UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GUILHERME GOMES, LUAN KLEIN

RELATÓRIO PROJETO FINAL

Disciplina: **Lógica Reconfigurável**
Turma: **S71**
Curso: **Engenharia da Computação**
Universidade Tecnológica Federal do Paraná - UTFPR
Avenida Sete de Setembro, 3165 - Curitiba/PR, Brasil - CEP 80230-901

Curitiba
Dezembro de 2021

RELATÓRIO - SISTEMA DE CRIPTOGRAFIA UTILIZANDO REDE ETHERNET COM CYCLONE 10

December 14, 2021

1 Introdução

A utilização da criptografia se faz cada vez mais fundamental no mundo atual nos mais diversos ramos onde a segurança da informação se faz necessária. Entretanto, o processo de criptografar e descriptografar, a depender da técnica a ser utilizada, se faz bastante custoso em termos computacionais. Diante disso, o presente projeto tem por objetivo implementar um sistema de criptografia utilizando uma Cyclone 10, no qual o propósito é fazer com que a placa se comunique via Ethernet com um computador, receba uma mensagem, realize a operação de criptografia/descriptografia sobre a mensagem e devolva a nova mensagem para o computador. Além disso, também será buscado realizar a implementação de uma quebra de criptografia por força bruta.

O presente trabalho se encontra divido nas seguintes seções: Metodologia, que busca explicar o processo de desenvolvimento do projeto onde cada uma das subseções trata de um dos temas relevantes para o projeto; Resultados e discussões, que trata dos principais resultados obtidos do projeto e das dificuldades encontradas, e por fim a conclusão do projeto como um todo.

2 Metodologia

2.1 Criptografia

Existem diversos métodos de criptografia, dos quais muitos já tem seu código desenvolvido na linguagem VHDL[1], como por exemplo o RSA (Rivest-Shamir-Adleman), SHA (*secure hash algorithm* entre outros. Para o presente projeto foi desenvolvido um método próprio de criptografia, visando além de praticar e melhorar a programação em VHDL, ter mais controle e domínio sobre o sistema.

O método de criptografia desenvolvido consiste em receber um caractere codificado em binário (no presente projeto, considerou-se a codificação ASCII

[1] Disponíveis em: $https://opencores.org/projects?expanded=Crypto\%20core$

estendida[2], que codifica cada caractere em 8 bits), inverte a ordem dos bits e soma 1 no final. Como exemplo, através da criptografia desenvolvida, o caractere *a*, que codificado em ASCII estendido é 01100001 seria criptografado para 10000111, que representa o caractere ç. Para a decodificação de um caractere criptografado, basta realizar o processo inverso: subtrai-se 1 do caractere recebido e inverte-se a ordem dos bits.

2.2 Comunicação via Ethernet

A comunicação a ser implementada entre o Kit FPGA e o computador é via ethernet em uma rede local. Essa conexão será feita diretamente entre os dois diapositivos, sem nenhum aparelho intermediário, como switch ou roteador.

O kit a ser utilizado é a FPGA QMTECH que contém, entre outras coisas, uma cyclone 10 e também uma interface com a ethernet a partir do RTL8211EG da realtek. Essa interface recebe os dados via protocolo de comunicação GMII da FPGA e envia os dados pela porta RJ-45. Da mesma maneira, todos os pacotes recebidos são enviados para a FPGA via o mesmo protocolo. O CI de interface possui uma larga capacidade de velocidade de transmissão que alcança até 1Gbps.

O protocolo GMII é uma das variantes *media-independent interface* (MII), que é padronizado pelo IEE802.3u e faz a conexão de diferentes tipos de PHY para MAC. Um protocolo *media-independent* pode ser utilizado em qualquer tipo de dispositivos PHY, como por exemplo fibra ótica e cabo ethernet, comunicando com qualquer MAC sem a necessidade de realizar alterações no hardware do dispositivo. Pela padronização o MII e suas variantes fazem a comunicação baseada em frames, no qual consiste em delimitadores, *ethernet header*, dados referentes ao protocolo utilizado e o CRC.

