}
                                                j++;
                                        }
                                        if
((camara==1)&&(pacote==12)&&(nGOB==3)){processaImagem1();Pacote.setState(!Pacote.getState());}
                                        if
((camara==2)&&(pacote==12)&&(nGOB==3)){processaImagem2();Pacote.setState(!Pacote.getState());}
                                        if
((camara==3)&&(pacote==12)&&(nGOB==3)){processaImagem3();Pacote.setState(!Pacote.getState());}
                                        if
((camara==4)&&(pacote==12)&&(nGOB==3)){processaImagem4();Pacote.setState(!Pacote.getState());}
                                }
                        }
                        fo.close();
                        if (trafegoControlado){try {enviaPacote((byte)1,(byte)4);
                                                                }catch (Exception e1){
        e1.printStackTrace();}}

                }
        }
        public void leBit (byte inBuffer []){
        //public byte[] huffman (byte[] msg){
                byte valorBit=0, Mascara_1=1;
                if (shift > 0){
                        valorBit = (byte)(inBuffer[in]>>shift) ;
                        valorBit = (byte)(valorBit & Mascara_1);
                        shift=shift-1;
                }
                else {
                        valorBit = (byte)(inBuffer[in] & Mascara 1);                //bit menos
significativo (ultimo bit do byte)
                        shift=7;in=in+1;
        //proximo bit esta no proximo byte
                }
                valorTestar = (short)(valorTestar<<1) ;
                valorTestar = (short)(valorBit | valorTestar);
        }
        public byte[]  huffman (byte[] inBuffer,int nGOB){
                byte outBuffer[] = new byte[2000];
                boolean encontradoDC=false,encontradoEOB=false;
                int Mascara_1=1,i,n,nEOB,nZeros,nAux,fimSeg,linha=0;
                short tabelaDC []
={0x0002,0x0003,0x0004,0x0005,0x0006,0x000E,0x001E,0x003E,0x007E,0x00FE,0x01FE};
                int tabelaAC [] ={0x0000,
                                0x0001,0x0004,0x000A,0x0018,0x0019,0x0038,0x0078,0x01F4,0x03F6,0x0FF4,
                                0x000B,0x0039,0x00F6,0x01F5,0X07F6,0x0FF5,0xFF88,0xFF89,0xFF8A,0xFF8B,
                                0x001A,0x00F7,0x03F7,0x0FF6,0X7FC2,0xFF8C,0xFF8D,0xFF8E,0xFF8F,0xFF90,
```

