Universidade Federal
de Campina Grande

Lucas Danrley Cajé de Souza

Relatório de Estágio Supervisionado: *Laboratório de Excelência em Microeletrônica do Nordeste - XMEN*

Campina Grande, PB

2021

Lucas Danrley Cajé de Souza

Relatório de Estágio Supervisionado: *Laboratório de Excelência em Microeletrônica do Nordeste - XMEN*

Relatório de estágio supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, *Campus* Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Universidade Federal de Campina Grande – UFCG

Centro de Engenharia Elétrica e Informática

Departamento de Engenharia Elétrica

Orientador: Gutemberg Gonçalves dos Santos Júnior

Campina Grande, PB

2021

Lucas Danrley Cajé de Souza

Relatório de Estágio Supervisionado: *Laboratório de Excelência em Microeletrônica do Nordeste - XMEN*

Relatório de estágio supervisionado submetido à Coordenação de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande, *Campus* Campina Grande, como parte dos requisitos necessários para obtenção do título de Graduado em Engenharia Elétrica.

Trabalho aprovado. Campina Grande, PB, 04/10/2021.

Gutemberg Gonçalves dos Santos Júnior
Orientador

Marcos Ricardo Alcântara Morais
Convidado

Campina Grande, PB
2021

Este trabalho é dedicado a minha família.

Agradecimentos

Agradeço aos meus pais, Edésio e Liduina, pela dedicação, sacrifício e apoio incondicional ao longo da minha formação. Agradeço a minha irmã, Yargles, por fazer parte e me apoiar na minha jornada.

Sou grato a todos os amigos e familiares que direta ou indiretamente me ajudaram na conclusão desse objetivo em minha vida. Aos amigos do IFRN, UFCG e Grenoble INP-ESISAR, isso não seria possível sem o apoio e o companheirismo de todos.

Por fim, agradeço ao professor Gutemberg Gonçalves e aos engenheiros Hugo Gayoso e Gabriel Villanova, pela ajuda e por tornarem possível a realização desse estágio.

Resumo

Este relatório descreve atividades realizadas pelo aluno Lucas Danrley Cajé de Souza durante o Estágio Supervisionado no Laboratório de Excelência em Microeletrônica do Nordeste - XMEN do Departamento de Engenharia Elétrica (DEE) da Universidade Federal de Campina Grande (UFCG), sob a orientação do Professor Gutemberg Gonçalves dos Santos Júnior. O estágio consistiu na realização de testes e na validação das funcionalidades de um *System-on-a-Chip* (SoC) desenvolvido no laboratório. O *chip* foi desenvolvido com base na arquitetura RISC-V para utilização de aplicações em *Domestic Internet of Things* (DIoT), possuindo módulos AES, e protocolos de comunicação I2C e SPI por exemplo.

Palavras-chaves: Desenvolvimento de *hardware*. Microeletrônica. Validação. RISCV.

Abstract

This report presents the activities developed by the student Lucas Danrley Cajé de Souza during his Supervised Internship at the Laboratório de Excelência em Microeletrônica do Nordeste - XMEN located in the Department of Eletrical Engineering (DEE) of the Federal University of Campina Grande (UFCG), under the guidance of professor Gutemberg Gonçalves dos Santos Júnior. The work consists of testing and validating the functionalities of a System-on-a-chip developed by the laboratory. The chip was developed based on RISCV architecture for use in Domestic Internet of Things (DIoT) applications, featuring AES module, and I2C and SPI communication peripherals, to list a few.

Key-words: Hardware Development. Microeletronics. Validation. RISCV.

Lista de ilustrações

Lista de abreviaturas e siglas

SoC	*System on Chip*
DES	*Data Encryption Standard*
IP	*Internal Peripheral*
ROM	*Read-Only Memory*
RAM	*Random Access Memory*
I^2C	*Inter-Integrated Circuit*
PCB	*Printed Circuit Board*
EVB	*Evaluation Board*
ULA	*Unidade Lógica Aritmética*

Sumário

1 Introdução

Este relatório apresenta as atividades realizadas pelo aluno Lucas Danrley Cajé de Souza durante a disciplina de Estágio Supervisionado no Laboratório de Excelência em Microeletrônica do Nordeste - XMEN do Departamento de Engenharia Elétrica (DEE) da Universidade Federal de Campina Grande (UFCG), mediante orientação do professor Gutemberg Gonçalves dos Santos Júnior. As atividades ocorreram entre o período de 28 de junho de 2021 a 17 de setembro de 2021, com carga horária de 20 horas semanais, totalizando 234 horas.

O laboratório XMEN, surgiu da iniciativa PEM - Programa de Excelência em Microeletrônica, realizada em 2016 e tinha como objetivo a capacitação de alunos do curso de graduação no domínio da microeletrônica. Por meio de aulas e treinamentos, os alunos faziam parte do fluxo de desenvolvimento de projetos de alto nível, aproximando-os do mundo profissional.

Atualmente o laboratório conta com a participação de discentes dos níveis de graduação, pós-graduação e engenheiros. Graças ao êxito nos seus primeiros projetos, além de continuar promovendo capacitação o laboratório também desenvolve diversos projetos para outras empresas, proporcionando um ambiente de crescimento tecnológico no nordeste do Brasil.

Um dos primeiros projetos efetuados no XMEN trata-se do PLC-SoC Chilipe, um *System-on-a-Chip* baseado em arquitetura RISC-V para aplicações em *Domestic Internet of Things* (DIoT). O *chip* contém diferentes módulos, chamados IPs (*Internal Peripherals*), que implementam diferentes funcionalidades no sistema, como por exemplo: núcleo baseado em RISC-V RI5CY, SPI, I2C, UART, Timer, AES, Viterbi, Reed-Solomon e PLC. Esse sistema foi inteiramente desenvolvido no laboratório.

O principal objetivo deste estágio é a realização de testes das funcionalidades desse SoC, tanto a nível de simulação como a nível de *hardware*. Para tanto, uma placa de circuito impresso foi desenvolvida e produzida, com o intuito de receber o chip e garantir acessibilidade à suas funcionalidades. Ademais, diferentes *firmwares* foram desenvolvidos afim de explorar os diferentes recursos da placa.

1.1 Plano de estágio

As atividades previstas para execução no plano de estágio e cumpridas ao longo dos meses foram:

- Design da placa de circuito impresso;
- Simulações das funcionalidades do sistema;
- Realização de testes de bancada em placas de circuito impresso e microprocessadores;
- Elaboração de documentação.

1.2 Organização do Trabalho

No capítulo 2 serão apresentados os principais conceitos acerca dos módulos e recursos presentes no SoC e que foram testados. No capítulo 3 serão apresentadas as atividades desenvolvidas ao longo do estágio. Por fim, no capítulo 4, serão apresentadas as considerações finais.

2 Embasamento Teórico

Nessa capítulo serão apresentados os conceitos fundamentais de alguns dos módulos e funcionalidades presentes no sistema e que foram simulados.

2.1 *Advanced Encryption Standard* (AES)

O AES é um dos algoritmos de cifra simétrica mais utilizados atualmente. Ele surgiu a partir de um concurso do NIST (Instituto Nacional de Padrões e Tecnologia dos EUA) que buscava uma melhor alternativa para o DES (*Data Encryption Standard*), que tinha chave menor. Esse algoritmo pode cifrar/decifrar blocos de 128 *bits* com chave (*key*) de 128, 192 ou 256 *bits*. A saída *texto cifrado* na Figura 1 é a saída do algoritmo, neste caso a informação encriptada.