2.3 Desenvolvimento

No desenvolvimento para o FPGA, para fazer a comunicação ethernet utilizou-se como base o exemplo que pode ser encontrado no site da QMTECH[3]. Este exemplo utiliza verilog como linguagem de descrição de hardware. Entretanto, ressalta-se que os resultados obtidos com o exemplo não foram satisfatórios. Mesmo seguindo de maneira minuciosa o tutorial não obteve-se o resultado esperado do exemplo. Em seguida alterou-se algumas das configurações do exemplo e realizou-se novamente alguns testes, e mesmo assim em nenhum momento obteve-se qualquer resultado satisfatório. Algumas hipóteses dos motivos desses problemas foram levantadas, como por exemplo o RTL8211EG pode estar danificado ou algum erro escondido no verilog que não pode ser facilmente detectado (considerando-se a falta de prática em verilog). Além disso, também buscou-se entender e encontrar possíveis falhas no circuito lógico, que pode ser visualizado na Figura 1, e a partir dele também não foi entendido o erro. É possível observar nessa figura que o grau de complexidade é bastante elevado, e sem uma

[2]https://desenvolvedorinteroperavel.wordpress.com/2011/09/11/tabela-ascii-completa/
[3]Disponível em: http://www.chinaqmtech.com/download$_fpga$

documentação eficiente, esse trabalho se mostrou uma tarefa demasiadamente complicada. Diante disso, a utilização da rede Ethernet foi abortada.

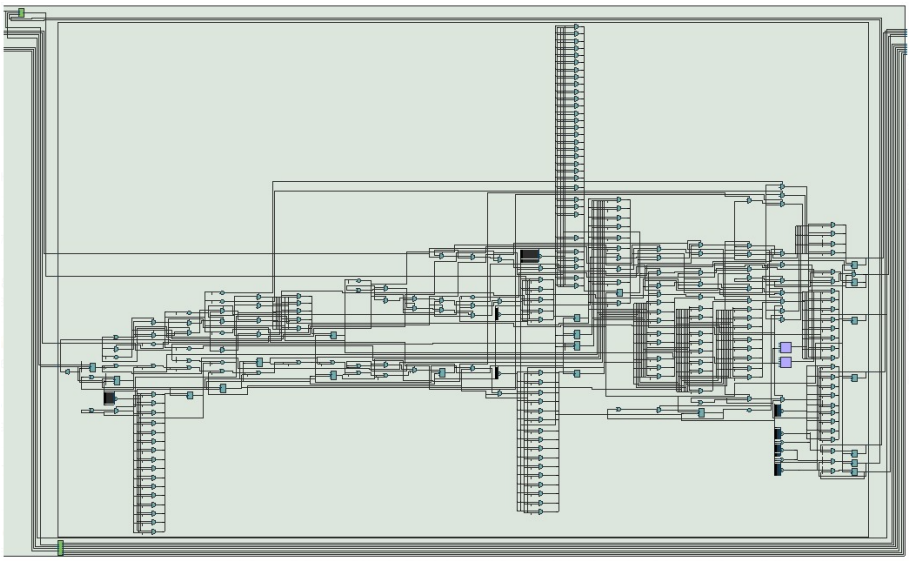

Figura 1: Circuito lógico referente ao exemplo do uso da Ethernet

Para o desenvolvimento de todos os processos envolvendo a criptografia, utilizou-se a linguagem VHDL para a construção de módulos específicos (como por exemplo, a codificação de um caractere em outro) e diagramas de bloco (esquemático) para a construção do funcionamento em geral de todo sistema (agregando todas as pequenas partes dele). Todos esses diagramas construídos serão explicados na seção de resultados.

