unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

**EDUARDO SEIJI AGUILERA NITO**

# Sistemas Embarcados com FPGA

Guaratinguetá – SP
2017

**Eduardo Seiji Aguilera Nito**

# Sistemas Embarcados com FPGA

Trabalho de Graduação apresentado ao Conselho de Curso de Graduação em Engenharia Elétrica da Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, como parte dos requisitos para obtenção do diploma de Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Leonardo Mesquita

Guaratinguetá - SP
2017

Nito, Eduardo Seiji Aguilera
N731s Sistemas embarcados com FPGA / Eduardo Seiji Aguilera Nito – Guaratinguetá, 2017.
107 f : il.
Bibliografia: f. 75-76

Trabalho de Graduação em Engenharia Elétrica – Universidade Estadual Paulista, Faculdade de Engenharia de Guaratinguetá, 2017.
Orientador: Prof. Dr. Leonardo Mesquita

1. Eletrônica. 2. Hardware. 3. Software. I. Título.

CDU 621.38

Luciana Máximo
Bibliotecária/CRB-8 3595

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
CAMPUS DE GUARATINGUETÁ

**EDUARDO SEIJI AGUILERA NITO**

ESTE TRABALHO DE GRADUAÇÃO FOI JULGADO ADEQUADO COMO PARTE DO REQUISITO PARA A OBTENÇÃO DO DIPLOMA DE
**GRADUADO EM ENGENHARIA ELÉTRICA**

APROVADO EM SUA FORMA FINAL PELO CONSELHO DE CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Prof. Dr. Leonardo Mesquita
Coordenador

**BANCA EXAMINADORA:**

Prof. Dr. Leonardo Mesquita
Orientador/UNESP-FEG

Prof. André Bianchi Laraia
Engenheiro Eletricista

Prof. José Marcelo de Assis Wendling Junior
Engenheiro Eletricista

Outubro de 2017

## AGRADECIMENTOS

Agradeço a toda minha família, fonte inesgotável de inspiração, amor, energia e motivação, pois sempre incentivaram meus estudos e apoiaram minhas escolhas. Agradeço principalmente aos meus pais, Sérgio e Sandra, que investiram tempo e dinheiro em minha educação, confiando em meu potencial para que eu possa atingir meus objetivos e, consequentemente, minha felicidade.

Agradeço à minha namorada, Tamai, por me apoiar em uma época tão fundamental e complexa da minha vida, sendo sempre compreensiva, atenciosa e carinhosa.

Aos companheiros de república: Eduardo, Gabriel, Gabriel (Fred), Matheus, Lucas, Pedro e Bruno pelos bons momentos vividos durante nossas vidas acadêmicas,

Agradeço às funcionárias domésticas, Cida e Ana, com as quais tive a oportunidade de conviver, por serem competentes e preocupadas com meu bem estar,

Agradeço a todos os docentes e funcionários da FEG, por prezarem pelo ensino e formação dos estudantes de Engenharia, em especial, ao meu orientador, Leonardo Mesquita, por me guiar e instruir para que eu pudesse realizar este trabalho.

## RESUMO

O presente trabalho tem como objetivo mostrar todas as etapas desenvolvidas no projeto de um sistema embarcado utilizando a plataforma FPGA, realizando um breve estudo comparativo entre as principais plataformas encontradas no mercado, citando as vantagens e desvantagens de se utilizar cada uma delas, em comparação com os FPGAs. O projeto utilizou um Kit de Desenvolvimento DE2-115, fabricado pela Terasic; VHDL como linguagem de descrição de hardware e a ferramenta de design, Quartus II 13.0SP1, desenvolvido pela Altera. O projeto pode ser dividido em duas partes: Hardware e Software. O hardware é composto por todos os componentes, periféricos, registradores e elementos físicos que constituem o sistema. Foi utilizada a ferramenta Qsys, intrínseca ao Quartus II, para instanciar esses componentes e interconectá-los. Para o Software, utilizou-se o ambiente de desenvolvimento integrado, Eclipse, tendo C como sua linguagem de programação. A aplicação final é o processamento do algoritmo criptográfico DES, utilizando o processador NIOS II, um teclado PS/2 para a entrada de dados e um monitor VGA como dispositivo de saída.



**PALAVRAS-CHAVE: S**istemas embarcados. FPGA. DE2-115. DES. NIOS II.

## ABSTRACT

This papers objective is to show all the steps of an Embedded System project, using platform FPGA, and also make a comparative analysis between the most common platforms available in the market, explaining the advantages and disadvantages of using each of them, compared to FPGA. The project used a DE2-115 Development Kit, by Terasic; VHDL as hardware description language, and the Quartus II 13.0 SP1 EDA tool, by Altera. The project can be divided in two parts: hardware and software; the hardware has all the components, peripherals, registers and physical elements that compose the system. The Qsys tool was used to instantiate, and interconnect these components. For the software, the Integrated Development Environment based on Eclipse was used, and C was the Programming language. The final application was to compute the DES cryptographic algorithm, using the NIOS II Processor, a PS/2 keyboard as input device, and a VGA monitor as output device.



**KEYWORDS:** Embedded systems. FPGA. DE2-115. DES. NIOS II.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS E QUADROS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| ABS | *Antilock Braking System* |
| ASCII | *American Standard Code for Information Interchange* |
| ASIC | *Application Specific Integrated Circuit* |
| BSP | *Board Support Package* |
| CAD | *Computer Aided Design* |
| CI | Circuito Integrado |
| CMOS | *Complementary Metal-Oxide-Semiconductor* |
| CPLD | *Complex Programmable Logic Device* |
| CPU | *Central Processing Unit* |
| CSV | *Comma Separated Value* |
| DES | *Digital Encryption Standard* |
| DRAM | *Dynamic Random Access Memory* |
| DSP | *Digital Signal Processor* |
| DVD | *Digital Video Disc* |
| ECB | *Electronic Code Book* |
| EDA | *Electronic Design Automation* |
| ELF | *Executable And Linking Format* |
| FIFO | *First in First out* |
| FIPS | *Federal Information Processing Standards* |
| FPGA | *Field Programmable Gate Array* |
| GPS | *Global Positioning System* |
| HAL | *Hardware Abstraction Layer* |
| HDTV | *High Definition Television* |
| JTAG | *Joint Test Action Group* |
| UART | *Universal Asynchronous Receiver Transmitter* |
| MB | *Megabytes* |
| MHz | *Megahertz* |
| NBS | *National Bureau of Standards* |
| NIST | *National Institute of Standards and Technology* |
| P-Box | *Permutation Box* |
| PC | *Personal Computer* |
| PCI | Placa de Circuito Impresso |
| PDA | *Personal Data Assistant* |
| PI | Permutação Inicial |
| PS/2 | *Personal System/2* |
| PUB | *Publication* |
| QIP | *Quartus II IP file* |
| RGB | *Red, Green & Blue* |
| S-Box | *Substitution Box* |
| SDRAM | *Synchronous Dynamic Random Access Memory* |
| SOF | *SRAM Object File* |
| SOPC | *System On Programmable Chip* |
| SRAM | *Static Random Access Memory* |
| USB | *Universal Serial Bus* |

| | |
|---|---|
| VCR | *Videocassete Recorder* |
| VGA | *Video Graphics Array* |
| VHDL | *Very High Speed Hardware Description Language* |
| XOR | *Exclusive OR* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Atualmente, grande parte dos dispositivos eletrônicos presentes no mercado são sistemas embarcados que utilizam *hardwares* e *softwares* com lógica digital. O advento da eletrônica digital fez com que as principais fabricantes de semicondutores conseguissem componentes com dimensões cada vez mais reduzidas e com baixo consumo de energia, como é o caso da família de portas lógicas CMOS (*Complementary Metal-Oxide-Semiconductor*), amplamente utilizada em circuitos digitais. Durante este período de progresso tecnológico, também foram desenvolvidos diversos métodos matemáticos que facilitaram o ensino e a aplicação da Eletrônica Digital, por exemplo, a álgebra booleana, utilizada em circuitos lógicos. Estes circuitos foram ficando cada vez menores e mais complexos, sendo encapsulados em Circuitos Integrados (CIs), que juntamente com outros componentes, eram soldados em uma Placa de Circuito Impresso (PCI), constituindo assim, um produto final. Esta tecnologia foi e continua sendo utilizada na produção de eletroeletrônicos, pois com a redução de custos e dimensões, começou-se uma produção em grande escala, tornando muitos produtos que antes eram utilizados apenas para fins científicos ou militares, acessíveis para o consumidor comum.

O conceito de *hardware* digital se iniciou com circuitos lógicos. Estes circuitos constituem-se de portas lógicas, que interligadas de maneira apropriada formam uma função lógica,como soma, multiplicação, divisão, multiplexação e até mesmo um processador, em sistemas mais complexos. Um *hardware* digital, do mais simples ao mais complexo, é composto de circuitos lógicos e a maneira como eles são organizados é o que define sua complexidade e funcionalidades. As empresas pioneiras no desenvolvimento de dispositivos eletrônicos começaram a desenvolver sistemas cada vez mais complexos, que realizavam uma variedade de funções. Ao final da década 70, já era possível encontrar no mercado diversos tipos de aparelhos e eletroeletrônicos.

> Para a maioria dos produtos digitais encontrados no mercado, é necessário projetar e construir alguns circuitos lógicos desde o início do projeto. Para implementar estes circuitos, três tipos de *chips* podem ser usados: dispositivos de lógica fixa, dispositivos de lógica programável e ASICs. (BROWN; VRANESIC, 2009, p.4, tradução nossa)

O primeiro grupo de *chips* realizava poucas funções e continham poucos circuitos lógicos por unidade. Foram muito populares durante o início da década de 80. À medida que a tecnologia de circuitos impressos melhorou, tornou-se economicamente inviável utilizar *chips* tão simples para ocupar uma placa de circuito impresso.

Os dispositivos de lógica programável, por sua vez, implementavam uma quantidade muito maior de circuitos lógicos e ainda permitiam uma programação por parte do usuário. Isto permitiu uma grande flexibilidade na criação de *hardwares*, além disso, reduzia custos de manufatura e montagem. Os tipos mais comuns de dispositivos de lógica programável são: *Field Programmable Gate Arrays* (FPGA) e *Complex Programmable Logic Devices* (CPLD).

Por último, estão os *Application Specific Integrated Chip* (ASIC), que são *chips* customizados para realizar uma aplicação específica. Geralmente são apropriados para produtos que serão produzidos em larga escala, pois esta tecnologia envolve maiores custos de manufatura. Cabe ao projetista decidir qual destas plataformas irá utilizar, sendo essa escolha dependente da complexidade da aplicação, custos e desempenho.

O presente trabalho irá utilizar a plataforma FPGA, e ao mesmo tempo, fazer uma comparação entre as principais plataformas presentes no setor. Este trabalho também tem por objetivo implementar um sistema embarcado, utilizando ferramentas de design e o processador embarcado da fabricante Altera, NIOS II.

## 1.1 MOTIVAÇÃO

Sistemas embarcados estão presentes em todos os setores do mercado e possuem inúmeras aplicações, além disso, é um tema recente e relevante, que traz um diferencial para a formação acadêmica do estudante. O projeto também irá agregar conhecimentos em *hardware* e *software* em plataformas FPGA, que são de grande valia para um profissional que trabalhe com Eletrônica. Todos os tópicos estudados e as etapas de desenvolvimento do projeto contribuem para que o estudante se aproxime de um nível de conhecimento que o permita desenvolver um produto final.

## 1.2 OBJETIVOS

- Apresentar todas as etapas necessárias no desenvolvimento de um projeto de um Sistema Embarcado
- Utilizar a plataforma FPGA e realizar uma comparação entre principais plataformas presentes no mercado
- Utilizar o *kit* de Desenvolvimento DE2-115 da Altera para implementação do *Hardware* e *Software* e apresentar uma aplicação piloto utilizando o processador embarcado NIOS II como elemento central do sistema
- Implementar o *Hardware* do sistema utilizando a ferramenta Qsys, pertencente à ferramenta de *design* Quartus II. Utilizar como linguagem de descrição de *hardware,* o VHDL
- Desenvolver o *Software* utilizando o ambiente de desenvolvimento Eclipse, tendo C como a linguagem de programação escolhida.
- Apresentar como aplicação piloto o algoritmo criptográfico DES (*Digital Encryption Standard*)

# 2 SISTEMAS EMBARCADOS

## 2.1 INTRODUÇÃO

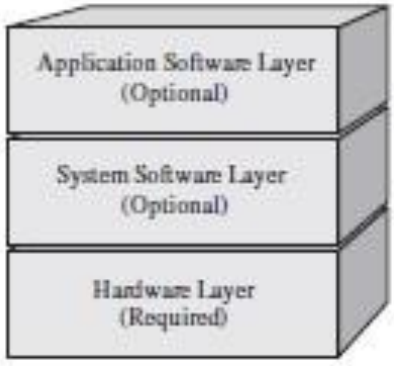
De acordo com Navabi (2007, p. 23, tradução nossa), um sistema embarcado pode ser definido como um sistema computacional com pelo menos um processador, que executa uma ou mais funções específicas para um *hardware*. Apesar de terem definições parecidas, Sistemas Embarcados diferem de computadores pessoais (PC); o primeiro grupo é destinado a executar uma aplicação específica, enquanto o segundo executa diversos tipos de funções, sendo denominados computadores de uso geral. Segundo Noergaard (2005), a definição de Sistemas Embarcados é fluida e está em constante mudança, por isso, existem muitas discussões entre profissionais da área a respeito dos critérios utilizados na classificação de um sistema digital, pois há uma linha tênue separando esses dois grupos. "[...] A fronteira exata entre um computador de uso geral e um sistema embarcado não é definida por "preto e branco", e sim, por um espectro" (SCHMIDT; SASS, 2010, p. 6, tradução nossa).

Ultimamente, muitas aplicações utilizam sistemas embarcados em plataforma FPGA para solucionar problemas da Engenharia, pois permitem a programação do *hardware* após o mesmo ter deixado a fábrica, oferecendo grande capacidade de processamento, rapidez e operação de forma paralela, sem a necessidade de um Sistema Operacional. É importante observar que aplicações que envolvem sistemas embarcados podem utilizar FPGAs, *Digital Signal Processors* (DSPs), microcontroladores ou até mesmo microprocessadores, pois todos utilizam um processador para implementar funções para o *hardware*, cabendo ao projetista analisar qual é a melhor plataforma a ser utilizada, sendo esta escolha dependente do nível de complexidade e desempenho desejados. O projeto de um sistema embarcado geralmente é sustentado por uma arquitetura, ou um modelo estrutural. A Figura 1 mostra o modelo simplificado de um sistema embarcado. A camada inferior é a de *hardware*, indispensável para todos os sistemas embarcados, pois contém os componentes físicos que estão na placa, a camada do meio é referente ao *software*, que consiste de um código escrito em linguagens de programação de baixo nível, definindo como o sistema se comporta com relação aos periféricos e a camada superior é a de aplicações, que pode ser caracterizada como a interface para o usuário final.

As duas camadas de cima são consideradas opcionais em um projeto de sistema embarcado, entretanto, são interessantes para o sistema, devido à flexibilidade e possibilidade de

incorporar novas funcionalidades. Em grandes projetos, dificilmente não há um *software*, pois a necessidade de atualizações, sistemas mais interativos com o consumidor e acesso a internet, faz com que sua presença seja praticamente obrigatória. Geralmente, há uma equipe dedicada à criação do *software* e outra destinada a projetar o *hardware* do sistema, pois a obtenção de um produto de qualidade é extremamente complexa.

Figura 1 – Modelo de um Sistema Embarcado




Fonte: Noergaard (2005)

O Quadro 1 a seguir apresenta a quantidade de aplicações de sistemas embarcados em diversos setores comerciais.

Quadro 1 – Sistemas Embarcados em diversos setores do Mercado

(continua)

| Mercado | Sistema Embarcado |
|---|---|
| Automotivo | Sistema de ignição |
| | Controle de motor |
| | Sistema de breque (exemplo: *Antilock braking system* (ABS)) |
| Eletroeletrônicos | Televisões digitais e analógicas |
| | Aparelhos de mídia (DVDs, VCR, etc) |
| | Assistentes pessoais de dados (PDAs) |
| | Eletrodomésticos de cozinha (geladeiras,torradeiras, micro-ondas) |
| | Automóveis |

Quadro 1 – Sistemas Embarcados em diversos setores do Mercado

(conclusão)

| | |
|---|---|
| Eletroeletrônicos | Brinquedos/Jogos |
| | Telefones/Telefones Celular/*Pagers* |
| | Câmeras |
| | GPS |
| Controle Industrial | Robótica e sistemas de controle |
| Médico | Bombas de infusão |
| | Máquinas de diálise |
| | Mecanismos protéticos |
| | Monitores cardíacos |
| Redes | Roteadores |
| | *Hubs* |
| | *Gateways* |
| Automação de escritório | Fax |
| | Fotocopiadora |
| | Impressoras |
| | Monitores |
| | *Scanners* |

Fonte: Noergaard (2005), tradução nossa

Sistemas embarcados estão presentes em todos os setores do mercado. É incontestável a importância destes dispositivos no cenário tecnológico atual. O tema discutido neste trabalho é a implementação de um sistema embarcado na plataforma FPGA, que possua *hardware* e *software* destinados a uma aplicação específica, além disso, será feito uma comparação entre as principais plataformas utilizadas no projeto de um sistema embarcado.

## 2.2 PLATAFORMAS: MICROCONTROLADOR, MICROPROCESSADOR, ASIC, DSP e FPGA

De acordo com o artigo Microcontrolador (2017), um microcontrolador é um pequeno computador em um circuito integrado, que possui uma CPU, além de componentes de memória e periféricos de entrada e saída, que podem ser programados através de funções pré-estabelecidas. Microcontroladores são bastante utilizados em sistemas embarcados, pois possuem componentes de *hardware* e *software*, destinados a executarem uma ou mais tarefas específicas, através do uso de um processador. Dentre várias aplicações, as principais estão em automóveis, *smartphones* e periféricos para computadores.

O conjunto de funções a serem executadas pelo microcontrolador é proveniente de um programa, geralmente escrito em linguagem C ou *Assembly*, no qual o processador irá executar cada uma das funções, de maneira sequencial. Para certas aplicações, é conveniente o processamento sequencial, principalmente se as funções já estão bem definidas e não necessitem de modificações futuras. Em aplicações que envolvem um grau de complexidade maior, os FPGAs são a plataforma mais indicada, pois conseguem realizar múltiplas tarefas em tempo real, caracterizando assim um *hardware* com operação em paralelo.

O desempenho de sistemas embarcados utilizando microcontroladores, de uma maneira geral, é satisfatório para uma grande variedade de aplicações, principalmente as que não exigem um grau de complexidade e processamento elevados. Existem muitos fabricantes no mercado que desenvolvem microcontroladores para entusiastas, estudantes de eletrônica e “hobbystas”, pois possuem um custo acessível e permitem que o projetista desenvolva projetos eletrônicos de diversos tipos.

Um microprocessador, por sua vez, é um processador que incorpora funções de uma CPU, em um único circuito integrado. Diferentemente dos microcontroladores, que incorporam memória e periféricos de entrada e saída em um único *chip*, os microprocessadores possuem apenas funções a serem realizadas por uma unidade de processamento, sendo que suas instruções são armazenadas em sua memória interna, geralmente composta por registradores. Os microprocessadores são muito utilizados para computadores de uso geral e são projetados com o intuito de processar instruções definidas por um sistema operacional. Possuem uma capacidade

de processamento maior que microcontroladores, e por realizarem funções mais genéricas, apresentam uma complexidade maior.

Apesar de estarem presentes majoritariamente em computadores de uso geral, os microprocessadores também podem ser utilizados em aplicações de sistemas embarcados, como: telefones celulares, aparelhos de DVD e Televisores de alta definição (HDTV). É conveniente observar que esta plataforma é mais recomendada para aplicações que exigem alto processamento e utilizem um sistema operacional.

Os ASICs são circuitos integrados que são customizados para realizarem uma função específica. Isto significa que são circuitos otimizados para executarem a tarefa com eficiência, consumirem menos energia e utilizarem menos recursos. Os ASICs são adequados para CIs que serão produzidos em larga escala, pois seu custo de manufatura é elevado, requerem cuidados criteriosos em sua fase de projeto e, além disso, não são reprogramáveis, uma vez que saem da fábrica.

As linguagens utilizadas nos ASICs geralmente são de descrição de *hardware*, VHDL ou Verilog, assim como nos FPGAs. A principal diferença é que ASICs não são reprogramáveis, já nos FPGAs, é possível reprogramá-los em campo, ou seja, após o mesmo ter deixado a fábrica, entretanto, a interconexão entre os blocos lógicos não é feita de forma otimizada.

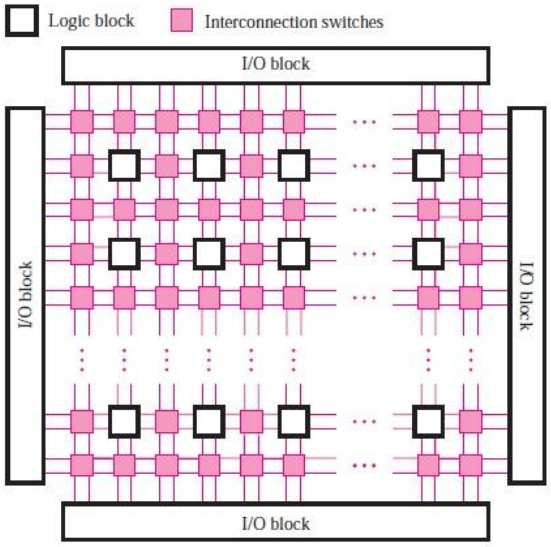
Um DSP é um tipo especial de microprocessador, que realiza funções específicas de processamento de sinais. A maioria dos microprocessadores de uso geral conseguem executar as funções de um DSP, porém, não de maneira otimizada. Os processadores de sinais digitais apresentam melhor eficiência e são adequados em aplicações que exigem processamento de algoritmos matemáticos complexos.

Sistemas embarcados utilizam DSPs para executar determinadas funções matemáticas e processamento de sinais analógicos, multiplicação de números complexos, conversão de dados, filtragem de sinais e até mesmo transformada de Fourier, entretanto, não costumam ser o elemento central de um sistema embarcado. Esta classe de circuitos evoluiu bastante ao longo do tempo e seu uso é indispensável em sistemas digitais mais complexos.

Os FPGAs, finalmente, são uma classe de dispositivos com lógica programável de campo. Esta plataforma vem sendo cada vez mais utilizada, por utilizar blocos lógicos com alta complexidade. Os FPGAs, juntamente com os CPLDs, dominam o mercado de sistemas embarcados, pois apresentam ferramentas e arquiteturas que permitem grande flexibilidade no

*design* de sistemas digitais. De certa maneira, essas duas tecnologias são muito parecidas, mas possuem uma diferença crucial, de acordo com Wilson (2016): enquanto CPLDs utilizam blocos lógicos fixos, os FPGAs utilizam blocos lógicos configuráveis, isto significa que a última opção é bem mais flexível que a primeira. Outra diferença está no número de portas lógicas que são encapsuladas em um bloco lógico: FPGAs conseguem comportar dezenas de milhares de circuitos lógicos, enquanto os CPLDs alcançam a ordem de 10000. A Figura 2 mostra a estrutura usual de um dispositivo FPGA.

Figura 2 – Estrutura e interconexões de um FPGA




Fonte: Brown; Vranesic (2009)

Da figura, podem-se identificar três tipos de recursos: blocos lógicos, blocos de entrada e saída, e as chaves de interconexão entre os blocos. Esta grande malha bidimensional opera de forma paralela e é totalmente reprogramável. A maneira como os blocos se interconectam pode ser feita tanto de maneira vertical quanto horizontal, dependendo geralmente do método de síntese utilizado pela ferramenta computacional de design (CAD).

## 2.3 ANÁLISE COMPARATIVA QUALITATIVA

Após uma introdução das principais plataformas utilizadas em sistemas embarcados, é possível fazer uma comparação no aspecto qualitativo entre elas. Todas possuem vantagens e desvantagens, sendo a escolha do projetista determinada pela aplicação e o tipo de função que o produto irá desempenhar.

As PCIs podem possuir mais de uma plataforma em seu *hardware*, por exemplo, alguns FPGA já possuem microprocessadores integrados em um único *chip*, portanto, há uma junção entre duas plataformas. A comparação será feita analisando o desempenho de cada uma de forma individualizada.

A principal diferença entre microcontroladores e FPGAs é com relação à unidade central de processamento (CPU). No primeiro grupo, utiliza-se um processador, que irá executar um programa, escrito em C ou *Assembly*, que contém todas as funções a serem executadas, de forma sequencial. Nos FPGAs, não é necessário o uso do processador, pois as funções a serem executadas são interconexões feitas de forma eletrônica entre os blocos lógicos, que tipicamente são sintetizados em linguagem de descrição de *hardware*. A escolha entre os dois é feita de acordo com o tipo de aplicação. Para aplicações que envolvem várias tarefas acontecendo de forma simultânea, os FPGAs são mais indicados, já em aplicações mais simples, onde o processamento sequencial é suficiente, os microcontroladores são mais adequados, até mesmo por terem custos e dimensões menores.

Os ASICs, como vistos anteriormente, possuem altos custos de manufatura e tempos longos de projeto e testes, já nos FPGAs, o custo é menor além de apresentarem grande flexibilidade e reprogramabilidade, o que torna as fases de protótipo mais eficientes. Uma desvantagem dos FPGAs em relação aos ASICs é a otimização do circuito, pois o método de síntese dos circuitos em FPGAs é feita de maneira genérica e a combinação entre os blocos lógicos é interconectada de maneira não otimizada, e nos ASICs, o CI é projetado para operar de maneira otimizada para executar uma determinada função. O consumo de energia consequentemente também será menor nos ASICs, visto que utilizam o mínimo de recursos e espaço para executarem a tarefa.

Os microprocessadores, via de regra, são utilizados em sua maioria para computadores de uso geral, enquanto os FPGAs são utilizados para Sistemas Embarcados. Existem produtos que necessitam de alta capacidade de processamento, portanto, é válido dizer que algumas aplicações

embarcadas utilizam microprocessadores como elemento central, todavia, é difícil fazer uma comparação entre duas plataformas que são utilizadas para aplicações distintas. No quesito poder de processamento, os microprocessadores se destacam perante os FPGAs, pois atingem frequências de *clock* na ordem de GHz, enquanto os FPGAs a ordem de centenas de MHz, mas isso não significa que os FPGAs são mais lentos, afinal, possuem operação em paralelo, podendo executar várias tarefas simultaneamente em um único pulso de *clock*. Por essa razão, os FPGAs são mais rápidos para certas aplicações, utilizando muito menos poder de processamento. Com relação ao consumo de energia, FPGAs tendem a consumir menos quando comparados com microprocessadores, pois operam a uma frequência de *clock* menor, aumentando a vida útil do componente.

Por último, resta comparar os DSPs com os FPGA. Na realidade, comparar essas duas tecnologias é de certa forma, infrutífero, pois se encontram em categorias diferentes. DSPs são microprocessadores especializados em processar sinais digitais que fazem parte de um sistema embarcado, portanto são apenas uma parte do todo. Muitas placas de circuito impresso e *kits* de desenvolvimento apresentam ambas as plataformas, pois um complementa o outro. Os DSPs são otimizados para processar sinais digitais e realizar algoritmos matemáticos, entretanto, possuem uma quantidade menor de funcionalidades em relação aos FPGAs.

# 3 MONTAGEM DO HARDWARE

A montagem do *hardware* é feita no Qsys, uma ferramenta intrínseca ao Quartus II. Através dela, é possível incorporar componentes no sistema, podendo eles ser de diversos tipos: memórias, periféricos de entrada e saída, controladores para entrada serial e paralela, pontes de comunicação, dentre outros. O Qsys possui componentes que se interconectam com o processador NIOS II, através de um sistema de barramentos de dados e controle chamado *Avalon Interconnect*. Este protocolo permite a criação rápida e eficaz de componentes para o *hardware* e possibilita o mapeamento deles na memória do sistema. Após adicionar todos os componentes que serão utilizados, são feitas as conexões entre eles, atribuições de endereços de memória e definição dos sinais que serão exportados para as conexões físicas da placa. O sistema pode ser gerado na própria interface gráfica. Durante a geração, são criados todos os arquivos, módulos e arquivos necessários para o *hardware*. Os procedimentos serão explicados nos próximos tópicos.

## 3.1 ALTERA UNIVERSITY PROGRAM

O *hardware* utilizado neste trabalho utiliza alguns dos controladores fornecidos pelo Programa de Universidades da Altera, que pode ser encontrado no site da Intel[1]. O *Altera University Program* traz um conjunto de exemplos de *hardwares* e *softwares* criados com o Quartus II, além de diversos controladores e componentes personalizados para o Qsys, que facilitam o trabalho do projetista na hora de estabelecer a comunicação entre os periféricos da placa. É conveniente lembrar que todos esses recursos são Propriedades Intelectuais (PI) pertencentes a Altera e que a utilização para fins comerciais deve ser autorizada pela mesma.