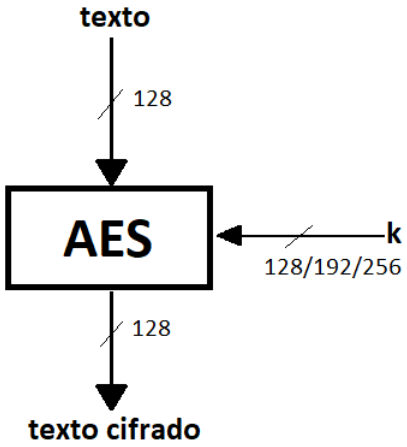

Figura 1 – Parâmetros de entrada e saída do algoritmo AES. Fonte: Elaborada pelo autor.

No caso do SoC utilizado no estágio, as chaves utilizadas são de 128 bits. No processo realizado pelo algoritmo, a informação é transformada em uma matriz de 16 *bytes*, assim como a chave. As operações serão realizadas em N vezes (N *rounds*), onde N depende do tamanho da chave, no caso desse SoC N=10 já que são 128 *bits*. A Figura 2 apresenta o fluxo do algoritmo:

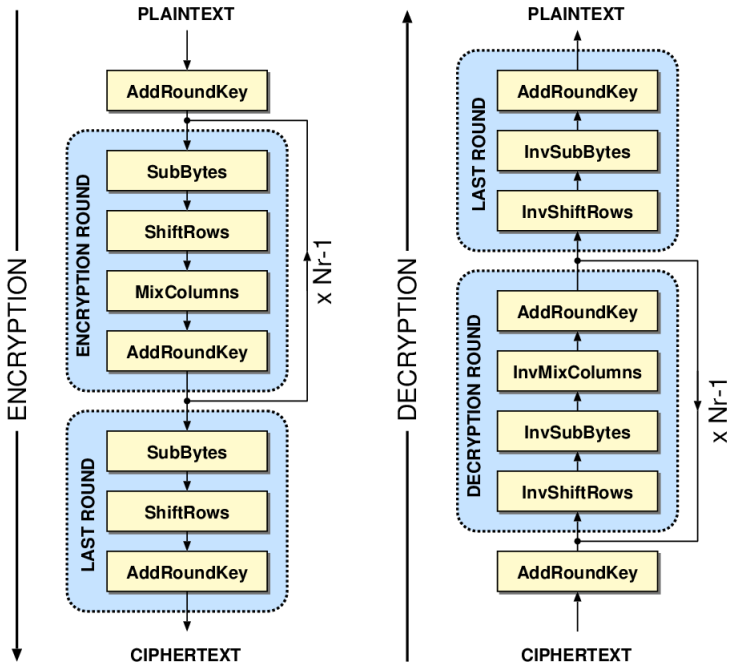

Figura 2 – Diagrama de blocos do algoritmo AES para os modos de encriptação e decriptação. Fonte: [7].

Em cada round são realizadas operações matriciais com os dados e com a chave com a finalidade de encriptar a informação. Essas operações são substituições de dados, com padrões definidos, difusão de linhas, mistura de colunas e por último um ou exclusivo (xor). No último round, a etapa de misturar as colunas não é feita. Em cada round, a chave inicial também é modificada.

No que diz respeito a otimização do algoritmo em diferentes situações, modos de cifra (ou de operação) podem ser empregados, ajudando a aumentar a segurança do algoritmo. Os modos de operação que podem ser implementados no *chip* são: ECB, CBC, PCBC, CFB, OFB e CTR. Em alguns desses modos de operação, um vetor de inicialização (IV) é requerido para somar com o primeiro bloco de dados, como é o caso do modo CBC.

2.2 *Universal Asyncronous Receiver-Transmitter* (UART)

UART é um tipo de protocolo de comunicação serial assíncrono, o que significa que não possui sinal de *clock*. Nesse sistema, dois dispositivos se comunicam diretamente por meio de dois fios: Rx para recepção dos bits e Tx para transmissão.

Como nesse protocolo não é utilizado um sinal temporal, são adicionados *bits* de início e fim para que seja conhecido quando a comunicação se inicia e quando todos os dados foram enviados. Além disso, o tamanho da mensagem enviada e a frequência de transmissão (*baud rate*) podem ser configurados, deixando os dois dispositivos com as mesmas configurações.

2.3 *Inter-Integrated Circuit* (I2C)

I2C é um protocolo de comunicação que utiliza um barramento serial de apenas dois fios (SCL – *clock* e SDA - dados). Por meio desse protocolo é possível estabelecer uma comunicação entre múltiplos dispositivos, chamados *masters* ou *slaves*.

Para se comunicar (enviar ou ler dados) o dispositivo *master* deve enviar um comando com o endereço do *slave* que deseja estabelecer uma conexão pelo barramento de dados, e o sinal de *clock* numa frequência típica de 100kHz, mas que pode ser maior a depender da aplicação. Assim, somente o dispositivo especificado irá responder, tornando possível o uso de diferentes componentes no mesmo barramento. O funcionamento desse protocolo pode ser melhor entendido com ajuda da figura abaixo:

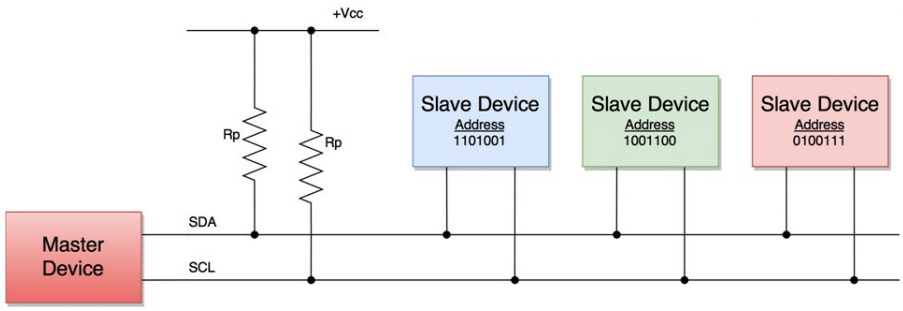

Figura 3 – Barramento I2C. Fonte: [9].

Cada dispositivo *slave* possui um endereço diferente. Cada mensagem enviada pelo barramento possui um *frame* com o endereço do dispositivo e um com o dado que está sendo enviado. Além disso possui um *bit* que especifica a leitura ou escritura, um bit de inicialização e um bit de ACK/NACK, que é retornado pelo receptor quando o *frame* de dados foi recebido [5].

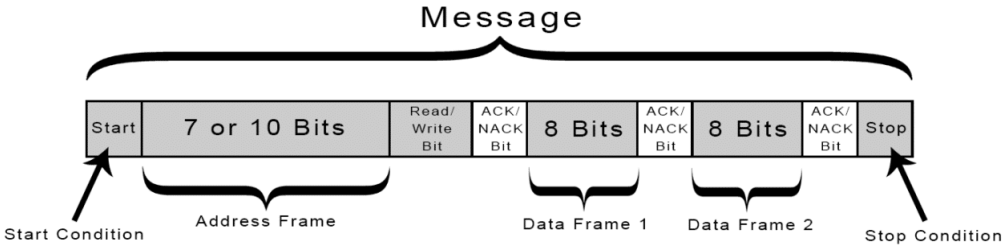

Figura 4 – Blocos de mensagem enviada via I2C. Fonte: [5].