3 Resultados e Discussões

3.1 Diagramas de Blocos, VHDL e Simulações

Uma vez que todo o processo de criptografia foi desenvolvido no Quartus 13, as simulações também foram realizadas utilizando o mesmo. Para a simulação da criptografia foi utilizado o diagrama de blocos presente na Figura 2, que é composto por duas memórias RAM (uma que contém a mensagem original e outra para conter a mensagem criptografada), um contador (para percorrer as memórias RAM's) e um bloco específico apenas para a criptografia. A principal entrada dessa simulação é o *datain*, que é um caractere da mensagem original, e a principal saída é a memória RAM2, que consiste nos caracteres da mensagem criptografada. O primeiro processo é realizar a inserção dos dados na memória RAM, e após, realizar a criptografia dos dados presentes na RAM, inserindo eles novamente em em outra memória RAM.

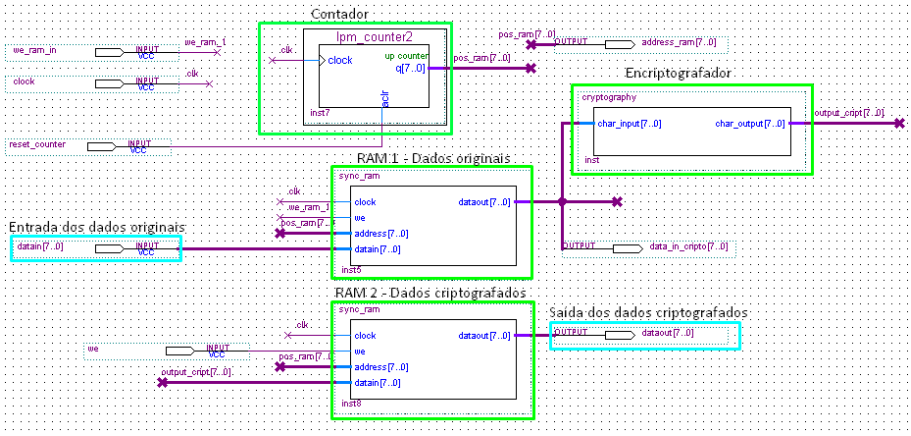

Figura 2: Diagrama de blocos construído para a simulação

Como exemplo de simulação, será criptografada a *string* "FPG@". Observando na tabela ASCII, tem-se a seguinte codificação:

- **F** : (70) - 01000110
- **P** : (80) - 01010000
- **G** : (71) - 01000111
- **@** : (64) - 01000000

Figura 3: Simulação da criptografia da mensagem FPG@.

Com isso, o resultado da simulação (presente no arquivo Waveform.vwf) da criptografia é apresentada na Figura 3, observa-se que a mensagem criptografada é:

- **c** : (99) - 01100011
- **Vertical Tab – VT** : (11) - 00001011
- **Ò** : (227) - 11100011
- **End of Text – ETX** : (03) - 00000011

Os demais sinais presentes na simulação da Figura 3 se referem a sinais necessários para o controle de inserção dos dados originais na memória RAM e posteriormente a criptografia e a nova inserção na RAM.

Para a simulação da descriptografia (presente no arquivo Waveform1.vwf), o diagrama de blocos é o mesmo apresentado na Figura 2, substituindo apenas o bloco de criptografia pelo de descriptografia. Como exemplo, será utilizada a mesma mensagem encontrada anteriormente pela criptografia. A simulação para a descriptografia pode ser visualizada na Figura 4. Com isso, observa-se que a mensagem é criptografada e descriptografada da maneira correta.

Figura 4: Simulação da descriptografia da mensagem encontrada anteriormente.

Por fim, há a parte da descriptografia por força bruta. Para tal, desenvolveu-se o diagrama de blocos presente na Figura 5. Observa-se que os elementos destacados em vermelho (MUX) e em verde (memória RAM) têm o papel de receber e armazenar a "resposta" da criptografia, ou seja, a mensagem correta que tentará ser descoberta pela força bruta.