## 3.2 CRIAÇÃO DO PROJETO

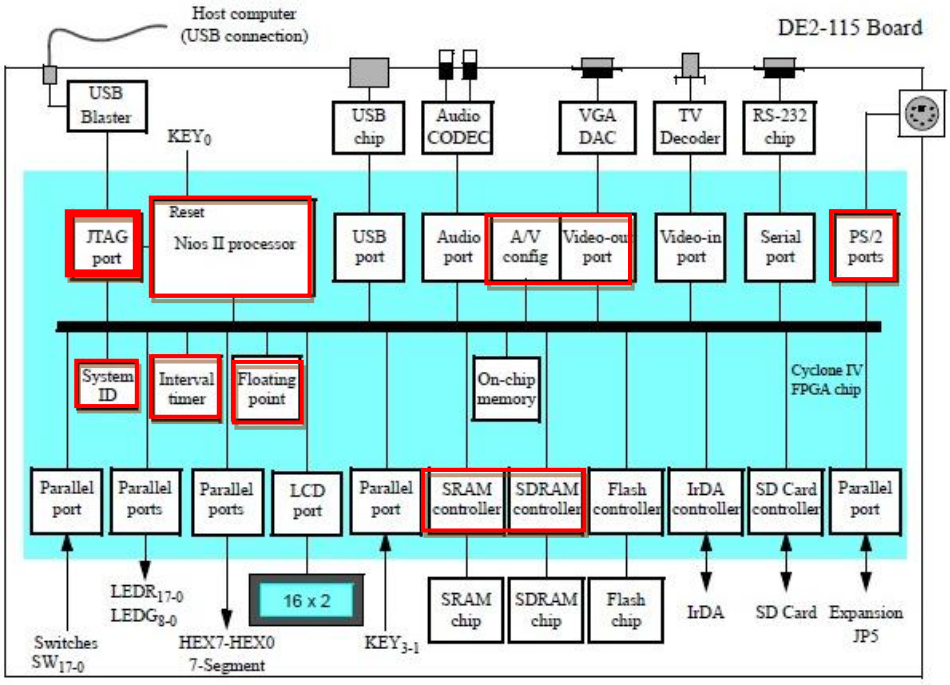
O primeiro passo é criar um projeto utilizando a ferramenta Quartus II 13.0 SP1, selecionando o *kit* de desenvolvimento DE2-115 e o FPGA Cyclone IV EP4CE115F29C7 Este procedimento é explicado em detalhes por Carbonara (2016). Após isso, abrir o Qsys, para iniciar a montagem do *Hardware*.

[1] https://www.altera.com/support/training/university/materials-computer-systems.html

## 3.3 SISTEMA DO QSYS

Primeiramente, é necessário saber os componentes que serão necessários para o projeto. A aplicação escolhida foi o algoritmo criptográfico DES, tendo como dispositivo de entrada um teclado PS/2 e como saída um monitor VGA. Para tal aplicação, serão utilizadas memórias, controladores de vídeo e teclado, processador Nios II, dentre outros. A Figura 3 apresenta um exemplo de *hardware* com diversos periféricos e controladores. Os componentes que serão utilizados no projeto estão destacados.

Figura 3 – Controladores e Componentes de *hardware*



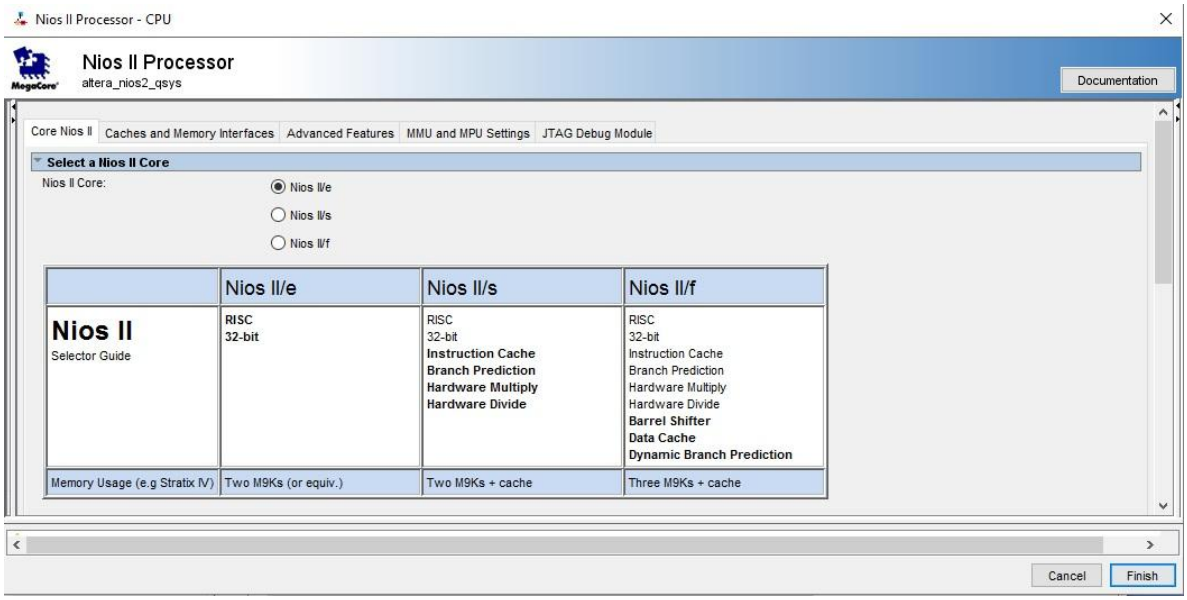

Fonte: Altera (2013)

### 3.3.1 Componentes

Os próximos itens irão explicar e descrever as configurações de cada componente que será adicionado no Qsys. Para adicionar um componente, basta procurar pelo nome na ferramenta de busca, ou encontrá-lo na biblioteca de componentes (*Component library*).

O primeiro componente a ser adicionado é o processador Nios II, que é o elemento central de processamento do sistema. Para adicioná-lo, deve-se expandir a opção *Embedded Processors* na biblioteca de componentes e selecione *Nios II Processor*. A janela da Figura 4 será aberta. Na aba *Core Nios II*, selecionar a versão *Economy (Nios II/e)*. Depois, clicar em *Finish* e renomear o componente para **CPU**. É possível notar que o Qsys irá mostrar erros, isso ocorre devido ao fato de o processador não estar associado a nenhum dispositivo de memória. Posteriormente este erro desaparecerá.

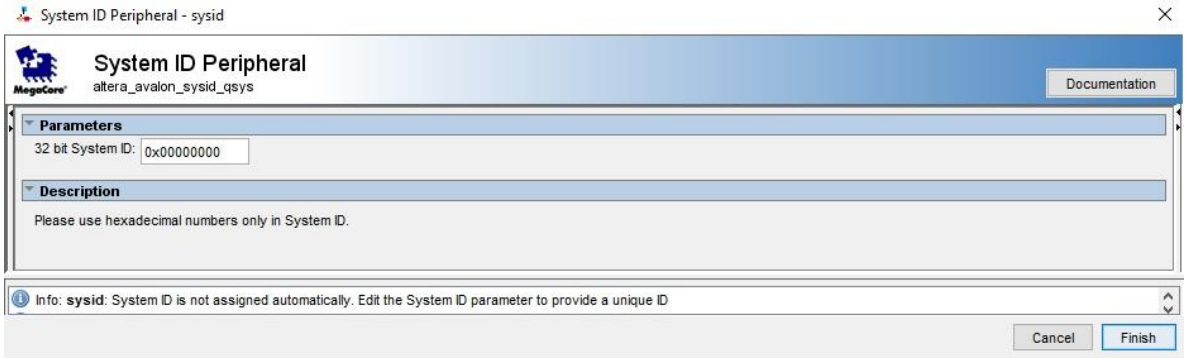
Figura 4 – Componente Nios II *Processor*




Fonte: Produção do próprio autor

A seguir, é necessário um componente que identifique o sistema. Procurar por *System ID Peripheral* na barra de busca e adicionar o componente. O valor no endereço é 0x00000000, como na Figura 5. O *System* ID é uma identificação do *hardware*, que é utilizada para fins de segurança. Renomear para **sysid**.

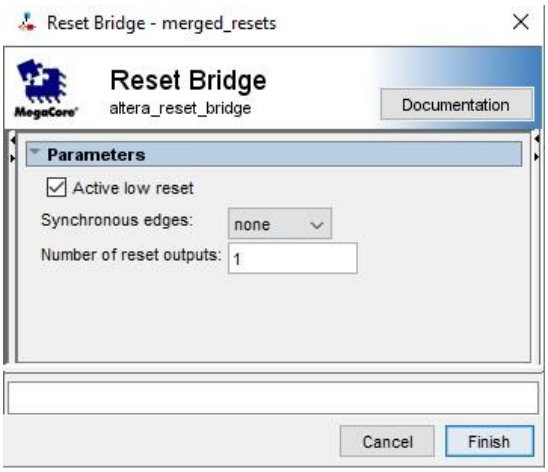
Figura 5 – *System* ID




Fonte: Produção do próprio autor

Após adicionar uma identificação, devem-se adicionar os sinais de *clock* e *reset* do sistema. Primeiramente, procurar em *Bridges>Reset Bridge* na biblioteca e adicionar o componente. A Figura 6 mostra suas configurações. Renomear para **merged_resets**.

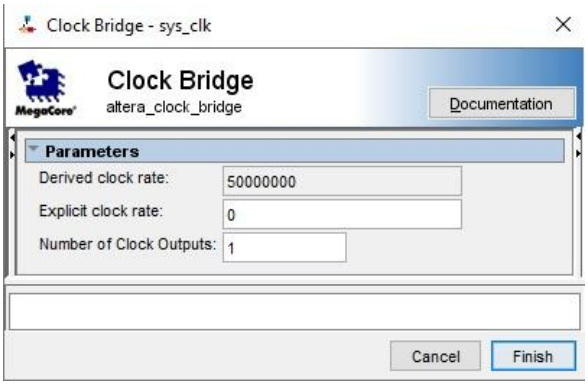
Figura 6 – Ponte para o sinal de *reset*




Fonte: Produção do próprio autor

A seguir, em *Bridges>Clock Bridges,* adicionar duas pontes de *clock.* Este componente serve como uma ponte para criar um sinal de *clock* com uma frequência desejada. No projeto, serão utilizados dois sinais de *clock*, um de 50 MHz e outro de 25 MHz para o monitor. A Figura 7 contém os parâmetros do componente, modifique apenas o valor da frequência de *clock* para os valores acima. Renomeá-los para **sys_clk** e **vga_clk**.

Figura 7 – Ponte para o sinal de *clock*

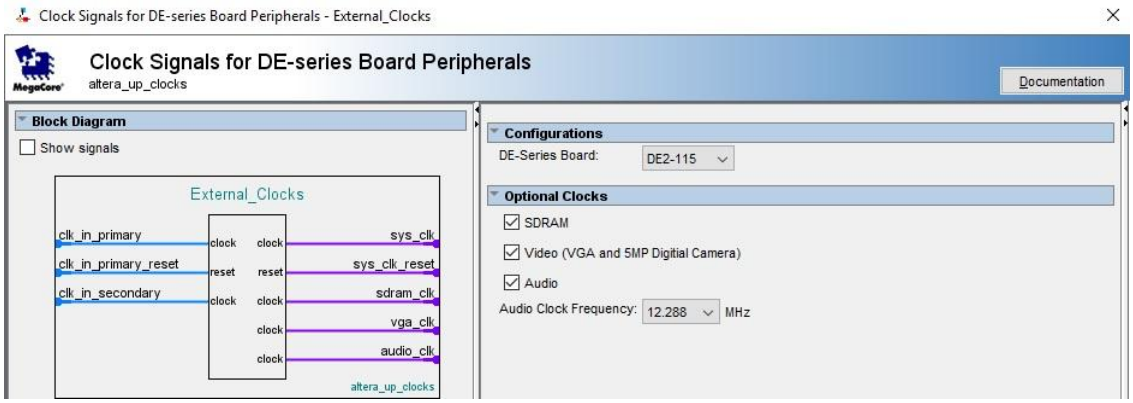



Fonte: Produção do próprio autor

As pontes para *clock*, adicionadas anteriormente necessitam das suas respectivas fontes de *clock*, que no caso, são os osciladores a cristal contidos na placa. Serão necessárias 2 fontes, uma de 50 MHz e outra de 27 MHz. Para adicioná-las, deve-se procurar em *Clock and Reset>Clock Source.* Renomear os componentes para **clk** e **clk_27**, respectivamente.

O próximo componente é um controlador que gera *clocks* externos a partir de diferentes entradas de *clock* na entrada. Encontra-se na aba *University Program> Clock Signals for DE-series Board Peripherals.* A Figura 8 contém as configurações. Renomear para **External_*Clock*s.**

Figura 8 – Componente que gera os sinais de *clock* do sistema



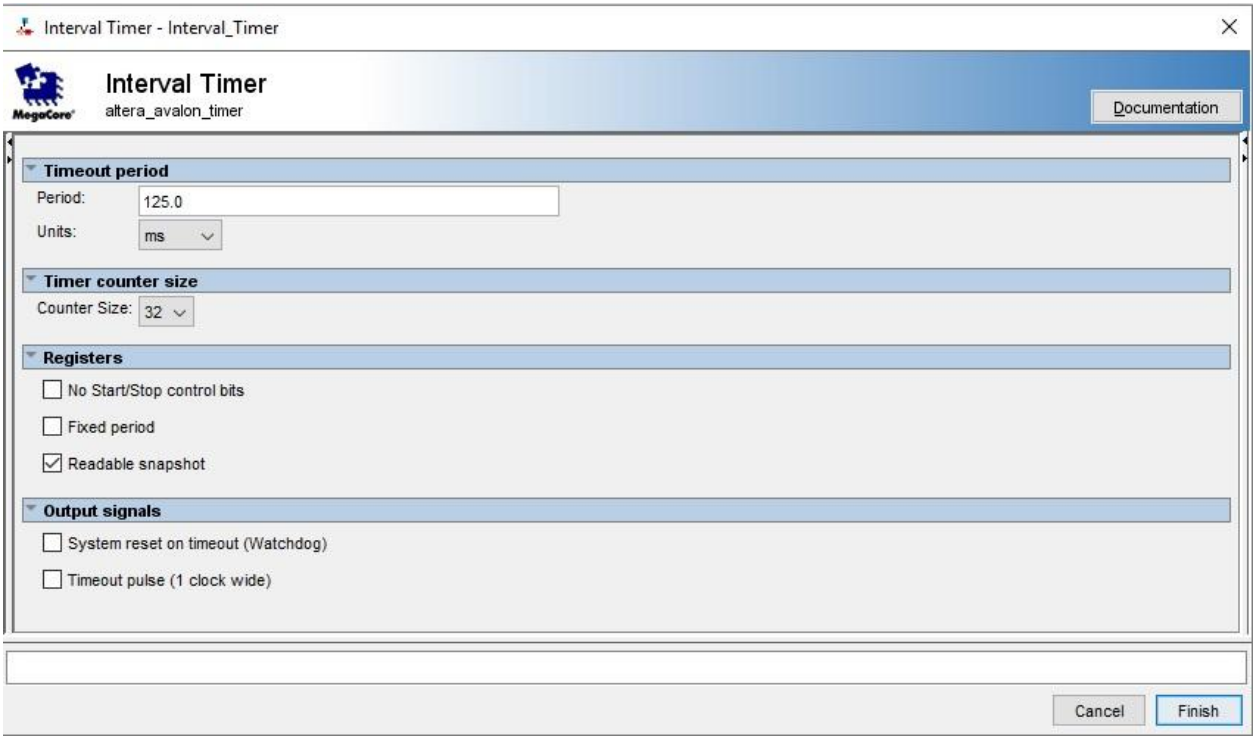

Fonte: Produção do próprio autor

É conveniente observar que o *clock* de áudio não será utilizado no projeto, mas pode ser útil para futuras implementações de *hardware*, por exemplo, um aviso sonoro ao terminar a

criptografia. É possível observar na Figura 8 o diagrama de blocos com todos os sinais de *clock* gerados. O sinal sdram_clk é responsável por sincronizar os ciclos da memória SDRAM.

O sistema irá incluir um *timer*, que é utilizado para medir intervalos de tempo. O tempo de contagem é calculado a partir do sinal de *clock* de 50 MHz, gerado pela DE2-115. Procurar em *Peripherals>Microcontroller Peripherals>Interval Timer.* A Figura 9 contém as configurações. Renomear para **Interval_Timer**.

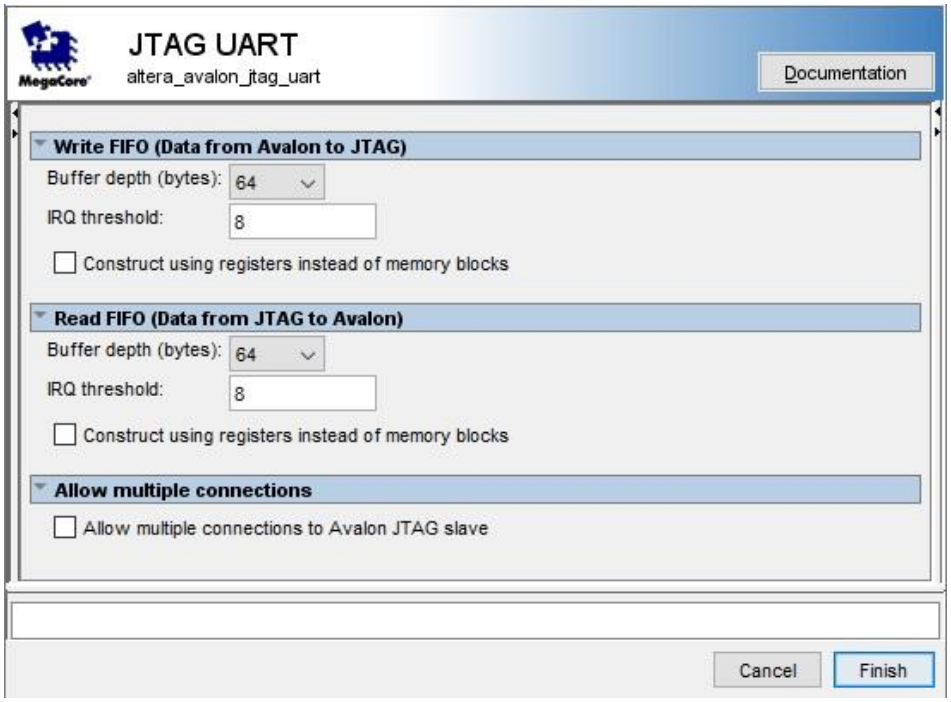
Figura 9 – Parâmetros utilizados para o *Timer* do sistema




Fonte: Produção do próprio autor

O Módulo que será adicionado a seguir é o JTAG UART e sua função é estabelecer a comunicação com um computador host (PC), através do cabo USB *Blaster*. Encontra-se em *Interface Protocols>Serial>JTAG UART*. As configurações já estão prontas e são encontradas na Figura 10. Renomeie para **JTAG_UART**.

Figura 10 – Módulo para o JTAG UART



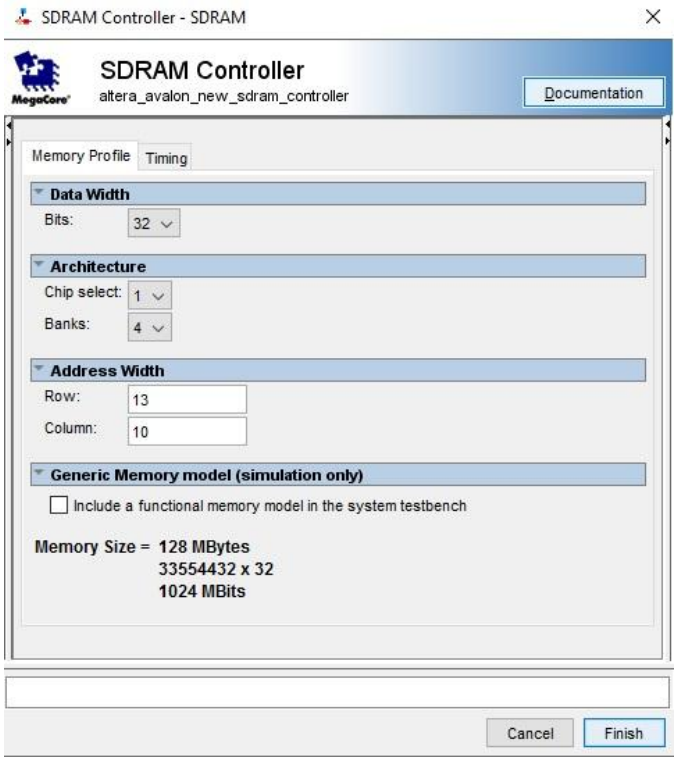

Fonte: Produção do próprio autor

O Próximo componente é um controlador que servirá de interface para o teclado. Encontra-em *University Program>Generic IO>PS2 Controller*. Adicione o módulo. Em seus parâmetros, selecione-o como um dispositivo *Memory Mapped,* ou seja, mapeado na memória. Renomeie para **PS2_port**.

Os próximos componentes a serem adicionados se referem aos dispositivos de memória do sistema. Dois tipos serão utilizados: SDRAM e SRAM.

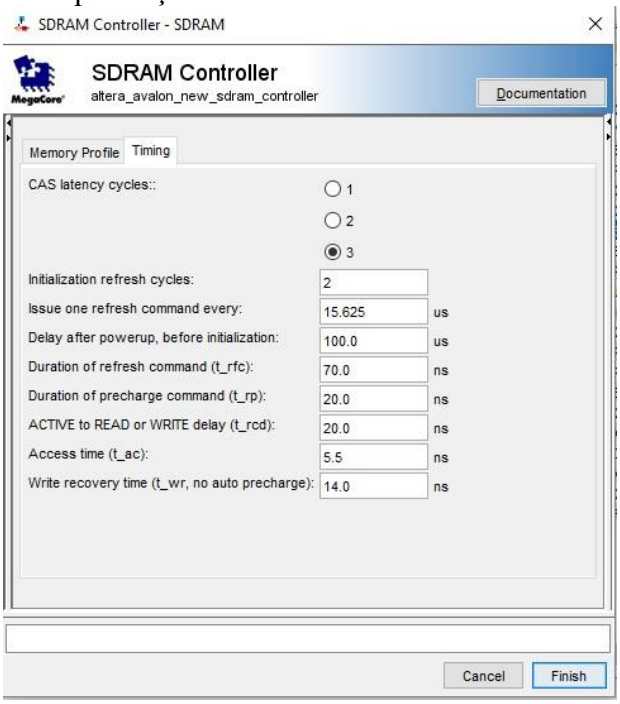
Para a memória SDRAM, que está associada ao processador, procure em *Memories and Memory Controllers>SDRAM Interfaces>SDRAM Controller* e adicione o componente. As Figura 11 e Figura 12 mostram as configurações de capacidade e temporização da memória, respectivamente. A quantidade de bits utilizada para o barramento de endereços e dados irá definir sua capacidade total; para o projeto, é interessante utilizar a capacidade máxima da memória (128 MB), pois dessa forma, é possível armazenar uma quantidade de dados maiores para serem encriptados e decriptados. Renomear o componente para **SDRAM**.

Figura 11 – Configurações do controlador para a memória SDRAM




Fonte: Produção do próprio autor

Figura 12 – Parâmetros de temporização e sincronismo da SDRAM




Fonte: Produção do próprio autor

A memória SRAM será utilizada para processar informações do *Buffer de Pixels* para o VGA. Encontra-se em *University Program>Memory>SRAM/SSRAM Controller*. Escolha a placa DE2-115 e marque a opção utilizar como um *buffer* de *Pixels*, como mostra a Figura 13. Após isso, clicar em *Finish* e renomear para **SRAM**.

Figura 13 – Controlador para a SRAM


Fonte: Produção do próprio autor

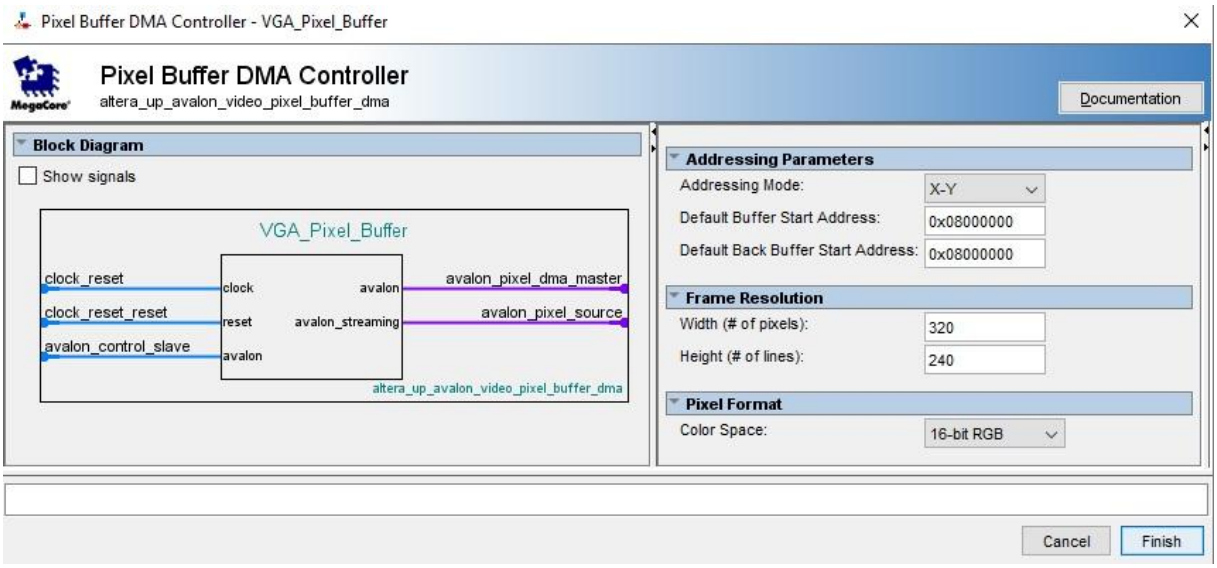
Os próximos componentes são controladores de vídeo, pertencentes ao Programa de Universidades da Altera. Este pacote fornece diversos componentes que realizam a interface entre o monitor VGA, bem como o processamento de imagens e vídeo.

O primeiro controlador é o *Buffer* de *Pixels*, que está associado à memória SRAM da placa. A memória SRAM é a mais adequada para esta função, pois exige menos sincronismo. Este componente encontra-se em *University Program>Audio & Video> Pixel Buffer DMA Controller*. Suas configurações são mostradas na Figura 14. Renomear para **VGA_Pixel_Buffer**.

No campo *Default Buffer Start Address* estará o endereço inicial da memória SRAM, pois é ela que irá processar os *Pixels* no monitor VGA.

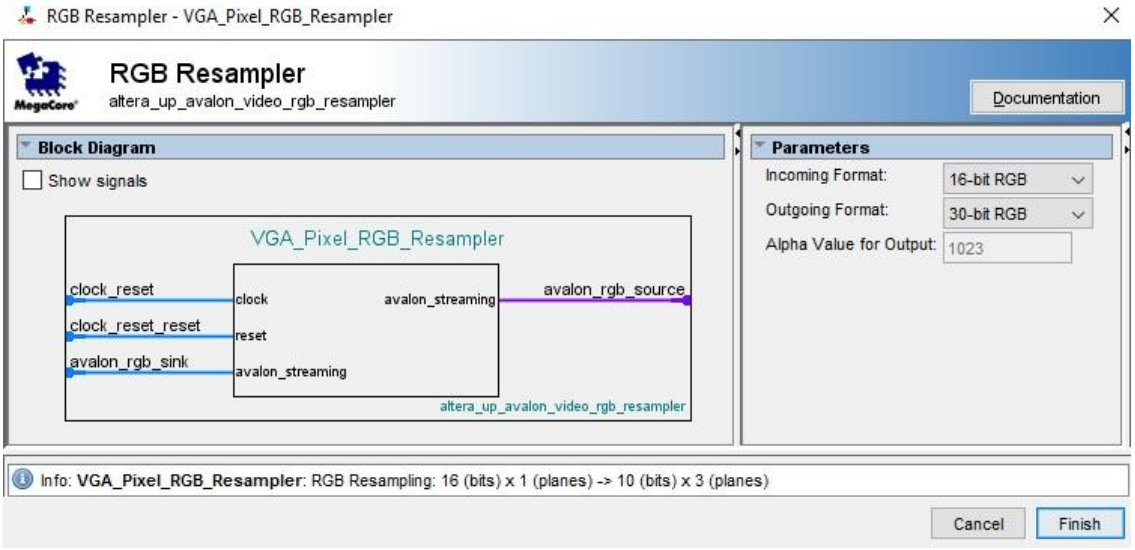
O *Resampler* é um componente que faz uma re-amostra de um *pixel*, e o converte em uma escala RGB. Encontra-se em *University Program>Audio & Video>RGB Resampler*. Suas configurações estão na Figura 15. Renomear para **VGA_Pixel_RGB_resampler**.

Figura 14 – Configurações para o *buffer* de *Pixels*




Fonte: Produção do próprio autor

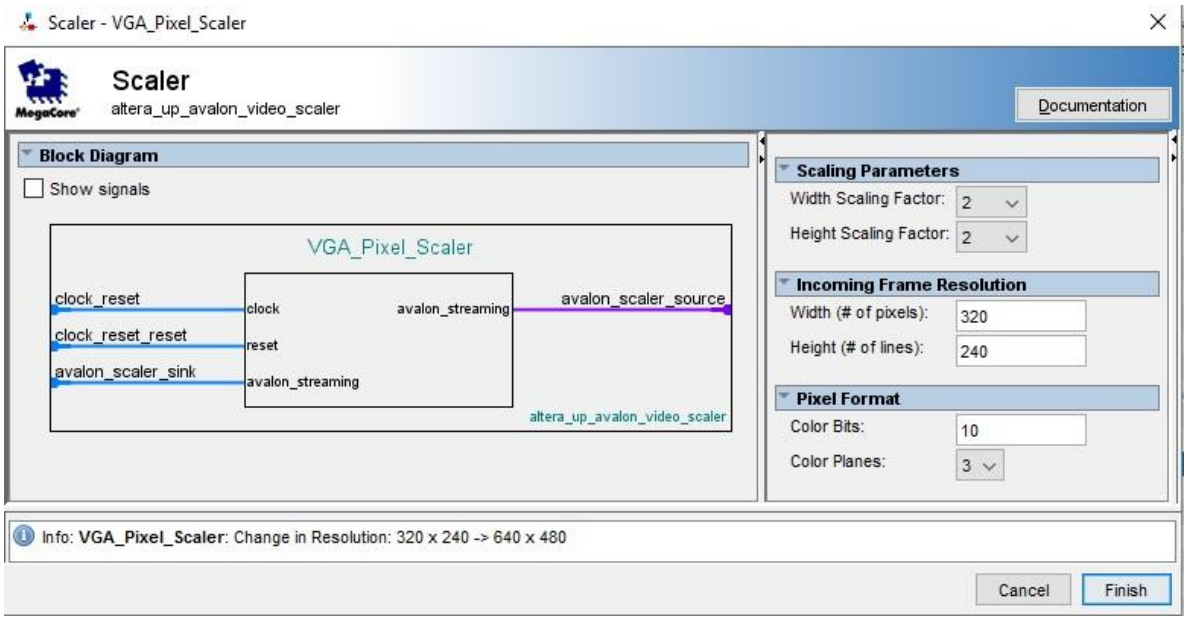
Figura 15 – Componente que realiza amostragem RGB




Fonte: Produção do próprio autor

O *Scaler* é responsável por ampliar ou reduzir o tamanho ou resolução de um *pixel*, através de um fator de multiplicação. Ele é utilizado, pois dessa forma, é possível ter entradas menores, reduzindo o tempo de processamento. A saída do *Scaler* será um *pixel* com resolução de 640 x 480, muito comum em monitores VGA. Encontra-se em *University Program>Audio & Video>Scaler*. As configurações estão na Figura 16. Renomear para **VGA_Pixel_*Scaler***.