2.4 *Serial Peripheral Interface* (SPI)

SPI é outro protocolo de comunicação síncrono e utiliza, além do *clock*, mais três sinais: MOSI (*master output – slave input*), MISO (*master input – slave output*) e CS

(*chip select*, para selecionar qual dispositivo *slave* vai receber os dados). Com esse sistema é possível transmitir e receber informações ao mesmo tempo, diferente do I2C.

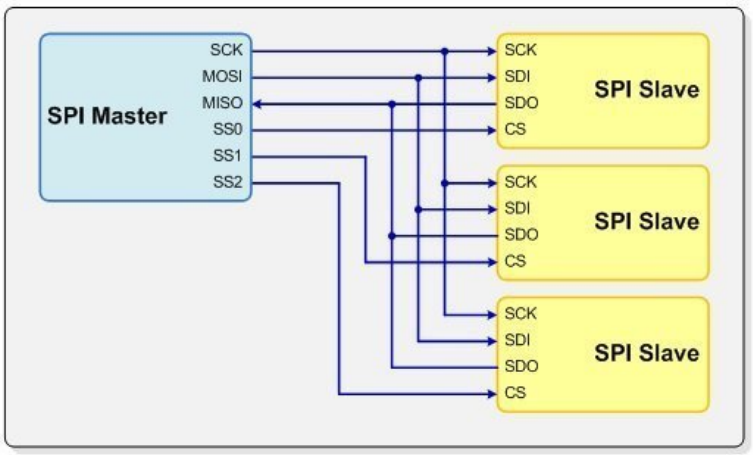

Figura 5 – Barramento SPI. Fonte: [3].

O dispositivo *master* pode selecionar para qual *slave* se deseja estabelecer uma comunicação por meio do CS e somente este irá responder. O dado é enviado *bit* por *bit* do *master* para o *slave* pelo fio MOSI e do *slave* para o *master* pelo MISO. Assim como no I2C, esse sistema também permite a conexão de diferentes dispositivos em paralelo.

2.5 *General Purpose Input/Output* (GPIO)

GPIO's são pinos genéricos que podem ser usados como entrada ou saída de dados. Seu comportamento (entrada ou saída) pode ser definido via software e eles podem assumir valores HIGH ou LOW. Esses componentes agregam ao sistema a possibilidade de serem configurados para diferentes situações, permitindo a troca de informações, controle ou conexão com diferentes dispositivos externos.

Um *firmware* foi construído para testar a funcionalidade de alguns GPIO's presentes no SoC, como será apresentado no próximo capítulo.

3 Atividades Desenvolvidas

As atividades ao longo do estágio foram desenvolvidas de maneira remota em virtude das condições causadas pela pandemia do Covid-19. Os trabalhos realizados são descritos a seguir.

Em uma primeira etapa foi desenvolvida uma placa de circuito impresso auxiliar onde o SoC seria conectado. Em um segundo momento foram realizadas simulações com objetivo de verificar o funcionamento do *chip* e prever o seu comportamento para que possa ser comparada com os resultados dos testes em *hardware*. Inicialmente foram detectados problemas com a versão da *toolchain* utilizada, uma vez resolvidos foi possível gerar as diferentes simulações.

3.1 Placa de Circuito Impresso (PCB)

O CI utilizado não possui memória *flash* interna. Assim, para que o programa pudesse ser carregado nele seria necessária uma memória externa. Além disso, outros componentes se fazem necessários para acessar seus recursos, tais como um adaptador do tipo *socket QFP64*, *level shifters*, pinos do tipo *headers* e um sistema de *reset* da placa. Dessa forma torna-se viável a construção de uma PCB.

A PCB foi confeccionada utilizando o *software* Kicad 5 no ambiente Windows. Para isso, foi utilizada a primeira versão projetada pela equipe do laboratório anteriormente. Nesse processo, partiu-se do esquema elétrico da placa já com todos os componentes e conexões necessárias para o correto funcionamento, para a confecção do *layout*.

3.1.1 Esquema elétrico

O esquema elétrico da placa é apresentado no Anexo 1. Como mencionado, estão presentes alguns componentes necessários para que seja possível explorar os recursos e funcionalidades do SoC. Conectores do tipo *pin headers* foram adicionados para verificação, por exemplo, do funcionamento dos protocolos UART, SPI e I2C. A memória *flash* utilizada foi a do modelo IS25LP128 de 128 Mb, onde, para sua programação, foram adicionados pinos de mesmo modelo.

Já com relação ao I2C, foi acrescentado um componente do tipo conversor bi-direcional PCA9306, afim de realizar uma mudança de nível de tensão, garantindo o funcionamento do protocolo. De maneira semelhante, para todos os outros pinos de comunicação (UART e SPI) e pinos GPIO, também foi acrescentado componentes do tipo *level shifters*, com a mesma finalidade do anterior.

Para explorar os recursos do PLC, todos os pinos dessa funcionalidade foram ligados a conectores externos. O sistema de *reset* da placa é feito por meio de um botão que permite de maneira forçada uma mudança de nível de 1 para 0 no pino de *RSTn* do SoC, reinicializando a aplicação.

3.1.2 *Layout*

Para o *layout* da placa de circuito impresso foram utilizados apenas duas camadas (superior e inferior), ambas com planos de terra (GND). As trilhas de cobre utilizadas foram de 0.25 mm com isolação de 0.2 mm. As *footprints* utilizadas foram adquiridas dos sites dos revendedores dos componentes, como por exemplo Mouser Electronics. Dessa forma não corria-se o risco de adquirir um componente e no projeto utilizar uma *footprint* diferente.

Para fabricação do protótipo foi escolhido, junto a um dos engenheiros responsáveis pelo projeto, a empresa PCBWay. Essa empresa tem sede na China e é uma das líderes mundiais em fabricação de PCBs. Para fabricação do produto se faz necessário o envio dos seguintes aquivos:

- Arquivos *Gerber*: contém as informações técnicas para fabricação de uma PCB, tais como dimensões das pistas de cobre e isolação e *footprints*.
- *Bill of Material* (BOM): contém as referências de cada componente do circuito, tais como fabricante, número de fabricação e tipo da *footprint*.
- *Centroid position*: arquivo que contém as posições e orientações dos componentes presentes na placa.

Normalmente todos esses arquivos são fornecidos pelo *software* utilizado, neste caso o Kicad. Uma vez prontos, eles podem ser enviados para fabricação. Na Figura 6 é apresentado o *layout* desenvolvido no *software* e na Figura 7 o resultado do projeto. Na fase atual, a empresa finalizou a fabricação e é aguardada a chegada desses protótipos no laboratório.