Já as partes em azul (um contador para gerar o caractere a ser comparado, um comparador e um contador para controlar as posições das memórias acessadas) tem a função de gerar caracteres e comparar com o esperado, até que o valor correto seja encontrado. Por fim, em roxo está a memória RAM, que armazena essa mensagem descriptografada. Essa descriptografia por força bruta foi implementada considerando a quebra de um caractere por vez, e assim, ao final, será obtida a mensagem completa.

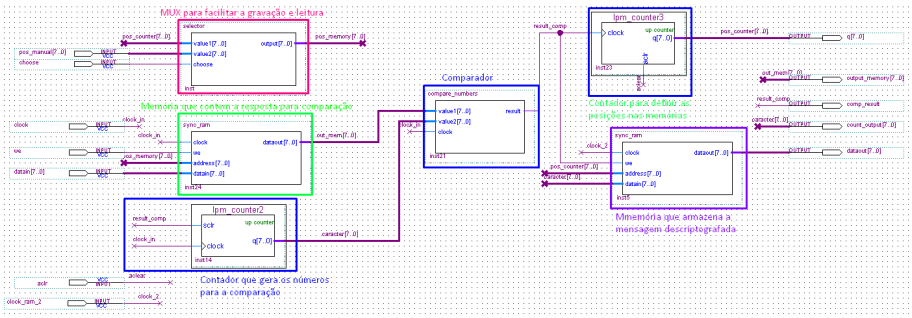

Figura 5: Diagrama de blocos da quebra de criptografia por força bruta.

Para a simulação da quebra de criptografia pela força bruta (presente no arquivo Waveform2.vwf), utilizou-se o diagrama de blocos presente na Figura 5. O resultado dessa simulação está presente na Figura 6, onde a parte em

vermelho se refere a inicialização da memória RAM (dado que a mesma precisa ser inicializada com os valores da mensagem correta, para poder realizar a comparação com os dados gerados pela força bruta). Já a parte em azul se refere aos resultados da comparação, entre o caractere real e o caractere que está sendo testado na força bruta. Quando ambos são iguais, esse valor é guardado em memória RAM e o *comp_result* tem o valor de 1. Observa-se que os valores na saída são os mesmos do que os da entrada e aparecem na mesma ordem. Dessa maneira é possível visualizar que a quebra da criptografia funciona.

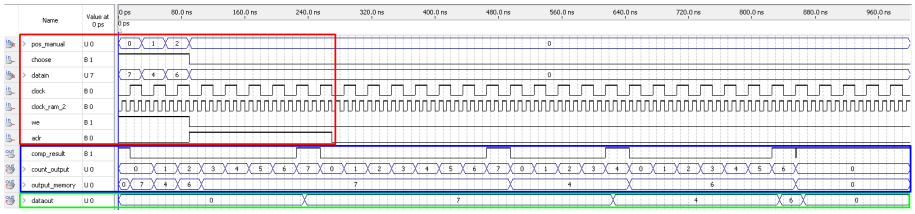

Figura 6: Exemplo da quebra de criptografia por força bruta.

3.2 Resultados extras

Junto com o exemplo do uso da Ethernet existe um exemplo para o HDMI que funcionou e nessa sessão será explicado o funcionamento desse exemplo. Essa explicação construída pode ser utilizado para outros casos, pois elucida a base do funcionamento do envio de dados de uma imagem para uma televisão ou monitor. Primeiramente sera mostrado sobre como funciona a imagem na televisão, como apresentado na Figura 7.

