

UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO DE JOINVILLE  
CURSO DE ENGENHARIA MECATRÔNICA

BRENO CANDIDO CONRADO

DESENVOLVIMENTO DE UM SISTEMA DE AQUISIÇÃO DE DADOS DE BAIXO  
CUSTO PARA MONITORAMENTO DE VIBRAÇÃO TORSIONAL

Joinville  
2020

BRENO CANDIDO CONRADO

DESENVOLVIMENTO DE UM SISTEMA DE AQUISIÇÃO DE DADOS DE BAIXO  
CUSTO PARA MONITORAMENTO DE VIBRAÇÃO TORSIONAL

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Prof. Dr. Anderson Wedderhoff Spengler

Joinville  
2020

## RESUMO

A medição de vibração torsional traz informações sobre conforto e durabilidade de um veículo, sendo necessário realizar a captação e armazenamento de dados referentes à frequência de rotação de um eixo de transmissão para analisar esta vibração. O processo de aquisição de dados se prova uma ferramenta de grande importância para validação de sistemas reais, necessitando de maiores velocidades de amostragem com a evolução. Este procedimento exige que o armazenamento de dados cresça da mesma forma, acompanhando sua taxa amostragem e garantindo segurança no registro de informações. O objetivo deste trabalho é projetar um sistema de armazenamento de dados em alta velocidade utilizando uma interface entre FPGA e memória Flash, com validação via simulação dos módulos gerados através do VHDL. Taxas de transferências desejadas foram alcançadas, atestando que o sistema escolhido demonstra grande eficiência no comando do chip de memória.

**Palavras-chave:** *Data logger*, Aquisição de dados, FPGA, VHDL, Memória *flash*, Vibração torsional.

## **ABSTRACT**

Torsional vibration measurement introduces information about comfort and durability of a vehicle, requiring the collecting and storing of data concerning the rotational frequency of a transmission shaft to analyse this vibration. The process of data acquisition proves to be an important tool when validating physical systems, requiring faster sampling speeds as we evolve. This procedure demands data storage to grow similarly, following its sampling rate e guaranteeing safety when storing data. This paper presents the design of a high-speed data storage system using an interface between flash memory and FPGA, validating its approach by simulating the modules designed in VHDL. Desired transfer rates were achieved, proving the system to be highly efficient in commanding the memory chip.

**Keywords:** Data logger, Data acquisition, FPGA, VHDL, Flash memory, Torsional vibration.

## **AGRADECIMENTOS**

Primeiramente, aos meus pais, Gisela de Candido e Geomar Tavares Conrado, pelo apoio e simpatia oferecidos durante toda a minha vida, e por terem me motivado a seguir a minha aspiração.

Aos meus avós, tios e toda a minha família, que estiveram sempre presentes.

À minha namorada, Mylena Simão, por ter me escutado e estado ao meu lado.

Ao professor Anderson Wedderhoff Spengler, que me ensinou, orientou e se disponibilizou a ajudar sempre que necessário.

Aos membros do LISHA, com quem convivi e aprendi durante anos.

A todos os meus amigos, especialmente os do curso de Engenharia Mecatrônica, por terem me dado suporte e acompanhado durante este trajeto.

## LISTA DE ILUSTRAÇÕES

Figura 1 – Trem de pulsos padrão da velocidade angular. . . . .	12
Figura 2 – Exemplo de configuração de sensor magnético para medição de vibração torsional. . . . .	13
Figura 3 – Medição de período utilizando ciclos de clock. . . . .	13
Figura 4 – Comparação entre o ROTEC RAS e equipamento de Korol (2015). Medição na transmissão (em vermelho) e no motor (em preto). . . .	15
Figura 5 – Comparação entre o PAK MKII (em preto) e equipamento de Korol (2015) (em vermelho) . . . . .	16
Figura 6 – Arranjo de células lógicas e chaves em um FPGA. . . . .	18
Figura 7 – Célula lógica com três entradas. . . . .	19
Figura 8 – Datalogger desenvolvido. . . . .	20
Figura 9 – Diagrama lógico do chip de memória S34ML02G1 . . . . .	21
Figura 10 – Estrutura dos blocos de memória do chip S34ML02G1 . . . . .	22
Figura 11 – Modo de endereçamento do chip S34ML02G1 . . . . .	23
Figura 12 – Diagrama para entrada de dados. . . . .	24
Figura 13 – Diagrama para saída de dados. . . . .	25
Figura 14 – Diagrama para escrita de comandos. . . . .	26
Figura 15 – Diagrama para escrita de endereços. . . . .	27
Figura 16 – Diagrama para leitura de página. . . . .	28
Figura 17 – Diagrama para programação de página. . . . .	29
Figura 18 – Diagrama para leitura de identificação. . . . .	29
Figura 19 – Code Composer Studio . . . . .	30
Figura 20 – ISE Project Navigator . . . . .	31
Figura 21 – iMPACT . . . . .	32
Figura 22 – Test Bench do software iSim . . . . .	33
Figura 23 – Simulação de uma porta OR utilizando o iSim . . . . .	33
Figura 24 – Máquina de estados do controlador de operações . . . . .	49
Figura 25 – Retorno da operação de leitura de identificação no osciloscópio. Canais CE# (em amarelo), WE# (em verde), RE# (em azul) e IO7 (em vermelho) . . . . .	51
Figura 26 – Simulação módulo timer . . . . .	53
Figura 27 – Simulação módulo de escrita . . . . .	53
Figura 28 – Simulação módulo de leitura . . . . .	54

Figura 29 – Simulação módulo de escrita e leitura com controle tri-state . . . . .	55
Figura 30 – Simulação operação de identificação . . . . .	55
Figura 31 – Simulação operação de programação de página . . . . .	56
Figura 32 – Simulação operação de leitura de página . . . . .	56
Figura 33 – Simulação da programação de um bloco inteiro . . . . .	57
Figura 34 – Simulação da leitura de um bloco inteiro . . . . .	58

## LISTA DE TABELAS

Tabela 1 – Relação entre <i>STF</i> e tempo de conservação da memória . . . . .	17
Tabela 2 – Relação entre <i>AT</i> e temperatura de conservação de memória . . . . .	17
Tabela 3 – Relação entre <i>WAF</i> e temperatura de conservação de memória . . . . .	17
Tabela 4 – Estrutura de memória do chip S34ML02G1 . . . . .	21
Tabela 5 – Pinos de comando do chip S34ML02G1 . . . . .	23
Tabela 6 – Definição dos possíveis modos dentro de operações . . . . .	45
Tabela 7 – Custo dos componentes . . . . .	50
Tabela 8 – Comparação entre os processos de escrita na memória Flash . . . . .	52



## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>10</b>
1.1	Objetivo Geral	11
1.2	Objetivos Específicos	11
<b>2</b>	<b>REVISÃO TEÓRICA</b>	<b>12</b>
2.1	Vibração torsional	12
2.2	Expectativa de vida para memória flash	16
<b>3</b>	<b>MATERIAIS E MÉTODO</b>	<b>18</b>
3.1	FPGA	18
3.2	Hardware desenvolvido	19
3.3	Determinação dos requisitos do sistema	20
3.4	Memória Flash S34ML02G1	21
3.4.1	Escrita de dados	24
3.4.2	Leitura de dados	24
3.4.3	Entrada de comandos	25
3.4.4	Entrada de endereços	26
3.4.5	Leitura de página	27
3.4.6	Programação de página	28
3.4.7	Leitura de identificação	29
3.5	Softwares utilizados	30
3.5.1	Code Composer Studio	30
3.5.2	ISE	30
3.5.3	iMPACT	31
3.5.4	iSim	32
<b>4</b>	<b>DESENVOLVIMENTO</b>	<b>34</b>
4.1	Projeto no microcontrolador Tiva	34
4.2	Projeto em VHDL para FPGA	37
4.2.1	Timer	37
4.2.2	Escrita e leitura	38
4.2.3	Controle tri-state	40
4.2.4	Controle de modos de entrada	43
4.2.5	Controle de operações	44
<b>5</b>	<b>RESULTADOS E DISCUSSÕES</b>	<b>50</b>
5.1	Custo de hardware	50

<b>5.2</b>	<b>Resposta em osciloscópio . . . . .</b>	<b>50</b>
<b>5.3</b>	<b>Velocidade de programação de memória com microcontrolador .</b>	<b>51</b>
<b>5.4</b>	<b>Simulações . . . . .</b>	<b>52</b>
5.4.1	Timer . . . . .	52
5.4.2	Módulo de escrita . . . . .	53
5.4.3	Módulo de leitura . . . . .	54
5.4.4	Módulo de escrita e leitura com tri-state . . . . .	54
5.4.5	Módulo de controle de operações . . . . .	55
<b>5.5</b>	<b>Tempo de escrita em toda a memória . . . . .</b>	<b>56</b>
<b>5.6</b>	<b>Tempo de leitura de toda a memória . . . . .</b>	<b>57</b>
<b>5.7</b>	<b>Tempo de vida da memória . . . . .</b>	<b>58</b>
<b>6</b>	<b>CONCLUSÕES . . . . .</b>	<b>59</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>60</b>
	<b>APÊNDICE A . . . . .</b>	<b>62</b>

## 1 INTRODUÇÃO

De acordo com Rajmond e Pitică (2010), fabricantes de componentes eletrônicos realizam milhares de testes físicos em seus equipamentos durante o projeto destes; por consequência disso, surge a necessidade do armazenamento de grandes quantidades de dados, para que falhas no sistema possam ser identificadas. *Data loggers* aparecem como uma ótima alternativa no monitoramento destes dados, também sendo usados em qualquer campo que haja a necessidade de acompanhar a evolução de um fenômeno. A constante necessidade do desenvolvimento de sistemas de aquisição de dados mais rápidos e maior capacidade de armazenamento é evidente.

"O registro e armazenamento de dados é um requerimento óbvio de qualquer sistema de aquisição"(BISHOP, 2007, p. 35-3, tradução nossa). Para o armazenamento em chip de memória, sua velocidade de escrita e leitura são um diferencial importante quando se busca amostrar sinais de alta frequência; isto inclui o intermédio a realizar a comunicação com o circuito integrado.

Para este projeto, foi utilizado como base a proposta de Korol (2015), que visa medir vibração torsional de *powertrain* automotivo através de um trem de pulsos gerado por sensor magnético. Em seu trabalho, Korol (2015) alcançou taxas de amostragem de 57 Hz, o que não tornou possível a análise espectral do sinal recebido, que se encontrava na faixa de 10 kHz, pois sua rotina de leitura incluía a transmissão dos dados recebidos a um microcontrolador que realizaria o processamento. Esta transmissão diminuiu a velocidade de coleta de amostras, ocasionando perdas de informações referentes aos sinais de maiores frequências.

Portanto, para contornar esta limitação, foi identificada a vantagem de efetuar o armazenamento prévio dos dados, transmitindo-os após o processo de amostragem ter finalizado. Desta forma, surge a necessidade de encontrar um modelo de chip de memória com velocidade de armazenamento capaz de superar esta taxa inicial de 57 Hz, obtendo velocidade superiores aos sinais de dezenas de kHz, assim como o hardware que fará a comunicação com este chip.

A escolha do hardware a ser utilizado varia de acordo com a aplicação. No trabalho feito por Mukaro e Carelse (1999), que visa medir radiação solar a cada dez minutos, utilizou-se um microcontrolador de oito bits, com frequência de clock de 8 MHz; isto se provou adequado para a situação.

Já datalogger comercial modelo V300 de Extech Instruments (2014), especializado em medição de acelerações em três eixos, fornece taxa de amostragem

de 200 Hz e memória de armazenamento de 4 Mbit, custando \$219,99. Enquanto o modelo CR1000X de Campbell Scientific (2020a) proporciona taxa de amostragem máxima de 1000 Hz de seus canais analógicos, com armazenamento de 4 MB em SRAM e 72 MB em flash (com possibilidade de expansão via cartão SD) e tem custo de \$2400,00.

Visando a criação de um método registro de dados comparável aos existentes no mercado, é proposto um sistema consistindo de um Field Programmable Gate Array (FPGA) realizando a interface com uma memória flash de 2 GBit, esta que possibilitaria altas velocidade de armazenamento com seus ciclos de escrita de 25 ns, assim como maior tamanho de memória.

O FPGA proporciona grande flexibilidade no projeto de circuitos digitais específicos a aplicações, permitindo que o mesmo chip seja reprogramado múltiplas vezes para diferentes lógicas sequenciais e combinatórias, oferecendo paralelismo e velocidade limitada apenas pela frequência de clock oferecida e suportada pelo dispositivo.

Como se vê, essa flexibilidade é extremamente valiosa quando são desenvolvidos sistemas que precisam alcançar requerimentos exigentes de sistemas de computação embarcados que estão se tornando tão comuns hoje em dia. (SASS; SCHMIDT, 2010, p. 2, tradução nossa)

## 1.1 Objetivo Geral

Criação de um sistema de armazenamento de dados de baixo custo comparável a *data loggers* comerciais, com parâmetros otimizados para um sistema de troca de dados com sensores de vibração torsional.

## 1.2 Objetivos Específicos

Os objetivos específicos são:

- a. Identificar parâmetros eficientes de um chip de memória para armazenar dados de medição de vibração torsional;
- b. Desenvolver um projeto eficiente para o comando do chip;
- c. Validar o sistema através de simulações do circuito obtido.

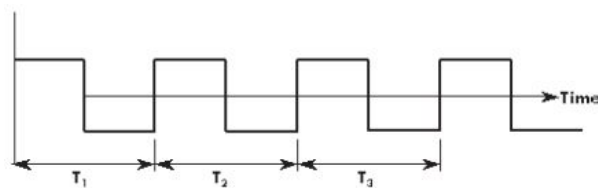
## 2 REVISÃO TEÓRICA

### 2.1 Vibração torsional

Korol (2015) pretendia medir vibração torsional em *powertrain* automotivo através da leitura de um trem de pulsos de lógica transistor-transistor (TTL). Esta medição, portanto, seria baseada no período dos pulsos do sinal de entrada.

"A eletrônica do sensor gera um sinal de velocidade angular na forma de trem de pulsos TTL. A frequência do trem de pulsos é diretamente proporcional à velocidade angular do eixo". (ADAMSON, 2004, p. 24, tradução nossa). Desta forma, para obter-se os dados da velocidade angular instantânea do *powertrain*, basta obter o valor do período de cada pulso gerado.

Figura 1 – Trem de pulsos padrão da velocidade angular.



Fonte: Adamson (2004).

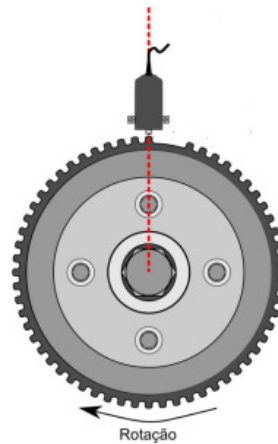
De acordo com Janssens e Britte (2014), vibrações torsionais são vibrações de um objeto, tipicamente de um eixo em rotação. Estas vibrações presentes podem afetar o conforto e a eficiência de um veículo. Sass e Schmidt (2010) diz que erros na medida de velocidades angulares podem vir de variações nos espaçamentos entre os dentes de uma engrenagem. Logo, isto resultaria em uma diferença no período de cada pulso do sinal, inclusive no tempo em que o mesmo pulso se encontra em nível baixo e alto. Esta variação é um resultado direto da vibração torsional.

Korol (2015) apontou que a frequência do trem de pulsos quando utilizado um sensor magnético instalado em frente a uma engrenagem do eixo do *powertrain*, que gera um pulso cada vez que um dente da engrenagem passa em frente ao sensor, é dada pela Equação 2.1.

$$f = \frac{w_{rpm} \cdot n}{60} \quad (2.1)$$

Onde  $f$  é a frequência de pulsos em hertz,  $w_{rpm}$  é o número de rotações por minuto do eixo, e  $n$  o número de dentes na engrenagem. Em seu trabalho, Korol (2015) considerou  $w_{rpm}$  na faixa de 1000 a 7000 rpm.

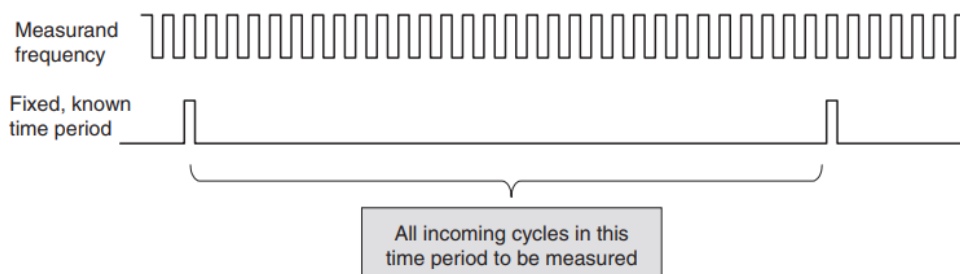
Figura 2 – Exemplo de configuração de sensor magnético para medição de vibração torsional.



Fonte: Pedroso (2017)

"Um contador pode ser usado em sistemas de aquisição de dados para aplicações de temporizador. Um sinal de clock conectado à entrada do canal e utilizando o sinal de entrada como disparo funciona bem". (MEASUREMENT COMPUTING, 2012, p. 123, tradução nossa). Logo, para obter o tempo de pulso basta medir o número de ciclos de clock que ocorreram a partir deste disparo e concluir o tempo baseado na frequência do oscilador. National Instruments (2020a) demonstra que este método de medição de frequência, chamado pela mesma de medição de período inverso, é efetivo quando a frequência do contador é muito maior que a frequência do sinal medido. Para a aplicação de medição de vibração torsional, esta técnica seria ideal para que se obtenha o período do pulso de cada dente da engrenagem, podendo fazer a análise destes períodos a fim de examinar a variação entre cada valor.