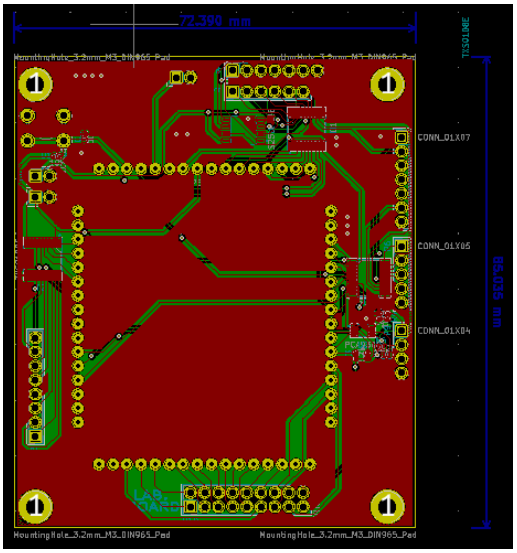

Figura 6 – PCB desenvolvida no ambiente Kicad. Fonte: Elaborada pelo autor.

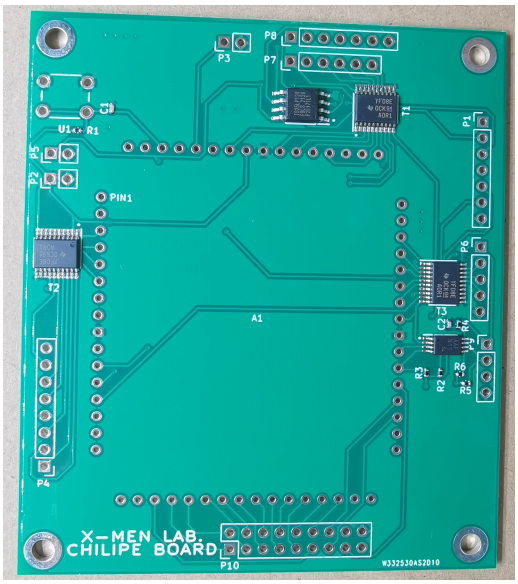

Figura 7 – PCB desenvolvida para testes. Fonte: Enviada por PCBWay.

3.2 Simulações

O objetivo de desenvolver simulações é prever o comportamento do sistema em questão antes de testá-los em hardware. Assim, é possível verificar se ele está ou não funcionando corretamente comparando os resultados de simulação com os dados reais.

Para isso, a equipe de engenheiros do laboratório já possuía alguns códigos em linguagem C, também chamados *firmwares*, que tem como objetivo acessar em alto nível os recursos do SoC.

Para executar uma simulação foram utilizados inicialmente esses códigos no ambi-

ente Linux CentOS. Nesse ambiente se faz necessário a instalação de uma *toolchain*, que é um conjunto de ferramentas de compilação dos códigos pelo processador em questão. No caso desse projeto, o núcleo do processador tem a base no projeto PULPino desenvolvido pela universidade *ETH Zürich* e da *University of Bologna*. Assim, essas instituições já fornecem esse conjunto de ferramentas adaptados para esse processador na plataforma *GitHub* disponível em [6].

Um problema encontrado no inicio do estágio foi justamente na geração dessas simulações devido à incompatibilidade entre a última versão da *toolchain* disponibilizada e a versão do *hardware* na qual o *chip* foi baseado. O que ocorre é que essas instituições realizam constantemente melhorias em seus projetos e atualizações são necessárias. Como o núcleo do *chip* foi baseado em uma versão mais antiga, o seu *core* não reconhecia algumas diretivas de compilação.

O problema foi resolvido importando a versão da *toolchain* antiga de outra máquina utilizada anteriormente. Depois de resolvido esse problema, as simulações podiam ser compiladas e geradas. Os resultados são analisados com ajuda do *software* SimVision da Cadence.

Ao executar qualquer *firmware* o resultado deve ser a criação de um arquivo no formato hexadecimal contendo todo o programa que será armazenado na memória flash e assim lido pelo *chip*.

3.2.1 *demo-aes.c*

A *demo-aes.c* implementa os diferentes modos de operação citados no capítulo anterior. As variáveis que especificam o valor de cada modo de operação estão definidas nas bibliotecas do projeto (*pem_aes.h*), assim como o escopo das funções necessárias para sua implementação. As principais funções são *AES_init*, *AES_writeKey*, *AES_writeData*, *AES_writeIV*. Cada uma delas escreve os valores passados como referência no registrador específico.

No programa principal da aplicação desenvolvida são escritas inicialmente a chave e o vetor de inicialização, que são constantes. Todos os valores são passados em palavras de 32 *bits*. Após isso, é possível testar cada modo de operação com uma função geral. Em cada chamada dessa função são feitos os mesmos cálculos para dois valores constantes: DATA1, que vale 00112233445566778899AABBCCDDEEFF e DATA2, que vale FFEEDDCCBBAA99887766554433221100, além das configurações que especificam:

- a inicialização da operação;
- qual operação será realizada (encriptação ou decriptação) e;
- o modo.

O resultado é aguardado usando a função *AES_getDone*, que retorna *true* caso a operação esteja pronta, seja de encriptação ou decriptação. Uma tabela com o resultado esperado de acordo com os diferentes modos de operação foi construída:

Modo	**Primeiro bloco cifrado**	**Segundo bloco cifrado**
ECB	69C4E0D86A7B0430D8CDB78070B4C55A	1B872378795F4FFD772855FC87CA964D
CBC	F6217F61B2D50A6AE6791f8C384B1E07	00E6F7C3F089B33AF8D701BEB170AF82
PCBC	F6217F61B2D50A6AE6791f8C384B1E07	1766EA65883AE0BE5DE323B1431CBD54
CFB	79C571F93E6E91590576009881A3AB8E	3F778AF1BA19E580C751137195BCFF23
OFB	79C571F93E6E91590576009881A3AB8E	03219503dd5973187FADC74474E3F047
CTR	79C571F93E6E91590576009881A3AB8E	5036C7D214C6E47119C48EEEC5219AA3

Tabela 1 – Resultado esperado para os dados encriptados. Fonte: adaptado do repositório do projeto.

Ao executar a aplicação original, todos os modos são testados de uma única vez, seguindo a sequência da tabela. O resultado é então a encriptação dos dois dados em todos os modos. Na Figura 8 é apresentado o resultado da execução desse código. Os valores são armazenados nos sinais internos:

- *r0*: guarda o valor do dado;
- *r1*: guarda o valor do vetor de inicialização;
- *r2*: guarda o resultado de cada operação.

Antes de começar as operação do módulo AES, é possível observar a inicialização da memória de instrução. Essa inicialização é feita simulando a memória *flash* e enviando o arquivo hexadecimal gerado. Só ao fim desse processo é que os cálculos são realizados pela ULA e passados pelos sinais descritos. Na imagem, devido a resolução, só é possível visualizar os valores finais. Portanto o valor de DATA2 e o resultado da encriptação desse valor são mostrados.