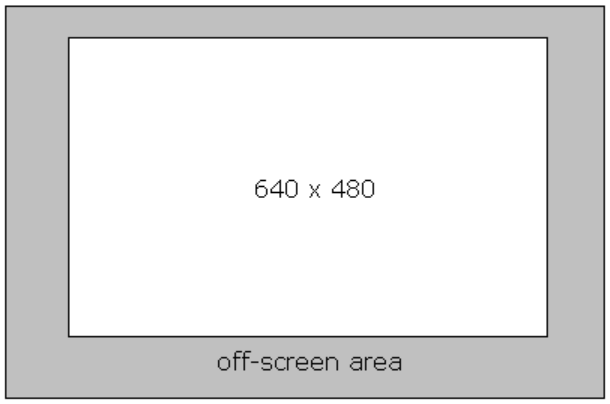

Figura 7: Área visível e a não visível da televisão.

Como pode ser observado na Figura 7, uma imagem de dimensões 640x480 pixels possui uma região escura, fazendo com que a quantidade real de pixels

necessários para essa imagem seja de 800x525. Essa quantidade extra é padrão na transmissão de imagens. A Figura 8 possui um *snippet* do código em verilog que faz a contagem dos pixels de 0 até 800 no contador X que se refere as colunas e 0 ate 525 no contador Y que se refere as linhas. Também pode ser visto que, no momento em que contadores indicam uma região visível, uma flag é setada, e que também há outras duas flags que são referentes ao sincronismo vertical (VSync), que mostra quando está chegando na última linha de pixels da tela, e sincronismo horizontal (HSync), que mostra quando está chegando na última coluna de pixels da tela.

```
////////////////////////////////////////////////////////////////////////
reg [9:0] CounterX, CounterY;
reg hSync, vSync, DrawArea;
always @(posedge pixclk) DrawArea <= (CounterX<640) && (CounterY<480);

always @(posedge pixclk) CounterX <= (CounterX==799) ? 0 : CounterX+1;
always @(posedge pixclk) if(CounterX==799) CounterY <= (CounterY==524) ? 0 : CounterY+1;

always @(posedge pixclk) hSync <= (CounterX>=656) && (CounterX<752);
always @(posedge pixclk) vSync <= (CounterY>=490) && (CounterY<492);

///////////////
```

Figura 8: Snippet do contador.

Para mostrar as cores de cada pixel basta ter um valor de 8 bits para cada canal do RGB. Como exemplo, para que um pixel seja branco basta colocar o valor 255 para vermelho, 255 para azul e 255 para verde no *snippet* da Figura 9. Nessa figura é possível visualizar que foi utilizado o valor dos contadores no momento para criar o RGB daquele pixel. No exemplo, existem dois elementos: A, que representa um retângulo (na região onde os 3 bits superiores do contador X e do contador Y são iguais a 2), e W que representa uma linha diagonal (quando o contador X é igual ao contador Y). Além disso, o retângulo definido por A terá uma coloração azul, enquanto que a linha diagonal indicada por W terá a cor branca. A cor de cada pixel pode ser obtida a partir de uma memória contendo uma imagem e assim transmitindo-a via HDMI.

```
///////////////
wire [7:0] W = {8{CounterX[7:0]==CounterY[7:0]}};
wire [7:0] A = {8{CounterX[7:5]==3'h2 && CounterY[7:5]==3'h2}};
reg [7:0] red, green, blue;
always @(posedge pixclk) red <= ({CounterX[5:0] & {6{CounterY[4:3]==~CounterX[4:3]}}, 2'b00} | W) & ~A;
always @(posedge pixclk) green <= (CounterX[7:0] & {8{CounterY[6]}} | W) & ~A;
always @(posedge pixclk) blue <= CounterY[7:0] | W | A;

////////////////////////////////////////////////////////////////////////
```

Figura 9: Snippet da cor do pixel.