Figura 3 – Medição de período utilizando ciclos de clock.



Fonte: Wilmshurst (2007)

Já no caso de Korol (2015), foi utilizado um método de medição de frequência baseado no número de pulsos em um intervalo fixo de tempo. Este método, chamado de contagem de pulsos em tempo determinado por National Instruments (2020a), pode ser usado quando a frequência do sinal medido aumenta, podendo até mesmo atingir

a frequência da fonte de contagem. Embora esta técnica possibilite a obtenção da frequência de rotação do eixo, ela não fornece dados instantâneos de frequência quando comparado à medição de período inverso, pois deixa de considerar o tempo independente de cada pulso. Isto resultaria na perda de informações de frequências maiores que o intervalo de contagem, incluindo dados de vibração torsional.

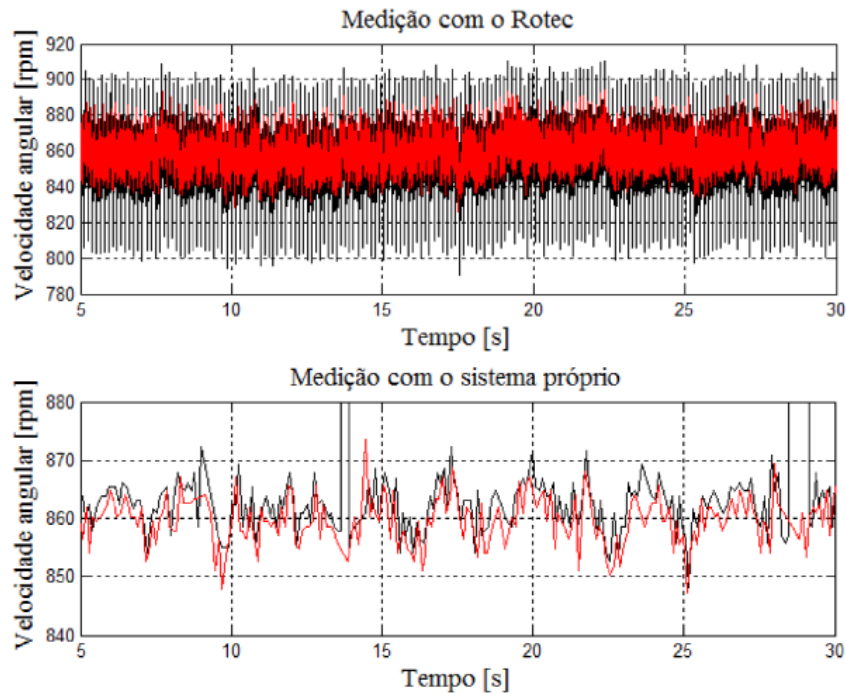
O *data logger* CR1000X de Campbell Scientific (2020a), avaliado em \$2400,00, fornece esta função de leitura de pulsos. De acordo com seu manual de produto, o sistema permite a detecção de bordas de transição entre níveis da tensão do sinal de entrada, apresentando dados de contagem de pulsos, frequência do sinal ou dados de tempo da onda. A maior frequência permitida para sinais pulsados de alta frequência é de 250 kHz. O armazenamento de seus dados é feito em memória de acesso randômico estática (SRAM) e com possibilidade de memória suplementar em cartão MicroSD.

Outro modelo de *data logger*, o CR310 de Campbell Scientific (2020b), com preço estimado de \$1650,00, é uma versão mais simples da anterior, mas que proporciona funções similares. O método de contagem de pulsos neste tem o mesmo mecanismo de disparo, entretanto sua frequência máxima permitida para pulsos se reduz a 35 kHz em um único canal de entrada. Seus dados de medidas são inicialmente registrados em memória de acesso randômico (RAM), sendo que os dados finais são destinados à memória flash serial do *data logger*.

Além dos *data loggers* mencionados, há os equipamentos comerciais utilizados por empresas automotivas e comparados no trabalho de Korol (2015). Foram listados dois sistemas de empresas diferentes: o *Rotation Analysis System* (RAS) fabricado pela ROTEC e utilizado na empresa ZF brasileira; e o PAK MKII, fabricante Müller-BBM, empregado na BWM do Brasil.

O RAS apresentou período de amostragem de 0,6 ms comparado a 114 ms alcançado pelo equipamento desenvolvido Korol (2015). Ele possui também um clock de frequência 10 GHz junto de um timer de 40 bits para medição do tempo de cada pulso, custando R\$220.000,00. (KOROL, 2015). ROTEC (2016) afirma em um modelo superior, com de clock 12 GHz, que a maior frequência de entrada para este é 20 MHz.

Figura 4 – Comparação entre o ROTEC RAS e equipamento de Korol (2015). Medição na transmissão (em vermelho) e no motor (em preto).

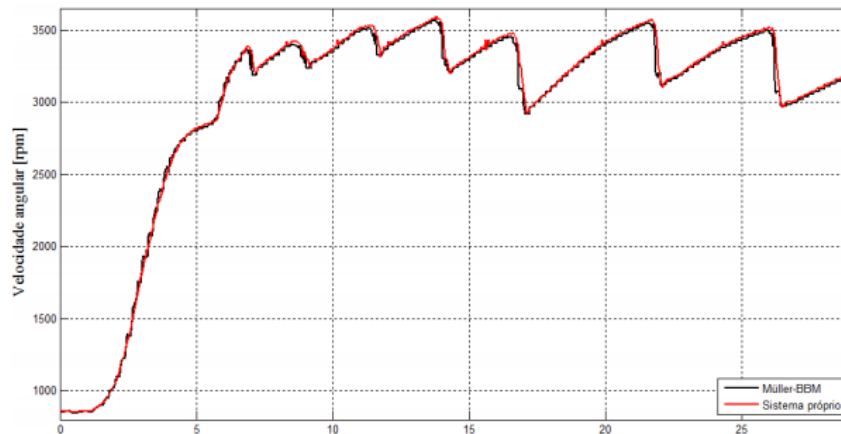


Fonte: Korol (2015).

O teste feito por Korol (2015), na empresa BWM do Brasil obteve resultados melhores. O PAK MKII, valor comercial de R\$92.000,00, estava conectado à *Controller Area Network* (CAN) do veículo através de um conversor de sinais com taxa de amostragem de 100 Hz; isto proporcionou melhor comparação, já que a CAN gerava um pulso por rotação do eixo. comparado ao equipamento desenvolvido que media sinais vindo dos dentes da engrenagem. (KOROL, 2015). De acordo com Müller-BBM (2015), o PAK MKII suporta até 1 MPulso/s (sendo este valor a soma da taxa entre dois canais), com resolução de 14 ns na sua amostragem de 4,9 MSa/s.



Figura 5 – Comparação entre o PAK MKII (em preto) e equipamento de Korol (2015) (em vermelho)



Fonte: Korol (2015).

## 2.2 Expectativa de vida para memória flash

National Instruments (2020b) atesta o tempo de vida esperado para uma um chip de memória flash. A Equação 2.2, demonstrada por National Instruments (2020b), relaciona ciclos de programar/apagar, com fatores referentes à temperatura de conservação do chip, à duração de armazenamento da memória e à amplificação devido à escrita ao total de escritas totais na flash.

$$Drive_{Endurance} = \frac{FlashCell_{Endurance}}{STF \cdot AT \cdot WAF} \quad (2.2)$$

Onde  $Drive_{Endurance}$  é o número de escritas na memória inteira,  $FlashCell_{Endurance}$  é o fator sobre vida da memória em ciclos de programação,  $STF$  é o fator relacionado ao tempo de armazenamento da memória,  $AT$  é o fator de aceleração devido à temperatura de armazenamento do chip, e  $WAF$  é a amplificação devido à eficiência de uso da flash. Logo,  $Drive_{Endurance}$  representa a vida total da memória, a partir da relação com o número de vezes que a flash pode ser programada em sua capacidade total.

Tabela 1 – Relação entre *STF* e tempo de conservação da memória

<b>Tempo de conservação</b>	<b><i>STF</i></b>
1 mês	0.08
3 meses	0.25
6 meses	0.5
1 ano	1
3 anos	3
5 anos	5
10 anos	10

Fonte: National Instruments (2020b) (tradução nossa).

Tabela 2 – Relação entre *AT* e temperatura de conservação de memória

<b>Temperatura de conservação (°C)</b>	<b><i>AT</i></b>
25	0.13
30	0.26
40	1
55	6.4
70	35
85	168

Fonte: National Instruments (2020b) (tradução nossa).

Tabela 3 – Relação entre *WAF* e temperatura de conservação de memória

<b>Carga de trabalho</b>	<b>Tipo de acesso</b>	<b><i>WAF</i> típico</b>
Empresa	Data randômica armazenada	~15
"Regra do polegar"	Mistura entre acesso randômico e acesso sequências como aproximação	4
Cliente	Típico de <i>laptops</i> de consumidores, consistindo de escritas sequenciais longas com alguns acessos randômicos	~2
"100% sequencial"	Todas as escritas são realizadas em sequência com arquivos grandes	~1

Fonte: National Instruments (2020b) (tradução nossa).

### 3 MATERIAIS E MÉTODO

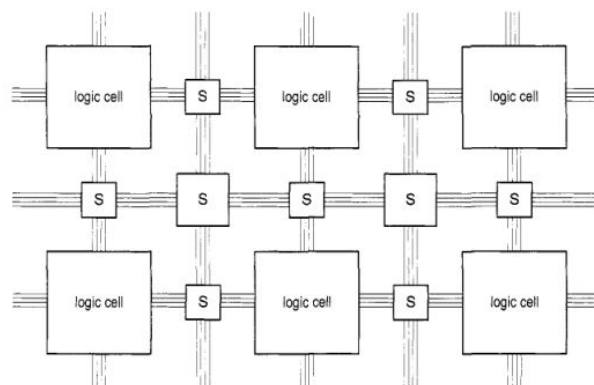
#### 3.1 FPGA

O Field Programmable Gate Array (FPGA) é um circuito integrado que pode ser programado em campo, à partir de uma configuração feita utilizando Hardware Description Language (HDL) (SASS; SCHMIDT, 2010).

De acordo com Dubey (2008), as vantagens de escolher um projeto baseado em FPGA ao invés de Circuitos Integrados de Aplicações Específicas (ASICs) incluem, a economia com custos de produção em pequeno volume, a reprogramabilidade, e um ciclo de projeto mais curto, partindo do conceito.

"FPGA é um dispositivo lógico que contém vetores bidimensionais de células lógicas genéricas e chaves programáveis."(CHU, 2011). Chu (2011) diz que um FPGA pode ser programado, configurando suas células lógicas e usando as chaves para ligar uma célula a outra.

Figura 6 – Arranjo de células lógicas e chaves em um FPGA.

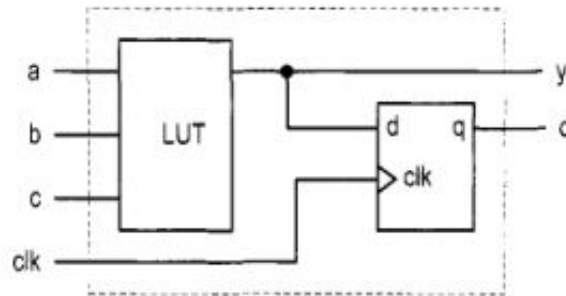


Fonte: Chu (2011).

Uma célula lógica contém uma Look Up Table (LUT) e um Flip Flop (FF) tipo D, sendo que a LUT pode ser usada para implementar uma função combinatória de  $n$  entradas (CHU, 2011).

A densidade de dispositivos FPGA tem crescido ao longo dos anos à mesma taxa que seu custo tem diminuído, fazendo do FPGA ser viável para diversas aplicações; FPGAs contemporâneas contêm milhares de LUTs e FFs para implementação de

Figura 7 – Célula lógica com três entradas.



Fonte: Chu (2011).

circuitos digitais complexos (DUBEY, 2008).

A maioria dos dispositivos FPGA também contém macro células ou macro blocos. Estes são projetados e fabricados no nível de transistores, e suas funcionalidades complementam as células lógicas. As macro células mais comuns incluem blocos de memória, multiplicadores combinatórios, circuitos de gerenciamento de clock, e circuitos de interface de entrada e saída. Dispositivos FPGA mais avançados podem até conter um ou mais núcleos de processadores pré-fabricados. (CHU, 2011, p. 13, tradução nossa)

Seguindo o trabalho desenvolvido por Korol (2015), onde não foi possível realizar uma análise das frequências maiores que representam a vibração torsional, devido a baixa velocidade de aquisição de dados, este sistema pretende superar essa dificuldade partindo da melhoria do Hardware, utilizando um FPGA.

Visando o armazenamento rápido e eficiente de dados lidos pela FPGA, foi escolhido o chip de memória Flash de 2 Gb, com ciclos de leitura de 25 ns, e oito entradas e saídas, S34ML01G2, referido no datasheet de Cypress Semiconductor Corporation (2018). Este chip que necessita de uma sequência de comandos e endereços para que sejam realizadas as operações desejadas, assim como o seguimento de diagramas de tempo específicos para a realização destes procedimentos, vistos na Seção 3.4.

### 3.2 Hardware desenvolvido

Para alcançar os requisitos necessários, foi desenvolvido um datalogger, cujo esquemático se encontra no Apêndice 6, que forneça funcionalidades essenciais de armazenamento de dados para o projeto, tendo o menor tamanho e custo possível para que os objetivos sejam cumpridos.

Figura 8 – Datalogger desenvolvido.



Fonte: O Autor.

Visando a alta taxa de processamento, a placa contém um clock de 150 MHz e o FPGA Spartan-3A. A alta velocidade do clock é devido aos problemas encontrados por Korol (2015), possibilitando a aquisição de dados em maior frequência.

O datalogger também contém um chip de memória S34ML02G1, para o armazenamento dos dados na velocidade necessária. As características deste chip são discutidas na Seção 3.4

### 3.3 Determinação dos requisitos do sistema

Foi considerado como aplicação base a medição dos períodos dos pulsos de um sinal TTL, que é pulsado cada vez que o dente da engrenagem de um eixo em rotação passa em frente a um sensor magnético. O dado armazenado pelo datalogger, portanto, será a contagem de ciclos de clock por período de pulso.

A capacidade da memória foi estimada a partir do trabalho de Korol (2015), que determinou a frequência de rotação mínima do eixo sendo 1000 rpm com 150 dentes na engrenagem. Utilizando a equação 2.1 obtém-se a frequência equivalente de 2,5 kHz. Conhecendo o oscilador empregado no hardware, visto na Seção 3.2, de 150 MHz, é possível determinar o número de ciclos que cada pulso comportaria: 60.000 ciclos, ou 16 bits; este seria o tamanho máximo de cada dado.

Em trabalho colaborativo com Erbs (2020), foi considerada medições realizadas durante 20 minutos. A fim de determinar a capacidade máxima da memória, deve-se assumir a maior frequência de rotação também como o tamanho de 16 bits para cada dado, como visto anteriormente. Korol (2015) considerou a velocidade de rotação máxima sendo 7000 rpm; substituindo este valor na equação 2.1, o resultado equivalente é 17,5 kHz. Portanto, com 17,5 mil dados por segundo, em 20 minutos atinge-se em torno de 321 Mbit.

### 3.4 Memória Flash S34ML02G1

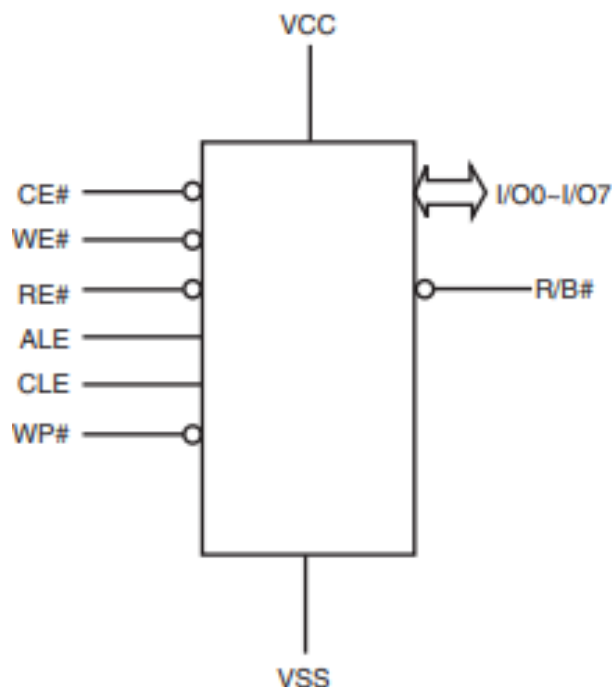
A fim de realizar o armazenamento dos dados lidos através do *Datalogger* em alta velocidade, foi utilizada a memória Flash S34ML02G1, que, de acordo com Cypress Semiconductor Corporation (2018) oferece 2 Gb de armazenamento e ciclos de leitura de 25 ns, com oito canais de entrada e saída, 100.000 ciclos de programar/apagar e 10 anos de retenção de dados. A estrutura de sua memória pode ser vista na Figura 9 e a estrutura dos blocos de memória do S34ML02G1 pode ser visto na Figura 10. O tamanho de cada página, bloco ou plano de memória pode ser visto na Tabela 4.