```
                                0x001B,0x00F8,0x03F8,0x0FF7,0XFF91,0xFF92,0xFF93,0xFF94,0xFF95,0xFF96,
                                0x003A,0x01F6,0xFF97,0xFF98,0XFF99,0xFF9A,0xFF9B,0xFF9C,0xFF9D,0xFF9E,
                                0x003B,0x03F9,0xFF9F,0xFFA0,0XFFA1,0xFFA2,0xFFA3,0xFFA4,0xFFA5,0xFFA6,
                                0x0079,0x07F7,0xFFA7,0xFFA8,0XFFA9,0xFFAA,0xFFAB,0xFFAC,0xFFAD,0xFFAE,
                                0x007A,0x07F8,0xFFAF,0xFFB0,0XFFB1,0xFFB2,0xFFB3,0xFFB4,0xFFB5,0xFFB6,
                                0x00F9,0xFFB7,0xFFB8,0xFFB9,0XFFBA,0xFFBB,0xFFBC,0xFFBD,0xFFBE,0xFFBF,
                                0x01F7,0xFFC0,0xFFC1,0xFFC2,0XFFC3,0xFFC4,0xFFC5,0xFFC6,0xFFC7,0xFFC8,
                                0x01F8,0xFFC9,0xFFCA,0xFFCB,0XFFCC,0xFFCD,0xFFCE,0xFFCF,0xFFD0,0xFFD1,
                                0x01F9,0xFFD2,0xFFD3,0xFFD4,0XFFD5,0xFFD6,0xFFD7,0xFFD8,0xFFD9,0xFFDA,
                                0x01FA,0xFFDB,0xFFDC,0xFFDD,0XFFDE,0xFFDF,0xFFE0,0xFFE1,0xFFE2,0xFFE3,
                                0x07F9,0xFFE4,0xFFE5,0xFFE6,0XFFE7,0xFFE8,0xFFE9,0xFFEA,0xFFEB,0xFFEC,
                                0x3FE0,0xFFED,0xFFEE,0xFFEF,0XFFF0,0xFFF1,0xFFF2,0xFFF3,0xFFF4,0xFFF5,
                                0x7FC3,0xFFF6,0xFFF7,0XFFF8,0xFFF9,0xFFFA,0xFFFB,0xFFFC,0xFFFD,0xFFFE,
                                0x03FA};
                byte tabelaValores [][]={{0},{-1,1},{-3,-2,2,3},{-7,-6,-5,-4,4,5,6,7},{-15,-14,-13,-12,-
11,-10,-9,-8,
                        8,9,10,11,12,13,14,15},{-31,-30,-29,-28,-27,-26,-25,-24,-23,-22,-21,-20,-19,-18,-
17,-16,16,17,18,
                        19,20,21,22,23,24,25,26,27,28,29,30,31},{-63,-62,-61,-60,-59,-58,-57,-56,-55,-
54,-53,-52,-51,-50,
                        -49,-48,-47,-46,-45,-44,-43,-42,-41,-40,-39,-38,-37,-36,-35,-34,-33,-
32,32,33,34,35,36,37,38,39,
                        40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63},{-127,-
126,-125,-124,-123,
                        -122,-121,-120,-119,-118,-117,-116,-115,-114,-113,-112,-111,-110,-109,-108,-107,-
106,-105,-104,-103,
                        -102,-101,-100,-99,-98,-97,-96,-95,-94,-93,-92,-91,-90,-89,-88,-87,-86,-85,-84,-
83,-82,-81,-80,-79,
                        -78,-77,-76,-75,-74,-73,-72,-71,-70,-69,-68,-67,-66,-65,-
64,64,65,66,67,68,69,70,71,72,73,74,75,76,

        77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,1
07,

        108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127}};

                out=6;shift=7;nEOB=0;valorTestar=0;encontradoDC=false;encontradoEOB=false;
                if (nGOB==0) in=6;
                for (n=0;n<6;n++) outBuffer[n]=inBuffer[n];
                        while (nEOB<88){
                        while ((!encontradoDC)&&(nEOB<88)){
                //inicio de teste de um valor (DC ou Fim bloco)
                                leBit(inBuffer);
                                if ((valorTestar!=2)&&(valorTestar!=3)){
                                        for (n=0;n<11;n++){
                                                if (tabelaDC[n]==valorTestar)
                                                        linha=n;
                                        }
                                }
                                if ((linha==0)&&(valorTestar==0)){
                                        leBit(inBuffer);
                //se é 0 o 2° bit tb e' 0 é o EOB se não é uma linha
                                        if (valorTestar==0){
                                                outBuffer[out]=-128;
                                                nEOB=nEOB+1;
                                        //      System.out.print(" EOBDC:"+nEOB);
                                                out=out+1;
                                        }
                                        else {
                                                leBit(inBuffer);
                                                if (valorTestar==2){
                                                        encontradoDC=true;
                                                        linha=0;
                                                        outBuffer[out]=0;
                                                        valorTestar=0;

                                                        out=out+1;
                                                }else linha=1;
                                        }
                                }
                                if (linha!=0){
                                        encontradoDC=true;
                        //"linha" tem  a linha da tabela e n° bits
                                        valorTestar=0;
                        //inicializa valorTestar para ler proximo valor (posicao)
                                        for (n=0;n<linha;n++) leBit(inBuffer);
        //vai entao ler a posicao que fica em valorTestar
                                        outBuffer[out]=tabelaValores[linha][valorTestar];
                                //      System.out.print(" DC:"+outBuffer[out]);
                                        valorTestar=0;
                                        out=out+1;
                                }
                                linha=0;
                        }
                        encontradoDC=false;
                        while ((!encontradoEOB)&&(nEOB<88)){
                //inicio de teste de um valor (AC ou Fim bloco)
                                leBit(inBuffer);
```