Agora será explicado um pouco sobre a velocidade da transmissão necessária para transmitir os pixels. Primeiramente, para transmitir um pixel é necessário enviar 8 bits de cada cor formando assim 24 bits por pixel. Em uma tela de 800x525 e com uma atualização de 60 quadros por segundo (60Hz) seria necessário enviar 0,44 Gbps de informação útil, que é utilizada na área visível. Na transmissão via HDMI existe um *clock* chamado *pixel clock*, que pode ser visualizado na Figura 10. Porém a transmissão não é de 8 bits porque além

deles, existem 2 bits de controle. Logo, é necessário a utilização de 10 bits, além de realizar o embaralhamento dos dados, como pode ser observado na Figura 11. Existem algumas fórmulas para calcular o valor do *pixel clock*, porém, todas as calculadas para esse tipo, que é pequeno, resultarem em valores abaixo de 25 MHz. Por padrão, o mínimo aceito por HDMI é 25 MHz, logo o *pixel clock* foi setado para esse valor padrão. Como a cada *pixel clock* é preciso ter 10 bits, o clock que irá estar nos canais de dados será de 250 MHz. Para chegar nesse clock é necessário criar um PLL para passar o clock da placa de 50 MHz para os 250MHz necessários.

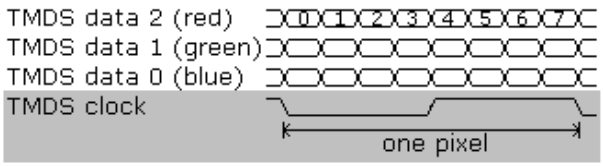

Figura 10: Pixel clock com 8 bits.

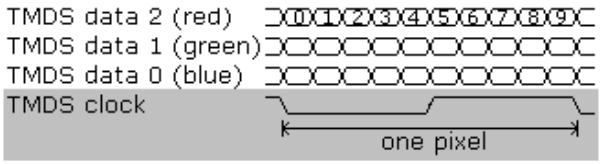

Figura 11: Pixel clock com 10 bits.

Por final o resultado obtido com esse exemplo é mostrado na Figura 12, onde é possível observar a linha branca diagonal e também o retângulo em azul, e as cores condizendo com o que foi colocado no código referente a posição do pixel na tela.

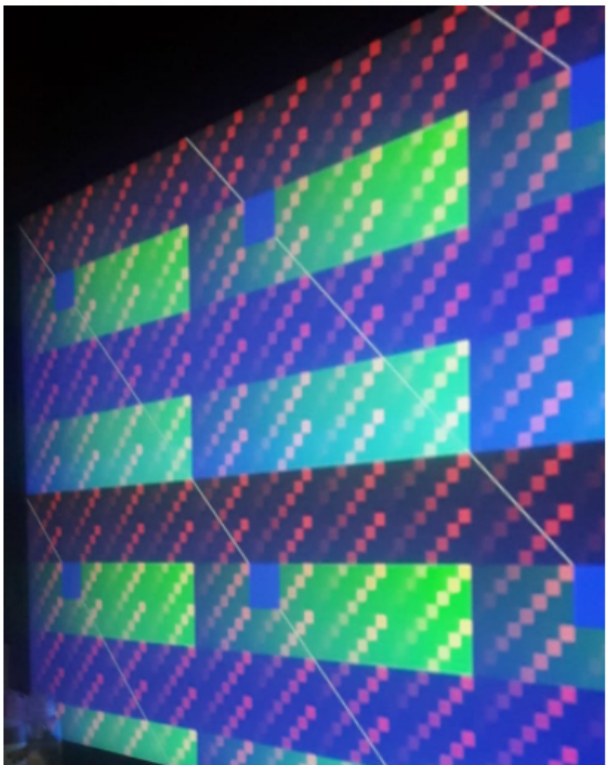

Figura 12: Imagem gerada na FPGA e recebida pela televisão via HDMI.