Figura 16 – Módulo que controla a resolução de um *Pixel*



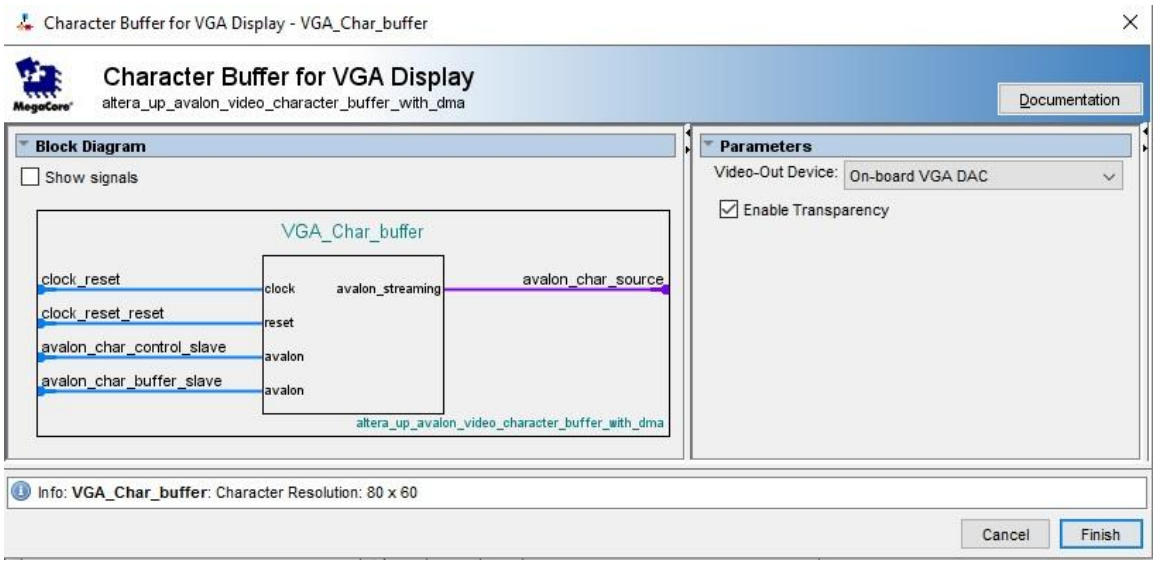

Fonte: Produção do próprio autor

É necessário para o projeto um controlador que armazene um caractere em um *buffer*, para que o mesmo possa ser mostrado em um monitor VGA. Para esta tarefa, utiliza-se um *buffer* de caracteres, que se encontra em *University Program>Audio & Video> Character Buffer for VGA Display*. A Figura 17 mostra como seus parâmetros são configurados. Renomear para **VGA_Char_buffer**.

O Alpha *Blender*, que é o próximo componente, tem a função de unir duas fontes de dados, uma proveniente do *buffer* de caracteres, outra do *Buffer* de *Pixels*, no mesmo dispositivo de saída, que é o monitor VGA. Caso houvesse apenas uma fonte de vídeo, este componente não seria necessário. Deve-se escolher o modo *Simple* nos parâmetros. Encontra-se em *University Program>Audio & Video>Video>Alpha Blender*. Renomear para **Alpha_Blending**.

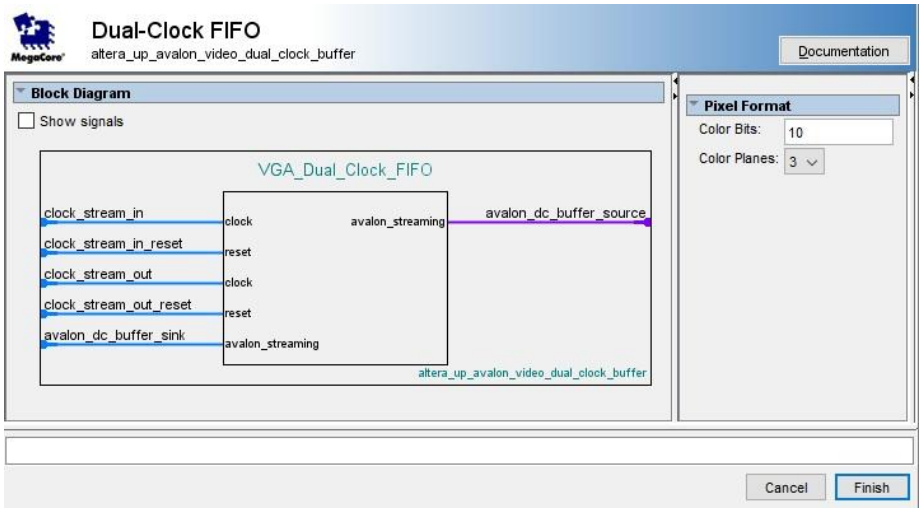
A seguir, busque por *University Program>Audio & Video>Video>Dual-Clock FIFO.* Este controlador tem a função de sincronizar o monitor com a fonte de dados. A Figura 18 contém as configurações. Renomear para **VGA_Dual_*Clock*_FIFO**.

Figura 17 – *Buffer* que armazena um caractere




Fonte: Produção do próprio autor

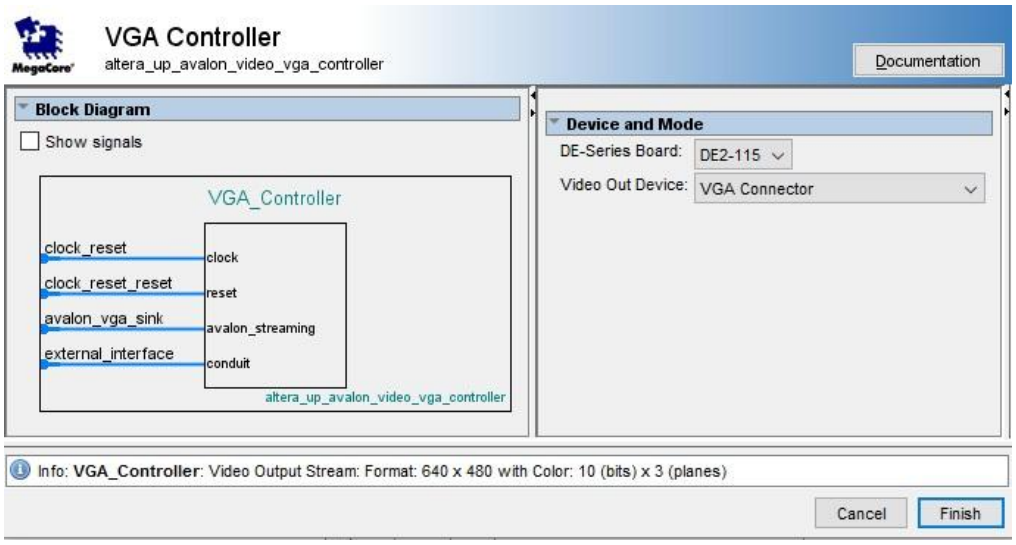
Figura 18 – Componente que realiza o sincronismo entre as fontes de dados.




Fonte: Produção do próprio autor

O projeto utilizará o controlador VGA, que realiza o controle de sincronismo vertical e horizontal do monitor VGA, além de outros parâmetros referentes ao monitor. Encontra-se em *University Program>Audio & Video>Video>VGA Controller*. As configurações estão na Figura 19. Renomear para **VGA_Controller**.

Figura 19 – Controlador para o Monitor VGA

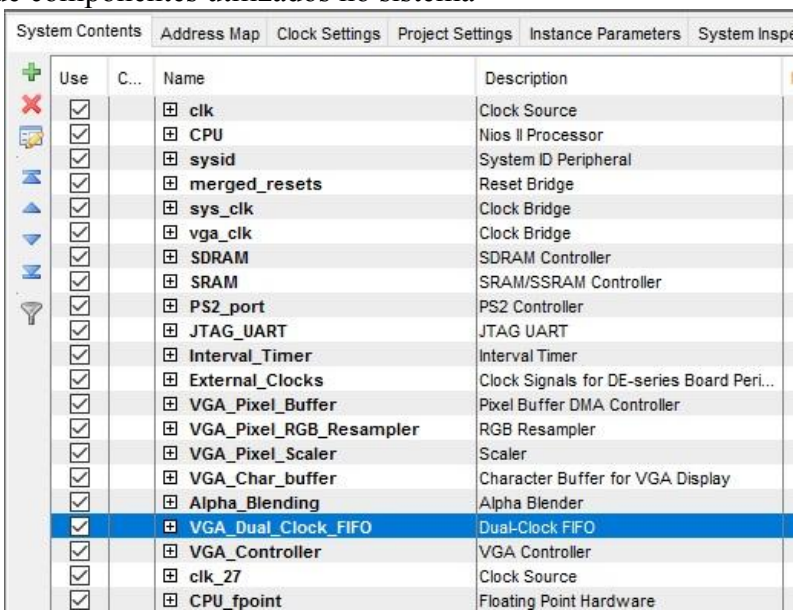



Fonte: Produção do próprio autor

Por último, foi adicionada uma instrução adicional para o Nios II, que realiza processamento com números reais. Encontra-se em *Floating point Hardware*. Renomear para **CPU_fpoint.**

O sistema Nios completo, após a adição de todos os componentes, deve ter o aspecto da Figura 20.

Figura 20 – Lista de componentes utilizados no sistema




Fonte: Produção do próprio autor

### 3.3.2 Conexões

A Quadro 2 a seguir mostra as conexões. Para realizar uma conexão entre dois sinais, basta clicar com o botão direito no sinal desejado e escolher o sinal ao qual ele será associado.

Quadro 2 – Conexões a serem feitas entre os sinais de cada componente

(continua)

| **Clk** | |
|---|---|
| clk_in | (Exported) |
| Clk_in_reset | Merged_resets.out_reset |
| Clk_clk | External_*Clock*s.clk_in_primary |
| Clk_reset | External_*Clock*s.clk_in_primary |
| | Alpha_Blending.*clock*_reset_reset |
| | PS2_port.*clock*_reset_reset |
| | SRAM.*clock*_reset_reset |
| | VGA_Char_*Buffer* |
| | VGA_Controller |
| | VGA_*Pixel_Buffer* |
| | VGA_*Pixel*_RGB_Resampler |
| | VGA_*Pixel_Scaler* |
| | VGA_Dual_*Clock*_FIFO |
| | VGA_Dual_*Clock*_FIFO |
| | Interval_*Timer* |
| | JTAG_UART |
| | SDRAM |
| | CPU |
| | Sysid.reset |
| **CPU** | |
| Clk | External_*Clock*s |

Quadro 2 – Conexões a serem feitas entre os sinais de cada componente

(continuação)

| | |
|---|---|
| Reset_n | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Data_master | (Conecta-se em todos) |
| Instruction_master | CPU.jtag_debug_module |
| | SDRAM.s1 |
| Jtag_debug_module_reset | (Conecta-se a todos, exceto: Clk.clk_in_reset,Clk_27.clk_in_reset e merged_resets.in_reset) |
| Jtag_debug_module | CPU.data_master |
| | CPU.instruction_master |
| Custom_instruction_master | CPU_fpoint.s1 |
| **Sysid** | |
| clk | External_*Clock*s.sys_clk |
| reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Control_slave | CPU.data_master |
| **Merged_resets** | |
| In_reset | (Exported) |
| Out_reset | External_*Clock*s.clk_in_primary_reset |
| | Clk.clk_in_reset |
| | Clk_27.clk_in_reset |
| **Sys_clk** | |
| In_clk | External_*Clock*s.sys_clk |
| Out_clk | (Exported) |
| **Vga_clk** | |
| In_clk | External_*Clock*s.vga_clk |
| Out_clk | (Exported) |

Quadro 2 – Conexões a serem feitas entre os sinais de cada componente

(continuação)

| | |
|---|---|
| **SDRAM** | |
| Clk | External_*Clock*s.sys_clk |
| Reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| S1 | CPU.Data_master |
| | CPU.instruction_master |
| Wire | (Exported) |
| **SRAM** | |
| *Clock*_reset | External_*Clock*s.sys_clk |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| External_interface | (Exported) |
| Avalon_sram_slave | VGA_*Pixel_Buffer*.avalon_*pixel*_dma_master |
| **PS2_port** | |
| *Clock*_reset | External_*Clock*s.sys_clk |
| | |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_ps2_slave | CPU.data_master |
| External_interface | (Exported) |
| **JTAG_UART** | |
| clk | External_*Clock*s.sys_clk |
| reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_jtag_slave | CPU.data_master |

Quadro 2 – Conexões a serem feitas entre os sinais de cada componente

(continuação)

| **Interval_Timer** | |
|---|---|
| clk | External_*Clock*s.sys_clk |
| reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| S1 | CPU.data_master |
| **External_*Clock*s** | |
| Clk_in_primary | Clk.clk |
| Clk_in_primary_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| | Merged_resets.out_reset |
| Sys_clk | CPU.clk |
| | Interval_*Timer*.clk |
| | JTAG_UART.clk |
| | SDRAM.clk |
| | Sysid.clk |
| | Alpha_Blending.*clock*_reset |
| | PS2_port.*clock*_reset |
| | SRAM.*clock*_reset |
| | VGA_Char_*buffer*.*clock*_reset |
| | VGA_*Pixel_Buffer*.*clock*_reset |
| | VGA_*Pixel*_RGB_Resampler.*clock*_reset |
| | VGA_*Pixel_Scaler*.*clock*_reset |
| | VGA_Dual_*Clock*_FIFO.*clock*_stream_in |
| | Sys_clk.in_clk |
| Sys_clk_reset | (Não conectado) |
| Sdram_clk | (Exported) |

Quadro 2 – Conexões a serem feitas entre os sinais de cada componente

(continuação)

| | |
|---|---|
| Vga_clk | VGA_Controller.*clock*_reset |
| | VGA_Dual_*Clock*_FIFO.*clock*_stream_out |
| | Vga_clk.in_clk |
| Clk_in_secondary | Clk_27.clk |
| Áudio_clk | (Exported) |
| **VGA_Pixel_Buffer** | |
| *Clock*_reset | External_*Clock*s.sys_clk |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_*pixel*_dma_master | SRAM. Avalon_sram_slave |
| Avalon_control_slave | CPU.data_master |
| Avalon_*pixel*_source | VGA_*Pixel*_RBG_Resampler.avalon_rgb_sink |
| **VGA_Pixel_RBG_Resampler** | |
| *Clock*_reset | External_*Clock*s.sys_clk |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_rgb_sink | VGA_*Pixel_Buffer*.avalon_*pixel*_source |
| Avalon_rgb_source | VGA_*Pixel_Scaler*.avalon_*Scaler*_sink |
| **VGA_Pixel_*Scaler*** | |
| *Clock*_reset | External_*Clock*s.sys_clk |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_*Scaler*_sink | VGA_*Pixel*_RBG_Resampler.avalon_rgb_source |
| Avalon_*Scaler*_source | Alpha_blending.avalon_backg*round*_sink |

Quadro 2 – Conexões a serem feitas entre os sinais de cada componente

(continuação)

| **VGA_Char_buffer** | |
|---|---|
| *Clock*_reset | External_*Clock*s.sys_clk |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_char_control_slave | CPU.data_master |
| Avalon_char_*buffer*_slave | CPU.data_master |
| Avalon_char_source | Alpha_Blending.avalon_foreg*round*_sink |
| **Alpha_Blending** | |
| *Clock*_reset | External_*Clock*s.sys_clk |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_foreg*round*_sink | VGA_Char_*buffer*. Avalon_char_source |
| Avalon_backg*round*_sink | VGA_*Pixel_Scaler*. Avalon_*Scaler*_source |
| Avalon_blended_source | VGA_Dual_*Clock*_FIFO.avalon_dc_*buffer*_sink |
| **VGA_Dual_*Clock*_FIFO** | |
| *Clock*_stream_in | External_*Clock*s.sys_clk |
| *Clock*_stream_in_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| *Clock*_stream_out | Extenal_*Clock*s.vga_clk |
| *Clock*_stream_out_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_dc_*buffer*_sink | Alpha_Blending.Avalon_blended_source |
| Avalon_dc_*buffer*_source | VGA_Controller.avalon_vga_sink |

Quadro 2 – Conexões a serem feitas entre os sinais de cada componente

(conclusão)

| **VGA_Controller** | |
|---|---|
| *Clock*_reset | Extenal_*Clock*s.vga_clk |
| *Clock*_reset_reset | Clk.clk_reset |
| | Clk_27.clk_reset |
| | CPU.jtag_debug_module_reset |
| Avalon_vga_sink | VGA_*Pixel*_RBG_Resampler.avalon_rgb_source |
| External_interface | (Exported) |
| **Clk_27** | |
| Clk_in | (Exported) |
| Clk_in_reset | Merged_resets.out_reset |
| clk | External_*Clock*s.clk_in_secondary |
| Clk_reset | (Conecta-se em todos, exceto: Clk.clk_in_reset e merged_resets.in_reset) |
| **CPU_fpoint** | |
| S1 | CPU.custom_instruction_master |

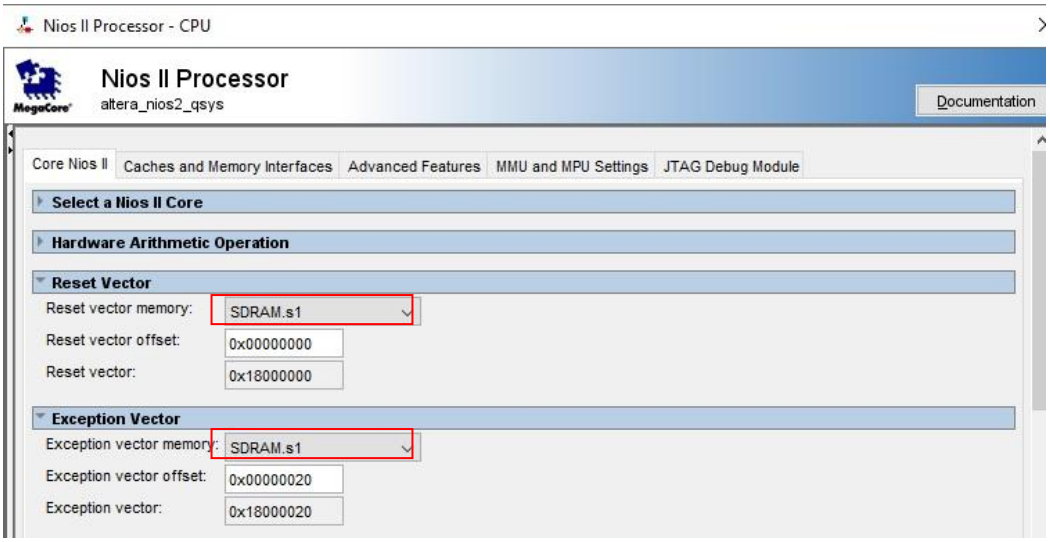
Fonte: Produção do próprio autor

Após realizar as conexões, deve-se conectar os dispositivos que possuem interrupção ao processador, na coluna IRQ, para eliminar possíveis erros de conexão.

### 3.3.3 Reset Vector e Exception Vector

É necessário associar os endereços de memória referentes ao *Reset Vector* e *Exception Vector* à memória SDRAM, para isso, clicar duas vezes no componente CPU e trocar os parâmetros, como indica a Figura 21.

Figura 21 – Configurando os vetores de inicialização e exceção do processador Nios II

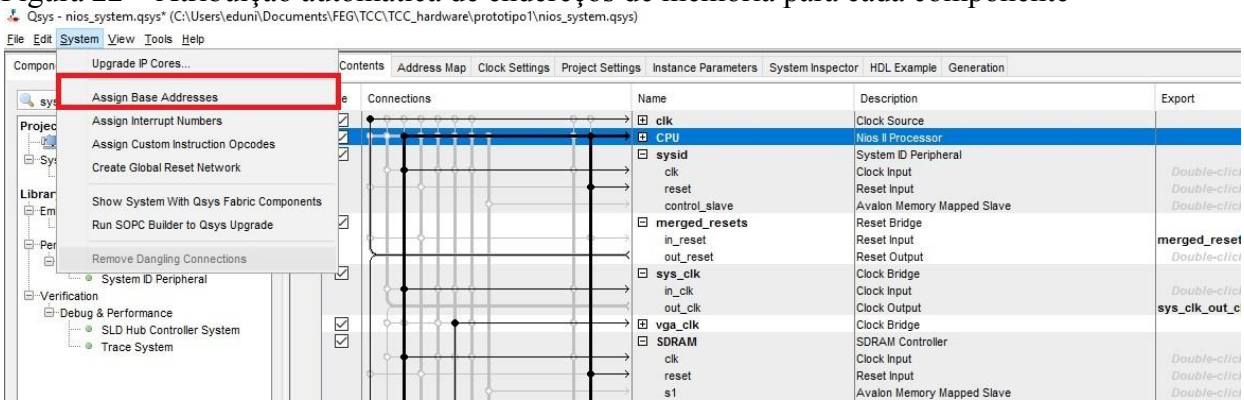



Fonte: Produção do próprio autor

### 3.3.4 Base Adressses

Na página do Qsys, é possível ver uma aba chamada *Address Map*. Lá estão todos os intervalos de endereços de memória associados para cada componente. Para definir os endereços, há duas opções, ou de forma manual, ou então atribuir endereços de forma automática. Para o projeto, escolheu-se a segunda opção, por questões de facilidade. Na aba *System*, selecionar a opção *Assign Base Addresses*, como mostra a Figura 22.

Figura 22 – Atribuição automática de endereços de memória para cada componente




Fonte: Produção do próprio autor

### 3.3.5 Configurações finais

Após todas as configurações terem sido feitas, deve-se ter na aba *System Contents* um sistema igual ao da Figura 20. É importante lembrar-se de exportar todos os sinais que terão conexões físicas com a placa[2].

Na aba *Address map* está o mapeamento de todos os componentes e os seus respectivos endereços de memória, de acordo com a Figura 23.

Em *Clock Settings*, estão todos os *clock*s que foram criados; os componentes ***clk*** e ***clk_27*** são fontes de *clock* e estão associados diretamente aos osciladores da placa. Sys_clk_out_clk e vga_clk_out_clk são sinais que realizam a transmissão do sinal para o componente **External_*Clock*s**, que é responsável pela geração dos *clock*s do sistema. A Figura 24 mostra estes sinais.

Na aba *Project Settings,* mostrada na Figura 25, é possível ver o diagrama de blocos do sistema, com todos os sinais de entrada e saída. Caso a opção *Show signals* esteja marcada, é possível ver com mais detalhes todos os sinais internos de cada componente.

Na aba *HDL Example*, pode-se gerar um código automaticamente, em VHDL ou Verilog, que pode ser utilizado para instanciar o sistema no projeto do Quartus. Escolheu-se trabalhar com VHDL.

Finalmente, na aba *Generation*, há algumas opções como criar arquivos de simulação, *testbench* e opções de síntese; deixe essas configurações como estão na Figura 26. Após isso, clicar em *Generate*, para que o sistema seja gerado. Salve o projeto na mesma pasta do projeto do Quartus.

Figura 23 – Mapeamento dos endereços de memória

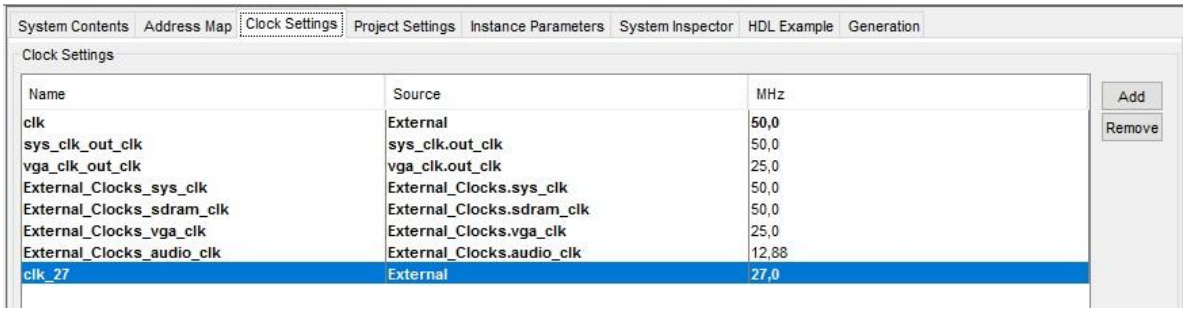
System Contents | Address Map | Clock Settings | Project Settings | Instance Parameters | System Inspector | HDL Example | Generation

| | CPU.data_master | CPU.instruction_master | VGA_Pixel_Buffer.avalon_pixel_dma_master |
|---|---|---|---|
| CPU.jtag_debug_module | 0x0820_0800 - 0x0820_0fff | 0x0820_0800 - 0x0820_0fff | |
| sysid.control_slave | 0x0820_1030 - 0x0820_1037 | | |
| SDRAM.s1 | 0x1800_0000 - 0x1fff_ffff | 0x1800_0000 - 0x1fff_ffff | |
| SRAM.avalon_sram_slave | 0x0800_0000 - 0x081f_ffff | | 0x0800_0000 - 0x081f_ffff |
| PS2_port.avalon_ps2_slave | 0x1000_0100 - 0x1000_0107 | | |
| JTAG_UART.avalon_jtag_slave | 0x0820_1038 - 0x0820_103f | | |
| Interval_Timer.s1 | 0x0820_1000 - 0x0820_101f | | |
| VGA_Pixel_Buffer.avalon_control_slave | 0x0820_1020 - 0x0820_102f | | |
| VGA_Char_buffer.avalon_char_contro... | 0x0820_1040 - 0x0820_1047 | | |
| VGA_Char_buffer.avalon_char_buffer... | 0x0900_0000 - 0x0900_1fff | | |

Fonte: Produção do próprio autor

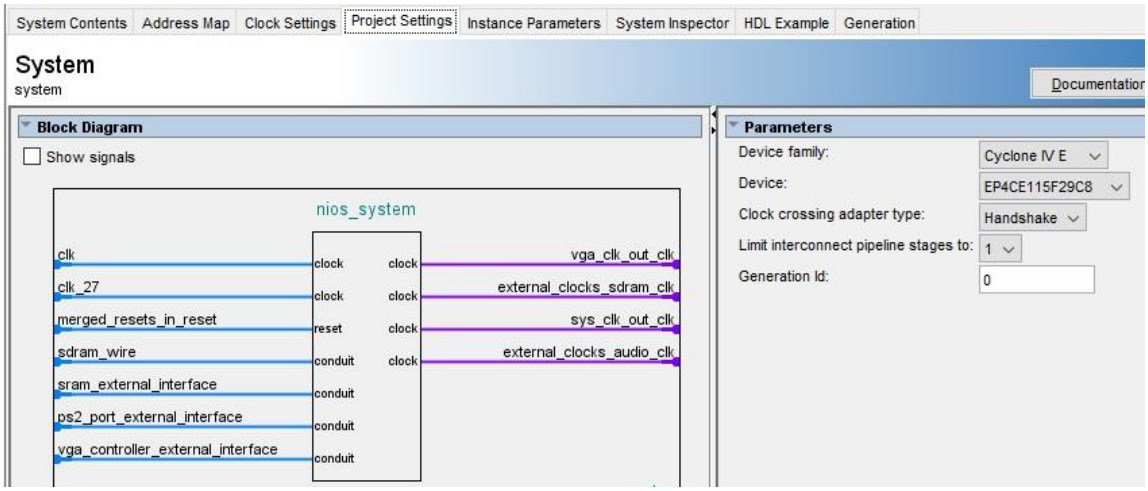
[2] No item 3.3.2, correspondem aos sinais com a palavra Exported em parênteses.