Tabela 4 – Estrutura de memória do chip S34ML02G1

Coluna	1 byte
Página	(2k + 64) bytes
Bloco	(2k + 64) bytes x 64 páginas = (128k + 4k) bytes
Plano	(128k + 4k) bytes x 1024 blocos = 1 Gb + 32 Mb

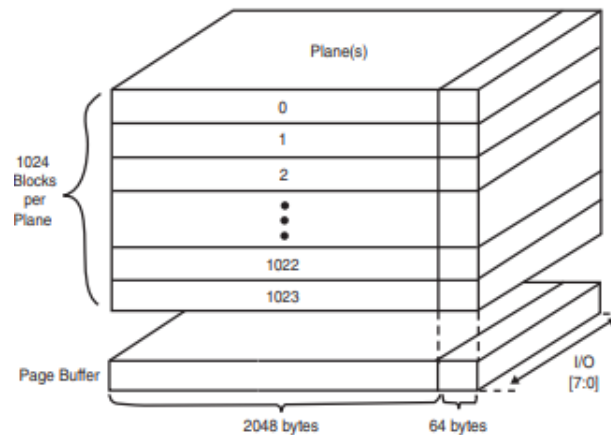
Fonte: O Autor.

Figura 9 – Diagrama lógico do chip de memória S34ML02G1



Fonte: Cypress Semiconductor Corporation (2018).

Figura 10 – Estrutura dos blocos de memória do chip S34ML02G1



Fonte: Cypress Semiconductor Corporation (2018).

Como visto na Tabela 4, cada plano possui 138.412.032 palavras. Considerando que cada palavra corresponde a 8 bits, obtemos 1 Gb + 32 Mb. No caso do S34ML02G1, há dois planos de memória, resultando, finalmente, em 276.824.064 palavras, ou 2 Gb + 64 Mb. Este tamanho de memória atende ao mínimo especificado na Seção 3.3, deixando espaços vazios caso seja necessário.

Na Figura 9, nota-se a presença de seis sinais de comando, oito canais de entrada e saída e um sinal de saída. Estes são apresentados na Tabela 5.

Tabela 5 – Pinos de comando do chip S34ML02G1

CE#	<b>Chip Enable.</b> Esta entrada controla a seleção do dispositivo. Quando o dispositivo não está ocupado, CE# baixo seleciona a memória.
WE#	<b>Write Enable.</b> Esta entrada trava comandos, endereços e dados. As entradas I/O são travadas na borda de subida do WE#.
RE#	<b>Read Enable.</b> A entrada RE# é o controle de saída de dados, e quando ativo leva os dados ao <i>bus</i> de I/O. Dados são válidos após a borda de descida de RE#, que também incrementa em um o contador interno do endereço de coluna.
ALE	<b>Address Latch Enable.</b> Esta entrada ativa o travamento das entradas I/O dentro do registro de endereços na borda de subida do WE#.
CLE	<b>Command Latch Enable.</b> Este sinal ativa o travamento das entradas I/O dentro do registro de comandos na borda de subida do WE#.
WP#	<b>Write Protect.</b> O pino WP#, quando baixo, fornece proteção em hardware contra modificações de dados indesejáveis (programação/apagamento).
R/B#	<b>Ready Busy.</b> A saída Pronto/Ocupado é um pino dreno aberto que sinaliza o estado da memória.
I/O0- I/O7	<b>Inputs/Outputs.</b> Os pinos I/O são usados para entrada de comandos, endereços, dados e saída de dados. Os pinos I/O flutuam para alta impedância quando o dispositivo está de-selecionado ou as saídas estão desativadas.

Fonte: Cypress Semiconductor Corporation (2018) (tradução nossa).

O modo de endereçamento no S34ML02G1 é descrito na Figura 11. Cada campo pode ser descrito da seguinte forma:

- A0 - A11: Endereço da coluna na página
- A12 - A17: Endereço da página no bloco
- A18: Endereço do plano
- A19 - A28: Endereço do bloco

Figura 11 – Modo de endereçamento do chip S34ML02G1

Bus Cycle	I/O [15:8] [14]	I/O0	I/O1	I/O2	I/O3	I/O4	I/O5	I/O6	I/O7
×8									
1st / Col. Add. 1	—	A0 (CA0)	A1 (CA1)	A2 (CA2)	A3 (CA3)	A4 (CA4)	A5 (CA5)	A6 (CA6)	A7 (CA7)
2nd / Col. Add. 2	—	A8 (CA8)	A9 (CA9)	A10 (CA10)	A11 (CA11)	Low	Low	Low	Low
3rd / Row Add. 1	—	A12 (PA0)	A13 (PA1)	A14 (PA2)	A15 (PA3)	A16 (PA4)	A17 (PA5)	A18 (PLA0)	A19 (BA0)
4th / Row Add. 2	—	A20 (BA1)	A21 (BA2)	A22 (BA3)	A23 (BA4)	A24 (BA5)	A25 (BA6)	A26 (BA7)	A27 (BA8)
5th / Row Add. 3	—	A28 (BA9)	Low	Low	Low	Low	Low	Low	Low

Fonte: Cypress Semiconductor Corporation (2018).

O endereço completo, portanto, deve ser enviada em 5 ciclos de escrita, no qual os dois primeiros representam a coluna (palavra), e os 3 seguintes representam a



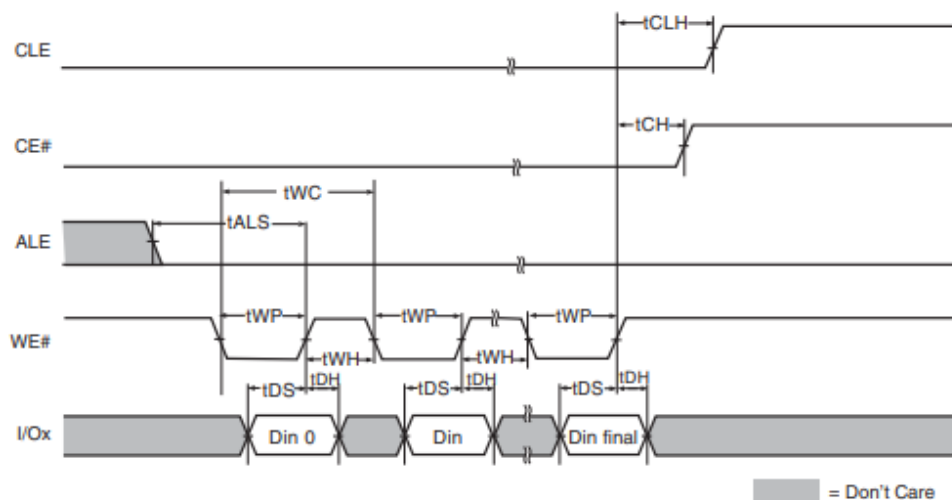
fileira da memória. Os endereços limitantes de cada campo é 2112 para a coluna, 64 para a página, 1024 para o bloco e 2 para o plano, como descrito na Tabela 4.

### 3.4.1 Escrita de dados

A operação de entrada de dados permite que os dados a serem programados sejam enviados ao dispositivo (CYPRESS SEMICONDUCTOR CORPORATION, 2018, p. 13, tradução nossa). Na Figura 12, é exposto o diagrama de sua operação. Como descrito na Tabela 5, os dados são escritos na borda de subida do WE#.

O período do pulso de WE#, descrito por  $t_{WP}$  é de 12 ns; o tempo total do ciclo de escrita,  $t_{WC}$ , é 25 ns; o tempo que os dados devem ser mantidos após a borda de subida de WE#,  $t_{DH}$ , é de 5 ns; o tempo total que o dado deve estar nos canais,  $t_{DS}$ , é de 10 ns; os tempos antes de subir os sinais de comando CLE e CE#,  $t_{CLH}$  e  $t_{CH}$  respectivamente, são ambos de 5 ns; e o tempo entre uma borda de descida de ALE e uma borda de subida de WE#,  $t_{ALS}$ , é de 10 ns. Estes são todos tempos mínimos, podendo levar períodos maiores para a comutação de sinais, mas não menores.

Figura 12 – Diagrama para entrada de dados.



Fonte: Cypress Semiconductor Corporation (2018).

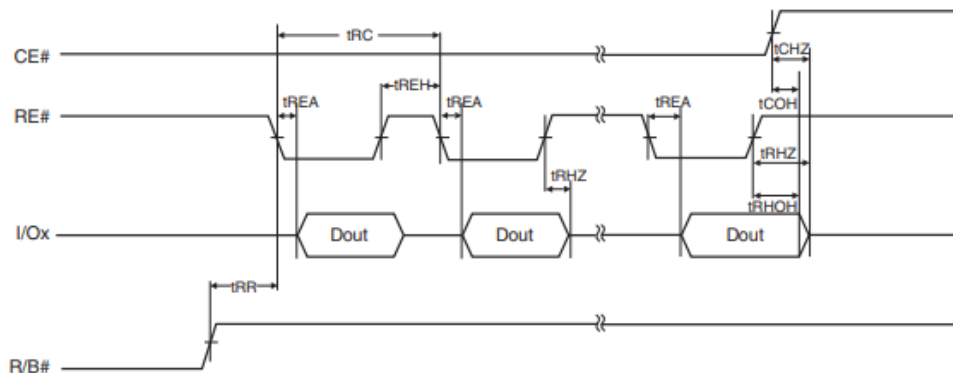
### 3.4.2 Leitura de dados

A operação de saída de dados permite que dados sejam lidos da memórias, assim como registradores de estado e dados de identificação (CYPRESS SEMICONDUCTOR CORPORATION, 2018, p. 13, tradução nossa). Os dados podem ser lidos nas bordas de subida do RE#. Na Figura 13 se encontra o diagrama para esta operação.

O tempo total do ciclo de leitura,  $t_{RC}$ , é de 25 ns; o tempo que RE# deve ser mantido em alto após sua borda de subida,  $t_{REH}$ , é de 10 ns; o tempo máximo para que

dados possam ser lidos após a borda de descida de RE#,  $t_{REA}$ , é de 25 ns; o tempo máximo para que o I/O volte a alta impedância,  $t_{RHZ}$ , é de 100 ns; o tempo mínimo de permanência da saída dos dados após a borda de subida de RE#,  $t_{RHOH}$ , é de 15 ns; o tempo mínimo a ser esperado entre o dispositivo estar pronto e a borda de descida de RE#,  $t_{RR}$ , é de 20 ns.

Figura 13 – Diagrama para saída de dados.



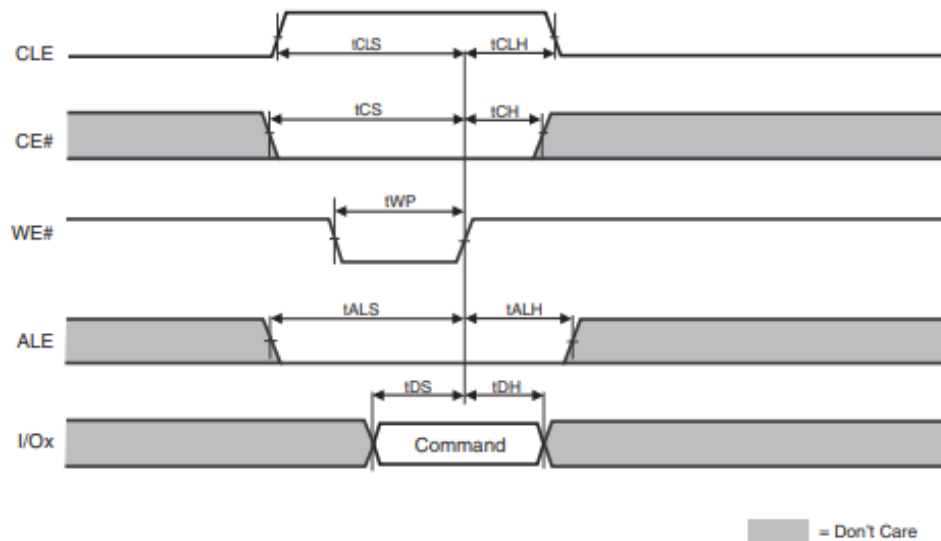
Fonte: Cypress Semiconductor Corporation (2018)

### 3.4.3 Entrada de comandos

A operação de entrada de comandos permite enviar comandos ao dispositivo de memória (CYPRESS SEMICONDUCTOR CORPORATION, 2018, p. 13, tradução nossa). Comandos são aceitos quando CLE estiver em estado lógico alto, e são travados na borda de subida de WE#. Na Figura 14 é mostrado o diagrama para realização deste comando. Nota-se que esta operação inclui a entrada de dados, como mostrado anteriormente. A única diferença, portanto, é o pulso do CLE.

Os tempos mínimos  $t_{CLS}$  e  $t_{CLH}$  são 10 e 5 ns, respectivamente.

Figura 14 – Diagrama para escrita de comandos.



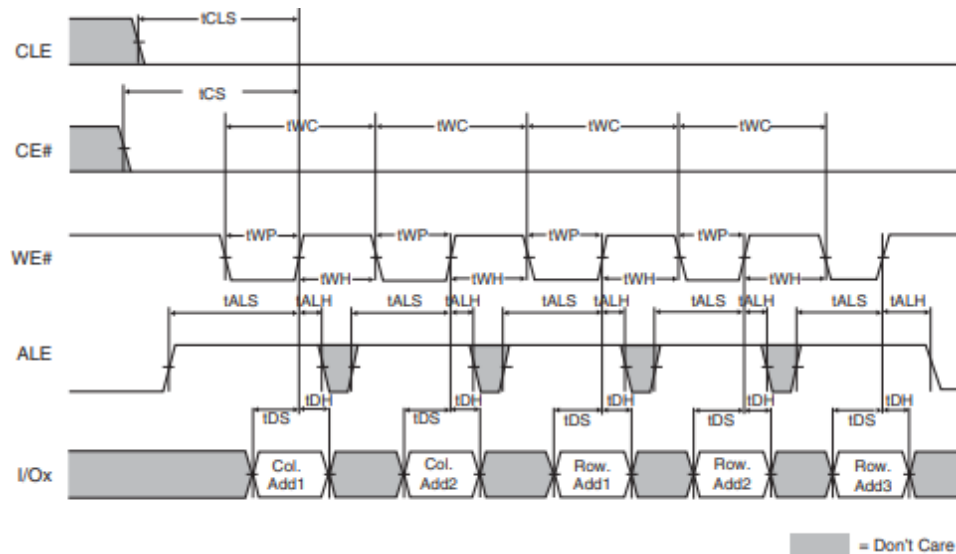
Fonte: Cypress Semiconductor Corporation (2018)

#### 3.4.4 Entrada de endereços

A operação de entrada de endereços permite a inserção de endereços de memória (CYPRESS SEMICONDUCTOR CORPORATION, 2018, p. 13, tradução nossa). Para isso, são utilizados cinco ciclos, como descrito na Figura 11. Endereços são aceitos com o sinal ALE em nível lógico alto, e travados na borda de subida de WE#. Seu diagrama pode ser visto na Figura 15 e, assim como na entrada de comandos, utiliza operação de escrita de dados. Cada ciclo é marcado por um pulso de WE#.

Os tempos mínimos  $t_{ALS}$  e  $t_{ALH}$  são 10 e 5 ns, respectivamente.

Figura 15 – Diagrama para escrita de endereços.



Fonte: Cypress Semiconductor Corporation (2018)

### 3.4.5 Leitura de página

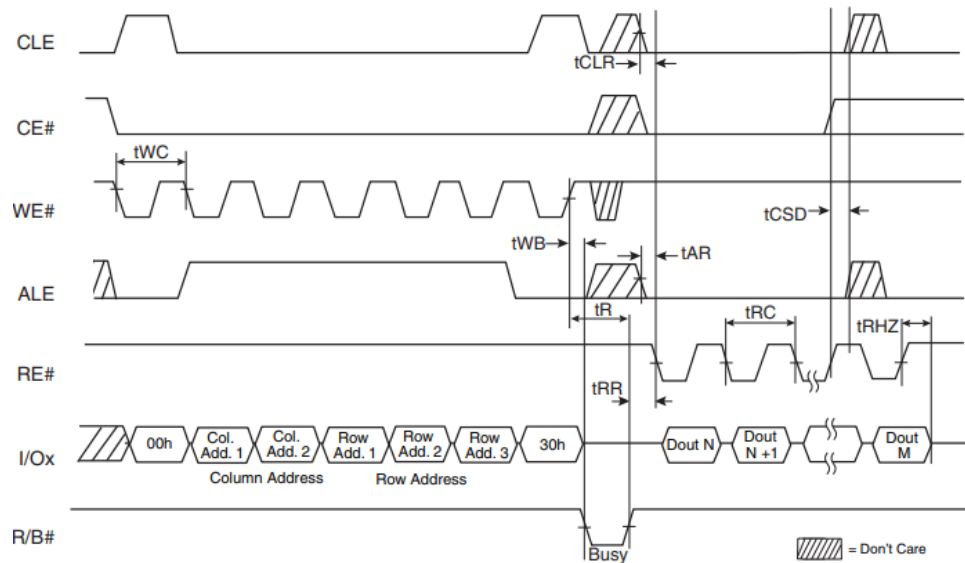
A operação de leitura de página permite a saída de dados em sequência de uma página desejada. Ela pode ser iniciada utilizando os comandos 00h e 30h, e deve ser realizada apenas quando o dispositivo não estiver ocupado, como indicado por R/B#. Todos os dados na página são transferidos aos registradores de dados; o controlador pode detectar a conclusão da transferência ao analisar a saída de R/B# (CYPRESS SEMICONDUCTOR CORPORATION, 2018, p. 15, tradução nossa).