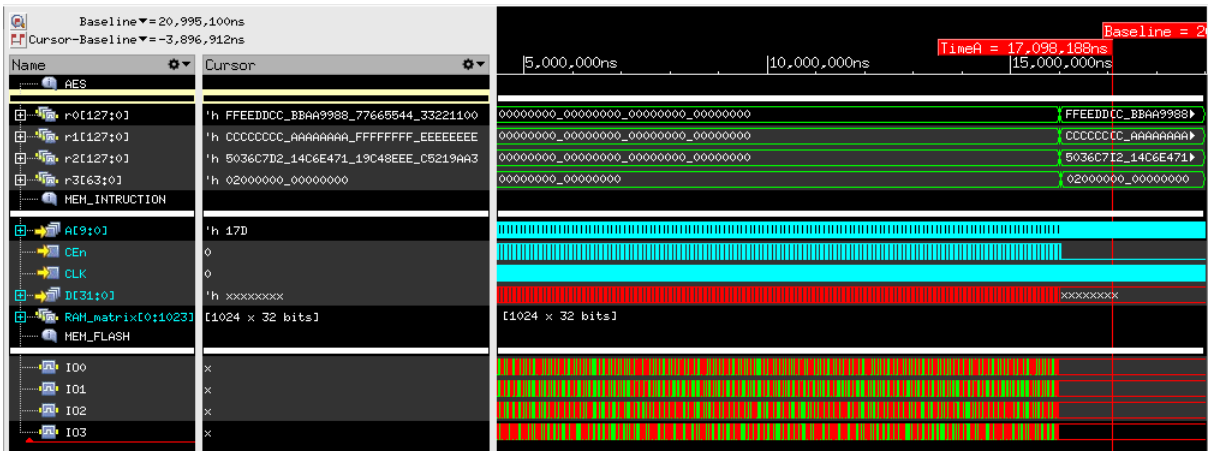

Figura 8 – Resultado da execução do código original. Fonte: Elaborada pelo autor.

Foi feito também testes isolados com cada modo de operação. As Figuras 11 e 12 apresentam o resultado para o modo CBC, onde é possível observar, além do resultado, a chave (*key*) e o modo, nesse caso 0b'001 para CBC.

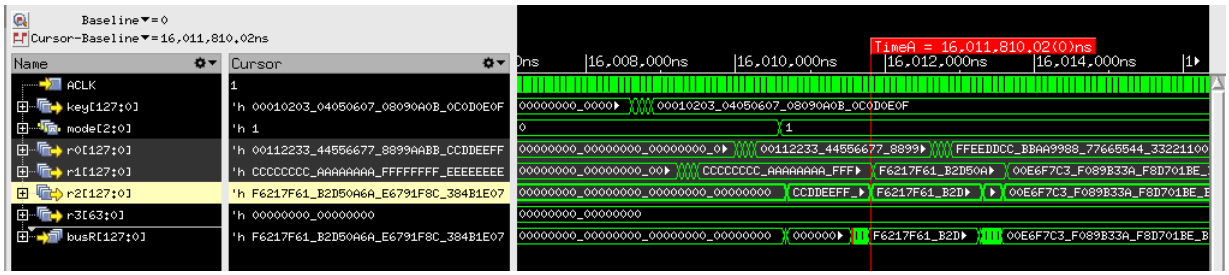

Figura 9 – Resultado do código adaptado: encriptação de DATA1. Fonte: Elaborada pelo autor.

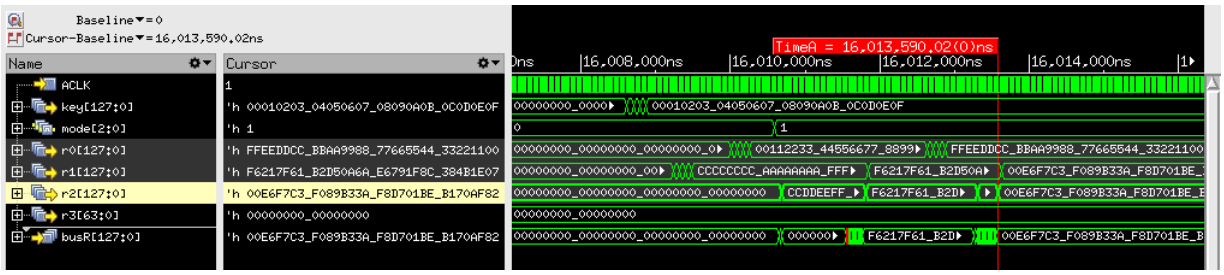

Figura 10 – Resultado do código adaptado: encriptação de DATA2. Fonte: Elaborada pelo autor.

Para o caso de uma operação de decriptação, o teste realizado foi semelhante aos anteriores. Aqui, após realizar a encriptação de DATA1 e DATA2, é armazenado na memória RAM os resultados da encriptação para logo em seguida decriptá-los, primeiro o resultado da operação 1 e logo em seguida o resultado da operação 2. As Figuras 11 e 12 apresentam os dois resultados para a encriptação e decriptação dos dados usando o modo ECB. Vale salientar que o sinal *r0* apresenta o resultado das operações. Logo, seguindo o fluxo do programa original, primeiro serão armazenados os valores da encriptação de cada dado e em seguida a decriptação.

Para ambos os casos os valores estavam compatíveis com os valores esperados.

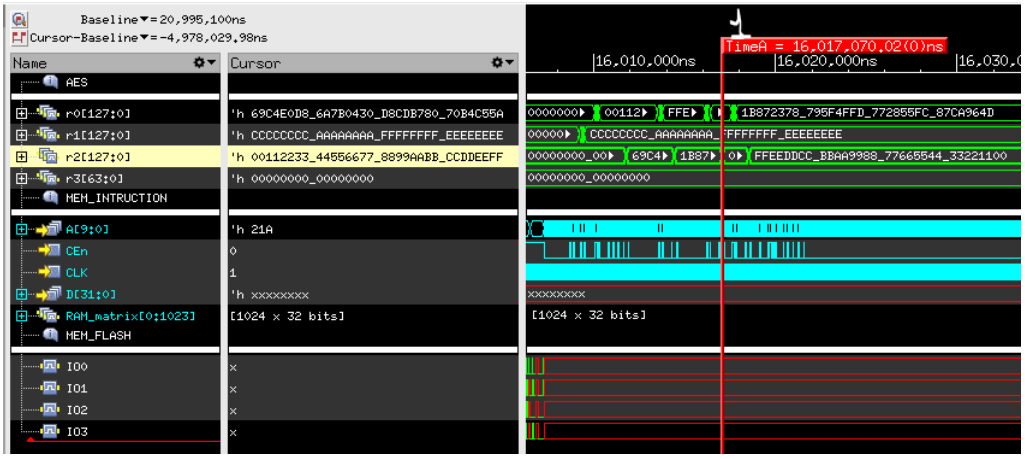

Figura 11 – Resultado do código adaptado: decriptação de DATA1. Fonte: Elaborada pelo autor.