Figura 24 – Sinais e fontes de *clock* do sistema



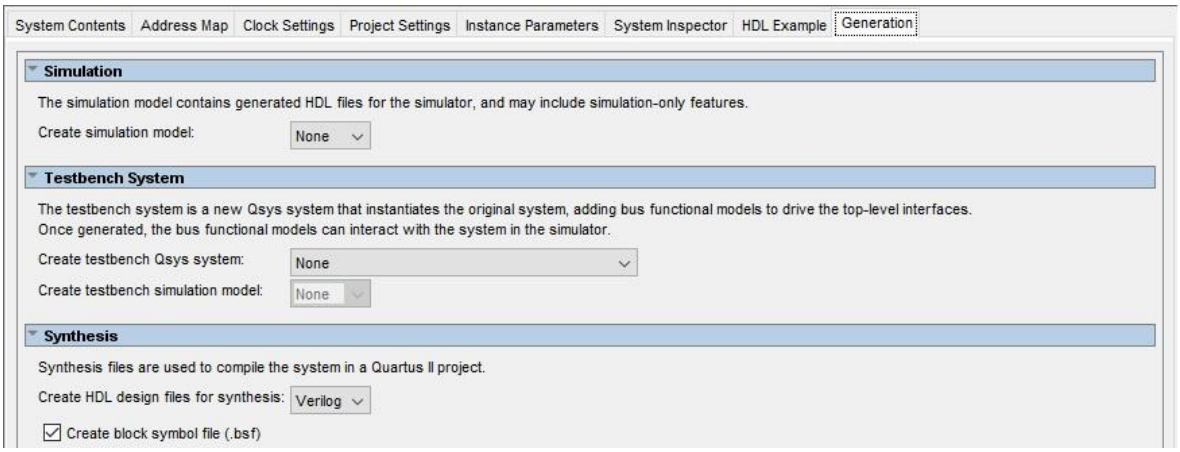

Fonte: Produção do próprio autor

Figura 25 – Configurações gerais do sistema e diagrama de blocos




Fonte: Produção do próprio autor

Figura 26 – Configurações de simulação, *testbench* e síntese do sistema

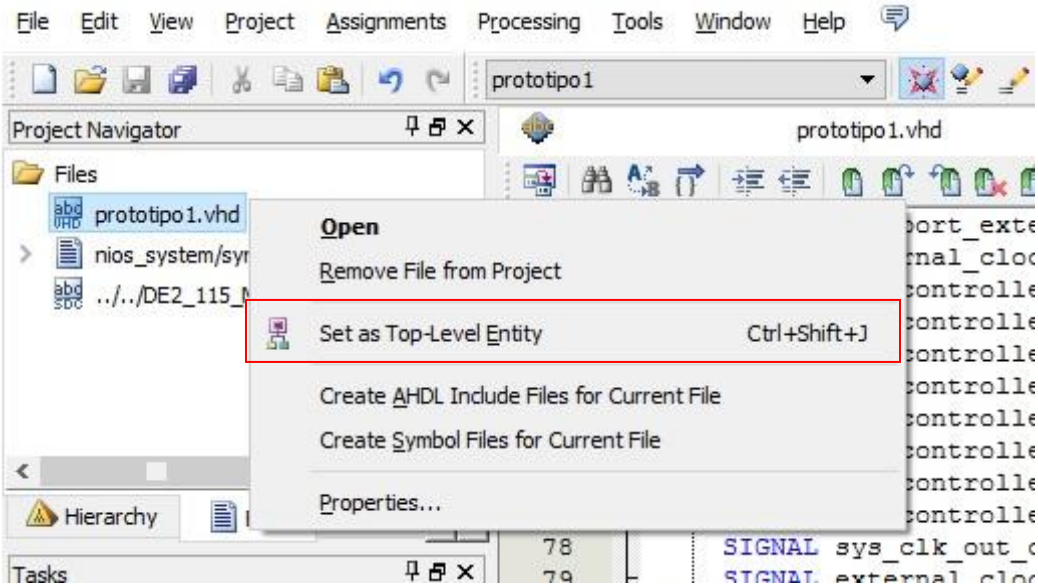



Fonte: Produção do próprio autor

## 3.4 CRIAÇÃO DO TOP LEVEL

Após a geração do Qsys, será criado um arquivo em VHDL no Quartus, que servirá como entidade TOP *level* para o projeto, lá estão todas as entradas e PORTS que serão utilizadas, além de conter uma instanciação do sistema gerado previamente. É importante definir o arquivo de descrição como TOP *Level*, para isso, vá à aba *Files* em *Project navigator*, clique com o botão direito no arquivo ".vhd" criado e selecione *Set as Top-Level Entity*, como indica a Figura 27.

Figura 27 – Definindo a entidade TOP *level*




Fonte: Produção do próprio autor

O Código em VHDL instancia todos os componentes criados no Qsys e também define as conexões de seus sinais com a placa. Está disponível abaixo.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;


ENTITY prototipo1 IS
PORT (
      -- Clocks Globais
      CLOCK_50          : IN STD_LOGIC;
      TD_CLK27          : IN STD_LOGIC;
      -- Reset
      TD_RESET_N  : OUT STD_LOGIC;
```

```
      -- Memória(SRAM)
      SRAM_DQ           : INOUT STD_LOGIC_VECTOR (15 DOWNTO 0);
      SRAM_ADDR   : OUT STD_LOGIC_VECTOR (19 DOWNTO 0);
      SRAM_CE_N   : OUT STD_LOGIC;
      SRAM_WE_N   : OUT STD_LOGIC;
      SRAM_OE_N   : OUT STD_LOGIC;
      SRAM_UB_N   : OUT STD_LOGIC;
      SRAM_LB_N   : OUT STD_LOGIC;
      -- Memória(SDRAM)
      DRAM_DQ           : INOUT STD_LOGIC_VECTOR (31 DOWNTO 0);
      DRAM_ADDR   : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
      DRAM_BA           : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
      DRAM_CAS_N  : OUT STD_LOGIC;
      DRAM_RAS_N  : OUT STD_LOGIC;
      DRAM_CLK          : OUT STD_LOGIC;
      DRAM_CKE          : OUT STD_LOGIC;
      DRAM_CS_N   : OUT STD_LOGIC;
      DRAM_WE_N   : OUT STD_LOGIC;
      DRAM_DQM          : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
      -- Teclado
      PS2_CLK           : INOUT STD_LOGIC;
      PS2_DAT           : INOUT STD_LOGIC;
      -- Monitor VGA
      VGA_CLK           : OUT STD_LOGIC;
      VGA_HS            : OUT STD_LOGIC;
      VGA_VS            : OUT STD_LOGIC;
      VGA_BLANK_N       : OUT STD_LOGIC;
      VGA_SYNC_N  : OUT STD_LOGIC;
      VGA_R             : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      VGA_G             : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
      VGA_B             : OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END prototipo1;

ARCHITECTURE behavior OF prototipo1  IS
COMPONENT nios_system
-- pode-se aproveitar o código gerado no Qsys, na aba HDL Example
PORT(SIGNAL clk_clk                                     : IN   STD_LOGIC;
     SIGNAL clk_27_clk                                  : IN   STD_LOGIC;
     SIGNAL merged_resets_in_reset_reset_n              : IN   STD_LOGIC;
     SIGNAL sdram_wire_addr                : OUT   STD_LOGIC_VECTOR(12 DOWNTO 0);
     SIGNAL sdram_wire_ba                   : OUT   STD_LOGIC_VECTOR(1 DOWNTO 0);
     SIGNAL sdram_wire_cas_n                            : OUT   STD_LOGIC;
     SIGNAL sdram_wire_cke                              : OUT   STD_LOGIC;
     SIGNAL sdram_wire_cs_n                             : OUT   STD_LOGIC;
     SIGNAL sdram_wire_dq                : INOUT   STD_LOGIC_VECTOR(31 DOWNTO 0);
     SIGNAL sdram_wire_dqm                  : OUT   STD_LOGIC_VECTOR(3 DOWNTO 0);
     SIGNAL sdram_wire_ras_n                            : OUT   STD_LOGIC;
     SIGNAL sdram_wire_we_n                             : OUT   STD_LOGIC;
     SIGNAL sram_external_interface_DQ : INOUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
     SIGNAL sram_external_interface_ADDR : OUT   STD_LOGIC_VECTOR(19 DOWNTO 0);
     SIGNAL sram_external_interface_LB_N                : OUT   STD_LOGIC;
     SIGNAL sram_external_interface_UB_N                : OUT   STD_LOGIC;
     SIGNAL sram_external_interface_CE_N                : OUT   STD_LOGIC;
     SIGNAL sram_external_interface_OE_N                : OUT   STD_LOGIC;
     SIGNAL sram_external_interface_WE_N                : OUT   STD_LOGIC;
     SIGNAL ps2_port_external_interface_CLK             : INOUT   STD_LOGIC;
     SIGNAL ps2_port_external_interface_DAT             : INOUT   STD_LOGIC;
     SIGNAL external_clocks_sdram_clk_clk               : OUT   STD_LOGIC;
     SIGNAL vga_controller_external_interface_CLK   : OUT   STD_LOGIC;
     SIGNAL vga_controller_external_interface_HS    : OUT   STD_LOGIC;
     SIGNAL vga_controller_external_interface_VS    : OUT   STD_LOGIC;
     SIGNAL vga_controller_external_interface_BLANK : OUT   STD_LOGIC;
     SIGNAL vga_controller_external_interface_SYNC  : OUT   STD_LOGIC;
```

```
      SIGNAL vga_controller_external_interface_R:OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      SIGNAL vga_controller_external_interface_G:OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      SIGNAL vga_controller_external_interface_B:OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      SIGNAL sys_clk_out_clk_clk                     : OUT   STD_LOGIC;
      SIGNAL external_clocks_audio_clk_clk           : OUT   STD_LOGIC);
END COMPONENT nios_system;