3.3 Dificuldades

Ao longo da construção do projeto, diversas dificuldades apareceram, necessitando de esforço e tempo para encontrar soluções plausíveis para tais. A seguir estão listados os principais empecilhos:

- **Utilização do Quartus 21.1 e 18.1:** Dado que o projeto consistia na utilização do kit que continha a Cyclone 10, foi necessário a utilização de uma versão do software Quartus adequada e compatível. Entretanto, não foi possível realizar simulações em nenhum desses softwares, impossibilitando assim o desenvolvimento neles. Um dos erros que ocorreu ao tentar realizar a simulação com o Quartus 21.1, foi que o simulador utilizado por ele não era o ModelSim (utilizado em outras versões) e sim o Questa, e ocorria alguns erros de licenças. **Solução:** Para a implementação e simulação de todo o projeto utilizou-se o Quartus 13, dado que todo o ferramental necessário neste funcionava, mesmo que está versão não fosse compatível com a placa. Em tese, com o projeto todo implementado e funcionando, bastaria migrar o código para outra versão do Quartus, e assim utilizar a Cyclone 10.

- **Exponenciação:** Inicialmente, o planejamento era utilizar um algoritmo de criptografia mais complexo, utilizando grandes exponenciações, como é o caso no RSA. Entretanto, não encontrou-se uma maneira eficiente e simples o suficiente para a realização. Além disso, era necessário que todo código fosse sintetizável, dado que o plano inicial era a real implementação dele em uma FPGA. **Solução:** Dado o pouco tempo para a construção da solução, buscou-se utilizar um algoritmo mais simples de criptografia que não utiliza-se exponenciação, para evitar possíveis erros que seriam decorrentes disso e que o objetivo seria apenas implantar um método de criprografia, mesmo que simples.

- **Falta de Documentação do kit de desenvolvimento:** A placa de desenvolvimento da QMTECH é recente então não há muito conteúdo na internet sobre ela. Essa empresa que desenvolveu a placa também não deu atenção devida para a documentação, deixando muitas dúvidas sobre como utilizar. Como foi o caso com o exemplo ethernet, que em nenhum momento explica sobre o software utilizado deixando a cargo do desenvolvedor entender o que acontece no código sozinho. **Solução:** Estudou-se o código o mais afundo afundo possível, porém, devido a uma linguagem nova e ser uma experiencia nova na análise e correção de erros em FPGA. Isso acabou acarretando em demasiadas complicações levando ao impedimento da utilização da comunicação via Ethernet. Para tentar compensar, buscou-se trazer informações sobre o processo da utilização de HDMI na mesma placa.

4 Conclusões

Diante dos resultados, constata-se que o objetivo inicial do trabalho não foi atingido devido aos problemas encontrados com a utilização da rede Ethernet com a placa. Isso se deve por uma gama de fatores distintos, e dentre eles destacam-se à dificuldade por se tratar de uma placa nova, da pouca documentação disponível, da baixa experiência do grupo no tratamento desse tipo de equipamento e do pouco tempo disponível para a construção de todo o projeto.

Por outro lado, a implementação dos módulos de criptografia foi realizado de maneira exitosa. Observa-se que através dos três diagramas de blocos implementados, é possível criptografar, descriptografar e descriptografar por força bruta uma mensagem. Vale ressaltar que essa última foi construída utilizando um modelo de quebra de criptografia por caractere, onde cada um dos caracteres é encontrado de maneira isolada. Para a construção das simulações foi necessário realizar alguns ajustes, como por exemplo, inicializar "na mão" a memória RAM com a mensagem a ser descriptografada. Essa necessidade surgiu devido a falta da rede Ethernet, que teria o papel de carregar a mensagem na memória.

Buscando compensar a falta da utilização do kit de desenvolvimento, pela falha na Ethernet, estudou-se o funcionamento do HDMI e documentado as funcionalidades bases para conseguir enviar uma imagem via HDMI para um

monitor ou televisão, baseando-se no exemplo que é dado pela QMTECH.

Por fim, é possível concluir que a realização dessa atividade foi bastante proveitosa para a elucidação dos conhecimentos adquiridos em aula, como por exemplo programação em VHDL. Mesmo sem ter alcançado o objetivo inicial do projeto, foi possível obter resultados significativos com a construção do mesmo, como por exemplo a construção dos módulos de criptografia e a melhoria da documentação do uso do HDMI da placa.