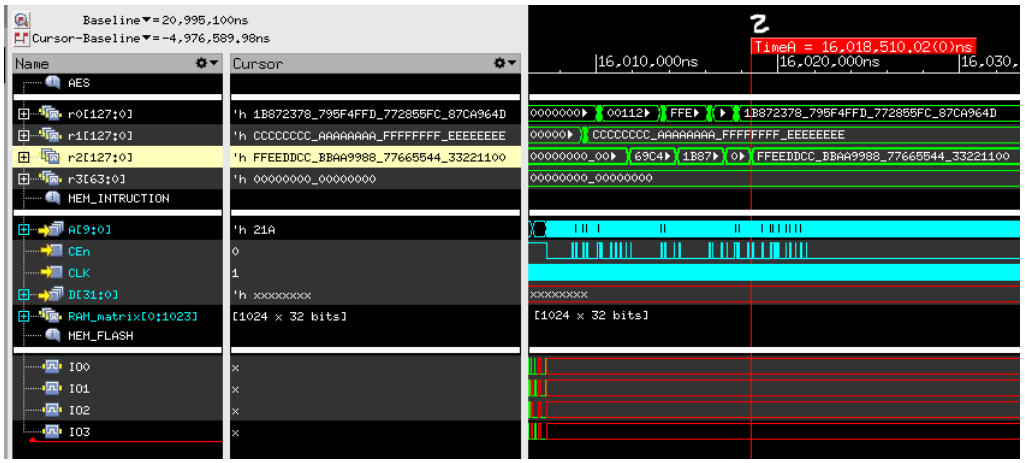

Figura 12 – Resultado do código adaptado: decriptação de DATA2. Fonte: Elaborada pelo autor.

3.2.2 *simple-gpio.c*

No código desenvolvido em *simple-gpio.c*, os pinos GPIOs são divididos em dois grupos onde metade dos pinos são configurados como entradas ([8:0]) e metade como saídas ([17:9]), com nível lógico 0.

Em seguida o valor das entradas é lido e o *LED* será aceso na saída respectiva. A Figura 13 apresenta o resultado desse código, onde as saídas ativadas foram de 9 a 17, por isso o resultado é o número em hexadecimal 0x03FE00. Logo após essa inicialização, o código verifica quais dos pinos de entrada estão recebendo nível 1 e ativa a saída respectiva.

Vale salientar que no *testbench* já têm 5 GPIOs sendo inicializados com 1, são eles que estão ativando as *outputs* de 9 a 14, por isso o resultado em hexadecimal 0x07E00. Nesse caso é possível ver os valores de saída sendo modificadas no registrador *r_gpio_out*.

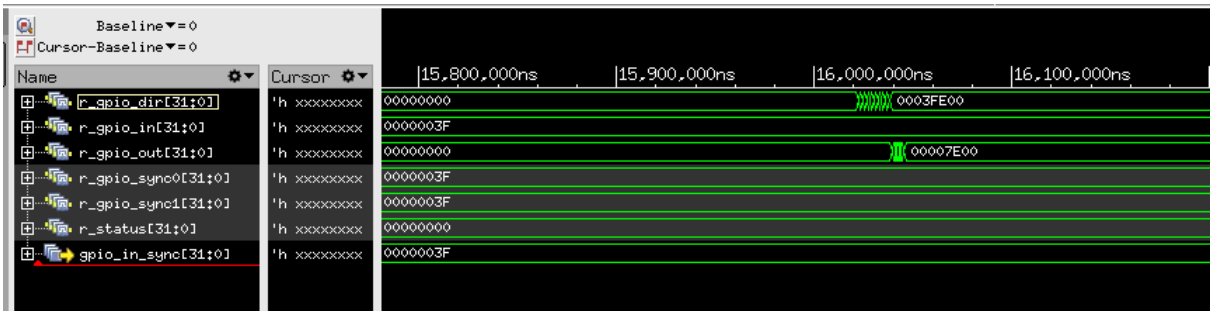

Figura 13 – Resultado de teste de execução da aplicação *simple-gpio.c*. Fonte: Elaborada pelo autor.

Com essa aplicação conseguimos verificar a funcionalidade dos GPIOs pelo SoC desenvolvido.

3.2.3 *demo-i2c.c*

A aplicação *demo-i2c.c* implementa a sequência de passos para o envio e leitura de dados via esse protocolo. A ideia é enviar uma sequência de dados e depois lê-los, assim é possível observar o funcionamento do protocolo I2C pelo SoC.

Podemos observar na Figura 14 a inicialização do protocolo configurando os valores de *I2C_PRESCALER* para 0x63 e *I2C_CRT_EN* para 0x80. O primeiro desses parâmetros serve para gerar o sinal de *clock* e o segundo para ativar o IP. Logo em seguida é possível observar nos registros *r_cmd* e *r_tx* os valores enviados pelas funções *i2c_send_command* e *i2c_send_data*, respectivamente.

É observado inicialmente o envio do primeiro valor 0xA0 e logo depois o *loop for* enviando os valores inteiros de 0 a 15 em formato hexadecimal. No fim da execução ainda observa-se o início da parte de leitura dos dados enviados via i2c.

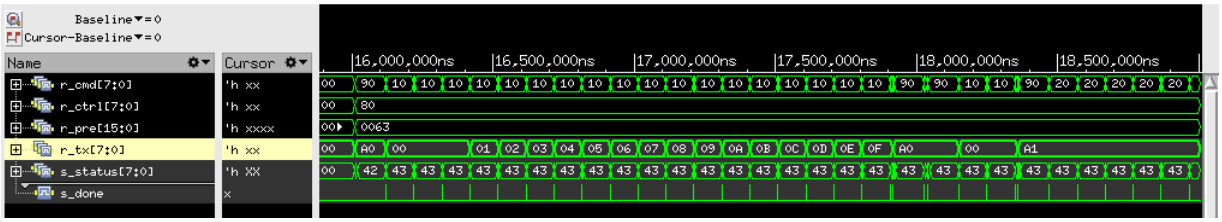

Figura 14 – Resultado de teste de execução da aplicação *demo-i2c.c*. Fonte: Elaborada pelo autor.

3.2.4 *demo-gpio-i2c-aes.c*

Até o momento, as aplicações desenvolvidas tinham como objetivo validar uma funcionalidade específica do sistema do chilipe, como os pinos GPIO, o módulo AES e o protocolo I2C. Nesta aplicação são utilizadas as três funcionalidades no mesmo código. A aplicação começa configurando o sistema para a encriptação de um dado pelo módulo AES. Ele usa o mesmo DATA1 e KEY do código *demo-aes.c* e o modo ECB.

Em seguida são inicializados 6 pinos GPIO, sendo dois configurados como entrada (*DIR_IN=0*) e 4 como saída (*DIR_OUT=1*). Após a encriptação ter sido realizada, o seu resultado é armazenado em 4 variáveis: data1, data2, data3 e data4. O endereço desse resultado pode ser conferido na definição da função *AES_writeResult()*.

Em seguida, em função dos valores de entrada dos GPIO 0 e 1 algumas operações são realizadas. As operações são sempre as mesmas: envio dos 4 dados de 8 *bits* que compõem o DATA via i2c. Esses dados dependem de quais entradas estarão ativas no momento, mas são sempre o mesmo dado deslocado de alguns bits.

- GPIO_0 =1 e GPIO_1 = 1: operações com data4;
- GPIO_0 =1 e GPIO_1 = 0: operações com data3;
- GPIO_0 =0 e GPIO_1 = 1: operações com data2;
- GPIO_0 =0 e GPIO_1 = 0: operações com data1;

No final de cada, o respectivo GPIO de saída é ativado.

Ao executar essa simulação pode-se ver claramente o módulo AES sendo inicializado, com a chave e o dado sendo escrito no respectivo registrador, assim como o resultado da operação. Em seguida o protocolo i2c é iniciado, os valores de *I2C_PRESCALER* e *I2C_CRT_EN* são escritos. Com a inicialização dos GPIO's, os pinos 0 e 1 são configurados como entradas e já estavam ativas, já que no *testbench* já estão definidos os sinais dos 5 primeiros GPIOs como 1, então o programa entra na primeira condição, onde os dados enviados via i2c são os do último conjunto de dados. No fim do primeiro *loop* é possível observar a alteração do respectivo GPIO (nesse caso o 5).