BEGIN
NIOS : nios_system  PORT MAP (
       clk_clk                                 => CLOCK_50,
       clk_27_clk                              => TD_CLK27,
       merged_resets_in_reset_reset_n          => '1',
       sdram_wire_addr                         => DRAM_ADDR,
       sdram_wire_ba                           => DRAM_BA,
       sdram_wire_cas_n                        => DRAM_CAS_N,
       sdram_wire_cke                          => DRAM_CKE,
       sdram_wire_cs_n                         => DRAM_CS_N,
       sdram_wire_dq                           => DRAM_DQ,
       sdram_wire_dqm                          => DRAM_DQM,
       sdram_wire_ras_n                        => DRAM_RAS_N,
       sdram_wire_we_n                         => DRAM_WE_N,
       sram_external_interface_DQ              => SRAM_DQ,
       sram_external_interface_ADDR            => SRAM_ADDR,
       sram_external_interface_LB_N            => SRAM_LB_N,
       sram_external_interface_UB_N            => SRAM_UB_N,
       sram_external_interface_CE_N            => SRAM_CE_N,
       sram_external_interface_OE_N            => SRAM_OE_N,
       sram_external_interface_WE_N            => SRAM_WE_N,
       ps2_port_external_interface_CLK         => PS2_CLK,
       ps2_port_external_interface_DAT         => PS2_DAT,
       external_clocks_sdram_clk_clk           => DRAM_CLK,
       vga_controller_external_interface_CLK   => VGA_CLK,
       vga_controller_external_interface_HS    => VGA_HS,
       vga_controller_external_interface_VS    => VGA_VS,
       vga_controller_external_interface_BLANK => VGA_BLANK_N,
       vga_controller_external_interface_SYNC  => VGA_SYNC_N,
       vga_controller_external_interface_R     => VGA_R,
       vga_controller_external_interface_G     => VGA_G,
       vga_controller_external_interface_B     => VGA_B);
END behavior;
```

A atribuição dos pinos pode ser feita de duas maneiras: importando um arquivo de configurações, ou manualmente. Para a primeira opção, deve-se ir à aba *Assignments* e selecionar a opção *Import Assignments*. Foi utilizado o arquivo **DE2_115_pin_assignments.csv**, que acompanha o DVD da DE2-115. É necessário lembrar que os pinos do arquivo devem ter o mesmo nome das PORTS declaradas na entidade, senão não serão identificados. Após importar o arquivo, remova todos os pinos que não serão utilizados.

Caso os pinos sejam configurados manualmente, deve-se consultar o manual da placa, para atribuir os pinos às devidas PORTS. O Quadro 3 mostra essa configuração.

Quadro 3 – Atribuição dos pinos às respectivas entradas e saídas

(continua)

| PORT | Pino | PORT | Pino | PORT | Pino |
|---|---|---|---|---|---|
| *CLOCK*_50 | PIN_Y2 | DRAM_DQ[25] | PIN_R7 | DRAM_DQM[3] | PIN_N8 |
| DRAM_ADDR[12] | PIN_Y7 | DRAM_DQ[24] | PIN_U5 | DRAM_DQM[2] | PIN_K8 |
| DRAM_ADDR[11] | PIN_AA5 | DRAM_DQ[23] | PIN_L7 | DRAM_DQM[1] | PIN_W4 |
| DRAM_ADDR[10] | PIN_R5 | DRAM_DQ[22] | PIN_M7 | DRAM_DQM[0] | PIN_U2 |
| DRAM_ADDR[9] | PIN_Y6 | DRAM_DQ[21] | PIN_M4 | DRAM_RAS_N | PIN_U6 |
| DRAM_ADDR[8] | PIN_Y5 | DRAM_DQ[20] | PIN_N4 | DRAM_WE_N | PIN_V6 |
| DRAM_ADDR[7] | PIN_AA7 | DRAM_DQ[19] | PIN_N3 | PS2_CLK[3] | PIN_G6 |
| DRAM_ADDR[6] | PIN_W7 | DRAM_DQ[18] | PIN_P2 | PS2_DAT | PIN_H5 |
| DRAM_ADDR[5] | PIN_W8 | DRAM_DQ[17] | PIN_L8 | SRAM_ADDR[19] | PIN_T8 |
| DRAM_ADDR[4] | PIN_V5 | DRAM_DQ[16] | PIN_M8 | SRAM_ADDR[18] | PIN_AB8 |
| DRAM_ADDR[3] | PIN_P1 | DRAM_DQ[15] | PIN_AC2 | SRAM_ADDR[17] | PIN_AB9 |
| DRAM_ADDR[2] | PIN_U8 | DRAM_DQ[14] | PIN_AB3 | SRAM_ADDR[16] | PIN_AC11 |
| DRAM_ADDR[1] | PIN_V8 | DRAM_DQ[13] | PIN_AC1 | SRAM_ADDR[15] | PIN_AB11 |
| DRAM_ADDR[0] | PIN_R6 | DRAM_DQ[12] | PIN_AB2 | SRAM_ADDR[14] | PIN_AA4 |
| DRAM_BA[1] | PIN_R4 | DRAM_DQ[11] | PIN_AA3 | SRAM_ADDR[13] | PIN_AC3 |
| DRAM_BA[0] | PIN_U7 | DRAM_DQ[10] | PIN_AB1 | SRAM_ADDR[12] | PIN_AB4 |
| DRAM_CAS_N | PIN_V7 | DRAM_DQ[9] | PIN_Y4 | SRAM_ADDR[11] | PIN_AD3 |
| DRAM_CKE | PIN_AA6 | DRAM_DQ[8] | PIN_Y3 | SRAM_ADDR[10] | PIN_AF2 |
| DRAM_CLK | PIN_AE5 | DRAM_DQ[7] | PIN_U3 | SRAM_ADDR[9] | PIN_T7 |
| DRAM_CS_N | PIN_T4 | DRAM_DQ[6] | PIN_V1 | SRAM_ADDR[8] | PIN_AF5 |
| DRAM_DQ[31] | PIN_U1 | DRAM_DQ[5] | PIN_V2 | SRAM_ADDR[7] | PIN_AC5 |
| DRAM_DQ[30] | PIN_U4 | DRAM_DQ[4] | PIN_V3 | SRAM_ADDR[6] | PIN_AB5 |
| DRAM_DQ[29] | PIN_T3 | DRAM_DQ[3] | PIN_W1 | SRAM_ADDR[5] | PIN_AE6 |
| DRAM_DQ[28] | PIN_R3 | DRAM_DQ[2] | PIN_V4 | SRAM_ADDR[4] | PIN_AB6 |
| DRAM_DQ[27] | PIN_R2 | DRAM_DQ[1] | PIN_W2 | SRAM_ADDR[3] | PIN_AC7 |
| DRAM_DQ[26] | PIN_R1 | DRAM_DQ[0] | PIN_W3 | SRAM_ADDR[2] | PIN_AE7 |

[3] Os pinos do teclado, PS2_CLK e PS2_DAT talvez tenham que ser modificados, no nível de tensão de 2,5V para 3,3V LVTTL.

Quadro 3 – Atribuição dos pinos às respectivas entradas e saídas

(conclusão)

| | | | | | |
|---|---|---|---|---|---|
| SRAM_ADDR[1] | PIN_AD7 | SRAM_OE_N | PIN_AD5 | VGA_G[2] | PIN_F8 |
| SRAM_ADDR[0] | PIN_AB7 | SRAM_UB_N | PIN_AC4 | VGA_G[1] | PIN_G11 |
| SRAM_CE_N | PIN_AF8 | SRAM_WE_N | PIN_AE8 | VGA_G[0] | PIN_G8 |
| SRAM_DQ[15] | PIN_AG3 | TD_CLK27 | PIN_B14 | VGA_HS | PIN_G13 |
| SRAM_DQ[14] | PIN_AF3 | TD_RESET_N | PIN_G7 | VGA_R[7] | PIN_H10 |
| SRAM_DQ[13] | PIN_AE4 | VGA_B[7] | PIN_D12 | VGA_R[6] | PIN_H8 |
| SRAM_DQ[12] | PIN_AE3 | VGA_B[6] | PIN_D11 | VGA_R[5] | PIN_J12 |
| SRAM_DQ[11] | PIN_AE1 | VGA_B[5] | PIN_C12 | VGA_R[4] | PIN_G10 |
| SRAM_DQ[10] | PIN_AE2 | VGA_B[4] | PIN_A11 | VGA_R[3] | PIN_F12 |
| SRAM_DQ[9] | PIN_AD2 | VGA_B[3] | PIN_B11 | VGA_R[2] | PIN_D10 |
| SRAM_DQ[8] | PIN_AD1 | VGA_B[2] | PIN_C11 | VGA_R[1] | PIN_E11 |
| SRAM_DQ[7] | PIN_AF7 | VGA_B[1] | PIN_A10 | VGA_R[0] | PIN_E12 |
| SRAM_DQ[6] | PIN_AH6 | VGA_B[0] | PIN_B10 | VGA_SYNC_N | PIN_C10 |
| SRAM_DQ[5] | PIN_AG6 | VGA_BLANK_N | PIN_F11 | VGA_VS | PIN_C13 |
| SRAM_DQ[4] | PIN_AF6 | VGA_CLK | PIN_A12 | | |
| SRAM_DQ[3] | PIN_AH4 | VGA_G[7] | PIN_C9 | | |
| SRAM_DQ[2] | PIN_AG4 | VGA_G[6] | PIN_F10 | | |
| SRAM_DQ[1] | PIN_AF4 | VGA_G[5] | PIN_B8 | | |
| SRAM_DQ[0] | PIN_AH3 | VGA_G[4] | PIN_C8 | | |
| SRAM_LB_N | PIN_AD4 | VGA_G[3] | PIN_H12 | | |

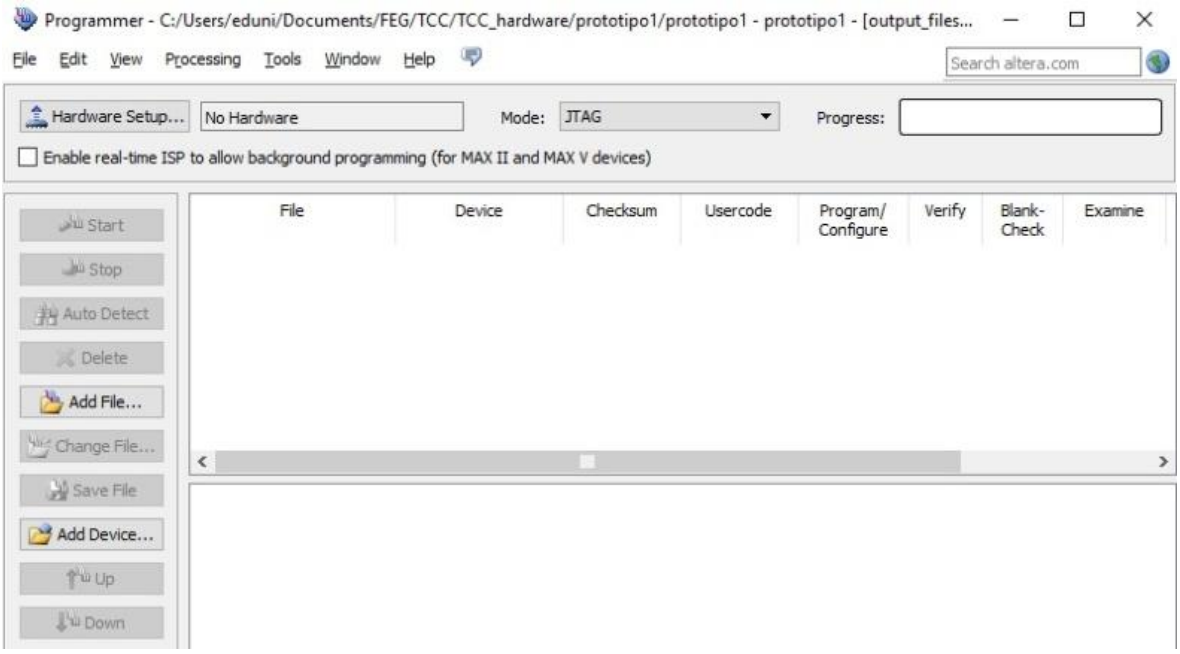
Fonte: Produção do próprio autor

Após a compilação do código e atribuição dos pinos, resta adicionar o arquivo referente aos componentes gerados no Qsys ao projeto, para isso, deve-se abrir a aba *Project>Add/Remove Files in Project*. Na janela que abrir, selecione o arquivo ".qip" referente ao sistema do Qsys, ele provavelmente estará na pasta *synthesis*, e terá o mesmo nome do arquivo com extensão ".qsys". Compilar o programa novamente.

## 3.5 IMPLANTAÇÃO DO HARDWARE NA DE2-115

Após a compilação do projeto, é necessário selecionar um dispositivo alvo, ou seja, a placa DE2-115, para implantar o *hardware* sintetizado pelo Quartus. Para isso, vá até a aba *Tools* e selecione a opção *Programmer*, lembrando que a placa deve estar ligada e o cabo *USB Blaster* deve estar instalado e conectado ao computador. A janela da Figura 28 irá aparecer.

Figura 28 – Tela para programar o hardware na DE2-115




Fonte: Produção do próprio autor

Clique em *Hardware Setup* e selecione o cabo *USB Blaster*. Após isso, clique em *Add File* e selecione o arquivo com a extensão ".sof", referente ao projeto, estes arquivos são gerados após a síntese do *hardware*, e podem ser encontrados na pasta *output_files*, localizada na mesma pasta do projeto. Após estes procedimentos, clique em *Start*. Quando for concluído, o *hardware* já estará configurado na placa, neste momento será possível ver algumas interações da placa com os periféricos.

A montagem do *hardware* foi concluída. O próximo capítulo irá discutir a respeito do desenvolvimento do *software*.

# 4 DESENVOLVIMENTO DO SOFTWARE

O código do *software* foi escrito em linguagem de programação C e foi desenvolvido na plataforma de desenvolvimento integrada para o NIOS II, Eclipse. A criação de um *software* permite que o *hardware* se comunique com os periféricos da placa, através de um conjunto de instruções executadas sequencialmente pelo processador.

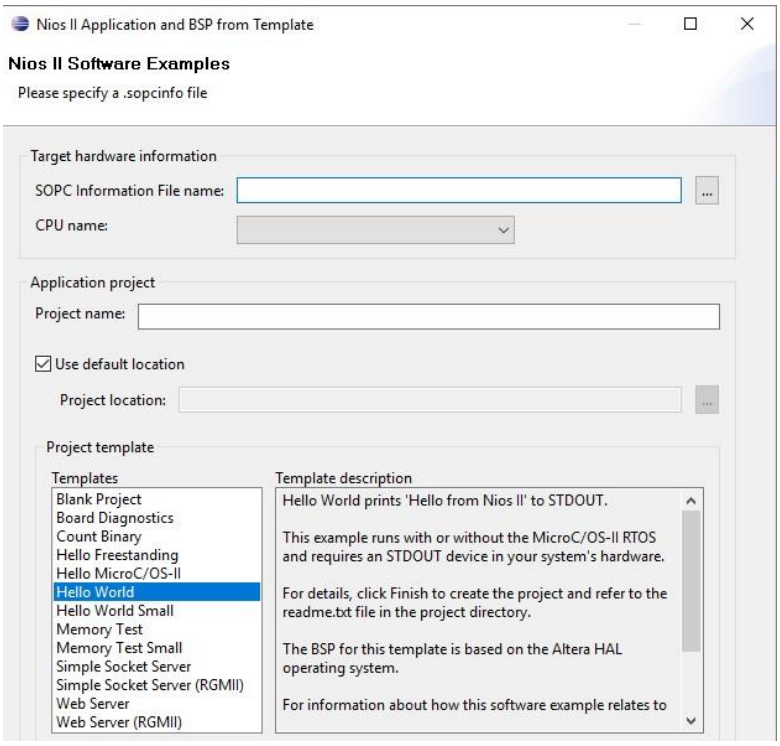
## 4.1 CRIAÇÃO DO SOFTWARE

Primeiramente, deve-se abrir o *Nios II 13.0 sp1 Software Build Tools for Eclipse*, que é a plataforma de desenvolvimento para o *software* do projeto. Antes de entrar na página inicial do Eclipse, será pedido um ambiente de trabalho. Crie uma pasta no mesmo diretório onde está contido o projeto do Quartus e selecione esta pasta como o *workspace.*

O *software* se divide em duas partes: a aplicação, que contém o código principal, bibliotecas e cabeçalhos, e o *Board Support Package* (BSP), que, de acordo com Altera (2010), é uma biblioteca especializada que contém arquivos de suporte para o sistema. Selecione a aba *File > New > Nios II Application and BSP from Template,* para criar um novo projeto. A janela da Figura 29 irá aparecer; nela um dos campos pedirá o arquivo SOPC, que pode ser encontrado na pasta do projeto. Defina um nome para o projeto. Em *Project template* é possível ver uma lista com vários exemplos de *softwares*. Foi utilizado o *template Hello World*, pois apresenta todos os recursos básicos para a criação de um *software* Nios II. Clique em *Finish*.

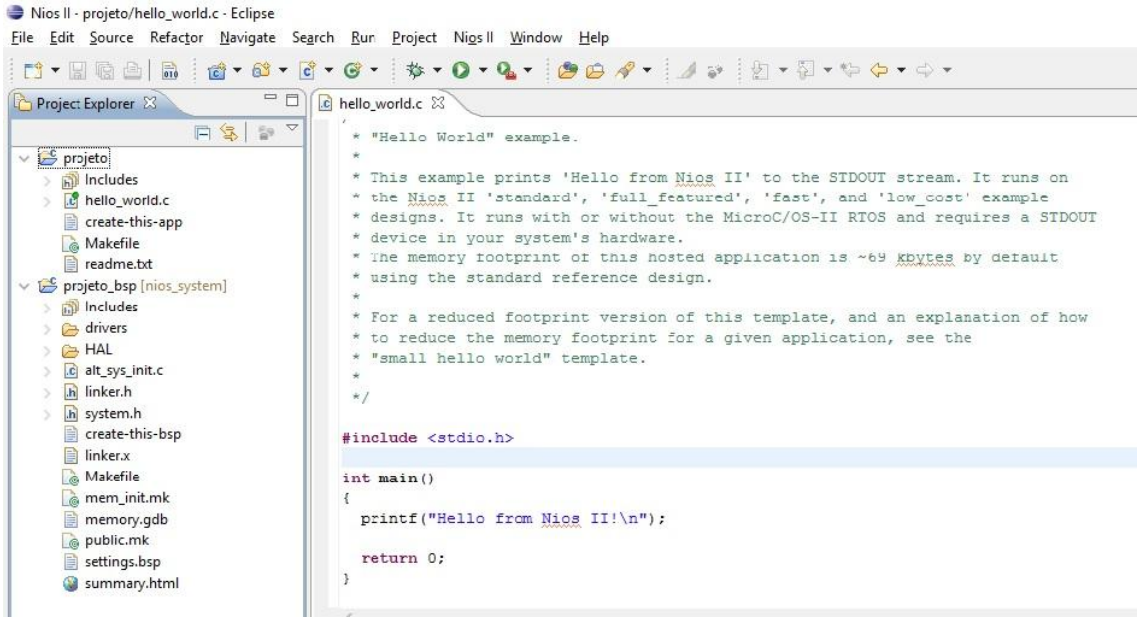
Após a criação do projeto, pode-se iniciar a criação do *software*, adicionando funcionalidades e posteriormente, a aplicação escolhida: executar o algoritmo criptográfico DES. A Figura 30 mostra como o projeto fica logo após ser criado.

Figura 29 – Criação de uma aplicação Nios II a partir de um modelo




Fonte: Produção do próprio autor

Figura 30 – Aparência inicial de um projeto recém-criado




Fonte: Produção do próprio autor

Observa-se que o projeto contém duas pastas, uma para a aplicação e a outra para o pacote BSP. Essas pastas se encontrarão na pasta *software*[4], que se encontra no mesmo lugar onde foi definido o *workspace*.

## 4.2 ADICIONANDO ARQUIVOS DE BIBLIOTECAS AO PROJETO

O *software* utilizado neste trabalho irá utilizar algumas bibliotecas desenvolvidas pelo Programa de Universidades da Altera. Estes arquivos são códigos escritos em C, que possuem um conjunto de funções para controlar o *hardware*. Para o projeto, serão utilizados os seguintes arquivos:

- altera_up_avalon_ps2.h
- altera_up_avalon_ps2_regs.h
- altera_up_ps2_keyboard.h
- altera_up_avalon_video_character_*buffer*_with_dma.h
- altera_up_avalon_video_character_*buffer*_with_dma_regs.h

Estes arquivos podem ser encontrados na pasta de propriedades intelectuais da Altera, nos seguintes diretórios:

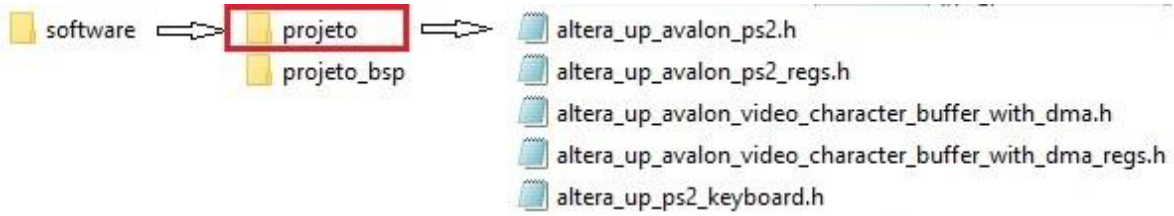
- "...\altera\13.0sp1\ip\University_Program\Input_Output\altera_up_avalon_ps2\HAL\inc"
- "...\altera\13.0sp1\ip\University_Program\Input_Output\altera_up_avalon_video_character_*buffer*_with_dma\HAL\inc"

Para adicionar estes arquivos no projeto, existem duas maneiras, ou copiar estes arquivos para o local da aplicação, ou então utilizando a própria interface do Eclipse. Os dois procedimentos serão descritos a seguir.

Primeiro método: copie todas as bibliotecas listadas acima e cole na pasta do projeto, que está dentro da pasta *software.* A Figura 31 mostra o procedimento.

[4] Esta pasta é criada automaticamente pelo Eclipse e contém as pastas do projeto.

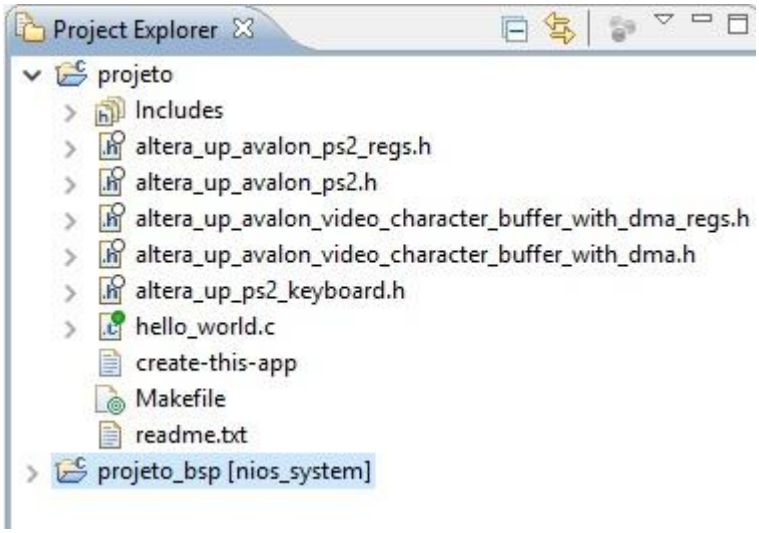
Figura 31 – Adicionando os arquivos de cabeçalho e bibliotecas ao projeto




Fonte: Produção do próprio autor

Após copiar os arquivos, clique com o botão direito no projeto e clique em *Refresh*. Os arquivos serão adicionados e a interface terá a aparência da Figura 32.

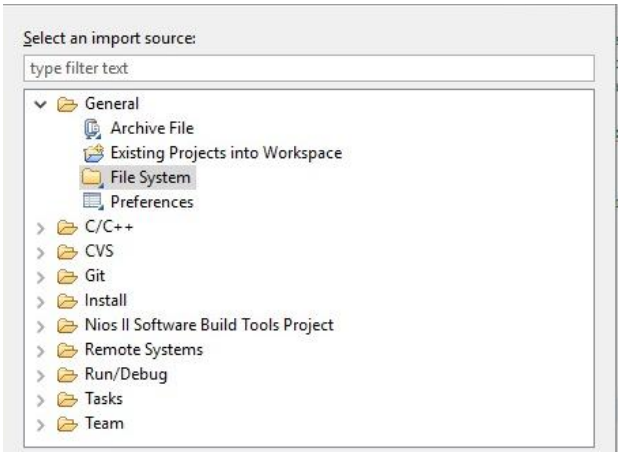
Figura 32 – Lista de arquivos do projeto




Fonte: Produção do próprio autor

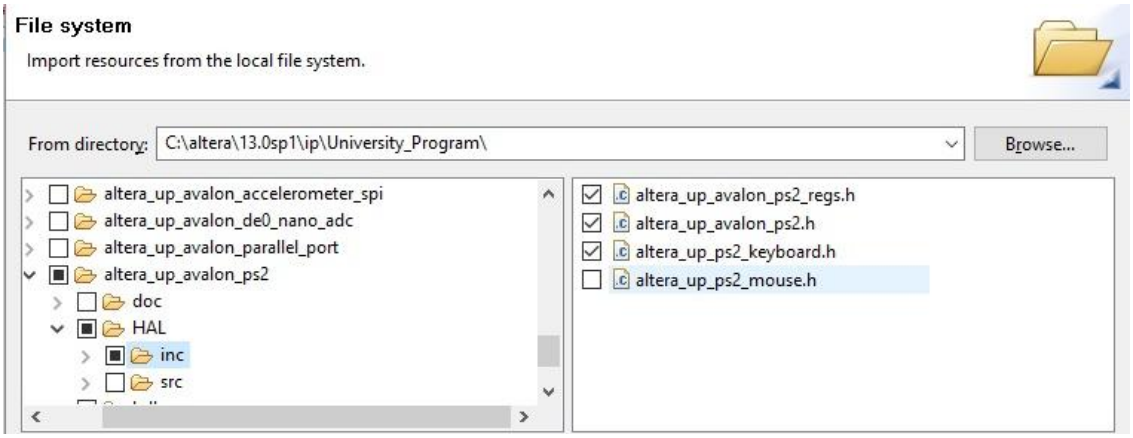
Segundo método: Clique com o botão direito no projeto, em *Project Explorer* e clique em *Import*. A Janela da Figura 33 aparecerá. Em seguida, na aba *General*, selecione *File System*. Na janela seguinte, será possível procurar os arquivos que se deseja adicionar, clique em *Browse* e adicione os arquivos. As Figura 34 eFigura 35 indicam o procedimento.

Figura 33 – Importando arquivos para o projeto



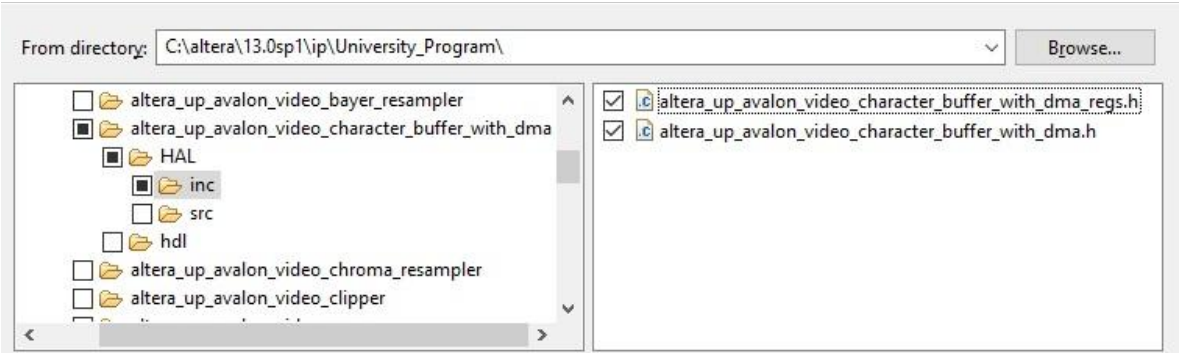

Fonte: Produção do próprio autor

Figura 34 – Localizando as pastas que contém as bibliotecas para o teclado PS/2




Fonte: Produção do próprio autor

Figura 35 – Localizando as pastas que contém as bibliotecas para o *buffer* de caracteres




Fonte: Produção do próprio autor

## 4.3 DRIVER PARA TECLADO

Encontre o arquivo “altera_up_ps2_keyboard.c” e adicione-o às pastas “HAL\src” e “Drivers\src”, que se encontram dentro da pasta do BSP. Estes arquivos são necessários para o funcionamento do teclado, pois contém funções predefinidas que realizam o controle do teclado.

## 4.4 CÓDIGO

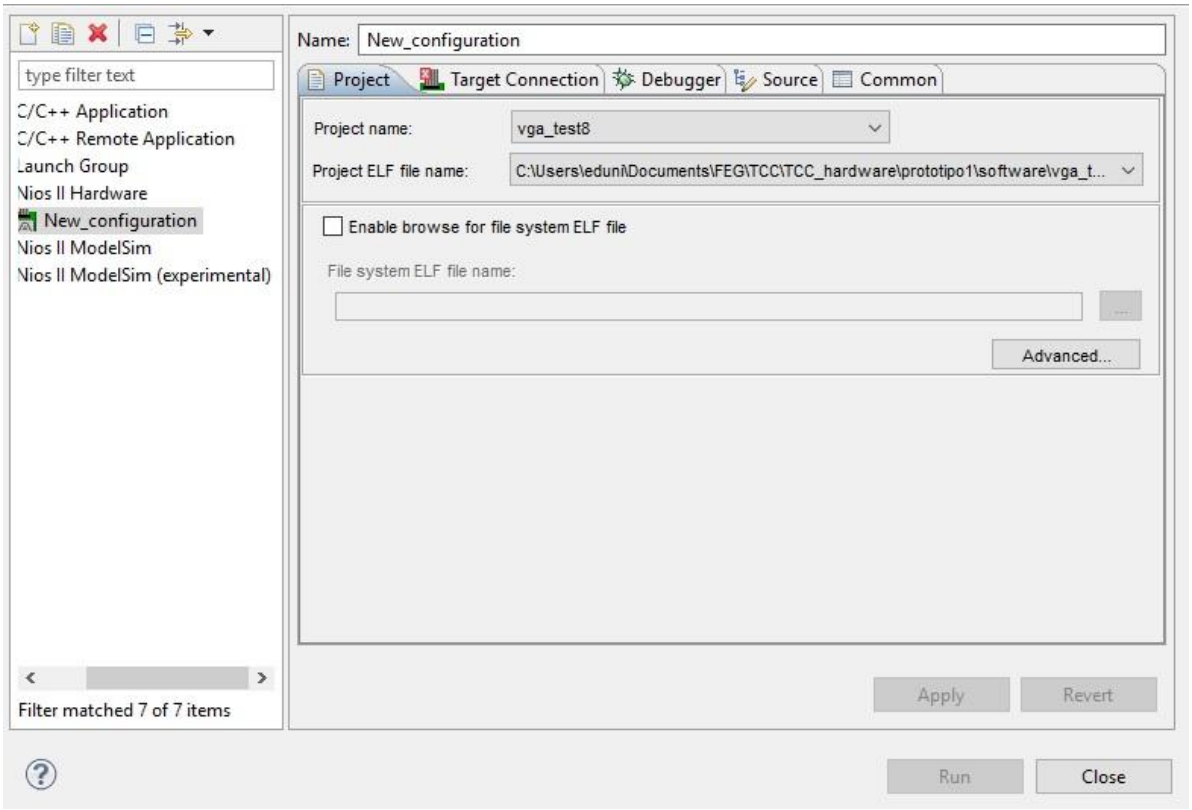
O código reúne todas as funcionalidades do projeto. O mesmo pode ser encontrado no Anexo A. Com o auxílio dos *drivers* e funções fornecidos pelo Programa de Universidades da Altera foi possível desenvolver um *software* que realiza a aplicação desejada: processar um algoritmo de criptografia DES, utilizando como dispositivo de entrada um teclado PS/2 e um monitor VGA para a saída.

## 4.5 COMPILAÇÃO E EXECUÇÃO DO CÓDIGO

Para compilar o código, selecione a opção *Build Project*, na aba *Project*, que se encontra na própria interface do Eclipse. Caso não haja erros, o compilador irá criar um arquivo com a extensão elf, responsável por efetuar a comunicação entre o *software* e a placa.

Após a compilação, será possível executar o projeto. Na aba *Run*, selecione a opção *Run Configurations*. A janela da Figura 36 irá aparecer. Selecione no menu esquerdo a opção *Nios II Hardware*. Na aba *Project*, selecione o arquivo ELF referente ao projeto. Na aba *Target Connection*, deve-se selecionar o dispositivo que irá transferir o programa para a placa, que no caso é o USB *blaster*, caso não esteja aparecendo, clique em *Refresh Connections*. Para realizar este procedimento a placa DE2-115 deve estar ligada e com o *hardware* programado na mesma, além disso, o cabo USB-*Blaster* deve estar conectado ao computador host. Ainda na mesma aba, marque as opções *Ignore mismatched system ID* e *ignore mismatched system timestamp* e certifique-se que a opção *Download ELF to selected target system* e *Start Processor* estejam marcadas. Aplique as configurações e em seguida, clique em ***Run.*** A partir deste momento, a placa fará o download do *software* e será possível conferir os resultados do projeto, que serão mostrados no capítulo a seguir.

Figura 36 – Compilação e execução do *software*



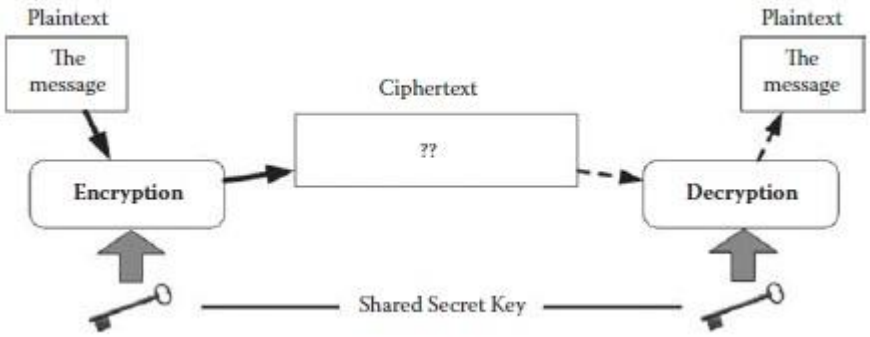

Fonte: Produção do próprio autor

O próximo capítulo apresentará a aplicação que foi desenvolvida e mostrará os resultados obtidos.

# 5 APLICAÇÃO: CRIPTOGRAFIA DES

A criptografia DES (*Digital Encryption Standard*) é um algoritmo criptográfico que tem por objetivo realizar a encriptação ou decriptação de um bloco com 64 bits de informação. Encriptar significa codificar uma informação, que é denominada de mensagem, ou *plaintext,* para que seu conteúdo seja protegido. A criptografia DES utiliza uma chave de 56 bits, que funciona como uma senha, apenas os usuários que possuem esta senha conseguirão ver o conteúdo original da mensagem. A Figura 37 mostra o processo básico de uma criptografia utilizando chave simétrica, ou seja, a mesma chave utilizada na encriptação é utilizada na decriptação.

Figura 37– Processo de criptografia utilizando chave simétrica



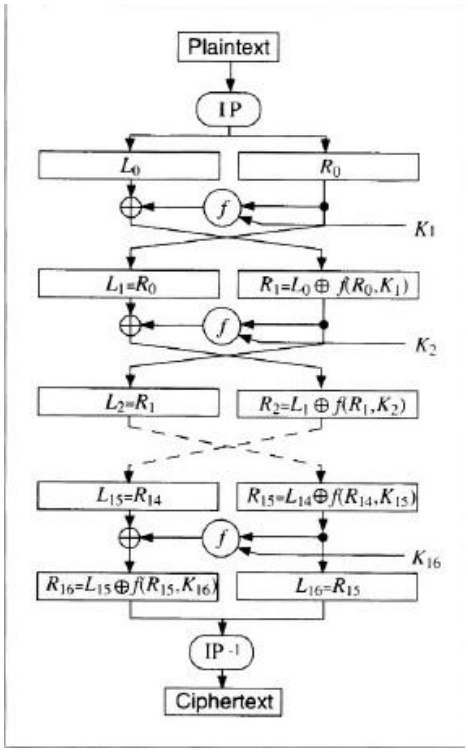

Fonte: Azad (2015)

O resultado de uma encriptação é um bloco com 64 bits, porém, codificado e ininteligível, a não ser que a chave correta seja utilizada. A mensagem encriptada também é chamada de *Ciphertext*. Para o destinatário da mensagem conseguir vê-la, o *ciphertext* passa pelo processo de decriptação, que é o inverso da encriptação. O resultado da decriptação é a mensagem original.

Algoritmos criptográficos são muito utilizados em todos os âmbitos da informática e tecnologia, pois têm a função de garantir a segurança e a confidencialidade da informação.

O fluxograma da Figura 38 mostra de maneira intuitiva, como funciona o algoritmo DES. Os detalhes e explicações de cada etapa estão no Anexo B.

Figura 38 – Fluxograma do algoritmo DES

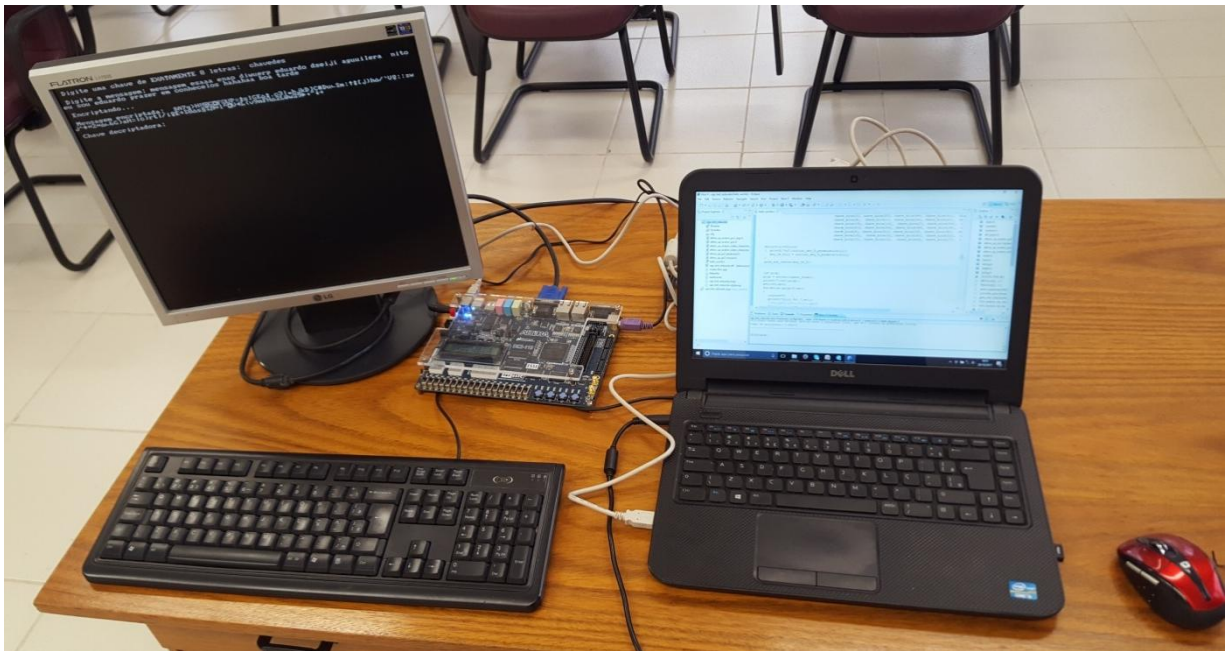



Fonte: Schneier (2015)

## 5.1 INTERFACE GRÁFICA

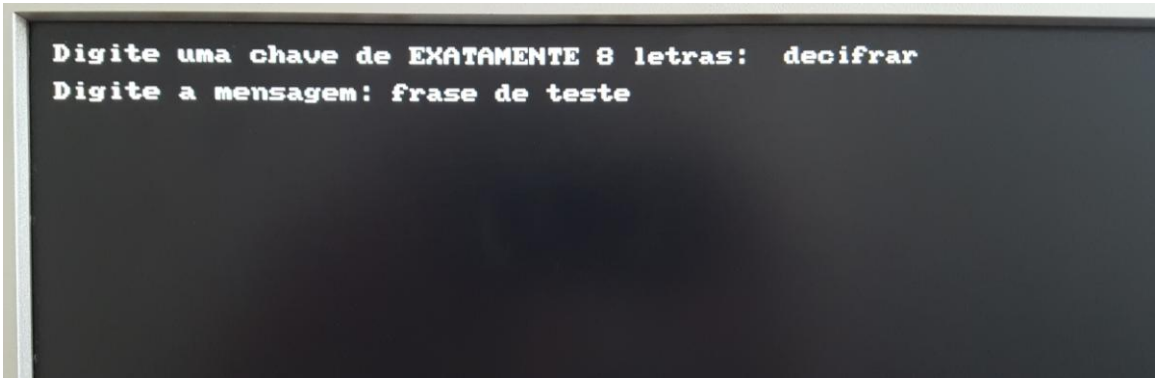
A interface utiliza um teclado PS/2 como dispositivo de entrada, na qual o usuário poderá inserir as informações necessárias para processar o algoritmo DES: a chave e a mensagem. O dispositivo de saída é um monitor VGA, que irá mostrar os resultados da aplicação. A Figura 39 mostra a configuração dos dispositivos e equipamentos utilizados, enquanto a Figura 40 mostra a aparência da interface gráfica do *software*.

Figura 39 – Bancada com os equipamentos utilizados



Fonte: Produção do próprio autor.

Figura 40 – Interface gráfica do projeto




Fonte: Produção do próprio autor

### 5.1.1 Funções VGA_text e VGA_box

Essas funções podem ser encontradas em alguns dos tutoriais de utilização dos controladores de vídeo do Programa de Universidades da Altera[3]. A função VGA_text imprime uma mensagem no monitor e possui como parâmetros as coordenadas do monitor e um ponteiro que aponta para uma string. Ao longo do código, esta função é utilizada algumas vezes, pois é através dela que o monitor mostra os resultados.

A função VGA_box desenha um retângulo preenchido e possui como parâmetros as dimensões do retângulo, cor e coordenadas. Esta função é utilizada para alterar o fundo da tela para a cor preta.

## 5.2 CÓDIGO E FUNÇÕES

O código deve realizar o processo de encriptografia e decriptografia de uma mensagem inserida pelo usuário, realizando a técnica de *Padding* e usar o *Electronic Code Book* (ECB) como modo de operação, para encriptar e decriptar múltiplos blocos de informação. O código foi dividido em algumas funções, tais como as substituições SBOX, permutações PBOX e a geração das subchaves, pois as mesmas são utilizadas mais de uma vez ao longo do código. Na função *main*, estão contidas as demais instruções; sua estrutura pode ser dividida em duas partes principais: encriptação e decriptação.

O código para encriptação é a primeira parte do programa. Após a definição da chave e mensagem, é necessário convertê-las para blocos de informações em binário, que serão armazenados em vetores. A DES, por definição, é feita em blocos de 64 bits, portanto, para mensagens onde o comprimento é maior que 8 bytes, é necessário dividir em mais de um bloco. A Equação 1 mostra a quantidade de blocos que são necessários para encriptar uma mensagem.

$$\text{Numero de blocos} = [(\text{nº de letras})*8]/64 + [1(\text{caso resto} \neq 0)] \quad (1)$$

Por exemplo, supondo que a mensagem inserida pelo usuário tenha sido a palavra “mensagem”. Ela possui 8 letras, o que totalizam 64 bits, exatamente. De acordo com a equação, o resultado seria 1, e como o resto é zero, não se deve somar 1. Outro exemplo seria se o usuário tivesse digitado a mensagem “estou fazendo um teste”; neste caso, há um total de 22 caracteres, contando os espaços, que totalizam 176 bits. Seguindo a equação, o resultado seria 3, pois o resto é diferente de zero. É conveniente lembrar que para o terceiro bloco, seria necessário adicionar dois caracteres, geralmente teclas de espaço ou zeros, para obter um bloco com 64 bits. A técnica de preenchimento automático é denominada de *Padding* e é utilizada para preservar o conteúdo da mensagem. O método de divisão em blocos iguais de 64 bits é o ECB, considerado o modo de operação mais simples da criptografia DES.

Após converter as duas entradas de string para vetores de números binários, o processo de criptografia irá iniciar. O código realiza todas as etapas do algoritmo, sendo que as principais foram divididas em funções. O *Ciphertext* obtido pela encriptação servirá como entrada para a decriptação. O sucesso ou falha da decriptação irá depender da chave de decriptação que será pedida ao usuário. Como o método de criptografia utilizado é o DES com chave simétrica, a mesma chave utilizada na encriptação é usada na decriptação. O algoritmo utilizado é o mesmo, a única diferença é a ordem das sub-chaves, que são utilizadas no sentido inverso, ou seja, a sub-chave 1, gerada na encriptação, será na verdade, utilizada no último *round* da decriptação, assim sucessivamente.

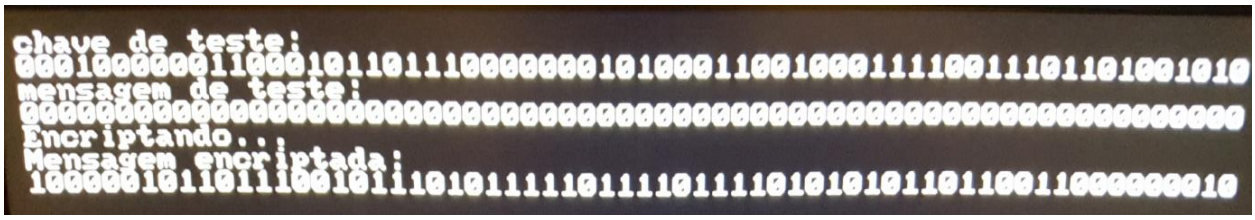
A chave irá definir o resultado de decriptação. Caso a chave inserida seja diferente. O processo de decriptação irá falhar, e o resultado será algo diferente da mensagem.

## 5.3 TESTANDO A VALIDADE DO SISTEMA

O resultado da criptografia DES deve estar de acordo com as especificações regulamentadas pelo órgão americano *National Bureau of Standard*s (NBS). Todos os requisitos de *hardware* e *software*, bem como requisitos para implementação do algoritmo estão descritos nos documentos fornecidos pela NBS. O documento que regulamenta o algoritmo DES é o FIPS PUB 46-3. Para fins de validação, foi criado o Programa de Validação para Módulos Criptográficos, que contém vetores de testes para validar o algoritmo e confirmar sua eficácia.

De acordo com o Apêndice A do documento publicado pelo NIST (*National Institute of Standards and Technology*), para uma chave definida pelo vetor 10316E028C8F3B4A e um *plaintext* definido por 0000000000000000, tem-se como *Ciphertext* o vetor 82DCBAFBDEAB6602. Como o projeto foi designado para encriptar informações no formato de texto foi necessário modificar o código da aplicação para realizar este teste, afinal, alguns dos números não correspondem a caracteres ASCII válidos. Pode-se observar na Figura 41 que para os vetores acima, obteve-se o mesmo *Ciphertext*, porém, em formato binário; conclui-se então que o algoritmo utilizado no *software* está correto. O Anexo D contém a página do documento com o vetor de testes.

Figura 41 – teste de validação do algoritmo DES




Fonte: Produção do próprio autor

## 5.4 DESEMPENHO DO SISTEMA

O sistema realiza a criptografia DES por *software*, ou seja, o algoritmo é executado de maneira sequencial através do processador. O desempenho dependerá do poder de processamento do processador, muitas vezes determinado pela frequência de *clock.*

O *software* levou aproximadamente 6 segundos para encriptar 1 bloco, utilizando-se a versão *economy* do processador Nios II. O tempo de encriptação é relativamente alto, comparado com os produtos disponíveis no mercado, além disso, implementações que utilizam *hardware* dedicado em linguagem VHDL são naturalmente mais rápidos, por utilizarem o paralelismo do FPGA.

Foram realizados testes utilizando o mesmo *software*, mas trocando-se o processador utilizado no *hardware*. A Tabela 1 abaixo mostra uma relação de tempo de encriptação por quantidade de blocos.

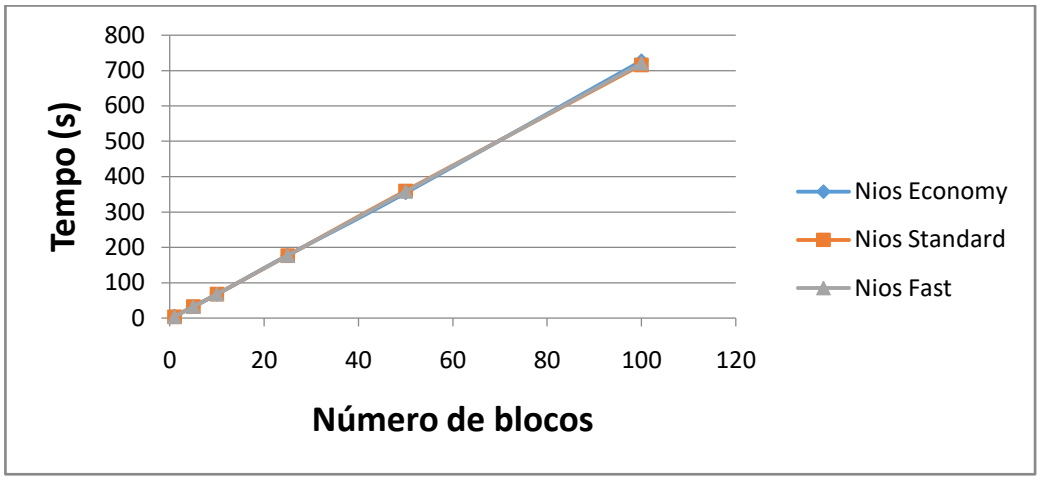
Tabela 1 – Tempos de encriptação e decriptação por quantidade de blocos

| Processador | Processo | Número de blocos | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 5 | 10 | 25 | 50 | 100 |
| Nios *Economy* | Encriptação | 5,72s | 32,7s | 1m 07 s | 2m 58s | 5m 54s | 12m 07s |
| | Decriptação | 6,31s | 31,41s | 1m 05s | 2m 47s | 5m 33s | 11m 04s |
| Nios *Standard* | Encriptação | 3,87s | 33,17s | 1m 08s | 2m 57s | 6m | 11m 56s |
| | Decriptação | 4,7s | 31s | 11m 05s | 2m 46s | 5m 30s | 11m 09s |
| Nios *Fast* | Encriptação | 3,67s | 32,8s | 1m 08s | 2m 57s | 5m 58s | 12m |
| | Decriptação | 4,25s | 31s | 1m 05s | 2m 46s | 5m 33s | 11m 09s |

Fonte: Produção do próprio autor

Pode-se observar que a versão do processador Nios II utilizada não influenciou no tempo de processamento do algoritmo, haja visto que os tempos estão muito próximos. Um resultado curioso é que a versão *Economy* do Nios obteve resultados iguais, ou até melhores, à medida que a quantidade de blocos aumentou. Os tempos de decriptação, no geral, foram menores que os de encriptação. A Figura 42 mostra o comportamento do sistema para o processo de encriptação.

Figura 42 – Tempos de encriptação de acordo com o número de blocos




Fonte: Produção do próprio autor

Observa-se que o gráfico possui um comportamento linear e proporcional. Para o sistema em questão, a versão do processador não alterou significativamente o desempenho, portanto, convém utilizar a versão *Economy*, pois utiliza menos recursos e elementos lógicos.

## 5.5 REDUÇÃO DO CÓDIGO

No item anterior, foi possível concluir que a versão do processador Nios utilizada não melhorou o desempenho do sistema. Isto indica que o desempenho dependerá de outros fatores, como por exemplo, a qualidade do código, o tamanho e sintaxe. Retirando as linhas de código que estão associadas ao terminal de saída do Nios, foi realizado um novo teste, para verificar se um código mais enxuto melhora a capacidade do sistema. A Tabela 2 abaixo contém os resultados.

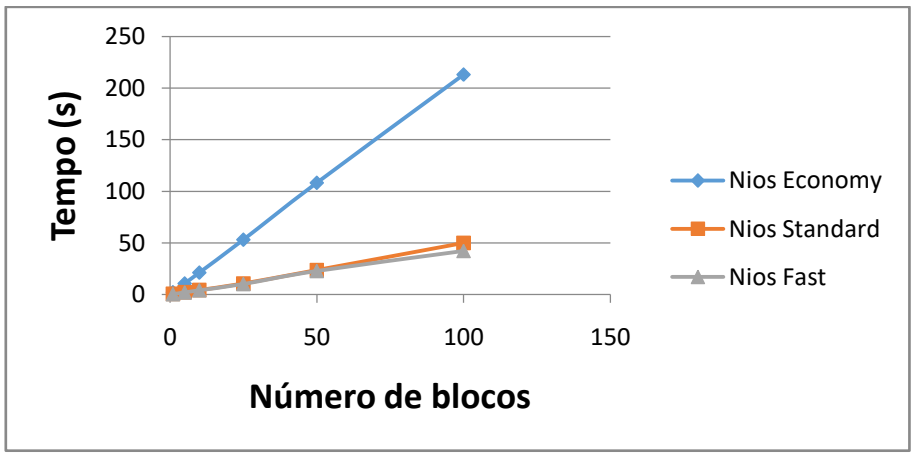
Tabela 2 – Tempos de encriptação e decriptação por quantidade de blocos

| Processador | Processo | Número de blocos | | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | 5 | 10 | 25 | 50 | 100 |
| Nios Economy | Encriptação | 2,18s | 10,67s | 21,21s | 53,09s | 108,18s | 213s |
| | Decriptação | 2,30s | 11,13s | 22,4s | 53,76s | 109,4s | 213,2s |
| Nios *Standard* | Encriptação | 0,48s | 2,06s | 4,24s | 10,55s | 23,63s | 50s |
| | Decriptação | 0,606s | 2,18s | 4,36s | 10,667s | 21,2s | 50,2s |
| Nios Fast | Encriptação | 0,363s | 2,06s | 4,12s | 10,18s | 22,79s | 48,24s |
| | Decriptação | 0,73s | 2,06s | 4,12s | 10,42s | 20,48s | 42,18s |

Fonte: Produção do próprio autor

Desta vez, os resultados foram diferentes. Utilizando-se o código reduzido, a versão *Economy* do Nios obteve resultados piores que as suas versões mais sofisticadas. A versão *Fast* apresentou o melhor desempenho, enquanto a versão *Standard* ficou em segundo lugar. É possível notar que os tempos de execução do algoritmo diminuíram drasticamente quando as linhas de código desnecessárias foram eliminadas. Funções que utilizam o terminal, tais como *printf* e *scanf* retardam o tempo de resposta do sistema. A Figura 43 abaixo mostra a nova relação.

Figura 43 – Tempos de encriptação com o código reduzido



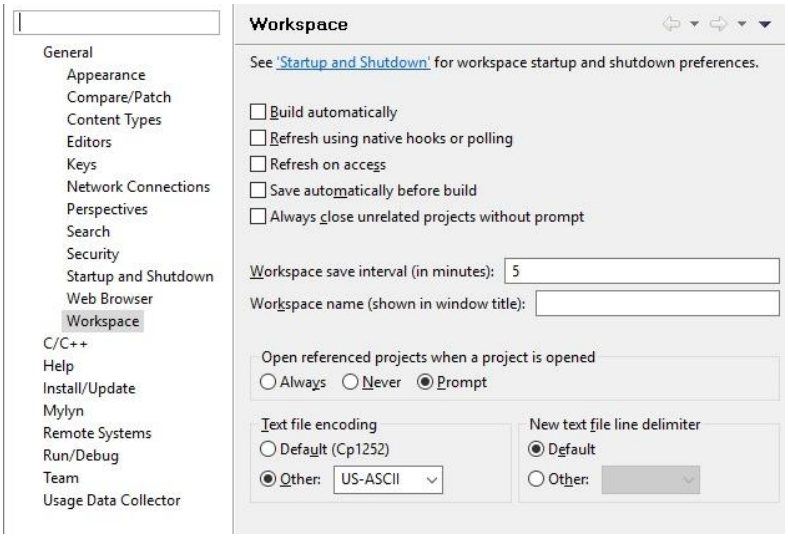

Fonte: Produção do próprio autor

## 5.6 PADRÃO DE CODIFICAÇÃO DO TEXTO

### 5.6.1 Alterando o sistema de codificação para o padrão ASCII

Na própria interface do Eclipse, selecione a aba *Window>Preferences*. Em seguida, dentro da aba *General*, selecione *Workspace*. Lá será possível configurar o padrão de codificação do texto a ser utilizado pelo compilador. Em *text file encoding*, marque a opção *Other* e selecione o padrão US-ASCII, como mostra a Figura 44.

Figura 44 – Configurando a codificação de texto para o padrão ASCII




Fonte: Produção do próprio autor

O sistema utilizará o padrão ASCII para codificação de textos, pois é o mais compatível com o *hardware*. O componente responsável por imprimir um caractere no monitor é o *Buffer* de caracteres e um de seus arquivos apresenta um mapeamento de 128 possíveis caracteres[5]. Os caracteres que serão mostrados no monitor VGA estão associados a este mapeamento. O Anexo C contém a tabela com todos os caracteres mapeados pelo *buffer*.

[5] Arquivo: altera_up_video_char_mode_rom_128.mif

### 5.6.2 Eliminação do oitavo bit

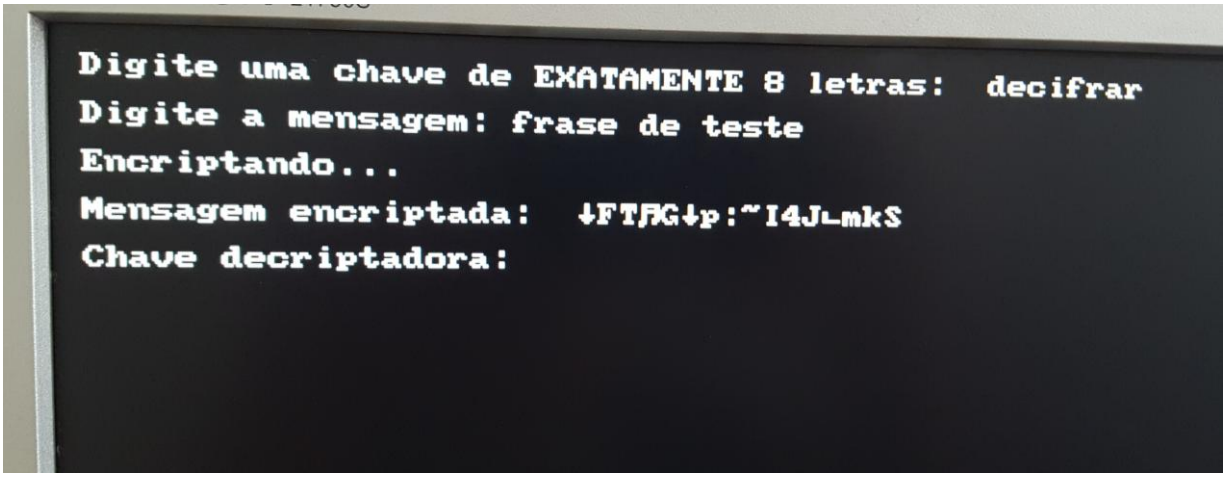
Um caractere ascii possui 7 bits, entretanto, a aplicação envolve a manipulação de blocos com 8 bits por caractere. Este bit adicional altera os resultados da criptografia, pois a quantidade de caracteres deixa de ser 128 e passa a ser 255. Por essa razão, todos os números maiores que 127, eliminam o bit mais significativo, para que o padrão ascii seja seguido. Por exemplo, supondo que um dos caracteres resultantes da encriptação seja o número 231, ele é maior que 128, o que indica que seu oitavo bit vale 1 e não está no padrão ascii. O próprio compilador elimina este bit, o que altera o valor de 231 para 103(231-128), que é representado pelo caractere "g" na tabela ascii do *buffer* de caracteres.

## 5.7 EXEMPLO DE EXECUÇÃO

Este tópico irá expor um exemplo prático do funcionamento do sistema, bem como os resultados obtidos.

Supondo que o usuário escolheu a frase "frase de teste" como a mensagem e que a chave foi definida pela palavra "decifrar". O resultado da encriptação será composto de dois blocos de 64 bits, sendo eles: 10011001 11000110 01010100 00001110 11000111 00011001 11110000 10111010 e 11111110 11001001 00110100 11001010 00011100 01101101 11101011 01010011. Convertendo os bytes para números decimais, tem-se a sequencia 153, 198, 84, 14, 199,25, 240, 186,254,201,52,202,28,109,235 e 83, da esquerda para a direita. Os números maiores que 128 serão convertidos para um caractere ascii, de acordo com o item anterior. Tem-se, portanto, uma nova sequencia, que será: 25,70,84,14,71,25,112,58,126,73,52,74,28,109,107 e 83. Os caracteres equivalentes estão mapeados na Tabela 3, que pode ser encontrada no Anexo C. Após consultar a tabela, observa-se que o *ciphertext* obtido é dado pela palavra "↓FT♫G↓p:~I4J └mkS" . A Figura 45 mostra o resultado obtido no monitor VGA.

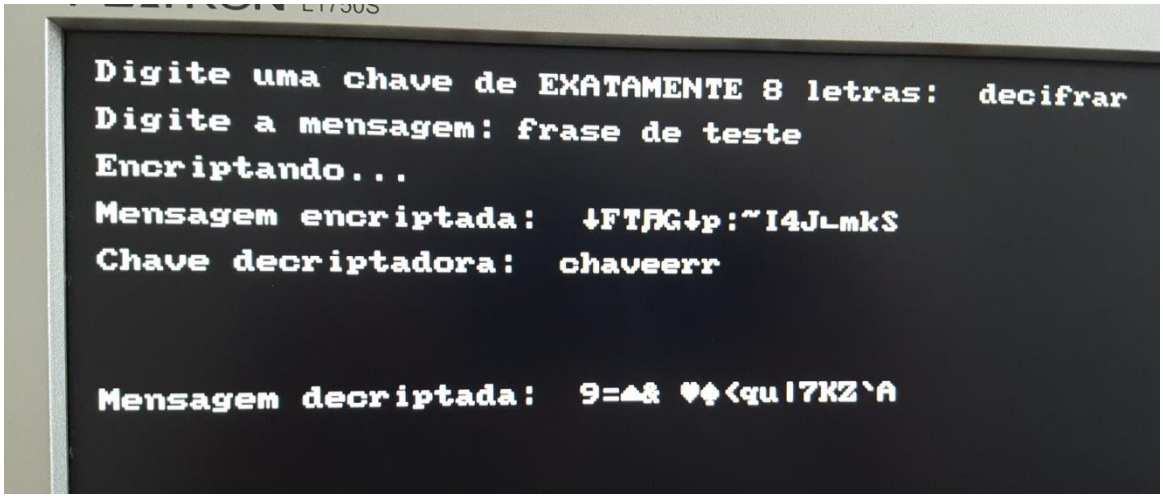
Figura 45 – Resultado da encriptação no monitor VGA




Fonte: Produção do próprio autor

Para o processo de decriptação o algoritmo é o mesmo. Será pedida a chave para o usuário, que terá algumas tentativas para decriptar a mensagem, antes do fim da execução do programa. Para uma primeira tentativa, foi inserida uma chave errada. Como consequência, o resultado da decriptação será algo distante da mensagem original. A Figura 46 mostra este procedimento.

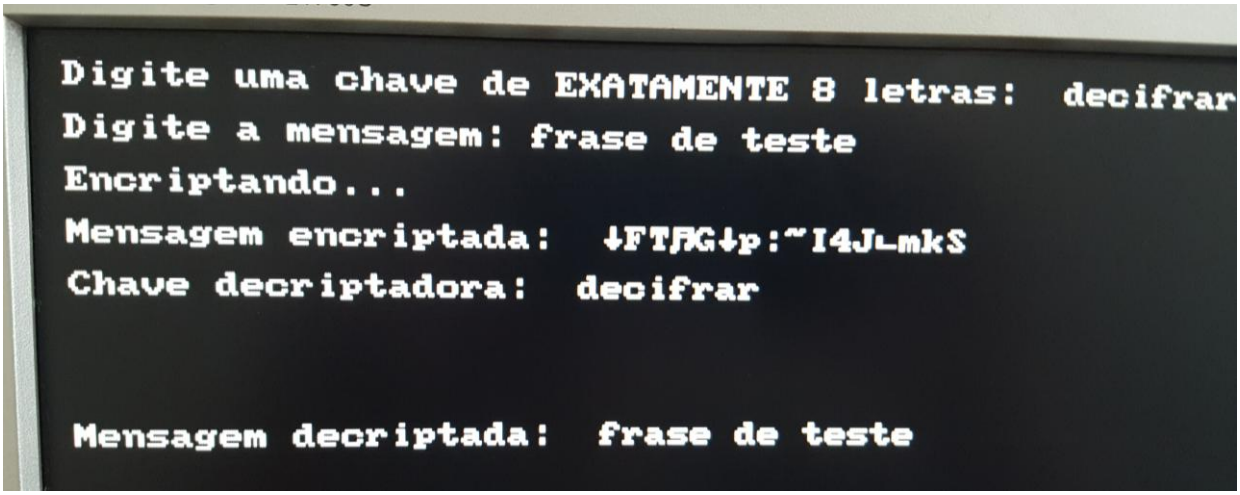
Figura 46 – Decriptação utilizando a chave errada




Fonte: Produção do próprio autor

Na segunda tentativa, foi inserida a chave correta. Como resultado obteve-se exatamente o conteúdo da mensagem, comprovando que o algoritmo está correto. A Figura 47 mostra o resultado.

Figura 47 – Decriptação utilizando a chave correta




Fonte: Produção do próprio autor.

# 6 CONCLUSÃO

Através da utilização de ferramentas adequadas, foi possível realizar todas as etapas de projeto de um sistema embarcado, utilizando como plataforma, o FPGA. O *kit* de desenvolvimento DE2-115 apresentou todos os recursos necessários para a execução do projeto. As ferramentas de design Quartus II 13.0SP1 e Qsys facilitaram a criação de um *hardware* que em sua essência, não é trivial e exige conhecimentos avançados em eletrônica e linguagem VHDL. O Programa de Universidades da Altera também contribuiu para a realização do projeto, pois foram utilizados controladores de vídeo e teclado pré-existentes. É conveniente lembrar que alguns recursos utilizados neste trabalho são propriedades intelectuais provenientes da Altera; o desenvolvimento de um produto com fins comerciais deveria cumprir com as leis de direitos autorais.

O ambiente integrado de desenvolvimento em Eclipse ofereceu todas as ferramentas para o desenvolvimento do *software*, que realiza um algoritmo criptográfico DES. O código realizou com sucesso a criptografia de uma mensagem comum, digitada no teclado, para que os resultados aparecessem no monitor VGA. A criptografia DES apresentou resultados interessantes com relação ao desempenho; para o código completo, que inclui funções de entrada e saída no terminal, a versão do processador Nios II não fez diferença no desempenho. Ao reduzir o código da aplicação, pôde-se perceber que a versão do Nios que obteve melhor desempenho foi a *fast*. Uma conclusão plausível e natural seria que o tamanho do código, suas funcionalidades, bem como a escolha do processador utilizado no sistema são fatores decisivos para um bom desempenho.

Por fim, foi possível perceber que a concepção e execução do projeto de um Sistema Embarcado requerem uma grande interdisciplinaridade e agregam conhecimentos em diversas áreas, além de exigir habilidades em programação. A possibilidade de reprogramar um circuito digital, certamente facilita a criação de sistemas digitais, reduzindo custos e dispensando tempo de montagem.

# REFERÊNCIAS

ALTERA CORPORATION. **Media computer system for the altera DE2-115 board**. 2013. 50 p.

ALTERA CORPORATION. **Nios II software developers handbook**. San Jose, 2010. 510 p.

BROWN, S.; VRANESIC, Z. **Fundamentals of digital logic with VHDL design**. 3rd ed. New York: McGraw-Hill, 2009. 939 p.

CARBONARA, L. P. **Estudo dos modos de configuração de FPGA.** 2016. 81 f. Trabalho de Graduação (Graduação em Engenharia Elétrica) – Faculdade de Engenharia do Campus de Guaratinguetá, Universidade Estadual Paulista, Guaratinguetá, 2016.

CHU, P. P. **Embedded SOPC design with NIOS II processor and VHDL examples.** Hoboken: John Wiley & Sons, 2011.703 p.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY.**Modes of operation validation system (MOVS):** requirements and procedures**.** Gaithersburg, 1998. 152 p.

NAVABI, Z. **Embedded core design with FPGA.** New York: McGraw-Hill, 2007. 440 p.

NOERGAARD, T. **Embedded systems architecture:** a comprehensive guide for engineers and programmers. Burlington: Elsevier Inc, 2005. 640 p. (Embedded Technology Series).

SASS, R.; SCHMIDT, A. G. **Embedded systems design with platform FPGAs.** Burlington: Elsevier Inc, 2010. 389 p.

SCHNEIER, B. **Applied cryptography.** Indianapolis: John Wiley & Sons, 2015. 754 p.

WIKIPÉDIA. **Microcontrolador**. 2017. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Microcontrolador&oldid=49055192>. Acesso em: 16 jun. 2017.

WILSON, P. R. **Design recipes for FPGAs using verilog and VHDL.** 2nd ed. San Diego: Elsevier Inc, 2016. 369 p.

## BIBLIOGRAFIA CONSULTADA

ALTERA CORPORATION. **PS/2 Core for altera DE2/DE1 boards.** 2006. 9 p

ALTERA CORPORATION. **Video IP cores for altera DE-series boards.** 2013. 46 p.

AZAD, S.; PATHAN, A.K. **Practical cryptography:** algorithms and implementations using C++. Boca Raton: CRC Press, 2015. 333 p.

JOAN, B. **Difference between FPGA and microcontroller.** 2009. Disponível em: < http://www.differencebetween.net/technology/difference-between-fpga-and-microcontroller/>. Acesso em: 20/09/2017.

TERASIC TECHNOLOGIES. **DE2-115 user manual**, 2012a. 117p. 1 CD-ROM.

TERASIC TECHNOLOGIES. **My first FPGA for altera DE2-115 board**. 2012b. 47p. 1 CD-ROM.

TERASIC TECHNOLOGIES. **My first nios II for altera DE2-115 board.** 2012c. 78p. 1 CD-ROM.

WELSCHENBACH, M. **Cryptography in C and C++.** New York: Apress, 2001. 432 p.

# ANEXO A – Código do *software*