De acordo com Cypress Semiconductor Corporation (2018), os dados podem ser lidos, através de pulsos no RE#, em ciclos de 25 ns após sua transferência aos registradores. Estes pulsos permitem a saída de dados do dispositivo, iniciando na coluna selecionada, até o último endereço de coluna.

Na Figura 16 é mostrado o diagrama para leitura de página. Observa-se a sequência necessária antes de realizar a leitura, consistindo de um comando, cinco endereços, e outro comando, os quais são os mesmos apresentados anteriormente.

Os tempos máximos  $t_{WB}$  e  $t_R$  são 100 ns e 25  $\mu$ s, enquanto o tempo mínimo  $t_{RR}$  é 20 ns.

Figura 16 – Diagrama para leitura de página.



Fonte: Cypress Semiconductor Corporation (2018)

### 3.4.6 Programação de página

A programação de página permite a escrita de dados sequenciais no dispositivo de memória.

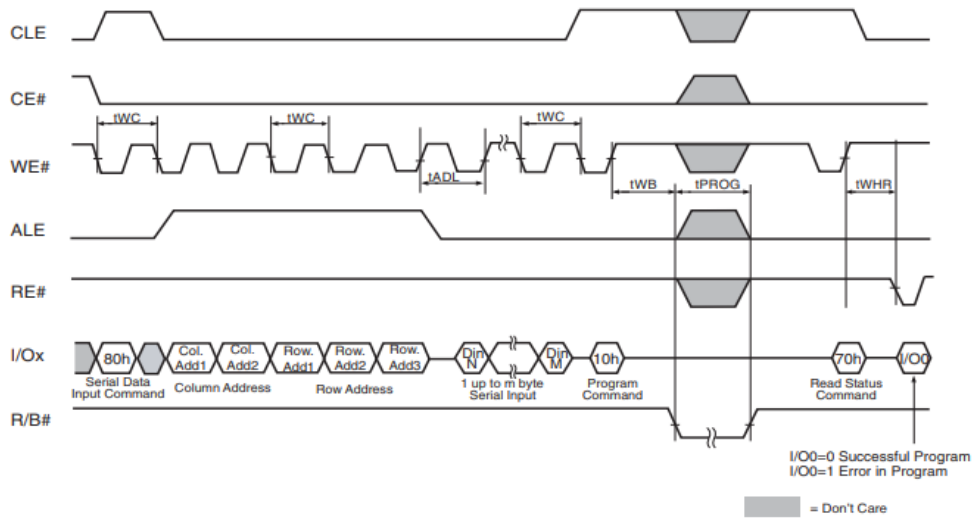
Um ciclo de programação de página consiste em um carregamento de dados seriais ao registrador de dados de até 2112 bytes, seguido de um período de programação não-volátil, onde os dados carregados são programados na célula apropriada (CYPRESS SEMICONDUCTOR CORPORATION, 2018, p. 15, tradução nossa)

De acordo com Cypress Semiconductor Corporation (2018), o procedimento consiste em enviar o comando 80h, seguido do endereço de memória, e dos dados seriais. Assim que terminado a entrada de dados, o comando 10h deve ser enviado para que a programação inicie. A conclusão da programação pode ser monitorada através do sinal R/B#.

Na Figura 17 é exposto o diagrama para programação de páginas. Nota-se a checagem do estado após a programação ser concluída. Isto é feito para garantir que não houve falhas durante a execução.

O tempo máximo  $t_{PROG}$  é de 700  $\mu s$ , mas sendo tipicamente de 200  $\mu s$ .

Figura 17 – Diagrama para programação de página.



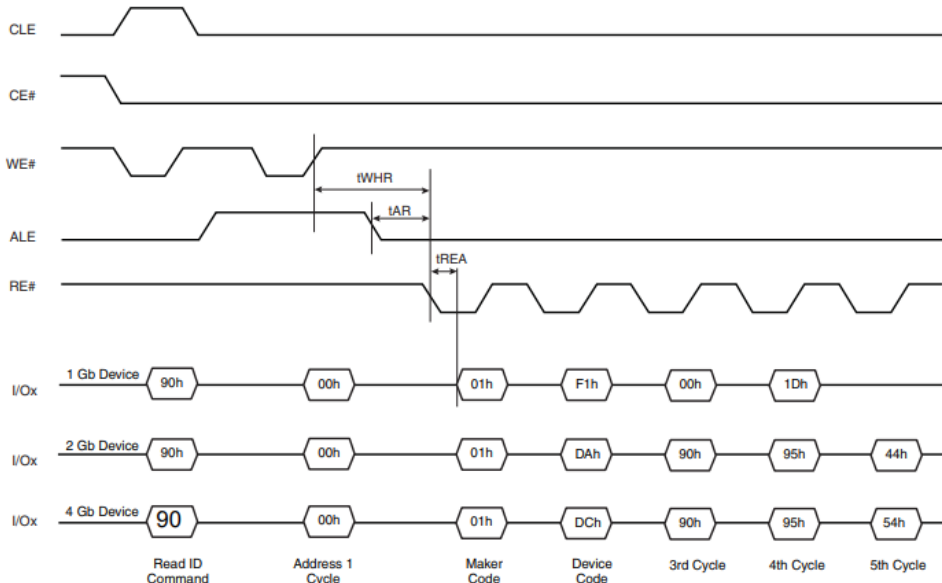
Fonte: Cypress Semiconductor Corporation (2018)

### 3.4.7 Leitura de identificação

O dispositivo possui um modo de identificação do produto, iniciado através da escrita do comando 90h, seguido do endereço 00h Cypress Semiconductor Corporation (2018, p. 26, tradução nossa). Para o chip utilizado, de 2 Gb, são enviados dados em cinco ciclos.

Seu diagrama, assim como os dados esperados, são mostrados na Figura 18.

Figura 18 – Diagrama para leitura de identificação.



Fonte: Cypress Semiconductor Corporation (2018)

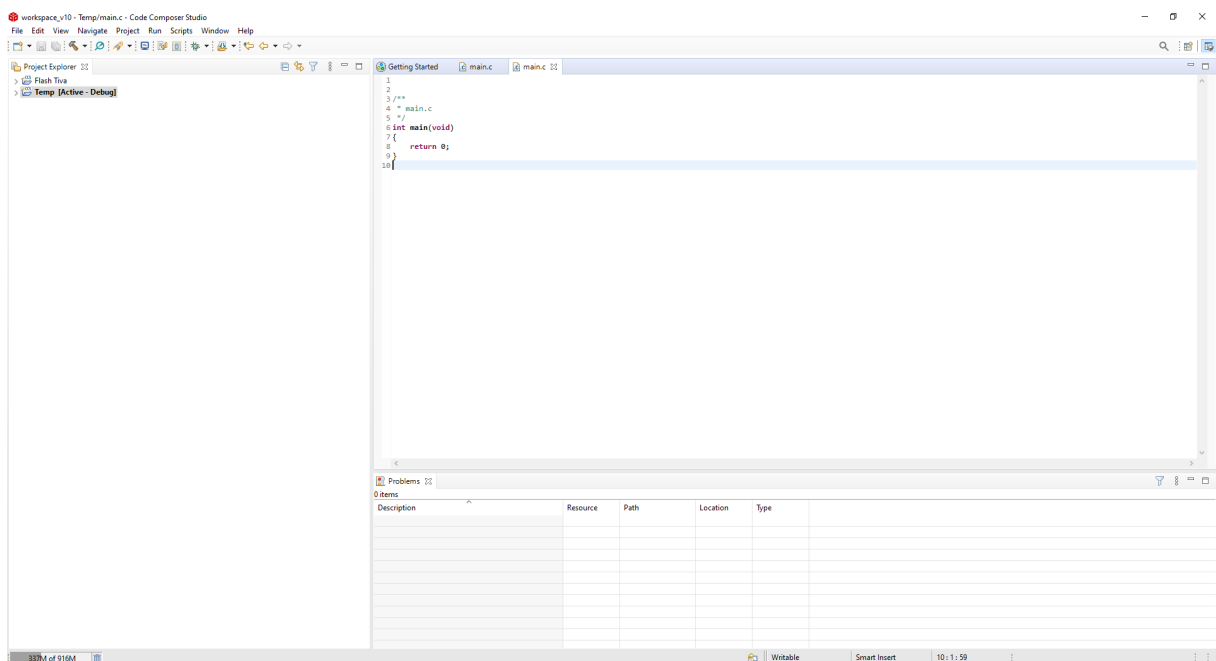
### 3.5 Softwares utilizados

#### 3.5.1 Code Composer Studio

O Code Composer Studio (CCS) é um Ambiente Integral de Desenvolvimento (IDE) desenvolvido pela Texas Instruments (TI). Este software é especializado em ferramentas de programação e debugging de microcontroladores desenvolvidos pela TI.

Assim como IDEs convencionais, o CCS oferece meios como *breakpoints* e visualização de variáveis em tempo real, além de artefatos mais complexos como constatação do estado da memória do microcontrolador após erros ocasionados.

Figura 19 – Code Composer Studio



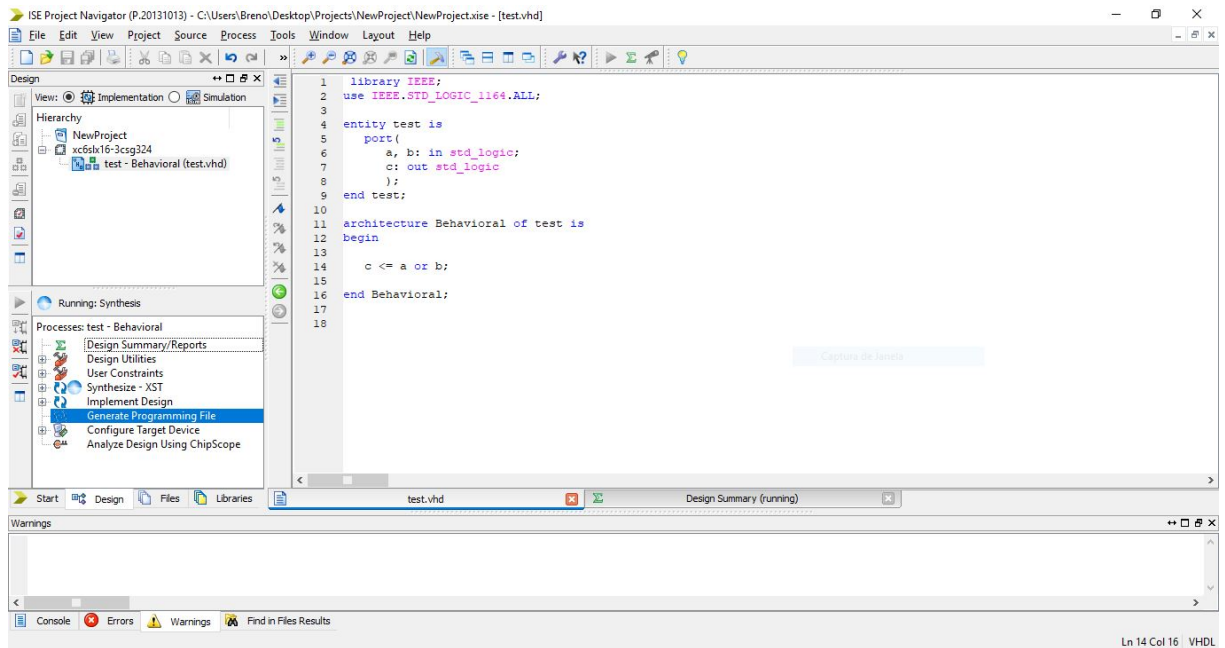
Fonte: O Autor.

#### 3.5.2 ISE

O ISE Project Navigator é utilizado para sintetização, implementação do projeto, e geração do arquivo de programação. É nele onde é feito todo o código em HDL.

O software proporciona, também, ferramentas de detecção de erros. Entretanto, estas ferramentas não são as mesmas de um IDE comum para linguagens de programação, já que o HDL é uma linguagem de descrição de circuitos digitais e deve garantir que os circuitos originados sejam possíveis quando traduzidos para o hardware após sintetizados. Portanto, o programa tende a apontar problemas que poderiam ser indesejados durante a programação, tais como latches inferidos, que costumam ser causados por declarações *if/else* ou *case* incompletas.

Figura 20 – ISE Project Navigator



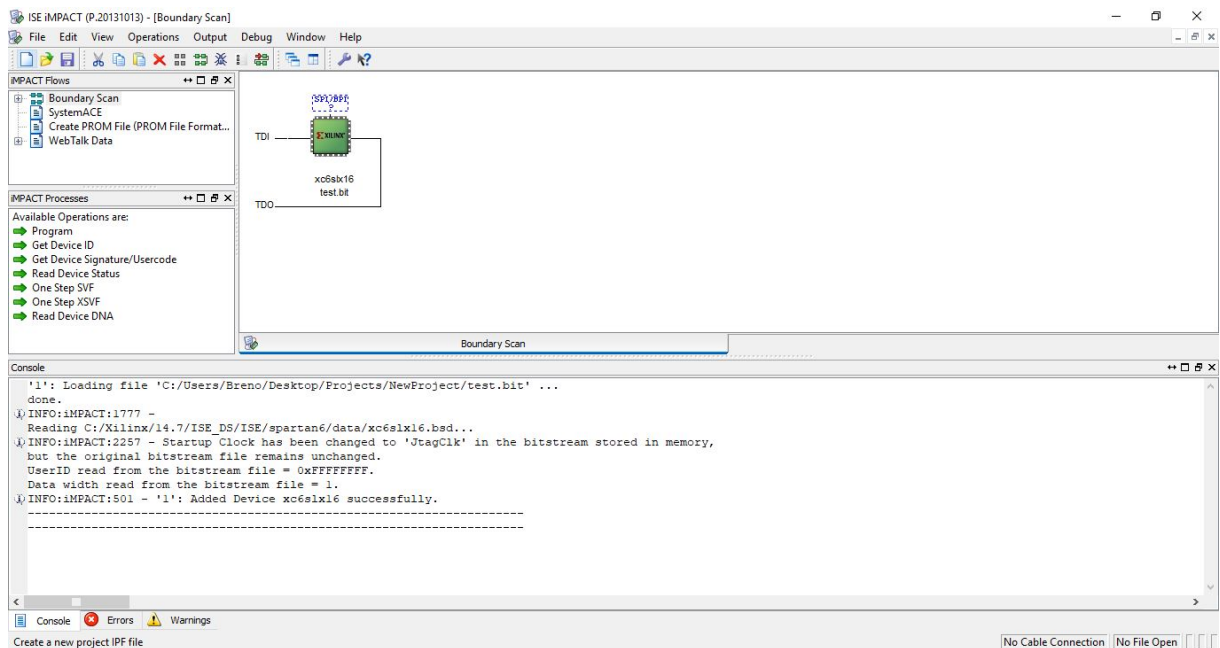
Fonte: O Autor.

### 3.5.3 iMPACT

O iMPACT é o software utilizado para programar o FPGA após o arquivo de programação ser gerado pelo ISE. O dispositivo FPGA necessita de uma plataforma de comunicação Joint Test Action Group (JTAG), a fim de realizar a programação. Ele também oferece a utilidade de escanear por dispositivos conectados, a fim de detectar erros de comunicação com o dispositivo FPGA.



Figura 21 – iMPACT



Fonte: O Autor.

### 3.5.4 iSim

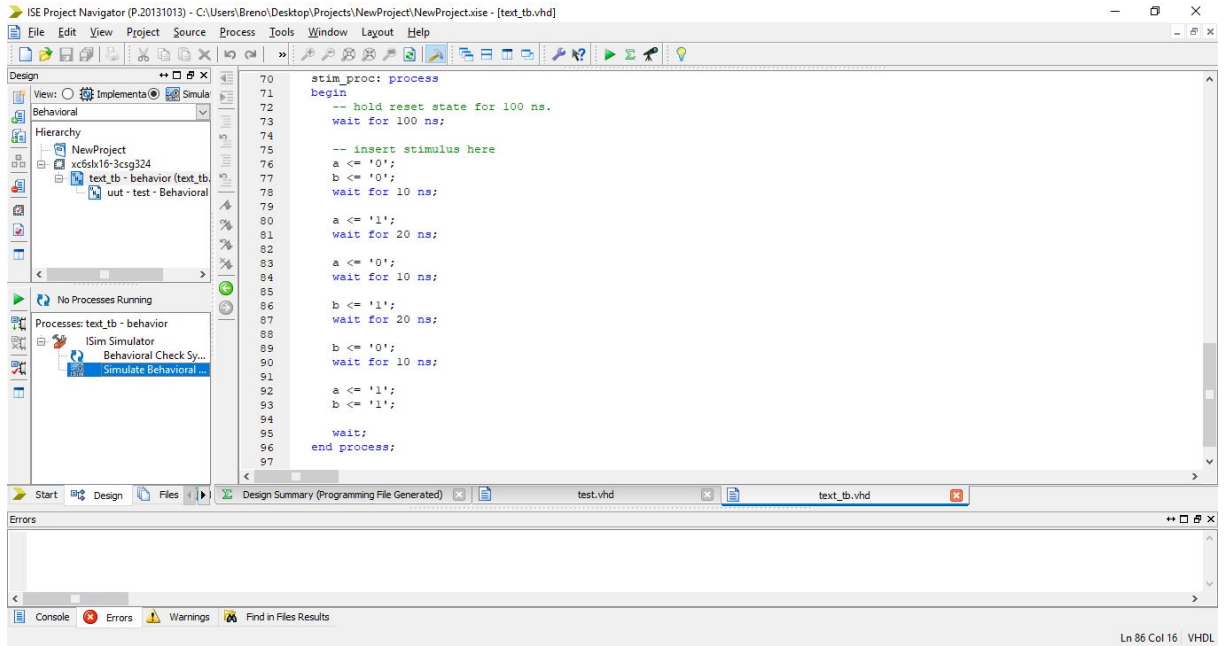
As simulações do circuito sintetizado são realizadas no iSim. Neste, são geradas entradas programaticamente e a forma de onda de suas saídas são mostradas em uma janela à parte. A simulação é ideal, não levando em conta imprevisibilidades causadas, por exemplo, por latches que podem estar presentes no circuito.