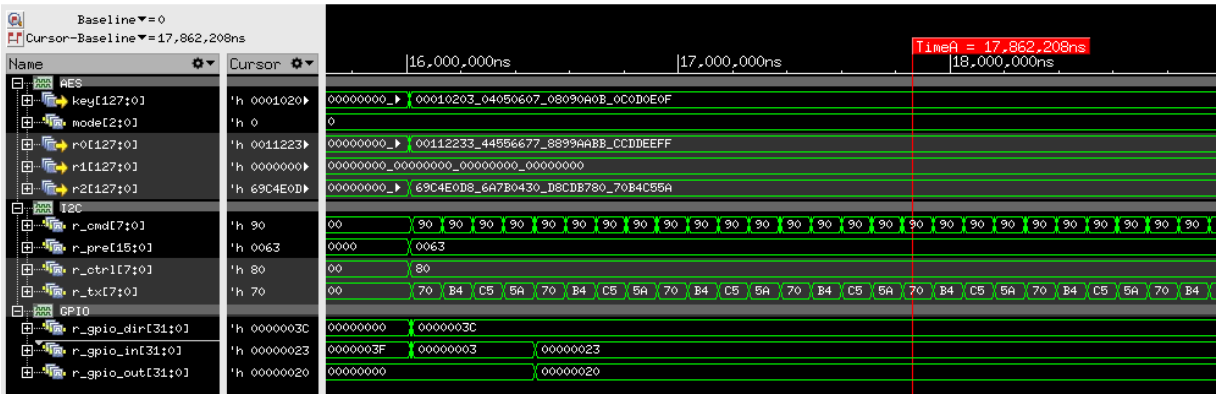

Figura 15 – Resultado de teste de execução da aplicação *demo-gpio-i2c-aes.c*. Fonte: Elaborada pelo autor.

3.2.5 *demo-gpio-uart-aes.c*

Para essa aplicação o objetivo é fazer o mesmo da anterior, porém utilizando o protocolo UART. As funções utilizadas no programa estão definidas também nas bibliotecas presentes no projeto (*uart.h* e *uart.c*) no repositório. São elas: *uart_set_cfg()* para a ativação ou não do bit de paridade e configuração da taxa de transmissão, *uart_sendchar()* para o envio dos dados propriamente ditos e *uart_wait_tx_done()* que aguarda até que o dado tenha sido enviado.

Na Figura 16 é possível visualizar novamente a inicialização do módulo AES e dos GPIOs, além do envio dos dados via protocolo UART pelo buffer *fifo_tx_data*.

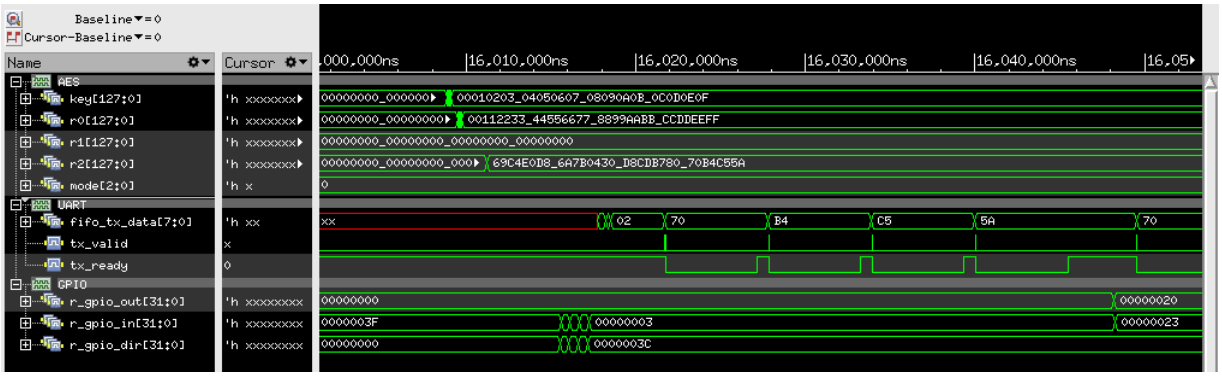

Figura 16 – Resultado de teste de execução da aplicação *demo-gpio-uart-aes.c*. Fonte: Elaborada pelo autor.

3.2.6 *demo-gpio-spi-aes.c*

Nessa aplicação são inicializadas as configurações do protocolo SPI, colocando o dispositivo como *master* e definindo o tamanho do dado para 32 *bits*. Aqui os dados

são enviados apenas uma vez, não sendo enviado os valores deslocados, como é possível visualizar na Figura 17, resultado da execução desse programa:

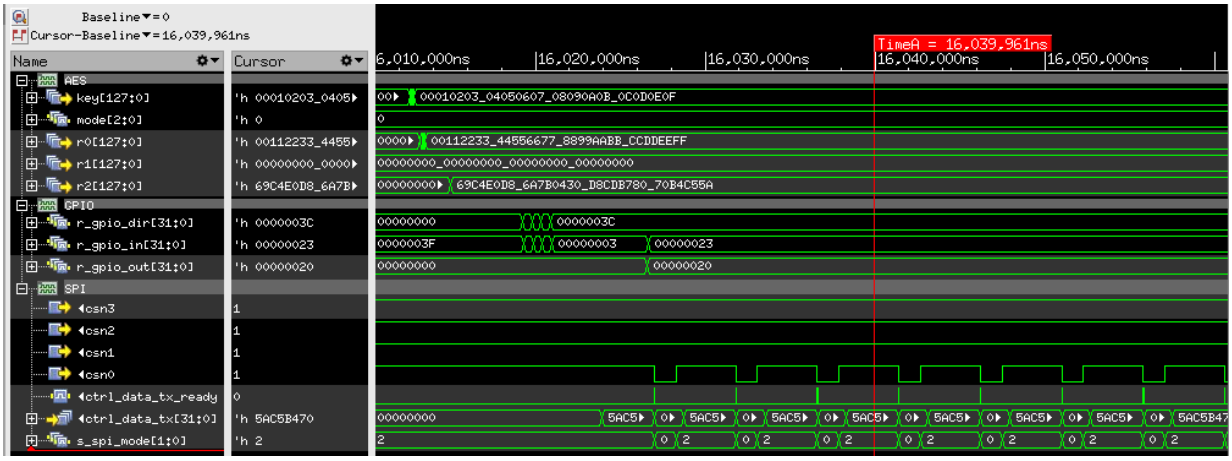

Figura 17 – Resultado de teste de execução da aplicação *demo-gpio-spi-aes.c*. Fonte: Elaborada pelo autor.

3.3 Tabela de testes

Uma tabela de testes iniciais foi desenvolvida pela equipe do laboratório. O objetivo era dispor de um conjunto de testes básicos que pudessem indicar o correto funcionamento do SoC. A Tabela 2 apresenta os testes básicos iniciais e a Tabela 3 apresenta três testes de periféricos.