```
//    bibliotecas e arquivos de cabeçalho
#include "stdio.h"
#include "unistd.h"
#include "system.h"
#include "alt_types.h"
#include "altera_up_avalon_ps2.h"
#include "altera_up_ps2_keyboard.h"
#include "altera_up_avalon_ps2_regs.h"
#include "altera_up_avalon_video_character_buffer_with_dma.h"
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <string.h>
//     Protótipos das funções
int * SBOX(int[48]);
int * PBOX(int[32]);
int * perm_expansao(int[32]);
int * converte_para_binario(char[8]);
void gera_sub_chaves(int[56]);
void VGA_text (int, int, char *);
void VGA_box (int, int, int, int, short);
//     Variáveis Globais
int K[16][48]; int x,y,z=0;
int contador_funcao=0;
unsigned char mensagem[1000]={};
unsigned char chave[8]={};
unsigned char chave_D[8]={};
unsigned char cipher_final[1000]={},cipher_temp[1000]={};
unsigned char mensagem_ascii[8]={},mensagem_string[20]={};
//     função read_keyboard
void read_keyboard(alt_up_ps2_dev *ps2_kb, alt_up_char_buffer_dev  *vga)
{
   KB_CODE_TYPE decode_mode;
   alt_u8 buffer ="00000000"; int count=0;
   char ascii;
   char *inputStr;
   alt_up_char_buffer_init(vga);
   if(contador_funcao == 0)
   {
      x=43;y=1;count=0;
      while(1){
     if(buffer == 90 && count== 8) break;
        if (decode_scancode(ps2_kb, &decode_mode, &buffer, &ascii)==0)
        {
          if (decode_mode == KB_ASCII_MAKE_CODE )
          {
            translate_make_code(decode_mode, buffer, inputStr);
            if(buffer == 102) // make code equivalente a tecla backspace
            {
              count--; //verifica se a tecla pressionada foi a barra de espaço
              x--;
              alt_up_char_buffer_string(vga, inputStr, x,y);
              printf("conteudo do buffer:%d ",buffer);
```

```
            printf("conteudo da variavel ascii:%c ",ascii);
            printf("contador: %d ",count);
            chave[count] = ascii;
            printf("conteudo da chave: %s\n",chave);
          }
          else
          {
            alt_up_char_buffer_string(vga, inputStr, x,y);
            printf("conteudo do buffer:%d ",buffer);
            printf("conteudo da variavel ascii:%c ",ascii);
            chave[count] = ascii; count++;
            printf("contador: %d ",count);
            printf("conteudo da chave: %s\n",chave);x++;
          }
          usleep(110000);
        }
        if (x > 78)
        {y++;
         x = 0;}
        if (y > 58)
        {y = x = 0;}
      }
    }

    int size=0;
    if(chave[count] == ' ')
    size=strlen(chave)-1;
    else size= strlen (chave);
    printf("\nChave: %s e tamanho: %d contador: %d",chave,size,count);

}// fim if contador_funcao

if(contador_funcao == 1) // mensagem como entrada
{
    x=20;y=3;count=0;
    while(buffer != 90)
    {
     if (decode_scancode(ps2_kb, &decode_mode, &buffer, &ascii)==0)
      {
         if (decode_mode == KB_ASCII_MAKE_CODE )
         {
           translate_make_code(decode_mode, buffer, inputStr);
           if(buffer == 102) // make code equivalente a tecla backspace
           {
             count--; //verifica se a tecla pressionada foi a barra de espaço
             x--;
             alt_up_char_buffer_string(vga, inputStr, x,y);
             printf("conteudo do buffer:%d ",buffer);
             printf("conteudo da variavel ascii:%c ",ascii);
             printf("contador: %d ",count);
             mensagem[count] = ascii;
             printf("conteudo da mensagem: %s\n",mensagem);
           }
           else
           {
             //else alt_up_char_buffer_string(vga, inputStr, x,y);
             alt_up_char_buffer_string(vga, inputStr, x,y);
```

```
            printf("conteudo do buffer:%d ",buffer);
            printf("conteudo da variavel ascii:%c ",ascii);
            mensagem[count] = ascii; count++;
            printf("contador: %d ",count);
            printf("conteudo da mensagem: %s\n",mensagem);x++;
          }
          usleep(110000);
        }
        if (x > 78)
        { y++;
          x = 0;z++;}
        if (y > 58)
        {y = x = 0;}
      }// fim if decode scan
    }//fim while
    int size=0;
    if(mensagem[count] == ' ')
       size=strlen(mensagem)-1;
    else size= strlen (mensagem);
    mensagem[count] ='\0';
    printf("\nMensagem: %s e tamanho: %d contador: %d",mensagem,size,count);

}//fim if contador_funcao
buffer = 0;
if(contador_funcao >= 2)
{
    y=9+z;x=22;count=0;

    while(1){
            if(buffer == 90 && count== 8) break;
      if (decode_scancode(ps2_kb, &decode_mode, &buffer, &ascii)==0)
      {
        if (decode_mode == KB_ASCII_MAKE_CODE )
        {
       translate_make_code(decode_mode, buffer, inputStr);

       if(buffer == 102) // make code equivalente a tecla backspace
       { count--; //verifica se a tecla pressionada foi a barra de espaço
         x--;
         alt_up_char_buffer_string(vga, inputStr, x,y);
         printf("conteudo do buffer:%d ",buffer);
         printf("conteudo da variavel ascii:%c ",ascii);
         printf("contador: %d ",count);
         chave_D[count] = ascii;
         printf("conteudo da chave: %s\n",chave_D);
       }
       else
       {
         alt_up_char_buffer_string(vga, inputStr, x,y);
         printf("conteudo do buffer:%d ",buffer);
         printf("conteudo da variavel ascii:%c ",ascii);
         chave_D[count] = ascii; count++;
         printf("contador: %d ",count);
         printf("conteudo da chaveD: %s\n",chave_D);x++;
       }
       usleep(110000);
        }
```

```
            if (x > 78)
            {y++;
             x = 0;}
            if (y > 58)
            {y = x = 0;}
          }
        }//fim while
        int size=0;
        if(chave_D[count] == ' ')
          size=strlen(chave_D)-1;
        else size= strlen (chave_D);
        printf("\nChaveD: %s e tamanho: %d contador: %d",chave_D,size,count);
    }// fim contador_funcao

}// fim funcao

int main(void)
{
//            declaracao das variaveis do algoritmo DES
int len=0,i,round=0,mc=0,md=0;
int compression_permutation_key[48]={},expansion_permutation[48]={};
int key_xor_right[48]={};
int key56_after_shift[56]={};
int pbox_permutation[32]={};
int xor_left_pbox[32]={};
int after_substitution[32]={};
VGA_box (0, 0, 400, 1000, 0x0);
alt_up_ps2_dev *ps2 = alt_up_ps2_open_dev("/dev/PS2_port"); //abre a PS/2
alt_up_ps2_init(ps2);
alt_up_ps2_clear_fifo(ps2);
alt_up_char_buffer_dev  *vga =
alt_up_char_buffer_open_dev("/dev/VGA_Char_buffer");//abre dispoitivo vga
alt_up_char_buffer_clear(vga); //apaga caracter anterior no monitor

char text0[50] = "Digite uma chave de EXATAMENTE 8 letras:\0";
VGA_text(1,1,text0);
if (ps2 ->device_type == PS2_KEYBOARD)//Checa se PS/2 é um teclado
   read_keyboard(ps2,vga);
else
{
   printf("O dispositivo conectado à entrada PS/2 nao eh um teclado!");
   return -1;
}
contador_funcao++;
//   Definição da Chave de 64 bits e suas 16 sub-chaves
int  chave_bits[64]={};
/*****bloco que converte a chave para um bloco de de 64 bits em binario******/
printf( "'%s' em binario:    " , chave );
int power,counter=0;
unsigned int c;
int *ch;
ch = converte_para_binario(chave);
for(i=0;i<=63;i++)
   chave_bits[i] = (int)(*(ch+i));
/*Na criptografia DES, a chave possui 56 bits, para isso, a chave inicial, que
possui geralmente 64 bits, passa por uma permutação inicial*/
int key_56[56]={};
```

```
int initial_key_permutation[56] =
{chave_bits[56],chave_bits[48],chave_bits[40],chave_bits[32],chave_bits[24],
chave_bits[16],chave_bits[8],chave_bits[0],chave_bits[57],chave_bits[49],
chave_bits[41],chave_bits[33],chave_bits[25],chave_bits[17],chave_bits[9],
chave_bits[1], chave_bits[58],chave_bits[50],chave_bits[42],chave_bits[34],
chave_bits[26],chave_bits[18],chave_bits[10],chave_bits[2],chave_bits[59],
chave_bits[51],chave_bits[43],chave_bits[35],chave_bits[62],chave_bits[54],
chave_bits[46],chave_bits[38],chave_bits[30],chave_bits[22],chave_bits[14],
chave_bits[6],chave_bits[61],chave_bits[53],chave_bits[45],chave_bits[37],
chave_bits[29],chave_bits[21],chave_bits[13],chave_bits[5],chave_bits[60],
chave_bits[52],chave_bits[44],chave_bits[36],chave_bits[28],chave_bits[20],
chave_bits[12],chave_bits[4],chave_bits[27],chave_bits[19],chave_bits[11],
chave_bits[3]};
printf("             KEY1:  ");
for(i=0;i<=55;i++)
{
   printf("%d",initial_key_permutation[i]);
   key_56[i] = initial_key_permutation[i];
}
gera_sub_chaves(key_56);

char text1[40] = "Digite a mensagem:";
VGA_text (1, (y+2), text1);
alt_up_ps2_init(ps2);
alt_up_ps2_clear_fifo(ps2);
if (ps2 ->device_type == PS2_KEYBOARD)//Checa se dispositivo é um teclado
  read_keyboard(ps2,vga);
else
{
  printf("O dispositivo conectado à entrada PS/2 nao eh um teclado!");
  return -1;
}
contador_funcao++;
char text2[40] = "Encriptando...\0";
VGA_text (1, (y+2), text2);
//            Definição da mensagem
int resto=0,w=0,nb=0,m=0;
int sh;
int plaintext[64]={}; // é um bloco de 64 bits
i = strlen(mensagem);
resto = i%8;
if(resto != 0)
     for(w=0;w<8-resto;w++,i++)
          mensagem[i] = ' ';
mensagem[i] = '\0';
sh = i;
printf("\n comprimento: %d",strlen(mensagem));
for(m=0;m<(sh/8);m++)
{
   counter=0;
   printf("Bloco %d: ",m+1);
   for(/*ib=0, */i=0 ; i< 8 ; i++)
   {
      c = mensagem[nb]; // c armazena o caracter que esta sendo convertido
       for( power=7 ; power+1 ; power-- )
           if( c >= (1<<power) ) //se c é maior ou igual , é 1
           {
```

```
                c -= (1<<power);
                printf("1");
                plaintext[counter]=1;
                counter++;
            }
            else //senao, é 1
            {
                printf("0");
                plaintext[counter] = 0;
                counter++;
            }
            nb++;
    }
    plaintext[counter] = '\0'; // ultimo char deve ser NULL

//Algoritmo DES
// Inicio da Encriptografia
/*A mensagem, denominada plaintext, passa por uma permutacao inicial*/
    int left_block[32],right_block[32];
    int initial_message_permutation[64] =
{plaintext[57],plaintext[49],plaintext[41],plaintext[33],plaintext[25],
plaintext[17],plaintext[9] ,plaintext[1], plaintext[59], plaintext[51],
plaintext[43],plaintext[35],plaintext[27],plaintext[19],plaintext[11],
plaintext[3], plaintext[61],plaintext[53],plaintext[45],plaintext[37],
plaintext[29],plaintext[21],plaintext[13],plaintext[5],plaintext[63],
plaintext[55],plaintext[47],plaintext[39],plaintext[31],plaintext[23],
plaintext[15],plaintext[7],plaintext[56], plaintext[48],plaintext[40],
plaintext[32],plaintext[24],plaintext[16],plaintext[8] ,plaintext[0],
plaintext[58],plaintext[50],plaintext[42],plaintext[34], plaintext[26],
plaintext[18], plaintext[10],plaintext[2],plaintext[60], plaintext[52],
plaintext[44],plaintext[36],plaintext[28],plaintext[20],plaintext[12],
plaintext[4], plaintext[62],plaintext[54], plaintext[46], plaintext[38],
plaintext[30], plaintext[22], plaintext[14],plaintext[6]};
    printf("\napos permutacao inicial:  ");
    round = 0;
    for(i=0;i<=63;i++)
       printf("%d",initial_message_permutation[i]);
    /* A mensagem após a permutacao inicial é dividida em 2 partes*/
    for(i=0;i<=31;i++)
       left_block[i] = initial_message_permutation[i]; //bloco da esquerda, com
32 bits
    for(i=32;i<=63;i++)
    {
       right_block[i-32] = initial_message_permutation[i]; //bloco da direita,
com 32 bits
    }
    printf("\n                    L%d:  ",round);
    for(i=0;i<=31;i++)
      printf("%d",left_block[i]);
    printf("\n                    R%d:  ",round);
    for(i=0;i<=31;i++)
    printf("%d",right_block[i]);
/* A partir daqui, os blocos passarão por 16 rounds iguais,que irão encriptar
a mensagem*/
    int *exp;
    while(round<16)
    {
```