Antes de gerar a simulação, é necessário criar uma *test bench*, gerada pelo próprio ISE, e associá-la ao módulo do circuito a ser testado. Esta será responsável pela sequência de sinais durante a simulação. Os sinais de entradas são determinados através da estipulação de seus níveis lógicos, junto de atrasos de tempo até a próxima transição. Então, a forma de onda é gerada seguindo a lógica interna do circuito e mostrada em uma janela dedicada.

Além da possibilidade de visualizar o comportamento das saídas, pode-se também mostrar os sinais auxiliares de módulos instanciados dentro do módulo principal. Isto facilita verificar, por exemplo, que as transições das máquinas de estados estão de acordo com o esperado.

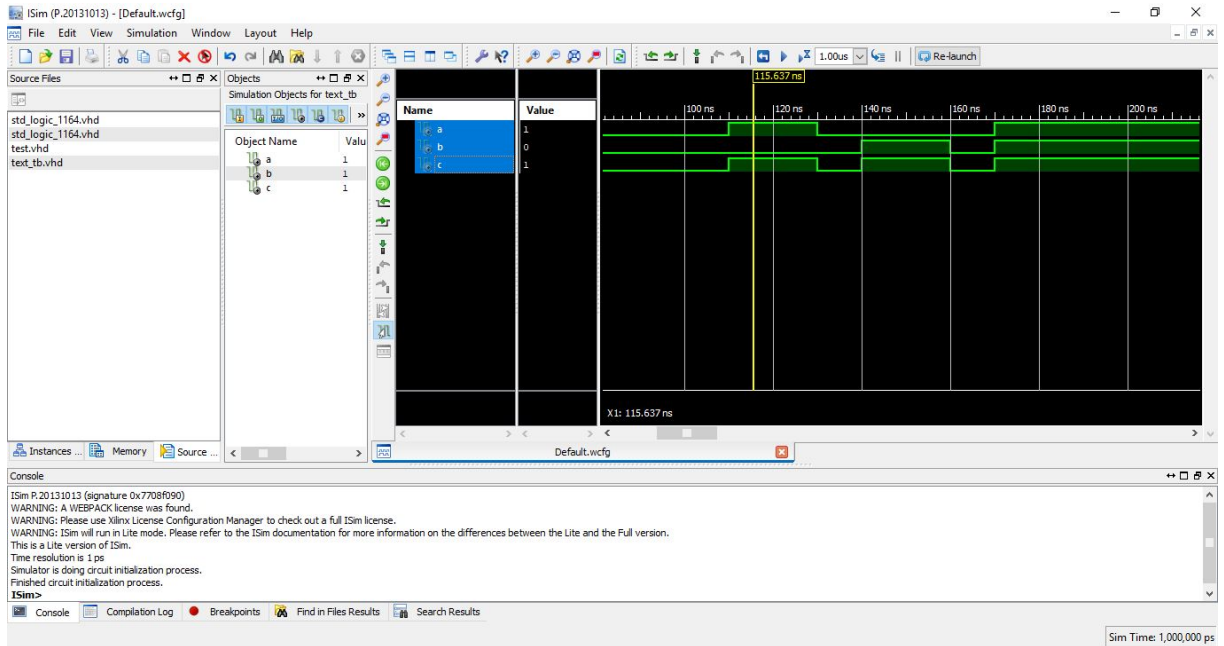
A test bench da simulação de uma porta OR pode ser vista na Figura 22, assim como suas formas de ondas geradas na Figura 23.

Figura 22 – Test Bench do software iSim



Fonte: O Autor.

Figura 23 – Simulação de uma porta OR utilizando o iSim



Fonte: O Autor.

## 4 DESENVOLVIMENTO

### 4.1 Projeto no microcontrolador Tiva

A fim de observar o funcionamento do chip e validar os métodos utilizados na escrita da memória, seus testes iniciais foram feitos utilizando o microcontrolador Tiva C Series TM4C123GH6PM.

Primeiramente, foram replicados os diagramas para entrada e saída de dados no chip, com respeito a seus sinais de comando. Na Figura 12, é exposto a operação de entrada de dados. Nota-se que os dados são escritos na borda de subida do sinal WE#.

Desta forma, o código no microcontrolador foi escrito como mostrado a seguir. Observe que o sinal WE# deve ficar em nível lógico baixo durante 12 ns durante a operação. No microcontrolador utilizado, a frequência de seu clock é de 80 MHz, logo um único ciclo é equivalente a 12.5 ns, o que seria suficiente para realizar o atraso necessitado. A função delay() é apenas uma função auxiliar caso haja a necessidade de aumentar este tempo com facilidade.

```

1 void mem_write(uint8_t data)
2 {
3     // Setting IO as output
4     io_configure_output();
5
6     // Set WE low to start write cycle
7     GPIOPinWrite(OUT_BASE, WE, 0x00);
8
9     // Writing to IO
10    io_write(data);
11
12    // Waiting for 12ns
13    SysCtlDelay(delay(1));
14
15    // Holding WE high for 12.5ns
16    GPIOPinWrite(OUT_BASE, WE, 0xFF);
17    SysCtlDelay(delay(1));
18
19    // Resetting IO
20    io_write(0x00);
21 }

```

Seguido, é mostrado o código para a saída de dados. Este, diferente da entrada de dados, tem como sinal principal o RE#.

A permanência dos dados frente à borda de subida de RE# é de 15 ns, como visto na Figura 13 e determinado pelo tempo  $t_{RHOH}$ . Portanto, a leitura de dados é feita imediatamente após o tempo mínimo de espera pós borda de descida de RE#, e antes de sua borda de subida, para evitar que este período seja ultrapassado devido ao overhead de cada função.

O código pode ser visto abaixo.

```

1 uint8_t mem_read()
2 {
3     // Setting IO as input
4     io_configure_input();
5
6     // Setting RE low to start read cycle
7     GPIOPinWrite(OUT_BASE, RE, 0x00);
8
9     // Delay to get a valid data
10    SysCtlDelay(1);
11
12    // Data read
13    uint8_t data = io_read();
14
15    // Setting RE high to complete cycle
16    GPIOPinWrite(OUT_BASE, RE, 0xFF);
17
18    // Waiting for cycle complete
19    SysCtlDelay(delay(1));
20
21    return data;
22 }

```

Tendo as rotinas de entrada e saída feitas, atentou-se às ações de escrita de comandos e endereços. Estas são realizadas utilizando uma combinação da função de entrada desenvolvida, junto com a operação sinais CLE e ALE para comandos e endereços, respectivamente. Os diagramas são vistos nas figuras 14 e 15. Nota-se que as operações devem ser realizadas enquanto os sinais CLE ou ALE se encontram em nível lógico alto.

Os períodos que CLE e ALE devem permanecer em nível lógico alto em relação à borda de subida de WE# são respeitados devido à própria função de escrita, que realiza atrasos internos. Consequentemente, as funções de entrada de comandos e endereços não precisam se preocupar com este fato, e apenas lidam com as transições dos sinais de comando.

Os códigos para estas operações são expostos abaixo.

```

1 void address_latch(uint8_t addr)
2 {
3     // Setting ALE high
4     GPIOPinWrite(OUT_BASE, ALE, 0xFF);
5     mem_write(addr);
6
7     // ALE low
8     GPIOPinWrite(OUT_BASE, ALE, 0x00);
9 }

```

```

1 void command_latch(uint8_t cmd)
2 {
3     // Setting CLE high
4     GPIOPinWrite(OUT_BASE, CLE, 0xFF);
5     mem_write(cmd);
6
7     // CLE low
8     GPIOPinWrite(OUT_BASE, CLE, 0x00);
9 }

```

Tendo todas as funções base desenvolvidas, o objetivo tornou-se validar o funcionamento do chip. Isso fez-se utilizando uma operação com saída previsível: a de leitura da identificação do chip. Seu diagrama pode ser visto na Figura 18.

Nesta operação, temos a escrita de um comando e um endereço. O comando, específico para cada operação, é 0x90, enquanto o endereço neste caso é fixado em 0x00. Após isso, são feitos cinco ciclos de leitura, sendo que cada ciclo irá retornar uma parte da identificação. Os dados retornados, em ordem, em cada ciclo são apresentados na Figura 18. Para o chip de 2 Gb utilizado, o esperado seria 0x01, 0xDA, 0x90, 0x95 e 0x44.

O código desenvolvido para esta operação pode ser visto abaixo.

```

1 void op_read_id(uint8_t *data)
2 {
3     // Command latch 0x90
4     command_latch(CMD_READ_ID);
5
6     // Address latch 0x00
7     address_latch(ADDR_READ_ID);
8
9     // 1st cycle: Maker code (0x01)           0000 0001
10    // 2nd cycle: Device code (0xDA)          1101 1010
11    // 3rd cycle: 0x90                        1001 0000
12    // 4th cycle: 0x95                        1001 0101
13    // 5th cycle: 0x44                        0100 0100
14    int cycle;
15    for(cycle = 0; cycle < 5; cycle++)
16    {

```

```

17     data[cycle] = mem_read();
18 }
19 }

```

## 4.2 Projeto em VHDL para FPGA

Após desenvolvido o código para o microcontrolador, foi feito o código em VHDL para sintetização no FPGA. Foram criados vários módulos, a fim de facilitar o desenvolvimento em níveis mais altos do projeto.

### 4.2.1 Timer

O código da máquina de estados do módulo de timer pode ser visto abaixo. Este é de grande importância, já que há atrasos específicos durante cada operação. Sua máquina de estados possui apenas dois estados: *idle* e *counting*. O timer irá esperar por um número de ciclos de clock, definido pelo sinal *counter*, e iniciará quando o sinal *start* estiver em nível lógico alto. O fim é determinado por um pulso no sinal *done*.

Observe que os tempos de atraso são dados em ciclos de clock, e não em segundos. Portanto, deve-se levar em conta a frequência do seu clock antes de utilizá-lo. Tendo isso em vista, o tempo mínimo de atraso se limita ao mínimo de um único ciclo de clock.

```

1  process(state_reg, counter, counter_reg, start)
2  begin
3      state_next <= state_reg;
4      counter_next <= counter_reg;
5      done <= '0';
6
7      case state_reg is
8          when idle =>
9              if start = '1' then
10                 counter_next <= unsigned(counter) - 1;
11                 state_next <= counting;
12             end if;
13         when counting =>
14             if counter_reg = 0 then
15                 done <= '1';
16                 state_next <= idle;
17             else

```

```

18         counter_next <= counter_reg - 1;
19     end if;
20 end case;
21 end process;

```

#### 4.2.2 Escrita e leitura

Assim como no código para o microcontrolador, os módulos de entrada e saída de dados são similares. Cada módulo possui dois timers, sendo um timer para os sinais WE# ou RE# em nível lógico baixo, e o outro para estes mesmos em nível lógico alto. Isto foi feito para poupar ciclos de clock que eram inseridos quando utilizava-se apenas um módulo.

Ambos os módulos possuem sinais *done* para indicar o fim da escrita ou leitura. Além disso, o módulo de leitura possui um sinal *data\_valid* que indica em qual momento é possível fazer a leitura do IO, já que estes dados são travados nas saídas após um tempo mínimo depois da borda de descida de RE#. O sinal *data\_valid*, portanto, é acionado na borda de subida de RE#.

```

----- Escrita -----
1 process(state_reg, timer_low_start_reg, timer_low_counter_reg, start,
  ↳ timer_low_done, timer_high_start_reg, timer_high_counter_reg,
  ↳ timer_high_done)
2 begin
3     timer_low_start_next <= '0';
4     timer_low_counter_next <= (others => '0');
5
6     timer_high_start_next <= '0';
7     timer_high_counter_next <= (others => '0');
8
9     state_next <= state_reg;
10    done <= '0';
11
12    case state_reg is
13        when idle =>
14            if start = '1' then
15                state_next <= we_low;
16            end if;
17
18        when we_low =>

```

```

19         timer_low_start_next <= '1';
20         timer_low_counter_next <= "1";
21
22         if timer_low_done = '1' then
23             state_next <= we_high;
24         end if;
25
26     when we_high =>
27         timer_high_start_next <= '1';
28         timer_high_counter_next <= "1";
29
30         if timer_high_done = '1' then
31             state_next <= idle;
32             done <= '1';
33         end if;
34
35     end case;
36
37 end process;
38
39 WE <= '0' when state_reg = we_low else
40     '1';

```

#### Leitura

```

1 process(state_reg, timer_low_start_reg, timer_low_counter_reg,
  ↳ start, timer_low_done, timer_high_start_reg, timer_high_counter_reg,
  ↳ timer_high_done, io)
2 begin
3     timer_low_start_next <= '0';
4     timer_low_counter_next <= (others => '0');
5
6     timer_high_start_next <= '0';
7     timer_high_counter_next <= (others => '0');
8
9     state_next <= state_reg;
10    data_valid <= '0';
11    done <= '0';
12

```



```

13     case state_reg is
14         when idle =>
15             if start = '1' then
16                 state_next <= re_low;
17             end if;
18
19         when re_low =>
20             timer_low_start_next <= '1';
21             timer_low_counter_next <= "1";
22
23             if timer_low_done = '1' then
24                 state_next <= re_high;
25             end if;
26
27         when re_high =>
28             timer_high_start_next <= '1';
29             timer_high_counter_next <= "1";
30             data_valid <= '1';
31
32             if timer_high_done = '1' then
33                 state_next <= idle;
34                 done <= '1';
35             end if;
36
37     end case;
38
39 end process;
40
41 RE <= '0' when state_reg = re_low else
42     '1';

```

#### 4.2.3 Controle tri-state

O módulo a seguir integra as duas funções de escrita e leitura, instanciando os módulos mostrados anteriormente. Este é responsável pelo controle tri-state do I/O. Ou seja: é o módulo que define quando os canais I/O devem estar em alta impedância, ou impondo um sinal de saída. A operação a ser realizada é definida por *rw*, onde 0 significa leitura, e 1 escrita.

O sinal de controle tri-state,  $T$ , determina que os I/Os devem permanecer em alta impedância apenas quando houver alguma ação de leitura ocorrendo. No caso contrário, os I/Os estarão atribuindo sinais de saída vindo de *io\_write*.

```
1  io <= io_write when T = '0' else (others => 'Z');
2  io_read <= io;
3
4  inst_read: entity work.read
5  port map(
6      clk => clk,
7      RE => read_re,
8      start => read_start,
9      data => read_data,
10     data_valid => read_data_valid,
11     done => read_done,
12     io => io_read
13 );
14
15 RE <= read_re;
16
17 inst_write: entity work.write
18 port map(
19     clk => clk,
20     data => write_data,
21     io => io_write,
22     WE => write_we,
23     done => write_done,
24     start => write_start
25 );
26
27 WE <= write_we;
28
29 process(clk)
30 begin
31     if rising_edge(clk) then
32         state_reg <= state_next;
33         read_data_reg <= read_data_next;
34         write_data_reg <= write_data_next;
35     end if;
```



```

73
74     when write_on =>
75         if write_done = '1' then
76             state_next <= idle;
77             done <= '1';
78         end if;
79
80     end case;
81 end process;
82
83 T <= '1' when state_reg = read_on or state_reg = read_valid else
84     '0';
85 data_in <= read_data_reg;
86 write_data <= write_data_reg;

```

#### 4.2.4 Controle de modos de entrada

O próximo módulo integra as funções de escrita e leitura, junto com o comando dos sinais CLE e ALE. As rotinas a serem realizadas são definidas a partir de um sinal de dois bits, *mode*. Este sinal admite os seguintes valores:

```

1  constant R: std_logic_vector(1 downto 0) := "00";
2  constant W: std_logic_vector(1 downto 0) := "01";
3  constant CMD: std_logic_vector(1 downto 0) := "10";
4  constant ADDR: std_logic_vector(1 downto 0) := "11";

```

As constantes *R*, *W*, *CMD* e *ADDR* representam, respectivamente: saída de dados, entrada de dados, entrada de comandos e entrada de endereços. A escolha por este meio se deu para evitar máquinas de estados maiores do que o necessário, finalmente se resumindo a apenas dois estados, com lógica explícita. O código pode ser visto a seguir.

```

1  process(mode, start, rw_done, state_reg, rw_start_reg)
2  begin
3      state_next <= state_reg;
4      rw_start_next <= '0';
5      done <= '0';
6

```

```

7     case state_reg is
8         when idle =>
9             if start = '1' then
10                state_next <= busy;
11                rw_start_next <= '1';
12
13            end if;
14
15        when busy =>
16            if rw_done = '1' then
17                state_next <= idle;
18                done <= '1';
19            end if;
20
21        end case;
22
23    end process;
24
25    rw <= '0' when mode = R else
26        '1';
27
28    ALE <= '1' when mode = ADDR and state_reg = busy else
29        '0';
30
31    CLE <= '1' when mode = CMD and state_reg = busy else
32        '0';

```

#### 4.2.5 Controle de operações

Finalmente, desenvolveu-se o módulo de controle de operações. A fim de evitar a criação a uma máquina de estados para cada operação, foi feita uma abordagem inspirada na linguagem de processadores, da forma que cada operação terá um vetor de ações a serem realizadas, e a máquina de estados será genérica para executar qualquer ordem necessária.