```
                                nZeros=0;
                                if (valorTestar!=1){
                                        for (n=0;n<162;n++){
                                                nAux=n;
                                                if ((short)tabelaAC[n]==valorTestar){
                                                        while (nAux>10){
                                                                nAux=nAux-10;
                                                                nZeros=nZeros+1;
                                                        }
                                                        linha=nAux;
                                                }
                                        }
                                }
                                if ((linha==0)&&(valorTestar==0)){
                        //se é 0 pode ser EOB se não é uma linha
                                        leBit(inBuffer);
        //se é 0 o 2° bit tb e' 0 é o EOB se não é uma linha
                                        if (valorTestar==0){
                                                outBuffer[out]=-128;
                                                nEOB=nEOB+1;
                                                out=out+1;
                                                encontradoEOB=true;
                                        //      System.out.print(" EOBAC");
                                        }else {
                                                linha=1;
                                                valorTestar=0;
                                        }
                                }
                                if (linha!=0){
                                        for (n=0;n<nZeros;n++){
                                                outBuffer[out]=0;
                                        //      System.out.print(" ACZ:"+outBuffer[out]);
                                                out=out+1;
                                        }
                                        if (valorTestar==0x03FA){
                                //mensagem de limite de zeros(15) o 16 zero
                                                valorTestar=0;
                                                //ja foi contabilizado nao e' preciso fazer nada
                                        //      System.out.print(" 15Zersos");
                                        }else{
                                                valorTestar=0;
                                                for (n=0;n<linha;n++) leBit(inBuffer);
                        //vai entao ler a posicao que fica em valorTestar
                                                outBuffer[out]=tabelaValores[linha][valorTestar];
                                        //      System.out.print(" AC:"+outBuffer[out]);
                                                valorTestar=0;
                                                out=out+1;
                                        }
                                }
                                linha=0;
                        }
                        encontradoEOB=false;
                }
                in=in+1;
        //      if (nGOB==3) System.out.print(" Tamanho:"+in);
/*      System.out.print("\nPacote (Blocos:"+nEOB+")\n");
                for (i=6;i<out;i++ )
                        System.out.print("|"+outBuffer[i]);*/
                return outBuffer;
        }

        public byte[] descomprime (byte[] msg, int nGOB){
        //Matriz Q ordem zigzag
         int MatQ[] = {
                                16,11,
                                12,14,12,10,16,
                                14,13,14,18,17,16,19,24,40,
                                26,24,22,22,24,49,35,37,29,40,58,51,61,
                                60,57,51,56,55,64,72,92,78,64,68,87,69,55,56,
                                80,109,81,87,95,98,103,104,103,62,77,
                                113,121,112,100,120,92,101,
                                103,99};
        //Matriz Q aumentada 25% ordem zigzag
         int MatQ1[] = {
                                20,14,
                                15,18,15,13,20,
                                18,16,18,23,21,20,24,30,50,
                                33,30,28,28,30,61,44,46,36,50,73,64,76,
                                75,71,64,70,69,80,90,115,98,80,85,109,86,69,70,
                                100,136,101,109,119,123,129,130,129,78,96,
                                141,151,140,125,150,115,126,
                                129,124};
        //Matriz Q aumentada 50% ordem zigzag
        int MatQ2[] = {
                                24,16,
                                18,21,18,15,24,
                                21,19,21,27,25,24,28,36,60,
                                39,36,33,33,36,73,52,55,43,60,87,76,91,
```

```
                            90,85,76,84,82,96,108,138,117,96,102,130,103,82,84,
                            120,163,121,130,142,147,154,156,154,93,115,
                            169,181,168,150,180,138,151,
                            154,148};
        //Matriz Q aumentada da compressao cromatica
        int MatQ3[] = {
                            17,18,
                            18,24,21,24,47,
                            26,26,47,99,66,56,66,99,99,
                            99,99,99,99,99,99,99,99,
                            99,99,99,99,99,99,99,99,
                            99,99,99,99,99,99,99,99,
                            99,99,99,99,99,99,99,99,
                            99,99,99,99,99,99,99,99,
                            99,99,99,99,99,99,99,99};
                //Matriz posiçoes inversas zigzag
                int MatZigzag[] = {
                            0,1,5,6,14,15,27,28,
                            2,4,7,13,16,26,29,42,
                            3,8,12,17,25,30,41,43,
                            9,11,18,24,31,40,44,53,
                            10,19,23,32,39,45,52,54,
                            20,22,33,38,46,51,55,60,
                            21,34,37,47,50,56,59,61,
                            35,36,48,49,57,58,62,63
                                    };
                int MatZigzagInversa[] = {
                            0,1,8,16,9,2,3,10,
                            17,24,32,25,18,11,4,5,
                            12,19,26,33,40,48,41,34,
                            27,20,13,6,7,14,21,28,
                            35,42,49,56,57,50,43,36,
                            29,22,15,23,30,37,44,51,
                            58,59,52,45,38,31,39,46,
                            53,60,61,54,47,55,62,63
                                    };