Test Name	Test	Description	Setup	Functionalities
1	Basic	Verificar a continuidade com os pares de alimentação VDD1/GND1, VDD2/GND2 e VDD3/GND3	Multimetro de mão/bancada; Configurar o mesmo para medição de continuidade.	Teste elétrico do chip
2	Basic	Verificar se o consumo é inferior <1mA	SMU; Alimentar em 1,8V; Limitar a corrente <=20mA	Teste elétrico do chip
3	Basic	Reproduzir um reset (reset ativo baixo) e verificar se o chip tenta fazer um boot via canal SPI	Setup test#2; Força um reset com jumper (VDD para GND) ou usar botão fisico.	Testa o reset, SPI e boot do chip

4	Basic	Carregar na flash o demo-spi-aes e verificar no canal SPI a resposta do AES calculado	Setup test#3; Gravar a aplicação na flash; Probear os sinais da SPI (per.) e GPIO (2,3,4,5)	Testa o boot, core, AES e SPI (per.)
5	Basic	Carregar na flash o demo-gpio-spi-aes e verificar no canal SPI a resposta do AES calculado multiplexado pelo GPIO	Setup test#3; Gravar a aplicação na flash; Probear os sinais da SPI (per.) e GPIO (2,3,4,5)	Testa o boot, core, AES, SPI (per.) e GPIO.
6	Basic	Carregar na flash o demo-uart-spi-aes e verificar no canal SPI a resposta do AES calculado multiplexado pelo GPIO	Setup test#3; Gravar a aplicação do demo na flash; Probear os sinais da UART (per.) e GPIO (2,3,4,5)	Testa o boot, core, AES, UART (per.) e GPIO.
7	Basic	Carregar na flash o demo-gpio-i2c-aes e verificar no canal SPI a resposta do AES calculado multiplexado pelo GPIO	Setup test#3; Gravar a aplicação na flash; Probear os sinais da I2C (per.) e GPIO (2,3,4,5)	Testa o boot, core, AES, I2C (per.) e GPIO.
8	Basic	Carregar na flash o demo-rs-encdec e verificar no canal SPI a resposta do RS calculado	Setup test#3; Gravar a aplicação na flash; Probear os sinais da SPI (per.).	Testa o boot, core, RS ENCDEC e SPI (per.)
9	Basic	Carregar na flash o demo-vi-encdec e verificar no canal SPI a resposta do Viterbi calculado	Setup test#3; Gravar a aplicação na flash; Probear os sinais da SPI (per.).	Testa o boot, core, Viterbi ENCDEC e SPI (per.)

10	Basic	Carregar na flash o demo-aes-rs-vi-encdec e verificar no canal SPI/UART a resposta calculada (AES, RS e Viterbi)	Setup test#3; Gravar a aplicação na flash; Probear os sinais da SPI/UART (per.).	Testa o boot, core, RS/Viterbi ENCDEC, AES ENCDEC e SPI/UART (per.)
11	Basic	Carregar na flash o demo-riscv (aplicação risc ex: bubble sort) e verificar no canal SPI/UART a resposta	Setup test#3; Gravar a aplicação na flash; Probear os sinais da SPI/UART (per.).	Testa o boot, core e SPI/UART (per.)

Tabela 2 – Testes básicos. Fonte: Repositório do projeto.

Test Name	Test	Description	Setup	Functionalities
1	Peripherals	Carregar na flash o demo-spi-aes (driver SPI) e verificar no canal SPI a resposta do AES calculado	Setup test#3; Gravar a aplicação na flash; Cabo USB-to-SPI; Probear os sinais da SPI in/out (per.).	Testa o boot, core, AES e SPI (per.)
2	Peripherals	Carregar na flash o demo-i2c-aes (driver I2C) e verificar no canal I2C a resposta	Setup test#3; Gravar a aplicação na flash; Cabo USB-to-I2C; Probear os sinais I2C (per.).	Testa o boot, core, AES e I2C (per.)
3	Peripherals	Carregar na flash o demo-uart-aes (driver UART) e verificar no canal UART a resposta	Setup test#3; Gravar a aplicação na flash; Cabo USB-to-UART; Probear os sinais da UART(per.).	Testa o boot, core, AES e UART (per.)

Tabela 3 – Testes iniciais dos periféricos. Fonte: Repositório do projeto.

A partir desses testes é possível inferir se, por exemplo, existe algum curto-circuito na placa e/ou no CI, se o sinal de *clock* é o correto, além de outros testes.

4 Conclusão

Este trabalho apresentou as atividades desenvolvidas pelo estagiário no laboratório XMEN do DEE da Universidade Federal de Campina Grande. Durante o período de estágio foi desenvolvido inicialmente uma PCB para os testes em *hardware* do SoC em questão e, em seguida, simulações de diferentes funcionalidades desse sistema.

Alguns problemas e desafios foram encontrados, mas conseguiram ser contornados com ajuda de outros membros da equipe, como foi o caso do problema de versionamento da *toolchain*. Com o progresso das atividades foi possível observar, então, o correto funcionamento do *chip* com diferentes *firmwares* simulados, validando-o do ponto de vista funcional.

Ademais, a oportunidade de realizar esse estágio foi de grande valia e muito proveitoso, pois proporcionou um crescimento tanto pessoal como profissional, no que diz respeito ao trabalho e colaboração de equipes para o alcance de objetivos. Essas atividades proporcionaram ao estagiário um contato maior com a área da microeletrônica e o emprego de diversas habilidades e conhecimentos adquiridos ao longo do curso de engenharia elétrica, como nas disciplinas de eletrônica, técnicas de programação, circuitos lógicos e arquitetura de sistemas digitais, e necessárias para o mercado de trabalho.

Referências Bibliográficas

[1] ARORA, M. **Embedded Sytem Design: Introduction to SoC System Architecture**. Learning Bytes Publishing, 2016.

[2] BARR, M.; MASSA, A. **Programming Embedded Systems**. O'Relly, 2009.

[3] CORELIS. **SPI Tutorial**. Disponível em: <https://www.corelis.com/education/tutorials/spi-tutorial/>. Acesso em 14 de setembro de 2021.

[4] BRITO, S. **Integração de IPs em SoC e interface com memória**. Orientador: Marcos Ricardo Alcântara Morais. 2018. 51 f. TCC (Graduação) – Curso de Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande, Campina Grande, 2018.

[5] CAMPBELL, Scott. **Circuit Basics**. Disponível em <https://www.circuitbasics.com/>. Acesso em 13 de setembro de 2021.

[6] ETH Zurich. **PULP Platform**. Disponível em: <https://pulp-platform.org/>. Acesso em 16 de setembro de 2021.

[7] LU, J. **AES-128 in Google Sheets**. Disponível em: <https://poisonninja.github.io/2016/12/07/AES-in-Google-Sheets/>. Acesso e m17 de setembro de 2021.

[8] MATHIAS, L. A. P. **Algoritmo de criptografia AES**. Disponível em: <https://www.gta.ufrj.br/grad/05_2/aes/>. Acesso em 12 de setembro de 2021.

[9] TEJA, R. **Arduino I2C Tutorial | How to use I2C Communication on Arduino?**. Disponível em: <https://www.electronicshub.org/arduino-i2c-tutorial/>. Acesso em 14 de setembro de 2021.

Anexo 1: Esquema elétrico