Portanto, foram definidos oito modos:

```

1    constant R: std_logic_vector(2 downto 0) := "000";
2    constant W: std_logic_vector(2 downto 0) := "001";

```

```

3  constant CMD: std_logic_vector(2 downto 0) := "010";
4  constant ADDR: std_logic_vector(2 downto 0) := "011";
5  constant CR: std_logic_vector(2 downto 0) := "100";
6  constant CW: std_logic_vector(2 downto 0) := "101";
7  constant WT: std_logic_vector(2 downto 0) := "110";
8  constant STP: std_logic_vector(2 downto 0) := "111";

```

Sendo que cada modo é definido por:

Tabela 6 – Definição dos possíveis modos dentro de operações

R	Leitura simples
W	Escrita simples
CMD	Entrada de comandos
ADDR	Entrada de endereços
CR	Leitura contínua.
CW	Escrita contínua
WT	Espera por R/B# válido
STP	Pare

Fonte: O Autor.

Portanto, cada operação deve ser definida utilizando uma combinação dentre estes oito modos em um vetor. O tipo do vetor, assim como a operação de identificação podem ser vistas a seguir.

```

1  type mode_file_type is array (0 to N_mode-1) of std_logic_vector(2
   ↪  downto 0);
2  constant OP_READ_ID: mode_file_type := (0=>CMD, 1=>ADDR, 2=>R, 3=>R,
   ↪  4=>R, 5=>R, 6=>R, others=>STP);

```

Onde  $N\_mode$  é um genérico que indica o número máximo possível de ações em uma única operação.

Para determinar quais comandos ou endereços devem ser entrados, ou se seus valores serão entradas de um módulo superior, há o vetor de entradas a seguir. Este vetor consiste de de nove bits, ao invés dos oito que costumam consistir uma palavra inteira; isto se dá com intenção de determinar se esta ação espera uma entrada externa, no bit mais significativo. A função *fullVector* retorna o vetor com o bit mais significativo '1' ou '0', dependendo da entrada esperada.

```

1  type dout_file_type is array (0 to N_MODE-1) of std_logic_vector(8
   ↪  downto 0);
2  constant OP_READ_ID_D: dout_file_type :=
   ↪  (0=>fullVector(orig=>CMD_READ_ID),
   ↪  1=>fullVector(orig=>ADDR_READ_ID), others => (others=>'0'));

```

A máquina de estados é definida no processo mostrado abaixo. Os estados *setup* e *busy* referem à execução em modos de escrita, leitura ou entrada de comandos e endereços. Já os estados *wait\_tadl*, *rb\_fixed\_wait* e *rb\_wait* são destinados à espera até que a memória esteja pronta para receber novos comandos; o tempo de espera fixo em *rb\_fixed\_wait* vem de  $t_{WB}$ , visto nas figuras 17 e 16, sendo no máximo 100 ns, enquanto  $t_{ADL}$  é encontrado em operações de entrada de dados logo após endereçamentos, sendo um mínimo de 70 ns.

```

1  process(start, state_reg, state_next_aux, done_t, step_reg, stop_edge,
   ↪  trigger_edge, continuous, rw_operation, timer_running, timer_done,
   ↪  RB, halt_mode)
2  begin
3      state_next <= state_reg;
4      start_t <= '0';
5      step_next <= step_reg;
6      timer_start <= '0';
7      timer_counter <= (others => '0');
8
9      case state_reg is
10         when idle =>
11             if start = '1' then
12                 state_next <= state_next_aux;
13             end if;
14
15         when setup =>
16             if rw_operation = '1' then
17                 if trigger_edge = '1' then
18                     start_t <= '1';
19                     state_next <= state_next_aux;
20                 end if;
21
22                 if stop_edge = '1' then

```

```
23         step_next <= step_reg + 1;
24     end if;
25     else
26         if halt_mode = '0' then
27             start_t <= '1';
28         end if;
29
30         state_next <= state_next_aux;
31     end if;
32
33     when busy =>
34         if done_t = '1' then
35             state_next <= state_next_aux;
36             if continuous = '1' then
37                 if stop_edge = '1' then
38                     step_next <= step_reg + 1;
39                 end if;
40             else
41                 step_next <= step_reg + 1;
42             end if;
43         end if;
44
45     when wait_tadl =>
46         if timer_running = '0' then
47             timer_counter <= "1011"; -- 11 clk cycles at 150 MHz
48             timer_start <= '1';
49         end if;
50
51         if timer_done = '1' then
52             state_next <= state_next_aux;
53         end if;
54
55     when rb_fixed_wait =>
56         if timer_running = '0' then
57             timer_counter <= "1111"; -- 15 clk cycles at 150 MHz
58             timer_start <= '1';
59         end if;
60
```



```

61         if timer_done = '1' then
62             state_next <= state_next_aux;
63         end if;
64
65     when rb_wait =>
66         if RB = '1' then
67             state_next <= state_next_aux;
68             step_next <= step_reg + 1;
69         end if;
70
71     when done =>
72         step_next <= 0;
73         state_next <= state_next_aux;
74     end case;
75 end process;

```

Para impedir que cada transição seja sobrecarregada, a lógica que dita o próximo estado permaneceu explícita, sendo definida pelo sinal *state\_next\_aux*. Este leva em consideração o estado atual, assim como o próximo modo a ser executado.

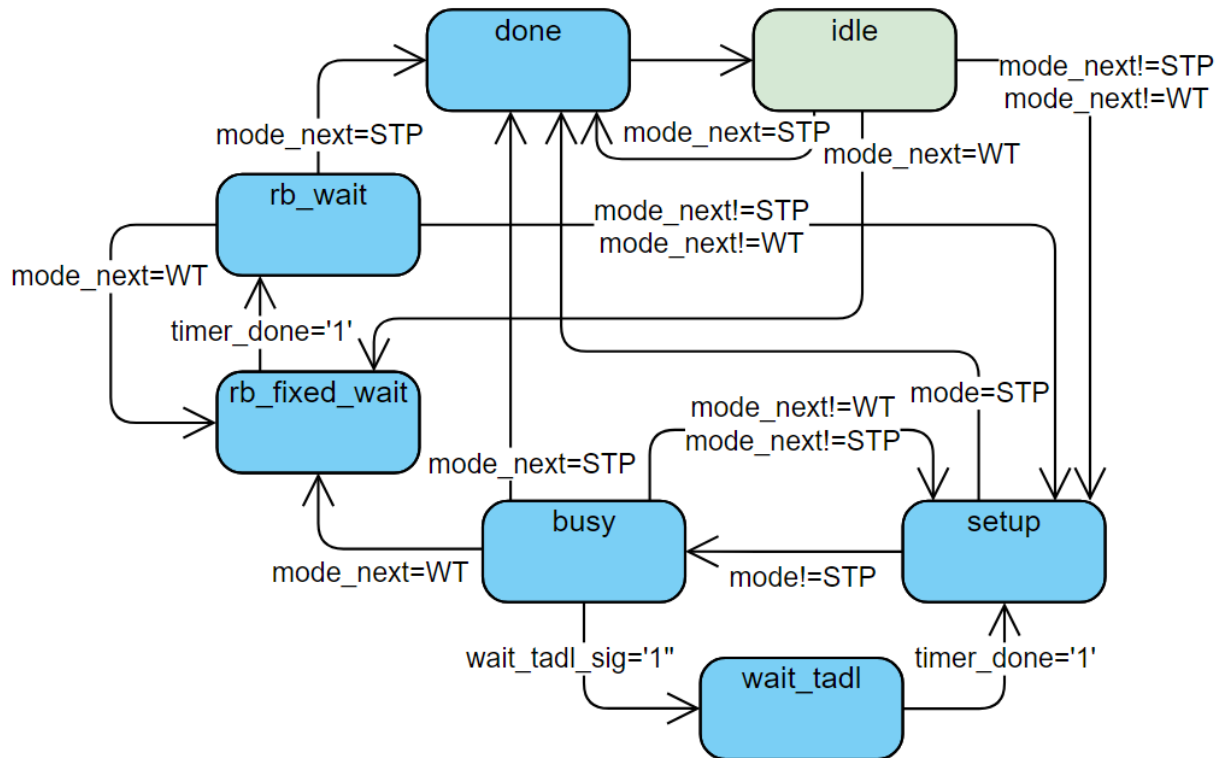
```

1 state_next_aux <=   idle when state_reg = done else
2                   rb_wait when state_reg = rb_fixed_wait else
3                   busy when state_reg = setup and mode /= STP else
4                   wait_tadl when wait_tadl_sig = '1' and state_reg =
5                       ↪ busy else
6                   setup when (state_reg = wait_tadl) or
7                       ((state_reg = idle or state_reg = busy or state_reg
8                       ↪ = rb_wait)
9                   and (mode_next /= STP and mode_next /= WT)) else
                   rb_fixed_wait when mode_next = WT else
                   done;

```

Os modos *CR* e *CW* determinam ações onde há a possibilidade de haver um número indefinido de escritas ou leituras. Como visto nas figuras 16 e 17, após os comandos e endereçamentos iniciais, é possível programar ou ler múltiplas palavras em sequência através do chaveamento de RE# e WE#. Considerado isso, definiu-se que essas ações podem ser controladas por sinais de *trigger* (inicia o próximo ciclo) e *stop* (parando a repetição e finalizando as ações restantes), que ditam o curso da operação nas suas bordas de subida.

Figura 24 – Máquina de estados do controlador de operações



## 5 RESULTADOS E DISCUSSÕES

### 5.1 Custo de hardware

Para a avaliação do custo do hardware, foi considerado o custo dos componentes principais na aplicação do *data logger*, sendo eles a memória flash, o FPGA e o oscilador, que fornece o sinal de clock.

Tabela 7 – Custo dos componentes

Componente	Custo (dólares)
Memória Flash S34ML02G1	4,57
FPGA Spartan-3A	8,33
Oscilador 150 MHz	0,89
<b>Total</b>	<b>13,79</b>

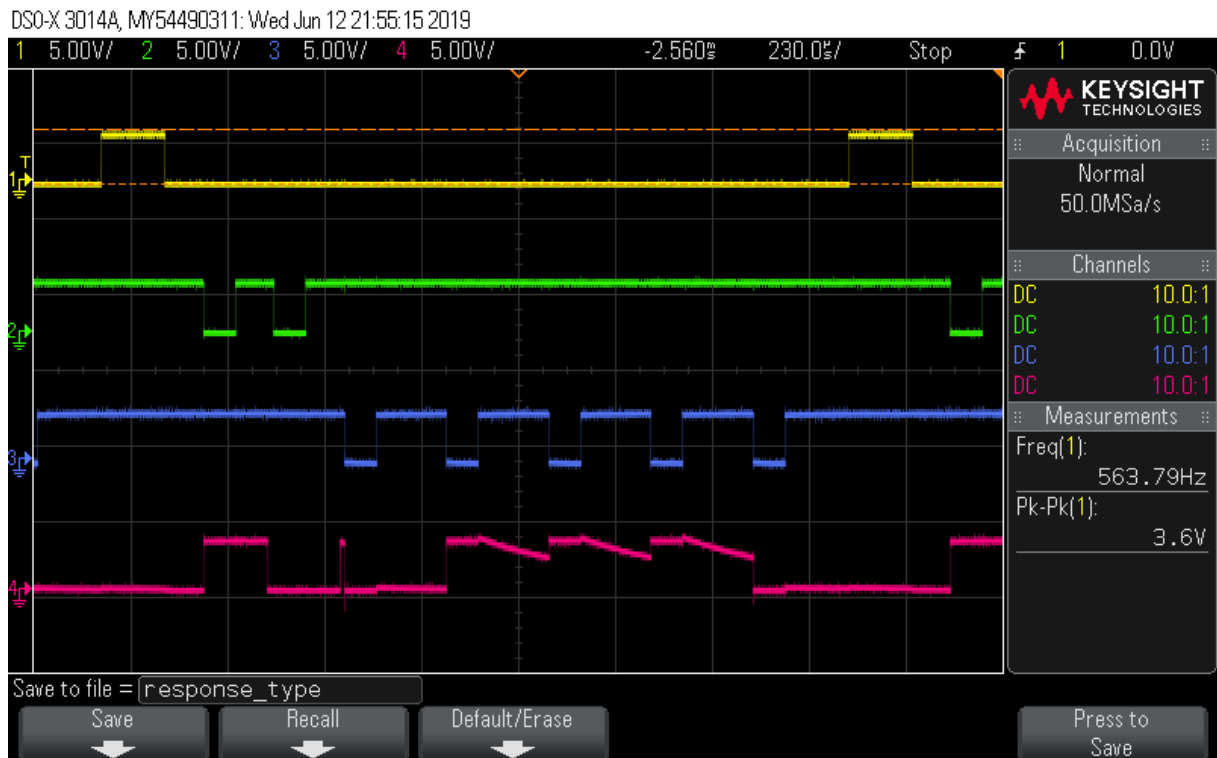
Fonte: O Autor.

Este preço obtido se encontra bem inferior quando analisado o *data logger* mais barato apresentado: o CR310 de Campbell Scientific (2020b), que tem custo de \$1650,00 e permite pulsos de entrada de até 35 kHz.

### 5.2 Resposta em osciloscópio

A fim de observar o funcionamento da interface com o chip, realizou-se a comunicação com o chip de memória utilizando o código feito para microcontrolador, conectando o controlador aos canais de comando da flash. Isto foi feito através um osciloscópio que monitorou os sinais principais da memória. Observa-se na Figura 25 a resposta obtida na operação de identificação, que retorna valores fixos. Para o teste realizado, foram necessários atrasos maiores do que o mínimo possível com o chip, a fim de fazer-se possível a visualização no osciloscópio, que não oferece taxa de amostragem alta o suficiente para a frequência normal de operação da memória.

Figura 25 – Retorno da operação de leitura de identificação no osciloscópio. Canais CE# (em amarelo), WE# (em verde), RE# (em azul) e IO7 (em vermelho)



Fonte: O Autor.

Foi lido a resposta de apenas um sinal I/O por conta da limitação do número de canais do osciloscópio. Os restantes I/O, entretanto, também foram analisados e demonstraram valores esperados.

Validou-se a abordagem utilizada, já que o sinal IO7 visto na Figura 25, em vermelho, teve o comportamento esperado. Utilizando a borda de subida de RE#, em azul como referência, conclui-se que os dados retornados foram: 0, 1, 1, 1, 0. Esta resposta condiz com o oitavo bit de cada ciclo (0x01, 0xDA, 0x90, 0x95 e 0x44) visto na Figura 18 para o chip de 2 Gb.

Os níveis de tensão visualizados fora do contexto próximo a borda de subida não são importantes, já que os canais se encontram em alta impedância quando não há saída de dados acontecendo. O aumento dos atrasos também não se provou um problema, pois os tempos recomendados no datasheet tratam-se de tempos mínimos; não havendo, em sua maior parte, limites superiores para estes períodos.

### 5.3 Velocidade de programação de memória com microcontrolador

Em trabalho colaborativo, Erbs (2020) expandiu as funcionalidades apresentadas anteriormente para simular a velocidade de programação da memória utilizando o mesmo microcontrolador. Nestes testes, fez-se uso de um timer de 64 bits e mediu o número de ciclos de clock de diferentes operações de programação.

Tabela 8 – Comparação entre os processos de escrita na memória Flash

<b>Processo de escrita</b>	<b>Período típico (ciclos/byte)</b>	<b>Período no pior caso (ciclos/byte)</b>
Escrita de apenas um byte	23.314,0	63.314,0
Escrita de uma página	171,0	189,9
Multiplane Cache Program para dois blocos	151,2	164,3
Multiplane Cache Program para toda a memória	164,1	174,6

Fonte: Erbs (2020).

Vale notar que os períodos calculados por Erbs (2020) consideram o tempo  $t_{PROG}$  para obter valores típicos ( $t_{PROG} = 200\mu s$ ) e no pior caso ( $t_{PROG} = 700\mu s$ ). Além disso, valor para escrita de apenas um byte não seria possível no caso de múltiplas escritas, já que o *datasheet* de Cypress Semiconductor Corporation (2018) especifica que a memória S34ML02G1 suporta no máximo 4 escritas parciais na mesma página.