                //Matriz A
                double MatA[] = {
                        0.7071,0.9808,0.9239,0.8315,0.7071,0.5556,0.3827,0.1951,
                        0.7071,0.8315,0.3827,-0.1951,-0.7071,-0.9808,-0.9239,-0.5556,
                        0.7071,0.5556,-0.3827,-0.9808,-0.7071,0.1951,0.9239,0.8315,
                        0.7071,0.1951,-0.9239,-0.5556,0.7071,0.8315,-0.3827,-0.9808,
                        0.7071,-0.9151,-0.9239,0.5556,0.7071,-0.8315,-0.3827,0.9808,
                        0.7071,-0.5556,-0.3827,0.9808,-0.7071,-0.1951,0.9239,-0.8315,
                        0.7071,-0.8315,0.3827,0.1951,-0.7071,0.9808,-0.9239,0.5556,
                        0.7071,-0.9808,0.9239,-0.8315,0.7071,-0.5556,0.3827,-0.1951};

                //Matriz temporaria
                double MatCompleta[] = new double[64];
                double MatNormal[] = new double[64];
                double MatTemp1[] = new double[64];
                double MatTemp2[] = new double[64];
                double Temp;
                byte MatDescompBlocos[] = new byte[5632];
                byte MatDescomprimida[] = new byte[5632];
                int i,j,n,linha,coluna,in,out, pMASC=0;
                if (nGOB ==0)pMASC=0x80;
                if (nGOB ==1)pMASC=0x40;
                if (nGOB ==2)pMASC=0x20;
                if (nGOB ==3)pMASC=0x10;
                in=6;
                            //in indexa entrada os 6 1º bytes sao de controlo
                for (int bloco=0;bloco<88;bloco++){
                        if (in < 1410){
                        //menor que 1458-48 para comp 25% só assim tera outro bloco
                            i=0;
                            while((int)msg[in]!= -128){
        //vai reproduzir a matriz completa e "desquantificada"
                                if ((msg[4]& (byte)pMASC)==0)
                                        MatCompleta[i] = ((int)msg[in])*MatQ[i];
                                else
                                        MatCompleta[i] = ((int)msg[in])*MatQ3[i];
                                i++;in++;
                            }
                            in++;
                            for (n=i;n<64;n++){
                                    MatCompleta[n]=0;
        //matriz completa na ordem zizag;
                            }
                            for (n=0;n<64;n++){
                                    MatNormal[n]=MatCompleta[MatZigzag[n]];
//matriz na ordem normal;
                            }
                            for (linha=0;linha<8 ;linha++ ){
                                    for (coluna=0;coluna<8;coluna++){
                                            for (j=0;j<8;j++){
                        //j indexa matrizes entrada
```

```
                                                        Temp = MatA[j+linha*8]*MatNormal[j*8+coluna];
                                                        MatTemp1[coluna+linha*8] =
MatTemp1[coluna+linha*8] + Temp;
                                                }
                                        }
                                }
                                for (linha=0;linha<8 ;linha++ ){
                                        for (coluna=0;coluna<8;coluna++){
                                                for (j=0;j<8;j++){
                                                        //todos os elementos da linha
                                                        Temp = MatTemp1[j+linha*8]*MatA[j+coluna*8];
                  //linha por linha por ser a transposta de A
                                                        MatTemp2[coluna+linha*8] =
MatTemp2[coluna+linha*8] + Temp;//MatNormal recebe a IDT
                                                }
                                        }
                                }
                                for (n=0;n<64;n++){
                                        MatTemp2[n]=MatTemp2[n]/4+128;
        //muliplica por 1/4 e soma de 128
                                        if (MatTemp2[n]>255){
                                                MatTemp2[n]=255;
                                        }
                                        if (MatTemp2[n]==128){
                                                MatTemp2[n]=0;
                                        }
                                        if (MatTemp2[n]<0){
                                                MatTemp2[n]=0;
                                        }
                                }
                                for (n=0;n<64;n++){
                                        MatDescompBlocos[n+bloco*64]=(byte) MatTemp2[n];    //Matriz
descomprimida na ordem dos blocos
                                        MatTemp2[n]=0;
                                //inicializações
                                        MatTemp1[n]=0;
                                }
                        }