```
   printf("\n=======================ROUND %d================\n",(round+1));
   exp = perm_expansao(right_block);
   for(i=0;i<=47;i++)
        expansion_permutation[i] = (int)(*(exp+i));

   /*operacao XOR entre expansion_permutation e
compression_permutation_subkey*/
   int key_xor_right[48];
   for(i=0;i<=47;i++)
   { if(expansion_permutation[i] == K[round][i])
        key_xor_right[i] = 0;
     else key_xor_right[i] = 1;
   }
   printf("\noperacao XOR entre a permutacao de expansao e a chave,ambas com
48bits\n               EXPANSAO:  ");
   for(i=0;i<=47;i++)
     printf("%d",expansion_permutation[i]);
   printf("\n    CHAVE COMPRIMIDA K%d:  ",(round+1));
   for(i=0;i<=47;i++)
     printf("%d",K[round][i]);
   printf("\n          XOR:  ");
   for(i=0;i<=47;i++)
     printf("%d",key_xor_right[i]);

   int *p;
   p = SBOX(key_xor_right);
   printf("Apos a substituicao: ");
   for(i=0;i<=31;i++)
   {  printf("%d",*(p+i));
      after_substitution[i] = (int)(*(p+i));
   }
   int *pbox;

   pbox = PBOX(after_substitution);
   for(i=0;i<=31;i++)
        pbox_permutation[i] = (int)(*(pbox+i));

     //apos a permutacao Pbox
      printf("\n          apos pbox:  ");
      for(i=0;i<=31;i++)
         printf("%d",pbox_permutation[i]);
      //OPERACAO XOR entre left_block e a pbox_permutation, ambas com 32 bits
      //int xor_left_pbox[32];
      for(i=0;i<=31;i++)
      {  if(left_block[i] == pbox_permutation[i])
               xor_left_pbox[i] = 0;
          else xor_left_pbox[i] = 1;
      }
      printf("\noperacao XOR entre a PBOX e bloco L%d \n  bloco da esquerda:
",round);
      for(i=0;i<=31;i++)
         printf("%d",left_block[i]);
      printf("\n               Pbox:  ");
      for(i=0;i<=31;i++)
         printf("%d",pbox_permutation[i]);
      printf("\n        XOR:  ");
      for(i=0;i<=31;i++)
```

```
         printf("%d",xor_left_pbox[i]);
      for(i=0;i<=31;i++)
         left_block[i] = right_block[i];
       for(i=0;i<=31;i++)
           right_block[i] = xor_left_pbox[i];
       printf("\n        L%d:  ",(round+1));
       for(i=0;i<=31;i++)
           printf("%d",left_block[i]);
       printf("\n        R%d:  ",(round+1));
       for(i=0;i<=31;i++)
           printf("%d",right_block[i]);
       round++;
   } //final do while rounds
   //AO FINAL DA LINHA ANTERIOR, FOI CONCLUIDO UM ROUND, QUE SERA REPETIDO
MAIS 16 VEZES
   int ciphertext[64];  // ciphertext é a mensagem encriptada

/* antes da permutacao final, que é a inversa da permutacao inicial, os blocos
da direita e da esquerda trocam de posição */
   for(i=0;i<=31;i++)
      ciphertext[i+32] = left_block[i];
   for(i=0;i<=31;i++)
      ciphertext[i] = right_block[i];
   printf("\nantes da permutacao final: ");
   for(i=0;i<=63;i++)
       printf("%d",ciphertext[i]);
   printf("\nutilizando essas duas metades, deve-se fazer a permutacao final,
que eh o inverso da permutacao inicial");
   int final_message_permutation[64] =
{ciphertext[39],ciphertext[7],ciphertext[47],ciphertext[15],ciphertext[55],
ciphertext[23],ciphertext[63],ciphertext[31],ciphertext[38],ciphertext[6],
ciphertext[46],ciphertext[14],ciphertext[54],ciphertext[22],ciphertext[62],
ciphertext[30],ciphertext[37],ciphertext[5],ciphertext[45],ciphertext[13],
ciphertext[53],ciphertext[21],ciphertext[61],ciphertext[29],ciphertext[36],
ciphertext[4],ciphertext[44],ciphertext[12],ciphertext[52], ciphertext[20],
ciphertext[60],ciphertext[28],ciphertext[35],ciphertext[3], ciphertext[43],
ciphertext[11],ciphertext[51],ciphertext[19],ciphertext[59],ciphertext[27],
ciphertext[34],ciphertext[2], ciphertext[42],ciphertext[10],ciphertext[50],
ciphertext[18],ciphertext[58],ciphertext[26],ciphertext[33],ciphertext[1],
ciphertext[41],ciphertext[9],ciphertext[49],ciphertext[17], ciphertext[57],
ciphertext[25],ciphertext[32],ciphertext[0],ciphertext[40], ciphertext[8] ,
ciphertext[48],ciphertext[16],ciphertext[56],ciphertext[24]};
   printf("\ntem-se portanto o bloco %d encriptado em binario: ",m+1);
   for(i=0;i<=63;i++)
      printf("%d",final_message_permutation[i]);
   //conversao de byte para ascii
   int bit=0,j,mensagem_decimal[8]={0};
   for(j=0;j<=7;j++)
   {
     for(i=7;i>=0;i--)
      {  mensagem_decimal[j] =
(final_message_permutation[bit]*pow(2,i))+mensagem_decimal[j];
          bit++;
      }
      cipher_final[mc++] = mensagem_decimal[j];
   }
} //final do for principal
```

```
cipher_final[mc] = '\0';

printf("\nA mensagem encriptada eh %s",cipher_final);
printf("\n%d",strlen(cipher_final));
/* Para a decriptografia, utiliza-se o mesmo algoritmo, porém, com as chaves
no sentido contrario, e os shifts sao para direita */
char text3[25] = "Mensagem encriptada:\0";
char cipher_vga[1000]={};
VGA_text (1, (y+4), text3);
for(i=0,x=23;i<=strlen(cipher_final);i++,x++){
     cipher_vga[i] = cipher_final[i];
     if((x%78)==0)
     {y++;x=0;z++;} //x=0;
     if(i==strlen(cipher_final)) cipher_vga[i]='\0';
     VGA_text(x,y+4,(cipher_vga)+i);
}
/*verificacao da chave*/
char text4[40] = "Chave decriptadora:\0";
//y = 3;
VGA_text (1, y+6, text4);

printf("\n-----------------------------------------------------------------");
printf("\n\nDECRIPTACAO\n");
round = 15;
int tentativas =0; int *ch_D;
while(tentativas <3)
{
    alt_up_ps2_init(ps2);
     alt_up_ps2_clear_fifo(ps2);
     read_keyboard(ps2,vga);
/***bloco que converte a chave para um bloco de de 64 bits em binario******/
    printf( "'%s' em binario:    " , chave_D );
     contador_funcao++;
     counter=0;
     ch_D = converte_para_binario(chave_D);
     for(i=0;i<=63;i++)
               chave_bits[i] = (int)(*(ch_D+i));

/** Na criptografia DES, a chave possui 56 bits, para isso, a chave inicial,
que possui geralmente 64 bits,passa por uma permutação */
     int key_56_D[56]={};
     int initial_key_D_permutation[56] =
{chave_bits[56],chave_bits[48],chave_bits[40],chave_bits[32],chave_bits[24],
chave_bits[16],chave_bits[8], chave_bits[0], chave_bits[57], chave_bits[49],
chave_bits[41],chave_bits[33],chave_bits[25],chave_bits[17], chave_bits[9],
chave_bits[1], chave_bits[58],chave_bits[50],chave_bits[42], chave_bits[34],
chave_bits[26],chave_bits[18],chave_bits[10],chave_bits[2], chave_bits[59],
chave_bits[51],chave_bits[43],chave_bits[35],chave_bits[62],chave_bits[54],
chave_bits[46],chave_bits[38],chave_bits[30],chave_bits[22],chave_bits[14],
chave_bits[6], chave_bits[61],chave_bits[53],chave_bits[45],chave_bits[37],
chave_bits[29],chave_bits[21],chave_bits[13],chave_bits[5], chave_bits[60],
chave_bits[52],chave_bits[44],chave_bits[36],chave_bits[28],chave_bits[20],
chave_bits[12],chave_bits[4], chave_bits[27],chave_bits[19],chave_bits[11],
chave_bits[3]};
     printf("KEY1:  ");
     for(i=0;i<=55;i++)
     {  printf("%d",initial_key_D_permutation[i]);
```

```
            key_56_D[i] = initial_key_D_permutation[i];
        }
        gera_sub_chaves(key_56_D);
        int po;
        po = strlen(cipher_final);
        printf("\n%d",po);
        m=0;c=0;nb=0;
        for(m=0;m<(po/8);m++)
        {
            counter=0;
            printf("Bloco %d: ",m+1);
            //for(ib=0,i=0;i<8;i++,nb++)
            for(/*ib=0, */i=0 ; i< 8 ; i++)//,nb++ )
            {
                c = (unsigned int)(unsigned char)cipher_final[nb]; // c armazena o
caracter que esta sendo convertido
                printf(" %d",c);
                for( power=7 ; power+1 ; power-- )
                if( c >= (1<<power) ) //se c é maior ou igual , é 1
                {
                   c -= (1<<power);
                   printf("1");
                   plaintext[counter]=1;
                   counter++;
                }
                else //senao, é 1
                {
                   printf("0");
                   plaintext[counter] = 0;
                   counter++;
                }
                nb++;
            }
            plaintext[counter] = '\0';
            int left_decryption_block[32]={},right_decryption_block[32]={};

            int initial_decryption_permutation[64] =
{plaintext[57],plaintext[49],plaintext[41],plaintext[33],plaintext[25],
plaintext[17], plaintext[9],plaintext[1], plaintext[59], plaintext[51],
plaintext[43],plaintext[35],plaintext[27],plaintext[19],plaintext[11],
plaintext[3], plaintext[61],plaintext[53],plaintext[45],plaintext[37],
plaintext[29],plaintext[21],plaintext[13],plaintext[5],plaintext[63],
plaintext[55],plaintext[47],plaintext[39],plaintext[31],plaintext[23],
plaintext[15],plaintext[7],plaintext[56], plaintext[48],plaintext[40],
plaintext[32],plaintext[24],plaintext[16],plaintext[8],plaintext[0],
plaintext[58],plaintext[50],plaintext[42],plaintext[34],plaintext[26],
plaintext[18],plaintext[10],plaintext[2],plaintext[60],plaintext[52],
plaintext[44],plaintext[36],plaintext[28],plaintext[20],plaintext[12],
plaintext[4], plaintext[62],plaintext[54],plaintext[46],plaintext[38],
plaintext[30], plaintext[22], plaintext[14],plaintext[6]};
             for(i=0;i<=31;i++)
             {  left_decryption_block[i] = initial_decryption_permutation[i];
                right_decryption_block[i] = initial_decryption_permutation[i+32];
             }
             printf("\napos permutacao inicial:  ");
             for(i=0;i<=63;i++) printf("%d",initial_decryption_permutation[i]);
             round = 15;
```

```
            int *exp_dec;
            /* Inicio dos rounds */
            while(round>=0)
            {
             exp_dec = perm_expansao(right_decryption_block);
             for(i=0;i<=47;i++){
                      expansion_permutation[i] = (int)(*(exp_dec+i));
                 }

              for(i=0;i<=47;i++)
              {
                 if(expansion_permutation[i] ==
K[round][i])//;compression_permutation_key[i])
                      key_xor_right[i] = 0;
                 else key_xor_right[i] = 1;
              }
              printf("\noperacao XOR entre a permutacao de expansao e a
chave,ambas com 48bits\n EXPANSAO  ");
              for(i=0;i<=47;i++)
                  printf("%d",expansion_permutation[i]);
              printf("\nK%d            ",(round+1));
              for(i=0;i<=47;i++)
                  printf("%d",K[round][i]);
              printf("\nXOR                        ");
              for(i=0;i<=47;i++)
                  printf("%d",key_xor_right[i]);
              int *q;
              q = SBOX(key_xor_right);
            printf("Apos a substituicao:  ");
                 for(i=0;i<=31;i++) {printf("%d",*(q+i));
                                     after_substitution[i] = (int)(*(q+i));
     }
            int *pbox_D;
            pbox_D = PBOX(after_substitution);
            for(i=0;i<=31;i++){
                      pbox_permutation[i] = (int)(*(pbox_D+i));
                 }

            printf("\napos pbox: ");
            for(i=0;i<=31;i++)
                 printf("%d",pbox_permutation[i]);
            //OPERACAO XOR entre left_block e a pbox_permutation
            for(i=0;i<=31;i++)
            {
               if(left_decryption_block[i] == pbox_permutation[i])
                   xor_left_pbox[i] = 0;
               else xor_left_pbox[i] = 1;
            }
            printf("\noperacao XOR entre a PBOX e L%d\n bloco da esquerda:
",(15-round));
            for(i=0;i<=31;i++)
                 printf("%d",left_decryption_block[i]);
            printf("\nPbox                              ");
            for(i=0;i<=31;i++)
                 printf("%d",pbox_permutation[i]);
            printf("\nXOR                            ");
            for(i=0;i<=31;i++)
```

```
                printf("%d",xor_left_pbox[i]);

          //AO FINAL DA LINHA ANTERIOR, FOI CONCLUIDO UM ROUND, QUE SERA
REPETIDO MAIS 16 VEZES
          for(i=0;i<=31;i++)
                left_decryption_block[i] = right_decryption_block[i];

          for(i=0;i<=31;i++)
                right_decryption_block[i] = xor_left_pbox[i];

          printf("\nL%d:                ",(round));
          for(i=0;i<=31;i++)
                printf("%d",left_decryption_block[i]);
          printf("\nR%d:             ",(round));
          for(i=0;i<=31;i++)
                printf("%d",right_decryption_block[i]);
          //printf("");

          round--;
          } //final do while rounds
          int ciphertext[64];
     /* Assim como na encriptografia, na decriptografia os blocos resultantes
da direita e esquerda trocam de lugar*/
          printf("\n apos os 16 rounds, tem-se as seguintes metades da
esquerda e direita\n");
          printf("esquerda: ");
          for(i=0;i<=31;i++){
                ciphertext[i+32] = left_decryption_block[i];
                printf("%d",left_decryption_block[i]);
          }
          printf("\ndireita: ");
          for(i=0;i<=31;i++){
                ciphertext[i] = right_decryption_block[i];
                printf("%d",right_decryption_block[i]);
          }
          printf("\nutilizando essas duas metades, deve-se fazer a permutacao
final, que eh o inverso da permutacao inicial");
          int final_decryption_permutation[64] =
{ciphertext[39],ciphertext[7],ciphertext[47],ciphertext[15],ciphertext[55],
ciphertext[23],ciphertext[63],ciphertext[31],ciphertext[38],ciphertext[6],
ciphertext[46],ciphertext[14],ciphertext[54],ciphertext[22],ciphertext[62],
ciphertext[30],ciphertext[37],ciphertext[5],ciphertext[45],ciphertext[13],
ciphertext[53],ciphertext[21],ciphertext[61],ciphertext[29],ciphertext[36],
ciphertext[4],ciphertext[44],ciphertext[12],ciphertext[52],ciphertext[20],
ciphertext[60],ciphertext[28],ciphertext[35],ciphertext[3],ciphertext[43],
ciphertext[11],ciphertext[51],ciphertext[19],ciphertext[59],ciphertext[27],
ciphertext[34],ciphertext[2],ciphertext[42],ciphertext[10],ciphertext[50],
ciphertext[18],ciphertext[58],ciphertext[26],ciphertext[33],ciphertext[1],
ciphertext[41],ciphertext[9],ciphertext[49],ciphertext[17],ciphertext[57],
ciphertext[25],ciphertext[32],ciphertext[0],ciphertext[40],ciphertext[8] ,
ciphertext[48], ciphertext[16], ciphertext[56], ciphertext[24]};
          printf("\ntem-se portanto o bloco decriptado em binario: ");
          for(i=0;i<=63;i++)
                printf("%d",final_decryption_permutation[i]);
          int bit=0;int mensagem_decriptada_decimal[8]={0}; int j;
          for(j=0;j<=7;j++)
          {
```

```
                for(i=7;i>=0;i--)
          {
                mensagem_decriptada_decimal[j] =
(final_decryption_permutation[bit]*pow(2,i))+mensagem_decriptada_decimal[j];
                bit++;
                }
                printf("%d ",mensagem_decriptada_decimal[j]);
                cipher_temp[md++] = mensagem_decriptada_decimal[j];
                //cipher_final[md++] = mensagem_decriptada_decimal[j];
          }
          for(i=0;i<=7;i++)
                printf("\n bit %d em inteiro %d\nbit em char
%c",i,mensagem_decriptada_decimal[i],mensagem_decriptada_decimal[i]);
          printf("\nnumero de bits:%d",(bit));
     } //fim do for principal
          cipher_temp[md] = '\0';
          md=0,nb=0;
          printf("\nresultado da decriptacao: %s",cipher_temp);
     //char cipher_
          char text5[25] = "Mensagem decriptada:\0";
          VGA_text (1, y+6, text5);
          char cipher_vgaD[1000]={};
          for(i=0,x=23;i<=strlen(cipher_temp);i++,x++){
          cipher_vgaD[i] = cipher_temp[i];
          if((x%78)==0)
          {y++;x=0;} //x=0;
          if(i==strlen(cipher_temp)) cipher_vgaD[i]='\0';
          VGA_text(x,y+6,(cipher_vgaD)+i);
          }
}//fim do while
}
/* Subrotina que desenha um retângulo preenchido no monitor VGA*/ ALTERA ©
void VGA_box(int x1, int y1, int x2, int y2, short pixel_color)
{
int offset, row, col;
volatile short * pixel_buffer = (short *) 0x08000000; // VGA pixel buffer

/* assume that the box coordinates are valid */
for (row = y1; row <= y2; row++)
{
col = x1;
while (col <= x2)
{
offset = (row << 7) + col;
*(pixel_buffer + offset) = pixel_color; // compute halfword Address, set pixel
++col;
}
}
}
/**Subrotina que imprime uma string no monitor VGA*/     ALTERA ©
void VGA_text(int x3, int y3, char * text_ptr)
{
     int offset;
     /* Declare volatile pointer to char buffer (volatile means that IO load
        and store instructions will be used to access these pointer locations,
        instead of regular memory loads and stores) */
```

```
     volatile char * character_buffer = (char *) 0x09000000;  // VGA character
buffer

     /* assume that the text string fits on one line */
     offset = (y3 << 7) + x3;
     while ( *(text_ptr) )
     {
          *(character_buffer + offset) = *(text_ptr);     // write to the
character buffer
          ++text_ptr;
          ++offset;
     }
}
int * SBOX(int XOR[48]){
     int S1[4][16] = {{14,4,13,1,2,15,11,8,3,10,6,12,5,9,0,7},
 {0,15, 7,4,14,2,13,1,10,6,12,11,9,5,3,8},
 {4,1,14,8,13,6,2,11,15,12,9,7,3,10,5,0},
 {15,12,8,2,4,9,1,7,5,11,3,14,10,0,6,13}};
     int S2[4][16] = {{15,1,8,14,6,11,3,4,9,7,2,13,12,0,5,10},
                      {3,13,4,7,15,2,8,14,12,0,1,10,6,9,11, 5},
                      {0,14,7,11,10,4,13,1,5,8,12,6,9,3,2, 15},
                      {13,8,10,1,3,15,4,2,11,6,7,12,0,5,14,9}};
     int S3[4][16] = {{10,0,9,14,6,3,15,5,1,13,12,7,11,4,2,8},
                      {13,7,0,9,3,4,6,10,2,8,5,14,12,11,15, 1},
                      {13,6,4,9,8,15,3,0,11,1,2,12,5,10,14,7},
                      {1,10,13,0,6,9,8,7,4,15,14,3,11,5,2,12}};
     int S4[4][16] = {{7,13,14,3,0,6,9,10,1,2,8,5,11,12,4,15},
                      {13,8,11,5,6,15,0,3,4,7,2,12,1,10,14,9},
                      {10,6,9,0,12,11,7,13,15,1,3,14,5,2,8,4},
                      {3,15,0,6,10,1,13,8,9,4,5,11,12,7,2,14}};
     int S5[4][16] = {{2,12,4,1,7,10,11,6,8,5,3,15,13,0,14,9},
                     {14,11,2,12,4,7,13,1,5,0,15,10,3,9,8, 6},
                      {4,2,1,11,10,13,7,8,15,9,12,5,6,3,0,14},
                      {11,8,12,7,1,14,2,13,6,15,0,9,10,4,5,3}};
     int S6[4][16] = {{12,1,10,15,9,2,6,8,0,13,3,4,14,7,5,11},
                      {10,15,4,2,7,12,9,5,6,1,13,14,0,11,3,8},
                      { 9,14,15,5,2,8,12,3,7,0,4,10,1,13,11,6},
                      {4,3,2,12,9,5,15,10,11,14,1,7,6,0,8,13}};
     int S7[4][16] = {{4,11,2,14,15,0,8,13,3,12,9,7,5,10,6,1},
                      {13,0,11,7,4,9,1,10,14,3,5,12,2,15,8,6},
                      {1,4,11,13,12,3,7,14,10,15,6,8,0,5,9,2},
                      {6,11,13,8,1,4,10,7,9,5,0,15,14,2,3,12}};
     int S8[4][16] = {{13,2,8,4,6,15,11,1,10,9,3,14,5,0,12,7},
                      {1,15,13,8,10,3,7,4,12,5,6,11,0,14,9,2},
                      {7,11,4,1,9,12,14,2,0,6,10,13,15,3,5,8},
                      {2, 1,14,7,4,10,8,13,15,12,9,0,3,5,6,11}};
     int
grupo1[6]={},grupo2[6]={},grupo3[6]={},grupo4[6]={},grupo5[6]={},grupo6[6]={},
grupo7[6]={},grupo8[6]={};
     int
grupo1_out[6]={},grupo2_out[4]={},grupo3_out[4]={},grupo4_out[4]={},grupo5_out
[4]={},grupo6_out[4]={},grupo7_out[4]={},grupo8_out[4]={};
     //int linha=coluna=0;
     int lin_bin[2]={},col_bin[4]={};
     int i,grupo;
     int elemento = 0,resto,l;
     static int apos_subst[32];
```

```
     /*Antes de passar pelas S-box, o bloco é separado em 8 grupos, com 6 bits
de entrada cada um e 4 bits na saída*/
     for(i=0;i<=47;i++){
          if(i<=5) grupo1[i]                    = XOR[i];
          if((i>=6) &&  (i<=11)) grupo2[i-6]    = XOR[i];
          if((i>=12) && (i<=17)) grupo3[i-12] = XOR[i];
          if((i>=18) && (i<=23))  grupo4[i-18] = XOR[i];
          if((i>=24) && (i<=29)) grupo5[i-24] = XOR[i];
          if((i>=30) && (i<=35)) grupo6[i-30] = XOR[i];
          if((i>=36) && (i<=41)) grupo7[i-36] = XOR[i];
          if((i>=42) && (i<=47)) grupo8[i-42] = XOR[i];
     }

     for(grupo=0;grupo<=7;grupo++)
     {
          printf("\n-----------------------------------------------------------
--\n");
          printf("grupo %d:",(grupo+1));
          for(i=0;i<=5;i++)
               printf("%d",grupo1[i]);
          switch(grupo)
          {
            case 0:
            {  for(i=0;i<=5;i++)
               {  switch(i)
                    { case 0: lin_bin[0]      = grupo1[i];break;
                      case 1: col_bin[0] = grupo1[i];break;
                      case 2: col_bin[1] = grupo1[i];break;
                      case 3: col_bin[2] = grupo1[i];break;
                      case 4: col_bin[3] = grupo1[i];break;
                      case 5: lin_bin[1]      = grupo1[i];break;
                    }
               }
             }break;
            case 1:
            {  for(i=0;i<=5;i++)
               {  switch(i)
                  { case 0: lin_bin[0]        = grupo2[i];break;
                    case 1: col_bin[0] = grupo2[i];break;
                    case 2: col_bin[1] = grupo2[i];break;
                    case 3: col_bin[2] = grupo2[i];break;
                    case 4: col_bin[3] = grupo2[i];break;
                    case 5: lin_bin[1]   = grupo2[i];break;
                  }
               }
            }break;
            case 2:
            {  for(i=0;i<=5;i++)
               {  switch(i)
                  { case 0: lin_bin[0]        = grupo3[i];break;
                    case 1: col_bin[0] = grupo3[i];break;
                    case 2: col_bin[1] = grupo3[i];break;
                    case 3: col_bin[2] = grupo3[i];break;
                    case 4: col_bin[3] = grupo3[i];break;
                    case 5: lin_bin[1]   = grupo3[i];break;
                  }
               }
```

```
}break;
case 3:
{  for(i=0;i<=5;i++)
   {  switch(i)
      { case 0: lin_bin[0]        = grupo4[i];break;
        case 1: col_bin[0] = grupo4[i];break;
        case 2: col_bin[1] = grupo4[i];break;
        case 3: col_bin[2] = grupo4[i];break;
        case 4: col_bin[3] = grupo4[i];break;
        case 5: lin_bin[1]   = grupo4[i];break;
      }
   }
}break;
case 4:
{  for(i=0;i<=5;i++)
   {  switch(i)
      { case 0: lin_bin[0]        = grupo5[i];break;
        case 1: col_bin[0] = grupo5[i];break;
        case 2: col_bin[1] = grupo5[i];break;
        case 3: col_bin[2] = grupo5[i];break;
        case 4: col_bin[3] = grupo5[i];break;
        case 5: lin_bin[1]   = grupo5[i];break;
      }
   }
}break;
case 5:
{  for(i=0;i<=5;i++)
   {  switch(i)
      { case 0: lin_bin[0]        = grupo6[i];break;
        case 1: col_bin[0] = grupo6[i];break;
        case 2: col_bin[1] = grupo6[i];break;
        case 3: col_bin[2] = grupo6[i];break;
        case 4: col_bin[3] = grupo6[i];break;
        case 5: lin_bin[1]   = grupo6[i];break;
      }
   }
}break;
case 6:
{  for(i=0;i<=5;i++)
   {  switch(i)
      { case 0: lin_bin[0]        = grupo7[i];break;
        case 1: col_bin[0] = grupo7[i];break;
        case 2: col_bin[1] = grupo7[i];break;
        case 3: col_bin[2] = grupo7[i];break;
        case 4: col_bin[3] = grupo7[i];break;
        case 5: lin_bin[1]   = grupo7[i];break;
      }
   }
}break;
case 7:
{  for(i=0;i<=5;i++)
   {  switch(i)
      { case 0: lin_bin[0]        = grupo8[i];break;
        case 1: col_bin[0] = grupo8[i];break;
        case 2: col_bin[1] = grupo8[i];break;
        case 3: col_bin[2] = grupo8[i];break;
        case 4: col_bin[3] = grupo8[i];break;
```

```
                    case 5: lin_bin[1]   = grupo8[i];break;
                  }
               }
             }break;
          }
          printf("\nlinha: ");
          for(i=0;i<2;i++) printf("%d",lin_bin[i]);
          printf("\ncoluna: ");
          for(i=0;i<4;i++) printf("%d",col_bin[i]);
          //BLOCO QUE CONVERTE A LINHA E COLUNA PARA DECIMAL, PARA SER
ENCONTRADA NA MATRIZ S
          int lin_dec=0,col_dec=0, j=0;
          //int after_substitution[32];
          for(i=1;i>=0;i--)
     {
          lin_dec = (lin_bin[i]*pow(2,j))+lin_dec;
          j++;
     }
     //j=0;
          printf("\nlinha: %d",lin_dec);
          for(i=3,j=0;i>=0;i--)
     {
          col_dec = (col_bin[i]*pow(2,j))+col_dec;
          j++;
     }
          printf("\ncoluna: %d",col_dec);
          elemento = 0;
          switch(grupo){
               case 0:elemento = S1[lin_dec][col_dec];break;
               case 1:elemento = S2[lin_dec][col_dec];break;
               case 2:elemento = S3[lin_dec][col_dec];break;
               case 3:elemento = S4[lin_dec][col_dec];break;
               case 4:elemento = S5[lin_dec][col_dec];break;
               case 5:elemento = S6[lin_dec][col_dec];break;
               case 6:elemento = S7[lin_dec][col_dec];break;
               case 7:elemento = S8[lin_dec][col_dec];break;
          }
          //dec=elemento;
          // o elemento eh o indice dos vetores das caixas S que farao a
substituicao
          printf("\nnumero da caixa %d: %d",(grupo+1),elemento);
          l=0;
          for(i=3;i>=0;i--)
          {
          resto = elemento % 2;//To
             switch(grupo)
             {
               case 0:
               {
                     if(resto > 0) {
                         grupo1_out[i]=1;} else{grupo1_out[i]=0;}
               }break;
               case 1:
               {
                     if(resto > 0) {
                         grupo2_out[i]=1;} else{grupo2_out[i]=0;}
               }break;
```

```
                case 2:
                 {
                       if(resto > 0) {
                          grupo3_out[i]=1;} else{grupo3_out[i]=0;}
                }break;
                case 3:
                 {
                       if(resto > 0) {
                          grupo4_out[i]=1;} else{grupo4_out[i]=0;}
                }break;
                case 4:
                 {
                       if(resto > 0) {
                          grupo5_out[i]=1;} else{grupo5_out[i]=0;}
                }break;
                case 5:
                 {
                       if(resto > 0) {
                          grupo6_out[i]=1;} else{grupo6_out[i]=0;}
                }break;
                case 6:
                 {
                       if(resto > 0) {
                          grupo7_out[i]=1;} else{grupo7_out[i]=0;}
                }break;
                case 7:
                 {
                       if(resto > 0) {
                          grupo8_out[i]=1;} else{grupo8_out[i]=0;}
                }break;

               }
             //l++;
             elemento = elemento/2;
             }
        }
        for(i=0;i<=3;i++)       apos_subst[i]=grupo1_out[i];
        for(i=4;i<=7;i++)     apos_subst[i]=grupo2_out[i-4];
        for(i=8;i<=11;i++)    apos_subst[i]=grupo3_out[i-8];
        for(i=12;i<=15;i++) apos_subst[i]=grupo4_out[i-12];
        for(i=16;i<=19;i++) apos_subst[i]=grupo5_out[i-16];
        for(i=20;i<=23;i++) apos_subst[i]=grupo6_out[i-20];
        for(i=24;i<=27;i++) apos_subst[i]=grupo7_out[i-24];
        for(i=28;i<=31;i++) apos_subst[i]=grupo8_out[i-28];

        return apos_subst;

}

void gera_sub_chaves(int chave_56[56]){
        int rot_sub_chaves[16] = {1,1,2,2,2,2,2,2,1, 2, 2, 2, 2, 2, 2,1};
        //static int **K[16][48];
        int round,i,j=0,tempEsq,tempDir;
        int perm_comp_chave[48]={};
        int sub_chave_esq[28]={},sub_chave_esq_apos_rot[28]={};
        int sub_chave_dir[28]={},sub_chave_dir_apos_rot[28]={};
        int chave_apos_rot[56]={};
```

```
     for(round=0;round<16;round++){
     for(i=0;i<=27;i++)
     {
          sub_chave_esq[i] = chave_56[i];
     }
     for(i=28;i<=55;i++){
          sub_chave_dir[i-28] = chave_56[i];
     }
     printf("\n          KEY%d L:  ",round);
     for(i=0;i<=27;i++)
          printf("%d",sub_chave_esq[i]);
     printf("\n          KEY%d R:  ",round);
     for(i=0;i<=27;i++)
          printf("%d",sub_chave_dir[i]);
     printf("\nas metades da esquerda e direita devem ser deslocadas de um ou
dois bits para a esquerda dependendo do round\n");
     for(i=0;i<=27;i++)
     {
          sub_chave_esq_apos_rot[i] = sub_chave_esq[i];
          sub_chave_dir_apos_rot[i] = sub_chave_dir[i];
     }

     for (j = 0; j <rot_sub_chaves[round];j++) //depois colocar o range para
i<= shift_subkeys[round]
    {
     tempEsq = sub_chave_esq_apos_rot[0];
          tempDir = sub_chave_dir_apos_rot[0];
     for(i=0;i<27;i++)
     {
               sub_chave_esq_apos_rot[i] = sub_chave_esq_apos_rot[i+1];
               sub_chave_dir_apos_rot[i] = sub_chave_dir_apos_rot[i+1];
     }
               sub_chave_esq_apos_rot[i] = tempEsq;
               sub_chave_dir_apos_rot[i] = tempDir;
    }
    for(i=0;i<=27;i++)
          chave_apos_rot[i]= sub_chave_esq_apos_rot[i];
     for(i=28;i<=55;i++)
          chave_apos_rot[i]= sub_chave_dir_apos_rot[i-28];


     printf("         KEY%d L<%d:  ",(round+1),rot_sub_chaves[round]);
     for(i=0;i<=27;i++)
          printf("%d",sub_chave_esq_apos_rot[i]);
     printf("\n           KEY%d R<%d:  ",(round+1),rot_sub_chaves[round]);
     for (i = 0; i <=27;i++)
          printf("%d",sub_chave_dir_apos_rot[i]);

          //permutação de compressão
     for(i=0;i<=47;i++)
     {switch(i)
      {case 0 : perm_comp_chave[i] = chave_apos_rot[13];break;/**/case 1:
perm_comp_chave[i] = chave_apos_rot[16];break;
       case 2 : perm_comp_chave[i] = chave_apos_rot[10];break;/**/case 3:
perm_comp_chave[i] = chave_apos_rot[23];break;
```

```
       case 4 : perm_comp_chave[i] = chave_apos_rot[0] ;break;/**/case 5:
perm_comp_chave[i] = chave_apos_rot[4] ;break;
       case 6 : perm_comp_chave[i] = chave_apos_rot[2] ;break;/**/case 7:
perm_comp_chave[i] = chave_apos_rot[27];break;
       case 8 : perm_comp_chave[i] = chave_apos_rot[14];break;/**/case 9:
perm_comp_chave[i] = chave_apos_rot[5] ;break;
       case 10: perm_comp_chave[i] = chave_apos_rot[20];break;/**/case 11:
perm_comp_chave[i] = chave_apos_rot[9];break;
       case 12: perm_comp_chave[i] = chave_apos_rot[22];break;/**/case 13:
perm_comp_chave[i]= chave_apos_rot[18];break;
       case 14: perm_comp_chave[i] = chave_apos_rot[11];break;/**/case 15:
perm_comp_chave[i] = chave_apos_rot[3];break;
       case 16: perm_comp_chave[i] = chave_apos_rot[25];break;/**/case 17:
perm_comp_chave[i] = chave_apos_rot[7];break;
       case 18: perm_comp_chave[i] = chave_apos_rot[15];break;/**/case 19:
perm_comp_chave[i] = chave_apos_rot[6];break;
       case 20: perm_comp_chave[i] = chave_apos_rot[26];break;/**/case 21:
perm_comp_chave[i] =chave_apos_rot[19];break;
       case 22: perm_comp_chave[i] = chave_apos_rot[12];break;/**/case 23:
perm_comp_chave[i] = chave_apos_rot[1];break;
       case 24: perm_comp_chave[i] = chave_apos_rot[40];break;/**/case 25:
perm_comp_chave[i] =chave_apos_rot[51];break;
       case 26: perm_comp_chave[i] = chave_apos_rot[30];break;/**/case 27:
perm_comp_chave[i] =chave_apos_rot[36];break;
       case 28: perm_comp_chave[i] = chave_apos_rot[46];break;/**/case 29:
perm_comp_chave[i] =chave_apos_rot[54];break;
       case 30: perm_comp_chave[i] = chave_apos_rot[29];break;/**/case 31:
perm_comp_chave[i] =chave_apos_rot[39];break;
       case 32: perm_comp_chave[i] = chave_apos_rot[50];break;/**/case 33:
perm_comp_chave[i] =chave_apos_rot[44];break;
       case 34: perm_comp_chave[i] = chave_apos_rot[32];break;/**/case 35:
perm_comp_chave[i] =chave_apos_rot[47];break;
       case 36: perm_comp_chave[i] = chave_apos_rot[43];break;/**/case 37:
perm_comp_chave[i] =chave_apos_rot[48];break;
       case 38: perm_comp_chave[i] = chave_apos_rot[38];break;/**/case 39:
perm_comp_chave[i] =chave_apos_rot[55];break;
       case 40: perm_comp_chave[i] = chave_apos_rot[33];break;/**/case 41:
perm_comp_chave[i] =chave_apos_rot[52];break;
       case 42: perm_comp_chave[i] = chave_apos_rot[45];break;/**/case 43:
perm_comp_chave[i] =chave_apos_rot[41];break;
       case 44: perm_comp_chave[i] = chave_apos_rot[49];break;/**/case 45:
perm_comp_chave[i] =chave_apos_rot[35];break;
       case 46: perm_comp_chave[i] = chave_apos_rot[28];break;/**/case 47:
perm_comp_chave[i] =chave_apos_rot[31];break;
      }
     }
     for(i=0;i<=47;i++){
          K[round][i]= perm_comp_chave[i];
          //printf("%d",K[round][i]);
     }
     for(i=0;i<=55;i++){
          chave_56[i] = chave_apos_rot[i];
     }
}

}
```

```
int * converte_para_binario(char ch[8]){
          /*****bloco que converte a chave para um bloco de de 64 bits em
binario******/
     printf( "'%s' em binario:    " , ch );
     int power,counter=0,i;
     static int ch_bin[64]={};
     unsigned int c;
     for( i=0 ; i<strlen(ch) ; i++ )
     {
          c = ch[i]; // c armazena o caracter que esta sendo convertido
          for( power=7 ; power+1 ; power-- )
          if( c >= (1<<power) ) //se c é maior ou igual , é 1
          {
            c -= (1<<power);
            printf("1");
            ch_bin[counter]=1;
            counter++;
          }
          else //senao, é 1
          {
            printf("0");
            ch_bin[counter] = 0;
            counter++;
          }
     }
     ch_bin[counter] = '\0'; // ultimo char deve ser NULL
     printf("\ncontador: %d\n",counter);
     return ch_bin;
}

int * perm_expansao(int dir[32]){
     int i;
     static int apos_exp[48]={};
     for(i=0;i<=47;i++){
     switch(i){
     case 0:apos_exp[i]=dir[31];break;/**/case 1:apos_exp[i]=dir[0]; break;
     case 2:apos_exp[i]=dir[1];break;/**/case 3:apos_exp[i]=dir[2]; break;
     case 4:apos_exp[i]=dir[3];break;/**/case 5:apos_exp[i]= dir[4]; break;
     case 6:apos_exp[i]=dir[3];break;/**/case 7:apos_exp[i]=dir[4]; break;
     case 8:apos_exp[i]=dir[5];break;/**/case 9:apos_exp[i]=dir[6]; break;
     case 10:apos_exp[i]=dir[7];break;/**/case 11:apos_exp[i]=dir[8]; break;
     case 12:apos_exp[i]=dir[7];break;/**/case 13:apos_exp[i]=dir[8]; break;
     case 14:apos_exp[i]=dir[9];break;/**/case 15:apos_exp[i]=dir[10];break;
     case 16:apos_exp[i]=dir[11];break;/**/case 17:apos_exp[i]=dir[12];break;
     case 18:apos_exp[i]=dir[11];break;/**/case 19:apos_exp[i]=dir[12];break;
     case 20:apos_exp[i]=dir[13];break;/**/case 21:apos_exp[i]=dir[14];break;
     case 22:apos_exp[i]=dir[15];break;/**/case 23:apos_exp[i]=dir[16];break;
     case 24: apos_exp[i]=dir[15];break;/**/case 25:apos_exp[i]=dir[16];break;
     case 26:apos_exp[i]=dir[17];break;/**/case 27:apos_exp[i]=dir[18];break;
     case 28: apos_exp[i]=dir[19];break;/**/case 29:apos_exp[i]=dir[20];break;
     case 30:apos_exp[i]=dir[19];break;/**/case 31:apos_exp[i]=dir[20];break;
     case 32: apos_exp[i]=dir[21];break;/**/case 33:apos_exp[i]=dir[22];break;
     case 34: apos_exp[i]=dir[23];break;/**/case 35:apos_exp[i]=dir[24];break;
     case 36: apos_exp[i]=dir[23];break;/**/case 37:apos_exp[i]=dir[24];break;
     case 38: apos_exp[i]=dir[25];break;/**/case 39:apos_exp[i]=dir[26];break;
     case 40: apos_exp[i]=dir[27];break;/**/case 41:apos_exp[i]=dir[28];break;
     case 42: apos_exp[i]=dir[27];break;/**/case 43:apos_exp[i]=dir[28];break;
```

```
     case 44: apos_exp[i]=dir[29];break;/**/case 45:apos_exp[i]=dir[30];break;
     case 46:apos_exp[i] = dir[31];break;/**/case 47:apos_exp[i]=dir[0];break;
          }
     }

     return apos_exp;
}

int * PBOX(int antes_pbox[32]){
     int i;
     static int apos_pbox[32]={};
     for(i=0;i<=31;i++){
          switch(i){
          case 0 : apos_pbox[i] =antes_pbox[15];break;/**/case 1 :
apos_pbox[i] =antes_pbox[6] ;break;
               case 2 : apos_pbox[i] =antes_pbox[19];break;/**/case 3 :
apos_pbox[i] =antes_pbox[20];break;
               case 4 : apos_pbox[i] =antes_pbox[28];break;/**/case 5 :
apos_pbox[i] =antes_pbox[11];break;
               case 6 : apos_pbox[i] =antes_pbox[27];break;/**/case 7 :
apos_pbox[i] =antes_pbox[16];break;
               case 8 : apos_pbox[i] =antes_pbox[0] ;break;/**/case 9 :
apos_pbox[i] =antes_pbox[14];break;
               case 10: apos_pbox[i] =antes_pbox[22];break;/**/case 11:
apos_pbox[i] =antes_pbox[25];break;
               case 12: apos_pbox[i] =antes_pbox[4] ;break;/**/case 13:
apos_pbox[i] =antes_pbox[17];break;
               case 14: apos_pbox[i] =antes_pbox[30];break;/**/case 15:
apos_pbox[i] =antes_pbox[9] ;break;
               case 16: apos_pbox[i] =antes_pbox[1] ;break;/**/case 17:
apos_pbox[i] =antes_pbox[7] ;break;
               case 18: apos_pbox[i] =antes_pbox[23];break;/**/case 19:
apos_pbox[i] =antes_pbox[13];break;
               case 20: apos_pbox[i] =antes_pbox[31];break;/**/case 21:
apos_pbox[i] =antes_pbox[26];break;
               case 22: apos_pbox[i] =antes_pbox[2] ;break;/**/case 23:
apos_pbox[i] =antes_pbox[8] ;break;
               case 24: apos_pbox[i] =antes_pbox[18];break;/**/case 25:
apos_pbox[i] =antes_pbox[12];break;
               case 26: apos_pbox[i] =antes_pbox[29];break;/**/case 27:
apos_pbox[i] =antes_pbox[5] ;break;
               case 28: apos_pbox[i] =antes_pbox[21];break;/**/case 29:
apos_pbox[i] =antes_pbox[10];break;
               case 30: apos_pbox[i] =antes_pbox[3] ;break;/**/case 31:
apos_pbox[i] =antes_pbox[24];break;
          }
     }
     return apos_pbox;
}
```

# ANEXO B – Criptografia DES

## B.1 Aspecto geral

O algoritmo DES, como explicado anteriormente, possui como entrada um bloco de 64 bits. Este bloco irá passar por uma permutação inicial, que irá trocar a posição dos bits de uma maneira pré-definida pela Figura 48. Após a permutação inicial, o bloco é dividido em duas partes: esquerda e direita, cada um com 32 bits. Esses blocos irão passar por 16 iterações, chamadas de *rounds*, que consistem de operações de permutação, compressão, expansão e função lógicas XOR. Além da mensagem, deve-se escolher uma chave, também com 64 bits, que será utilizada ao longo do algoritmo para realizar a encriptação e decriptação; cada *round* gera uma sub-chave. Ao final destas iterações, o bloco resultante passa por uma permutação final, que é o inverso da permutação inicial. Este processo é denominado de encriptação e possui como resultado um ciphertext.

## B.2 Definir a chave e a mensagem

Primeiramente, é necessário escolher uma chave, que é o elemento que garante a segurança da informação. Diferentes chaves geram diferentes *Ciphertexts*, por isso a encriptação DES é denominada segura para certas aplicações; somente os detentores da chave podem ter acesso à mensagem.

Após é chave, é necessário definir a mensagem, ou *plaintext,* que é a entrada do algoritmo. Deve ter obrigatoriamente 64 bits. Um bloco de 64 bits é o equivalente a uma palavra de 8 bytes, ou seja, 8 caracteres. Caso o bloco da informação seja menor que 64 bits, devem-se utilizar métodos especiais que redimensionem para a quantidade correta.

## B.3 Permutação inicial

A permutação inicial da mensagem é feita trocando-se as posições dos bits em um vetor. A Figura 48 mostra este procedimento. Por exemplo, após a PI, o bit da posição 58 irá para a posição 1, o bit 50 para a posição 2 e assim por diante.

Figura 48 – Permutação inicial da mensagem

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

Fonte: Shneier (2015)

A permutação inicial da chave segue o mesmo padrão da PI da mensagem, entretanto, alguns bits são eliminados, para que a chave possua 56 bits. A Figura 49 mostra o padrão.

Figura 49 – Permutação Inicial da chave

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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, |
| 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 |

Fonte: Shneier (2015)

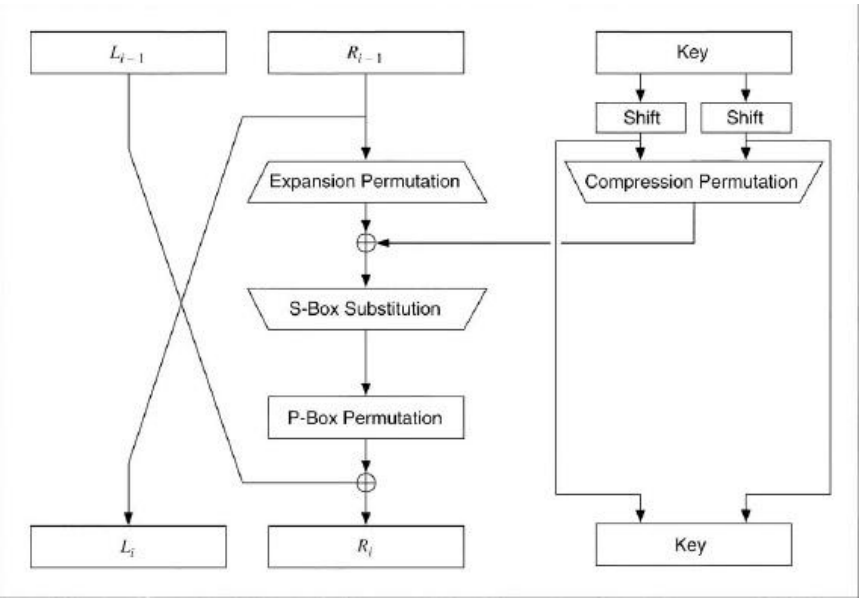
B.4 Divisão dos Blocos

Após a permutação inicial, o bloco resultante de 64 bits é dividido em 2 partes, um da esquerda e outra da direita, cada um com 32 bits. Este procedimento é feito para a mensagem e para a chave.

B.5 Iterações(ou *Round*s)

A criptografia DES é feita com 16 *round*s repetidos através de permutações e operações XOR entre os bits. A Figura 50 mostra o diagrama de um *round*.

Figura 50 – Fluxograma de um *round* da criptografia DES




Fonte: Shneier (2015)

No começo de um *round*, a chave possui 2 partes, esquerda e direita, cada uma com 28 bits. Esses blocos sofrem uma operação de rotação de bits para esquerda, podendo ser uma rotação de 1 ou 2 bits, dependendo do *round*. A Figura 51 a seguir, mostra a quantidade de bits rotacionados para cada *round*.

Figura 51 – Número de bits a serem rotacionados de acordo com o *round*

| Round | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Number | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Fonte: Shneier (2015)

Após o *Shift*, os blocos resultantes se unem em um bloco de 56 bits e passam por uma compressão, de 56 bits para 48. Essa compressão é feita através de uma permutação. A Figura 52 mostra os índices para cada bit permutado.

Figura 52 – Permutação de Compressão da chave

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

Fonte: Shneier (2015)

No início de cada *round*, o bloco da direita da mensagem passa por uma permutação de expansão, que irá transformar um bloco de 32 bits em um de 48. A Figura 53 mostra o resultado.

Figura 53 – Permutação de Expansão do bloco da direita

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

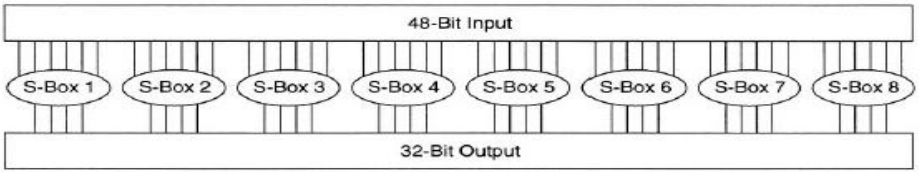
Fonte: Shneier (2015)

Os resultados da expansão do bloco direito da mensagem e a compressão da chave passam por uma operação XOR. Ambas as partes devem ter 48 bits.

O próximo passo é o bloco das substituições S-box. São consideradas uma etapa crucial da encriptação DES. Essas “caixas” são matrizes 4x16, projetadas para substituir valores através de um índice para linha e um índice para a coluna. O resultado da primeira XOR é um bloco com 48 bits, que será divido em 8 grupos de 6 bits. Estes grupos constituem a entrada de uma S-box, sendo que em sua saída, tem-se 8 grupos de 4 bits, totalizando 32 bits no total.

Para cada um dos 8 grupos, há uma S-box associada. Por exemplo, o grupo de bits b1 a b6 corresponde à S1, b7 a b13 à S2, assim por diante. A Figura 54 ilustra de maneira intuitiva as entradas e saída das S-box.

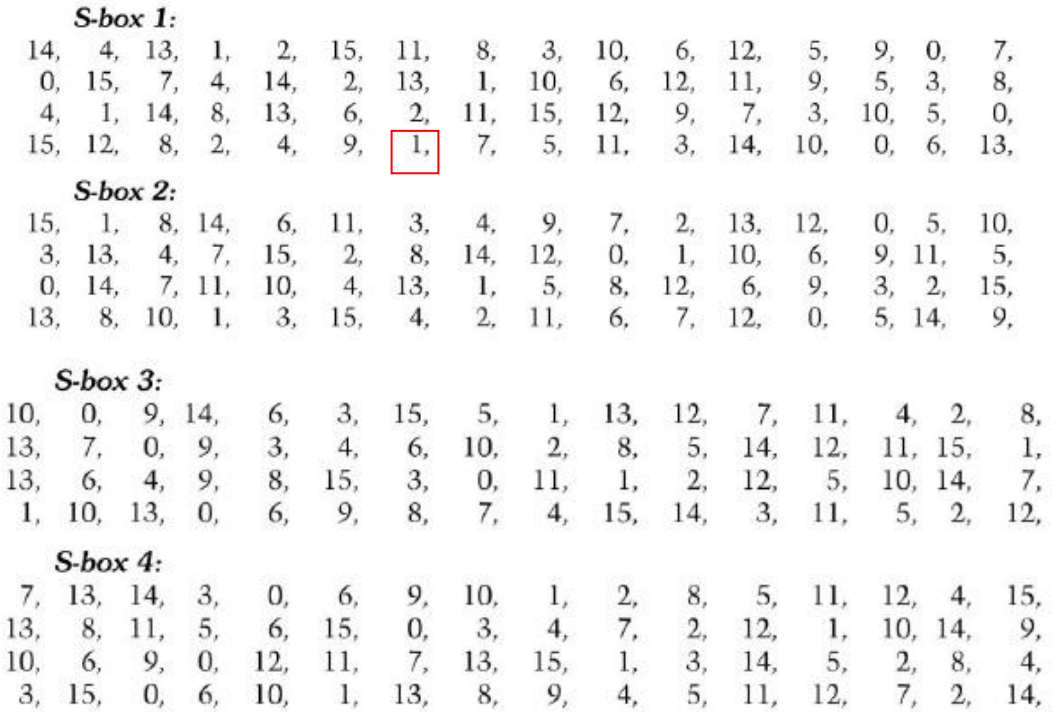
Figura 54 – Relação de entradas e saídas para as S-Box




Fonte: Shneier (1996)

O funcionamento de uma S-box é relativamente simples, entretanto, as substituições que ocorrem dentro de cada uma delas são funções não lineares e foram projetadas cuidadosamente para que o processo seja correto. Considerando o primeiro grupo de bits, b1 a b6, supondo que este grupo seja em binário: 101101. Separando-se os dois bits extremos, tem-se o conjunto 11, e o outro grupo seria 0110. Estes dois conjuntos correspondem aos índices para linha e coluna respectivamente, em outras palavras, eles apontam para uma localização da S-box. Portanto, tem-se para o grupo 1, o elemento correspondente a linha 3, e coluna 6, que são os valores decimais correspondentes para 11 e 0110. Tendo esses dois índices, basta procurar na matriz da caixa S1, o elemento correspondente, que no caso corresponde ao número 1. O elemento escolhido nada mais é que a saída da S-box, ou seja, 0001. Em suma, substituiu-se o grupo 101101 na entrada por 0001 na saída. Como são 8 saídas, tem-se um bloco resultante de 32 bits ao final desta operação. A Figura 55 a seguir mostra as matrizes de cada S-box.

Figura 55 – Elementos das matrizes de cada S-Box



*S-box 5:*

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

*S-box 6:*

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

*S-box 7:*

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

*S-box 8:*

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

Fonte: Shneier (1996)

A permutação P-box é feita com o bloco resultante das substituições das S-box. É um mapeamento de cada bit de entrada para uma posição de saída. A Figura 56 mostra as posições. Possui como resultado um bloco de 32 bits.

Figura 56 – Permutação P-Box

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

Fonte: Shneier (1996)

A segunda operação XOR utiliza o bloco resultante da P-box e a metade da esquerda da mensagem. O resultado desta operação é o bloco da direita do próximo *round*.

Ao final do *Round*, tem-se como a metade da esquerda o bloco da direita que foi utilizado no início do *round*, enquanto para o bloco da direita, tem-se o resultado da operação XOR entre a P-box e o bloco da esquerda inicial.

Ao final de 16 repetições, haverá um bloco da esquerda e da direita, que podem ser chamados de L16 e R16, que trocam de posição, antes de passar pela permutação final.

B.6 Permutação Final

A Permutação Final é a etapa que gera como resultado a mensagem encriptada. Esta etapa é exatamente o inverso da permutação inicial. A tabela que define a permutação final é dada na Figura 57.

Figura 57 – Permutação final

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

Fonte: Shneier (1996)

O Ciphertext é um bloco de 64 bits, assim como a mensagem, mas ininteligível. A única maneira de ver o conteúdo da mensagem é decriptando-a com a mesma chave utilizada na encriptação.

B.7 Decriptação

A decriptação é o processo inverso da encriptação. Utiliza como entrada a mensagem encriptada(ciphertext) e possui como saída a mensagem. A chave utilizada é a mesma, mas são utilizadas na ordem inversa. Por exemplo, a sub-chave do 16º *round* da encriptação será utilizada no 1º *round* da decriptação. A 15ª sub-chave no 2º e assim por diante. Ao final dos 16 *round*s, trocam-se os lados e realiza-se a permutação final, obtendo-se a mensagem original.

## ANEXO C – Mapeamento de caracteres do buffer de caracteres

Este mapeamento também pode ser encontrado no arquivo da nota de rodapé 11.

Tabela 3 – Mapeamento do *buffer* de caracteres

| Nº | Char | Nº | Char | Nº | Char | Nº | Char | Nº | Char |
|---|---|---|---|---|---|---|---|---|---|
| **0** | | 26 | → | 52 | 4 | 78 | N | 104 | h |
| **1** | ☺ | 27 | ← | 53 | 5 | 79 | O | 105 | i |
| **2** | ☻ | 28 | └ | 54 | 6 | 80 | P | 106 | j |
| **3** | ♥ | 29 | ↔ | 55 | 7 | 81 | Q | 107 | k |
| **4** | ♦ | 30 | ▲ | 56 | 8 | 82 | R | 108 | l |
| **5** | ♣ | 31 | ▼ | 57 | 9 | 83 | S | 109 | m |
| **6** | ♠ | 32 | | 58 | : | 84 | T | 110 | n |
| **7** | • | 33 | ! | 59 | ; | 85 | U | 111 | o |
| **8** | ◘ | 34 | " | 60 | < | 86 | V | 112 | p |
| **9** | ○ | 35 | # | 61 | = | 87 | W | 113 | q |
| **10** | ◙ | 36 | $ | 62 | > | 88 | X | 114 | r |
| **11** | ♂ | 37 | % | 63 | ? | 89 | Y | 115 | s |
| **12** | ♀ | 38 | & | 64 | @ | 90 | Z | 116 | t |
| **13** | ♪ | 39 | ´ | 65 | A | 91 | [ | 117 | u |
| **14** | ♫ | 40 | ( | 66 | B | 92 | \ | 118 | v |
| **15** | | 41 | ) | 67 | C | 93 | ] | 119 | w |
| **16** | ► | 42 | * | 68 | D | 94 | ^ | 120 | x |
| **17** | ◄ | 43 | + | 69 | E | 95 | _ | 121 | y |
| **18** | ↕ | 44 | , | 70 | F | 96 | ` | 122 | z |
| **19** | ‼ | 45 | - | 71 | G | 97 | a | 123 | { |
| **20** | ¶ | 46 | . | 72 | H | 98 | b | 124 | \| |
| **21** | § | 47 | / | 73 | I | 99 | c | 125 | } |
| **22** | ■ | 48 | 0 | 74 | J | 100 | d | 126 | ~ |
| **23** | ↕ | 49 | 1 | 75 | K | 101 | e | 127 | ⌂ |
| **24** | ↑ | 50 | 2 | 76 | L | 102 | f | | |
| **25** | ↓ | 51 | 3 | 77 | M | 103 | g | | |

## ANEXO D – Vetor de teste utilizado para validação do algoritmo

Figura 58 – Vetor de teste

**Appendix A Sample Round Outputs for the DES**

**INPUT**
**KEY = 10316E028C8F3B4A**
**PLAINTEXT = 0000000000000000**

| L | R |
|---|---|
| 00000000 | 47092B5B |
| 47092B5B | 53F372AF |
| 53F372AF | 9F1D158B |
| 9F1D158B | 8109CBEE |
| 8109CBEE | 60448698 |
| 60448698 | 29EBB1A4 |
| 29EBB1A4 | 620CC3A3 |
| 620CC3A3 | DEEB3D8A |
| DEEB3D8A | A1A0354D |
| A1A0354D | 9F0303DC |
| 9F0303DC | FD898EE8 |
| FD898EE8 | 2D1AE1DD |
| 2D1AE1DD | CBC829FA |
| CBC829FA | B367DEC9 |
| B367DEC9 | 3F6C3EFD |
| 3F6C3EFD | 5A1E5228 |

**OUTPUT**
82DCBAFBDEAB6602

Fonte: NIST (1998)