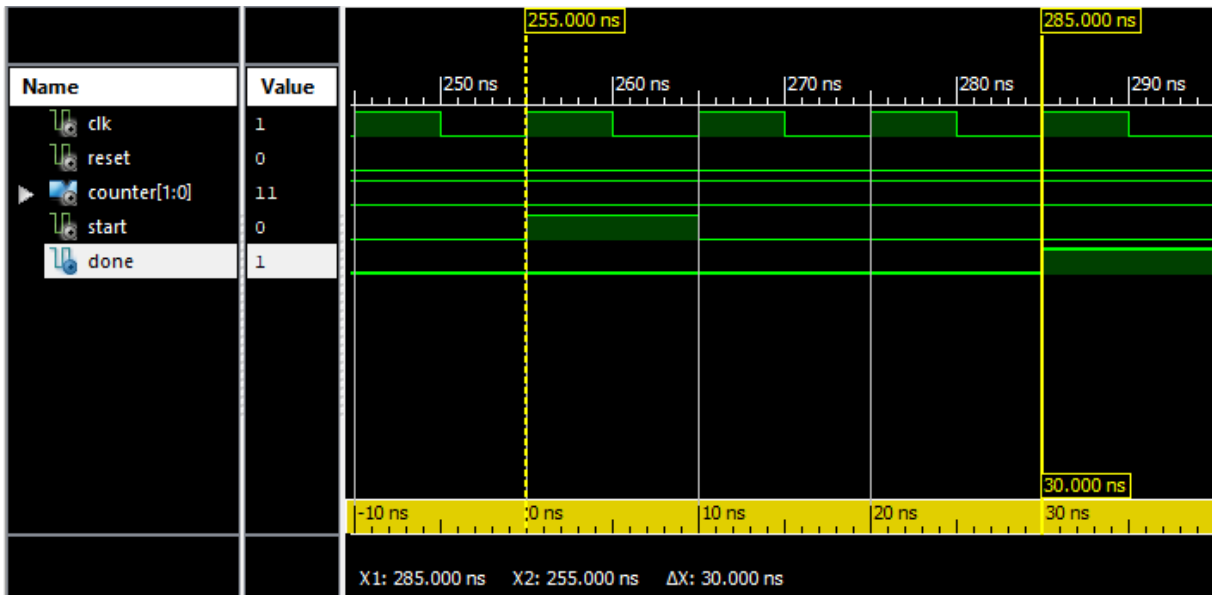
## 5.4 Simulações

Através do software de simulações iSim, validou-se os códigos feitos. As simulações foram realizadas através da imposição entradas arbitrárias, observando se a lógica interna e as saídas do circuito obtido condizem com o esperado.

### 5.4.1 Timer

Utilizou-se um clock com período de 10 ns para simular o módulo de timer, a Figura 26 mostra o resultado obtido. Determinado um atraso de 3 ciclos de clock, espera-se que a diferença de tempo entre o sinal *start* e o sinal *done* seja de 30 ns.

Figura 26 – Simulação módulo timer

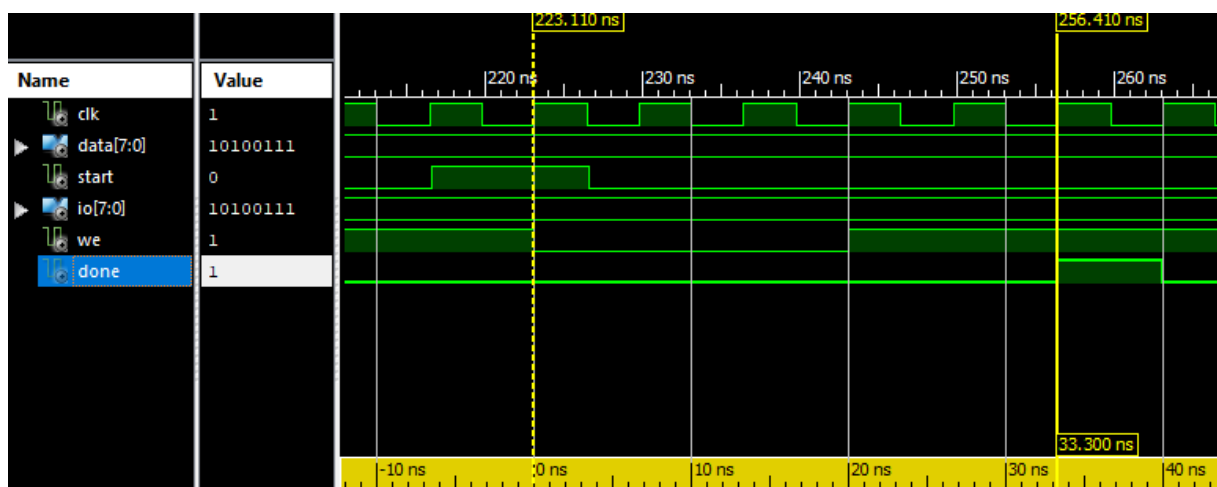


Fonte: O Autor.

#### 5.4.2 Módulo de escrita

Assumindo o período de clock de 6,66 ns, foi simulado o módulo de escrita. Esperava-se alcançar os tempos mínimos necessários de WE# em nível lógico baixo (12 ns) e do ciclo de leitura (25 ns). O ideal para isto ser alcançado seria determinar dois ciclos de clock para o nível lógico baixo, e dois ciclos para o nível lógico alto, atingindo 26,64 ns totais. Entretanto, as transições de estados acabam inserindo ciclos indesejados, alcançando tempos de 39,8 ns entre os sinais *start* e *done*, e 33,3 ns entre a borda de descida de WE# e *done*.

Figura 27 – Simulação módulo de escrita

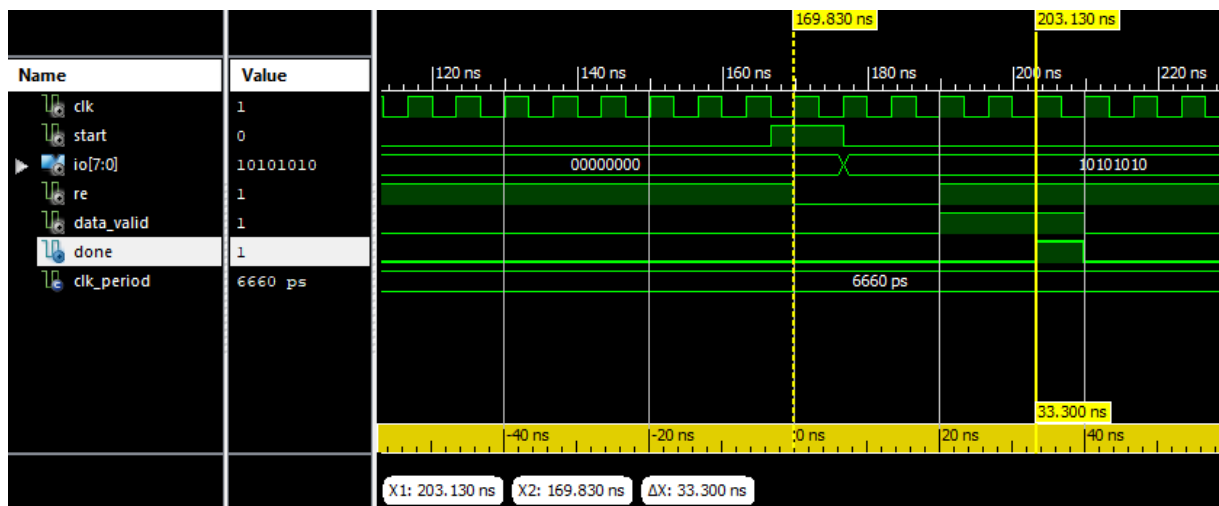


Fonte: O Autor.

### 5.4.3 Módulo de leitura

Simulou-se, também, o módulo de leitura com período de clock de 6,66 ns. Este apresenta os mesmos pontos apresentados no módulo de escrita, onde ciclos de clock são inseridos devido à transição de estados. O sinal *data\_valid*, que indica que os canais I/O possuem saídas válidas para serem lidas, segue a borda de subida de RE#, e acontece após 26,64 ns da borda de descida do mesmo.

Figura 28 – Simulação módulo de leitura

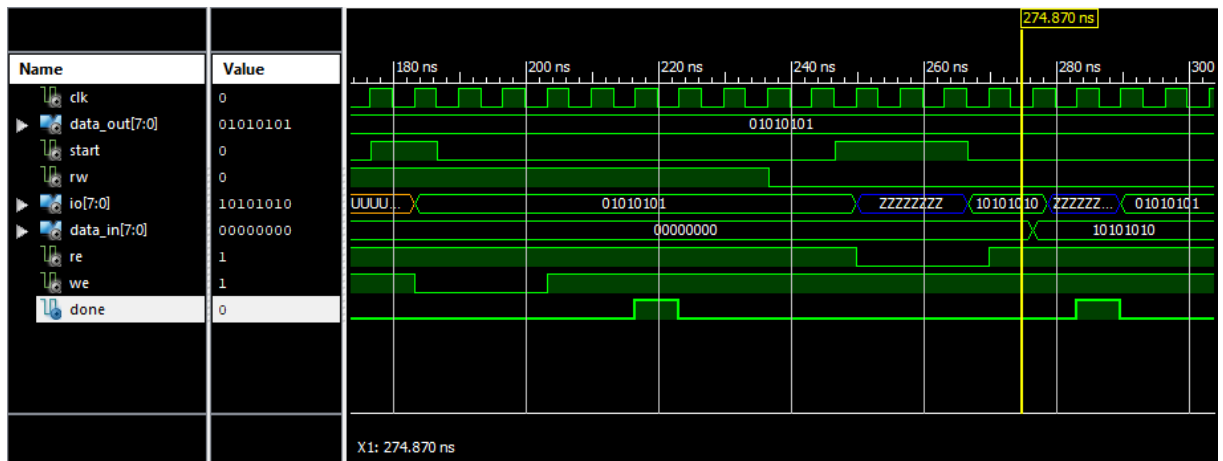


Fonte: O Autor.

### 5.4.4 Módulo de escrita e leitura com tri-state

Para o módulo de escrita e leitura, foi simulado em sequência a rotina de escrita seguida da rotina de leitura. Os estados dos I/O são arbitrários, e foram definidos de forma condizente ao esperado. Note que seus valores importam apenas enquanto o módulo estiver atuando, portanto qualquer valor residual não irá interferir no conjunto inteiro. O valor a ser escrito é determinado por *data\_out*, e o valor lido é exibido em *data\_in*.

Figura 29 – Simulação módulo de escrita e leitura com controle tri-state



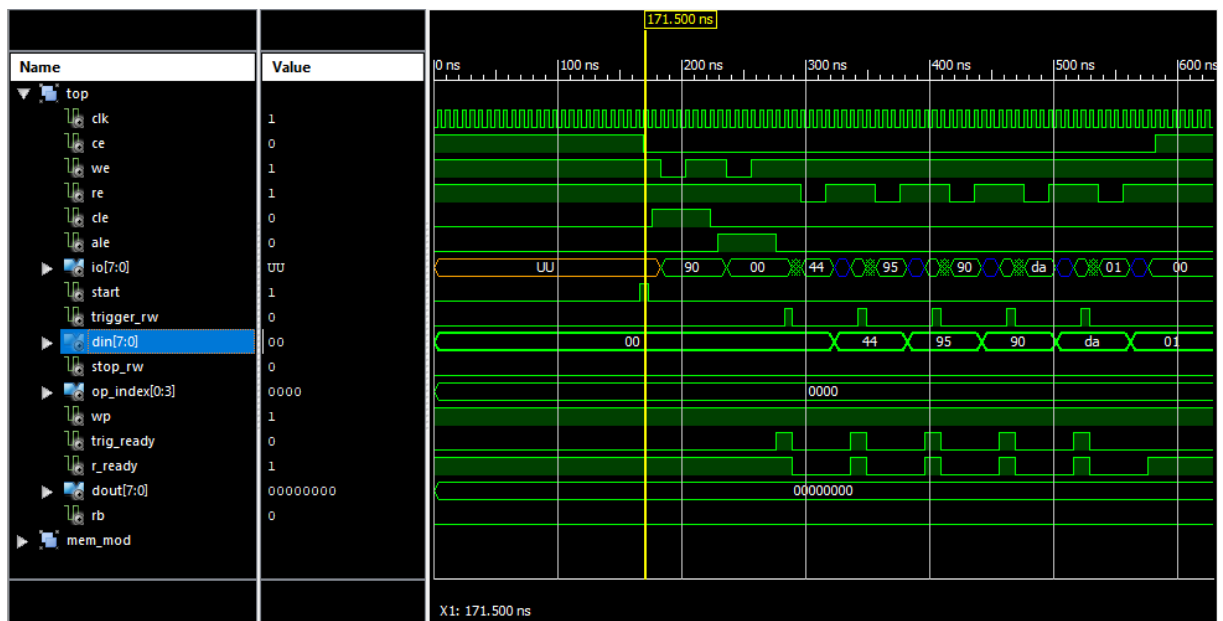
Fonte: O Autor.

#### 5.4.5 Módulo de controle de operações

A simulação deste módulo prevê que as formas de onda sejam similares às operações apresentadas nos diagramas de Cypress Semiconductor Corporation (2018). A fim validar o método utilizado, simulou-se as operações de identificação (Figura 18), programação de página (Figura 17) e leitura de página (Figura 16).

Na leitura de identificação foi representado o mesmo retorno de dados do chip S34ML02G1.

Figura 30 – Simulação operação de identificação



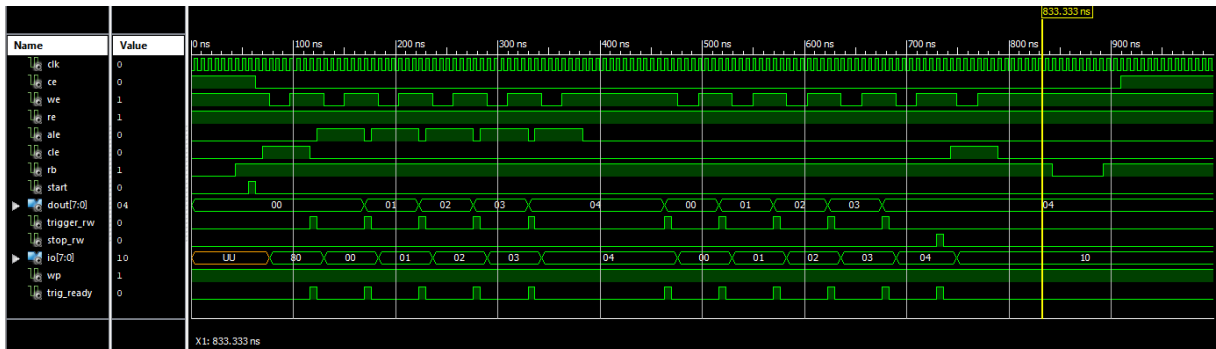
Fonte: O Autor.

O período em que R/B# permanece em nível lógico baixo foi decrescido, comparado ao seu valor típico de 200  $\mu$ s, com propósito de observar as transições dos



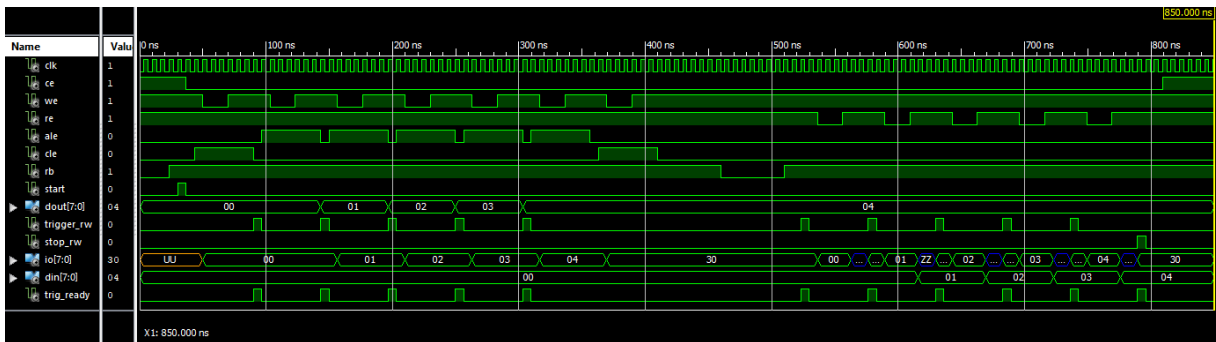
demais sinais de controle.

Figura 31 – Simulação operação de programação de página



Fonte: O Autor.

Figura 32 – Simulação operação de leitura de página



Fonte: O Autor.

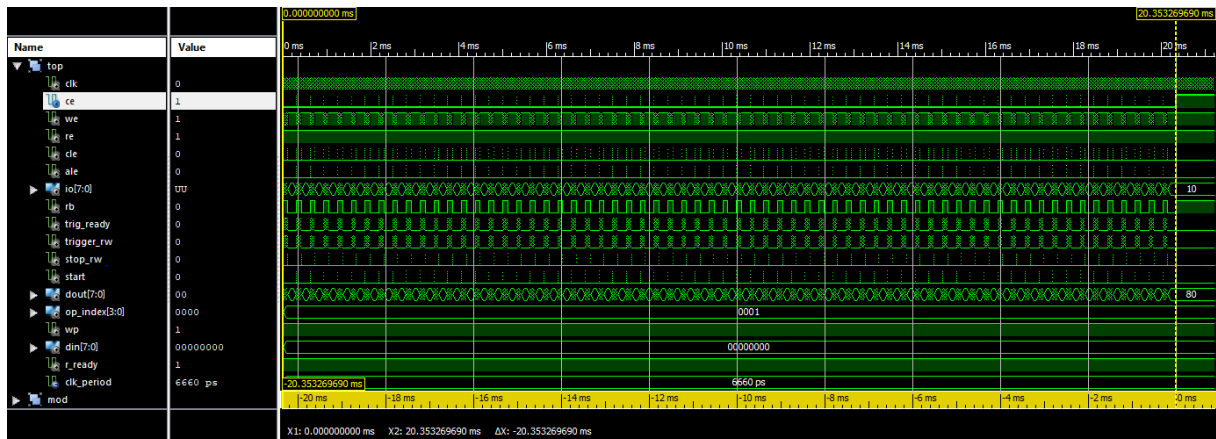
## 5.5 Tempo de escrita em toda a memória

Com intenção de obter o tempo de programação total da memória com os módulos criados simulou-se a programação de apenas um bloco, extrapolando o tempo obtido para a capacidade total de armazenamento. Executar este procedimento em sua totalidade tomaria tempo demasiado, e teria resultados similares à aproximação obtida.