                }
                // a Matriz deve ser reescrita na ordem normal
                out=0;
                for (linha=0;linha<16;linha++){
        //para todas as linhas
                        for (coluna=0;coluna<44;coluna++){                                      //para todas
as colunas referentes a cada bloco
                                for (i=0;i<8;i++){
                //para todos os píxeis da linha de cada bloco
                                        MatDescomprimida[out]=MatDescompBlocos[i+coluna*128+linha*8];
                                        out++;
                                }
                        }
                }
                return MatDescomprimida;
        }

        // This function produces Nb(Nr+1) round keys. The round keys are used in each round to decrypt
the states.
    public void KeyExpansion(){
                int i,j;
                byte temp[]=new byte [4],k;

                        // The first round key is the key itself.
                for(i=0;i<Nk;i++)
                {
                        RoundKey[i*4]=Key[i*4];
                        RoundKey[i*4+1]=Key[i*4+1];
                        RoundKey[i*4+2]=Key[i*4+2];
                        RoundKey[i*4+3]=Key[i*4+3];
                }

                // All other round keys are found from the previous round keys.
                while (i < (Nb * (Nr+1)))
                {
                        for(j=0;j<4;j++)
                        {
                                temp[j]= (byte)RoundKey[(i-1) * 4 + j];
                        }
                        if (i % Nk == 0)
                        {
                                // This function rotates the 4 bytes in a word to the left once.
                                // [a0,a1,a2,a3] becomes [a1,a2,a3,a0]

                                // Function RotWord()
                                {
                                        k = temp[0];
                                        temp[0] = temp[1];
                                        temp[1] = temp[2];
```

```
                                        temp[2] = temp[3];
                                        temp[3] = k;
                                }

                                // SubWord() is a function that takes a four-byte input word and
                                // applies the S-box to each of the four bytes to produce an output
word.

                                // Function Subword()
                                {
                                        if (temp[0]<0)temp[0]=(byte) sBox[temp[0]+256];else
temp[0]=(byte) sBox[temp[0]];
                                        if (temp[1]<0)temp[1]=(byte) sBox[temp[1]+256];else
temp[1]=(byte) sBox[temp[1]];
                                        if (temp[2]<0)temp[2]=(byte) sBox[temp[2]+256];else
temp[2]=(byte) sBox[temp[2]];
                                        if (temp[3]<0)temp[3]=(byte) sBox[temp[3]+256];else
temp[3]=(byte) sBox[temp[3]];
                                }

                                temp[0] =  (byte) (temp[0] ^ Rcon[i/Nk]);
                        }
                        else if (Nk > 6 && i % Nk == 4)
                        {
                                // Function Subword()
                                {
                                        if (temp[0]<0)temp[0]=(byte) sBox[temp[0]+256];else
temp[0]=(byte) sBox[temp[0]];
                                        if (temp[1]<0)temp[1]=(byte) sBox[temp[1]+256];else
temp[1]=(byte) sBox[temp[1]];
                                        if (temp[2]<0)temp[2]=(byte) sBox[temp[2]+256];else
temp[2]=(byte) sBox[temp[2]];
                                        if (temp[3]<0)temp[3]=(byte) sBox[temp[3]+256];else
temp[3]=(byte) sBox[temp[3]];
                                }
                        }
                        RoundKey[i*4+0] =(byte) (RoundKey[(i-Nk)*4+0] ^ temp[0]);
                        RoundKey[i*4+1] =(byte) (RoundKey[(i-Nk)*4+1] ^ temp[1]);
                        RoundKey[i*4+2] =(byte) (RoundKey[(i-Nk)*4+2] ^ temp[2]);
                        RoundKey[i*4+3] =(byte) (RoundKey[(i-Nk)*4+3] ^ temp[3]);
                        i++;
                }
        }

        // The SubBytes Function Substitutes the values in the
        // state matrix with values in an S-box.
        void InvSubBytes()
        {
                int i,j;
                for(i=0;i<4;i++)
                {
                        for(j=0;j<4;j++)
                        {
                                if ((state[i][j])<0) state[i][j] =
(byte)(sBoxInv[(int)(state[i][j]+256)]);
                                else state[i][j] = (byte)(sBoxInv[(int)(state[i][j])]);
                        }
                }
        }

        // This function adds the round key to state.
        // The round key is added to the state by an XOR function.
        public void AddRoundKey(int round){
                int i,j;
                for(i=0;i<4;i++)
                {
                        for(j=0;j<4;j++)
                        {
                                state[i][j] ^= RoundKey[(int)(round * Nb * 4 + i * Nb + j)];
                        }
                }
        }

        // The ShiftRows() function shifts the rows in the state to the left.
        // Each row is shifted with different offset.
        // Offset = Row number. So the first row is not shifted.
        public void InvShiftRows(){
                byte temp;

                // Rotate first row 1 columns to right
                        temp=state[1][3];
                state[1][3]=state[1][2];
                state[1][2]=state[1][1];
                state[1][1]=state[1][0];
                state[1][0]= temp;

                // Rotate second row 2 columns to right
                        temp=state[2][0];
```

```
                state[2][0]=state[2][2];
                state[2][2]=temp;

                temp=state[2][1];
                state[2][1]=state[2][3];
                state[2][3]=temp;

                // Rotate third row 3 columns to right
                temp=state[3][0];
                state[3][0]=state[3][1];
                state[3][1]=state[3][2];
                state[3][2]=state[3][3];
                state[3][3]=temp;
        }

        // xtime is a macro that finds the product of {02} and the argument to xtime modulo {1b}
        public int xtime(int x){
                int y;
                y=((x<<1) ^ (((x>>7) & 1) * 0x1b));
                return y;
        }

        // Multiplty is a macro used to multiply numbers in the field GF(2^8)
        public int Multiply(int x, int y){
                int z;
                z=(((y & 1) * x) ^ ((y>>1 & 1) * xtime(x)) ^ ((y>>2 & 1) * xtime(xtime(x))) ^ ((y>>3 & 1)
* xtime(xtime(xtime(x)))) ^ ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))));
                return z;
        }

        // MixColumns function mixes the columns of the state matrix.
        // The method used to multiply may be difficult to understand for beginners.
        // Please use the references to gain more information.
        public void InvMixColumns(){

                int i;
                int a,b,c,d;
                for(i=0;i<4;i++)
                {

                        a = state[0][i];
                        b = state[1][i];
                        c = state[2][i];
                        d = state[3][i];


                        state[0][i] = (byte)(Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^
Multiply(d, 0x09));
                        state[1][i] = (byte)(Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^
Multiply(d, 0x0d));
                        state[2][i] = (byte)(Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^
Multiply(d, 0x0b));
                        state[3][i] = (byte)(Multiply(a, 0x0b) ^ Multiply(b, 0x0d) ^ Multiply(c, 0x09) ^
Multiply(d, 0x0e));
                }
        }

        // InvCipher is the main function that decrypts the CipherText.
        public void InvCipher(){
                int i,j,round=0;

                //Copy the input CipherText to state array.
                for(i=0;i<4;i++)
                {
                        for(j=0;j<4;j++)
                        {
                                state[i][j] = (AES_in[i*4 + j]);
                        }
                }

                // Add the First round key to the state before starting the rounds.
                AddRoundKey(Nr);



                                // There will be Nr rounds.
                // The first Nr-1 rounds are identical.
                // These Nr-1 rounds are executed in the loop below.
                for(round=Nr-1;round>0;round--)
                {
                        InvShiftRows();
                        InvSubBytes();
                        AddRoundKey(round);
                        InvMixColumns();
                }

                        // The last round is given below.
                // The MixColumns function is not here in the last round.
```

```
                InvShiftRows();
                InvSubBytes();
                AddRoundKey(0);

                // The decryption process is over.
                // Copy the state array to output array.
                for(i=0;i<4;i++)
                {
                        for(j=0;j<4;j++)
                        {
                                AES_out[i*4+j]= (byte) state[i][j];
                        }
                }
        }

        public void enviaPacote(byte comando,byte valor) throws Exception {
                // Preparacao para enviar um pacote
                byte buffer_out[] = new byte[3];
                byte [] msg_out = new byte [3];
                msg out [0]=(byte)136;                          //valor de confirmação de mensagem (88
em hexadecimal)
                msg_out [1]=comando;
                msg_out [2]=valor;
                for( int i = 0; i < 3 ; i++ )
                        buffer_out[i] = (byte) msg_out[i];
                pacoteEnviado = new DatagramPacket( buffer_out, buffer_out.length,
                IPpar, PortoPar);
                sd.send ( pacoteEnviado );
        }

}


class novaJanela extends JFrame{
        TextField IP;
        JanelaVideoVigi Referencia=null;
        public novaJanela(String titulo,String texto){
                setTitle(titulo);
                setSize(450, 100);
                setResizable(false);
                JPanel painel1 = new JPanel();
                Container c1 = getContentPane();
                c1.add(painel1);
                JLabel campoTexto = new JLabel();
                campoTexto.setText(texto);
                painel1.add(campoTexto);
                setVisible(true);
        }
        public novaJanela(String titulo,String texto, JanelaVideoVigi Referencia){
                this.Referencia=Referencia;
                setTitle(titulo);
                setSize(450, 100);
                setResizable(false);
                JPanel painel1 = new JPanel();
                Container c1 = getContentPane();
                c1.add(painel1);
                JLabel campoTexto = new JLabel();
                campoTexto.setText(texto);
                painel1.add(campoTexto);
                IP = new TextField(15);
                IP.setEditable(true);
                painel1.add(IP);
                reporIP();
                JButton botaoOK = new JButton("OK");
                painel1.add(botaoOK);
                Processa pOK = new Processa();
                botaoOK.addActionListener(pOK);
                setVisible(true);
        }
        public void reporIP(){
                IP.setText(Referencia.endereco);
        }
        public void devolveIP(){
                Referencia.endereco=IP.getText();
        }
        class Processa implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        devolveIP();
                                try     {Referencia.IPpar = InetAddress.getByName(Referencia.endereco);
                                }catch (Exception e2){    e2.printStackTrace();}
                }
        }
}
```

Anexo E. Configuração pDSLAM8 e router ADSL

- Descriptor ADSL:

Name	Latency	Rate Mode	SNR (tenth dB)						Tx Rate (Kbps)			
			DownStream			UpStream			DownStream		UpStream	
			Target	Min	Max	Target	Min	Max	Min	Max	Min	Max
3	fast	startUp	60	0	310	60	0	310	64000	2048000	32000	256000

- Interfaces ADSL:

	Profile				
Int	Line	Alarm	Type	Oper Status	Admin Status
adsl.01	3	1	fast	handshake	up
adsl.02	3	1	fast	handshake	up

- Interfaces ATM:

		Maximum bits		Available BW		Oversubs						Num	Num	Status
Int	If Index	VPI	VCI	Rx	Tx	CBR	rVBR		nVBR	UBR		VPCs	VCCs	Oper
							SCR	P-S		MDCR	P-M			
atm.001	adsl.01	8	16	256000	2048000	100	100	200	100	100	200	0	0	LLDown
atm.002	adsl.02	8	16	256000	2048000	100	100	200	100	100	200	0	0	LLDown

- Descriptor tráfego ATM:

IND	CLASSE	PCR	SCR	MBS	CDVT us	Frame	MCR UBR
1	CBR.1	174	-	-	10	Y	-

- Cruzamento ATM:

Porto de Origem								Porto de Destino						
					Traf	UPC mode						Traf	UPC mode	
Ind	Int	Phy	Vp	Vc	Tx	Rx	Tx	Int	Phy	Vp	Vc	Tx	Rx	Tx
1	atm.001	adsl.001	0	32	1	pass	pass	atm.002	adsl.002	0	32	1	pass	pass

Os *routers* utilizados são router ADSL anexo A da *conexant*, foram ambos configurados da seguinte forma:

- VPI/VCI: `0/32`
- ATM Service Category: `UBR`
- Encapsulation: `1483 Bridged IP LLC`

Histórico

- 21 de Janeiro de 2008, Versão 1.0, mailto:jbm@isep.ipp.pt
- 21 de Abril de 2008, Versão 1.1, mailto:jbm@isep.ipp.pt
- 4 deNovembro de 2008, Versão 2.0, mailto:jbm@isep.ipp.pt

$Id:MEEC-TEDI.dot v1.0b Date:16-11-2007$