Considerando  $t_{PROG}$  típico de  $200 \mu s$ , a Figura 33 mostra que o tempo levado na programação de um bloco inteiro durou, aproximadamente, 20,353 ms. Tendo 1024 blocos por plano (Tabela 4) e assumindo dois planos para o S34ML02G1, o tempo total para programar a memória seria de aproximadamente 41,683 segundos, ou 6,33 MB/s.

Nesta velocidade de 6,33 MB/s e considerando sinais da mesma frequência que o máximo de 250 kHz suportado pelo *data logger* CR1000X de Campbell Scientific (2020a), o tamanho de cada dado poderia ser expandido para 212 bits.

Figura 33 – Simulação da programação de um bloco inteiro



Fonte: O Autor.

Considerando o teste realizado por Korol (2015) com engrenagem de 112 dentes e frequência de rotação de 6000 rpm, obtém-se a frequência do trem de pulsos de 11.2 kHz (equação 2.1) podemos saber se o módulo desenvolvido seria suficiente para realizar as medições desejadas.

Assumindo o tamanho de cada dado sendo 16 bits, como discutido na Seção 3.4, e a velocidade de 6,33 MB/s, constata-se que a frequência de escrita de dados seria equivalente a 3,32 MHz, extremamente acima da taxa necessária.

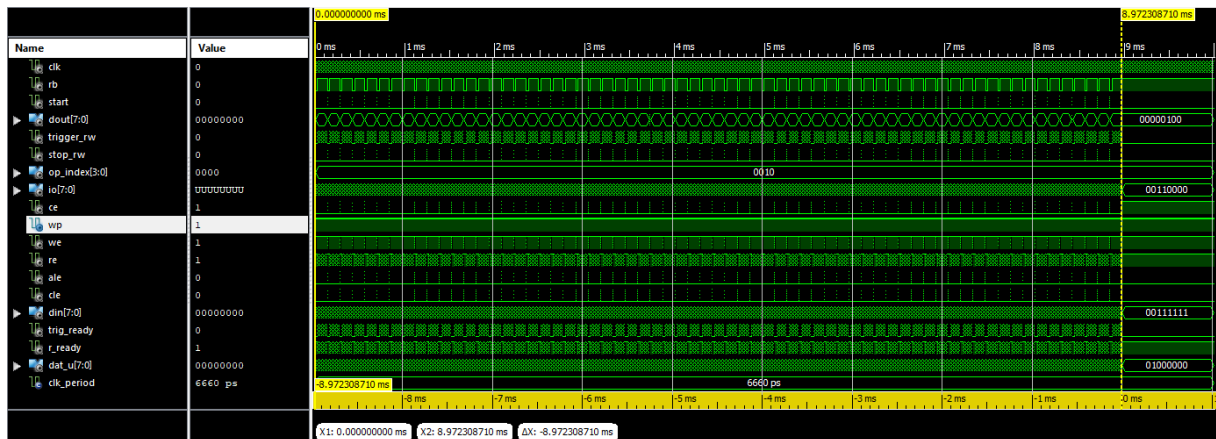
O fator que mais contribui para diminuir a velocidade total possível é  $t_{PROG}$ , que tem valor típico de 200  $\mu s$  e máximo de 700  $\mu s$ . Para o valor típico, este tempo consiste 63,90% de um ciclo de escrita de página. Enquanto para o pior caso esta proporção se tornaria 69,10%.

Esta frequência máxima de 3,32 MHz seria possível de atingir, entretanto, para a medição do período de pulsos utilizando o mesmo *clock* de 150 MHz e tamanho de dados de 16 bits, haveria um máximo de 45 ciclos de clock por pulso.

## 5.6 Tempo de leitura de toda a memória

Tendo como base a simulação para a programação de um bloco inteiro, elaborou-se a mesma abordagem para a simulação da leitura de página. Suas formas de ondas estão na Figura 34, e foi atingido o tempo de 8,97 ms. Consequentemente, a leitura da memória inteira resultaria em 18,37 segundos, ou 14,37 MB/s.

Figura 34 – Simulação da leitura de um bloco inteiro



Fonte: O Autor.

O grande diferencial entre os tempos de leitura e programação é o período em que o chip de memória fica ocupado após a entrada de comandos, já que seus ciclos são idênticos. No caso da leitura de página,  $t_R$  é no máximo  $25 \mu s$ , enquanto  $t_{PROG}$  tem valor típico de  $200 \mu s$  e máximo de  $700 \mu s$ .

Para a leitura de páginas,  $t_R$  representa no máximo 18,12% do tempo total de cada página.

## 5.7 Tempo de vida da memória

Embora a velocidade de armazenamento de dados tenha sido maior que o necessário, outro fator importante de memórias *flash* é a quantidade de ciclos de programação suportados. Utilizando a equação 2.2, é viável inferir o total de memória total capaz de ser escrita em uma *flash*.

Supondo valores de temperatura de conservação de  $30^\circ C$ , escrita totalmente sequencial e tempo de armazenamento de 10 anos (tempo máximo de retenção de acordo com Cypress Semiconductor Corporation (2018)) pode obter-se os valores a serem empregados na equação através das Tabelas 1, 2 e 3. Sabendo que a vida de programar/limpar do S34ML02G1 é de 100.000 ciclos, a equação 2.2 resulta em 38.461 de escritas completas na memória, aproximadamente.

Com a capacidade de 2 Gbit + 64 Mbit, o total programado até o fim da vida do chip depois 38.461 escritas inteiras equivaleria a 9,68 TB. Tendo a velocidade de 6,33 MB/s obtida na Seção 5.5, levaria 18,57 dias de testes e amostragens contínuas até atingir este limite de ciclos, embora este tempo possa aumentar consideravelmente, reduzindo o tempo de retenção de dados considerado.

## 6 CONCLUSÕES

O presente trabalho buscou criar um sistema de armazenamento de dados em alta velocidade, a fim de atingir requisitos descritos por temas anteriores que não conseguiram atingir taxas de amostragem altas o suficiente. O método de armazenamento conseguiu rivalizar as taxas mostradas por *data loggers* comerciais, mesmo que o presente processo não consta com os múltiplos canais, atingindo o objetivo geral do trabalho.

O uso do FPGA proporcionou alta eficiência na interface com a memória, provando ser excelente quando se necessita realizar comunicação em nível de hardware. Sua capacidade de realizar e de controlar diversos sinais paralelos a alta frequência se mostra melhor que um microcontrolador, e foi um dos motivos principais que tornou possível a realização do processo.

A escolha pela memória *flash* resultou em um ponto positivo, já que esta proporcionou tempos de escrita e leitura, e capacidade de armazenamento excelentes. Sua velocidade de escrita de 6,33 MB/s obtida após feita a simulação do HDL desenvolvido se mostrou mais que suficiente para realizar os testes de Korol (2015), que via sinais de até 17,5 kHz. Esta velocidade seria capaz de até mesmo superar as leituras de pulsos de alta frequência do CR1000X de Campbell Scientific (2020a), ou podendo escolher por armazenar dados de 212 bits quando trabalhando na mesma frequência de 250 kHz.

Dos equipamentos apresentados nos testes práticos de Korol (2015), RAS e PAK MKII, apenas o RAS não teria sua frequência alcançada, tendo em vista que este emprega um clock de 10 GHz e custa R\$220.000,00. Já o PAK MKII de Müller-BBM (2015) e valor R\$92.000,00, com entrada suportada em 1 MPulso/s e resolução de 14 ns, teria sua eficiência confrontada com a velocidade de escrita de 6,33 MB/s, e resolução de 6,66 ns a 150 MHz.

Embora o objetivo tenha sido alcançado, para alcançar velocidades de transferência ainda maiores, outras operações de escrita poderiam ser escolhidas. Visto como  $t_{PROG}$  foi o maior tempo consumido durante a programação, métodos de programação de página como o Cache Program ou Multiplane Page Program contornariam este fato, possibilitando razões menores entre  $t_{PROG}$  e cada ciclo de programação de página.

## REFERÊNCIAS

- ADAMSON, S. **Improved approaches to the measurement and analysis of torsional vibration**. Munique, Alemanha, 2004.
- BISHOP, R. H. **Mechatronic system control, logic, and data acquisition**. Boca Raton, Flórida: CRC press, 2007.
- CAMPBELL SCIENTIFIC. **CR1000X**: Measurement and Control Data Logger. Logan, Utah, 2020. Disponível em: <<https://s.campbellsci.com/documents/us/manuals/cr1000x-product-manual.pdf>>.
- CAMPBELL SCIENTIFIC. **CR300 Series**: Compact Datalogger. Logan, Utah, 2020. Disponível em: <<https://s.campbellsci.com/documents/us/manuals/cr300.pdf>>.
- CHU, P. P. **FPGA prototyping by VHDL examples**: Xilinx Spartan-3 version. Hoboken, Nova Jersey: John Wiley & Sons, 2011.
- CYPRESS SEMICONDUCTOR CORPORATION. **1 Gb, 2 Gb, 4 Gb, 3 V, 4-bit ECC, SLC NAND Flash Memory for Embedded**. San Jose, Califórnia, 2018. Rev. \*Q. Disponível em: <<https://docs.rs-online.com/2ff7/0900766b816ad030.pdf>>.
- DUBEY, R. **Introduction to embedded system design using field programmable gate arrays**. Berlim, Alemanha: Springer Science & Business Media, 2008.
- ERBS, V. K. **Monitoramento sem fio da vibração torsional de powertrain automotivo (parte II)**. 23 p. — Relatório de Iniciação Científica – Centro Tecnológico de Joinville, Universidade Federal de Santa Catarina, Joinville, SC, 2020.
- EXTECH INSTRUMENTS. **3-Axis G-Force Datalogger**. Massachusetts, Estados Unidos, 2014. Disponível em: <[http://www.extech.com/products/resources/VB300\\_DS-en.pdf](http://www.extech.com/products/resources/VB300_DS-en.pdf)>.
- JANSSENS, K.; BRITTE, L. Comparison of torsional vibration measurement techniques. In: **Proceedings of THE THIRD INTERNATIONAL CONFERENCE ON CONDITION MONITORING OF MACHINERY IN NON-STATIONARY OPERATIONS CMMNO 2013**. Berlin, Heidelberg: Springer, 2014. p. 453–463.
- KOROL, D. **Monitoramento sem fio da vibração torsional de powertrain automotivo**. 61 p. — Trabalho de conclusão de curso (Graduação em Engenharia Mecatrônica) – Centro Tecnológico de Joinville, Universidade Federal de Santa Catarina, Joinville, SC, 2015.
- MEASUREMENT COMPUTING. **Data Acquisition Handbook**. 3. ed. Massachusetts, Estados Unidos, 2012. Disponível em: <<https://www.mccdaq.com/pdfs/anpdf/Data-Acquisition-Handbook.pdf>>.

MUKARO, R.; CARELSE, X. F. A microcontroller-based data acquisition system for solar radiation and environmental monitoring. **IEEE transactions on instrumentation and measurement**, IEEE, v. 48, n. 6, p. 1232–1238, 1999.

MÜLLER-BBM. **Acquire dynamic signals**. Planegg, Alemanha, 2015. Disponível em: <[https://www.pakbybbm.com/wp-content/uploads/2015/06/PAK\\_MKII\\_Data\\_Acquisition\\_Hardware\\_Brochure.pdf](https://www.pakbybbm.com/wp-content/uploads/2015/06/PAK_MKII_Data_Acquisition_Hardware_Brochure.pdf)>.

NATIONAL INSTRUMENTS. **Making Accurate Frequency Measurements**. Austin, Texas, 2020. Disponível em: <<https://www.ni.com/pt-br/support/documentation/supplemental/06/making-accurate-frequency-measurements.html>>.

NATIONAL INSTRUMENTS. **Understanding Life Expectancy of Flash Storage**. Austin, Texas, 2020. Disponível em: <<https://www.ni.com/pt-br/support/documentation/supplemental/12/understanding-life-expectancy-of-flash-storage.html>>.

PEDROSO, J. **Sensor de Efeito Hall**. 2017. Disponível em: <<https://analisordevibracoes.wordpress.com/author/jefersonpedroso/>>.

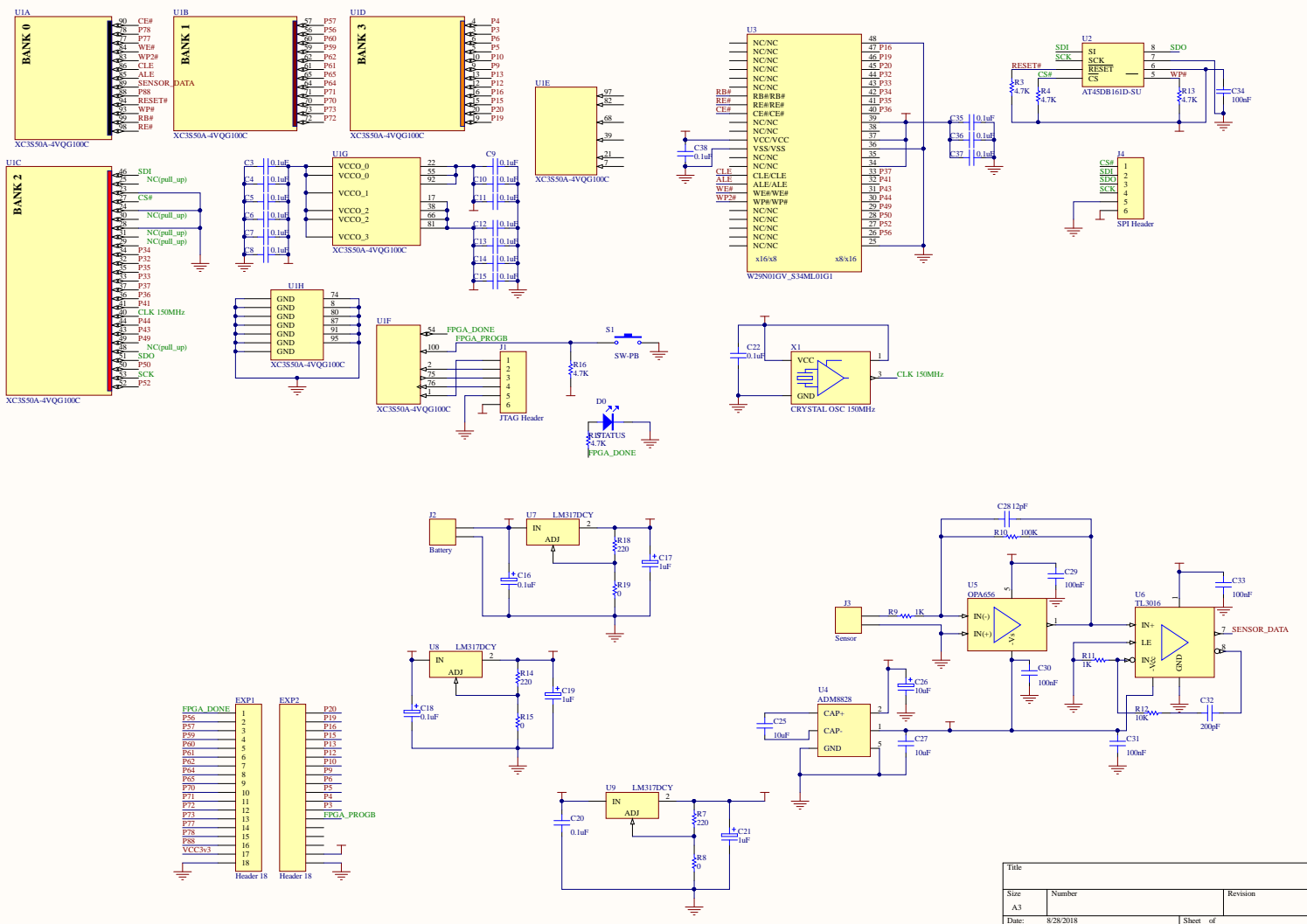
RAJMOND, J.; PITICĂ, D. Data logger for signal analysis and failure prediction. In: IEEE. **2010 IEEE 16th International Symposium for Design and Technology in Electronic Packaging (SIITME)**. Cluj-Napoca, Romania, 2010. p. 263–266.

ROTEC. **ROTEC RAS delta** – The fourth generation of ROTEC measurement systems. Munique, Alemanha, 2016. Disponível em: <[https://www.vispiron.de/fileadmin/data/Content/03\\_Business\\_Units/02\\_Messtechnik/01\\_Messgeraete/Factsheet\\_ROT\\_EC\\_RAS\\_delta\\_2016\\_04\\_EN.pdf](https://www.vispiron.de/fileadmin/data/Content/03_Business_Units/02_Messtechnik/01_Messgeraete/Factsheet_ROT_EC_RAS_delta_2016_04_EN.pdf)>.

SASS, R.; SCHMIDT, A. G. **Embedded systems design with platform FPGAs: principles and practices**. Burlington, Massachusetts: Morgan Kaufmann, 2010.

WILMSHURST, T. **Designing Embedded Systems with PIC Microcontrollers: Principles and Applications**. Oxford, Reino Unido: Elsevier, 2007.

# APÊNDICE A



File	Number	Revision
A3		
Date:	8/28/2018	Sheet of
File:	C:\Users\Vinici\Desktop\schematic_SchDk	Drawn By: