Problema #1-Microarquitetura

Álan Bruno[1], **Laercio Rios** [2], **Pedro Mota**[1], **Ramon Silva**[3]

[1]DTEC – Universidade Estadual de Feira de Santana (UEFS)

alan.bruno@gmail.com[1]

laercio.rios13@hotmail.com[2]

pedromotafsa@gmail.com[3]

ramondecerqueirasilva@gmail.com[4]

Abstract. *This paper reports the whole process of developing a system requested by the Soc-IP company that will serve as pilot for the company's start in the field of embedded platforms using FPGA devices. The project involves the use of technologies such as Quartus software and the NIOS II processor for system modeling using the hardware description language, verilog, to develop the solution. At the end of the report you will be able to understand all the processes involved in the final product, the results obtained and the conclusions of the team.*

Resumo. *Este trabalho relata todo o processo de desenvolvimento de um sistema solicitado pela empresa Soc-IP que servirá de piloto para o início da empresa no ramo de plataformas embarcadas utilizando dispositivos FPGA. O projeto envolve o uso de tecnologias como o software Quartus e o NIOS II para modelagem do sistema utilizando a linguagem de descrição de hardware, verilog, para desenvolver a solução. Ao final do relatório será possível entender todos os processos envolvidos no produto final, os resultados obtidos e as conclusões da equipe.*

1. Introdução

Nos últimos anos é possível observar a grande velocidade com que a tecnologia está evoluindo. É intuitivo que para sobreviver e sustentar-se no mercado é necessário que as empresas adaptem-se nestas novas tecnologias.

A empresa Soc-IP, atuante no desenvolvimento de *cores* licenciáveis, deseja entrar na área de plataformas embarcadas a partir de dispositivos FPGA. Afim de iniciar esta trajetória, uma equipe foi designada para desenvolver uma *Interface Homem-Máquina* (IHM) que utiliza botões como meio entrada e para saída utiliza LED's e um display LCD. Para desenvolver o sistema, o processador NIOS II será utilizado.

A descrição do sistema foi toda feita em Verilog. Como o NIOS II é baseado em um modelo RISC a equipe utilizou uma técnica conhecida como Instrução Personalizada (Custom Instruction) para criação de uma instrução que funcionasse como um auxiliar na utilização do display LCD. Para execução/compilação e testes foi utilizado o Quartus.

Este trabalho visa, então, contextualizar todo o processo de desenvolvimento feito pela equipe, documentando os passos utilizados e as dificuldades encontradas durante o

processo. Ao fim, este relatório permitirá que o leitor entenda o funcionamento do sistema criado e torná-lo capaz de operar o sistema desenvolvido.

2. Fundamentação Teórica

2.1. FPGA

Atualmente as FPGAs são bastante usadas em diversos setores da indústria, estando presente em setores onde desempenho, paralelismo e tempo real são cruciais. Dito isso, o que se pode-se dizer é que as FPGAs são um dispositivo lógico programável que suporta a implementação de circuitos digitais[Prado 2019].

Isso posto, já falando da sua estrutura, pode-se destacar que ela possuí três componentes básicos, conforme mostrado na imagem abaixo(Figura 1)[Prado 2019],onde os retângulos representam blocos de entrada e saída, os quadrados brancos representam blocos lógicos e os azuis switches. Vale destacar, que hoje já são comuns blocos de memória, DSP e até mesmo um processador ARM na arquitetura de FPGAs[Prado 2019].

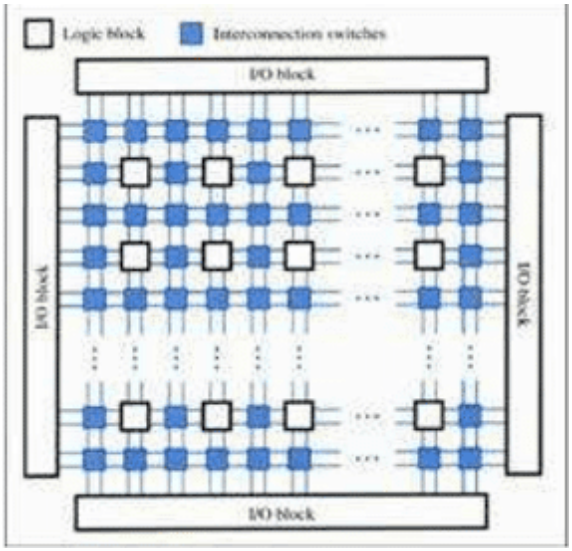


Figura 1. Componentes básicos.

Dito isso, com relação aos blocos lógicos, pode-se dizer que estes estão arranjados em uma matriz bidimensional, e os fios de interconexão são organizados como canais de roteamento horizontais e verticais entre as linhas e colunas do bloco lógico. Ademais, vale dizer que cada bloco lógico em uma FPGA tipicamente tem um pequeno número de entradas e saídas, um dos blocos lógicos normalmente se chama LookUp Table(LUT) que contém células de armazenamento que são usadas para implementar uma pequena função lógica[Prado 2019].

2.2. CISC vs RISC

O termo CISC vem do termo em inglês Complex Instruction Instruction Set Computer, que em português seria, Computador com Conjunto Complexo de Instrução. Isto posto, pode-se destacar que o CISC é uma tecnologia mais antiga e usada para família de computadores compatíveis em nível de software[Souza 2019].

Em suma, o que se pode afirmar sobre o CISC é que têm um conjunto de instruções grande, de tamanhos variáveis, com formatos complexos. Muitas dessas instruções são bastante complicadas, executando múltiplas operações quando uma única instrução é dada. Ademais, pode-se dizer que o problema do com máquinas CISC é que um conjunto pequeno de instruções complexas torna o sistema consideravelmente mais lento[TI 2019].

Já falando de RISC, que vem do termo inglês Reduced Instruction Set Computer, que em português, seria Computador com Conjunto Reduzido de Instruções. O principal objetivo do RISC objetivo é simplificar as instruções de modo que elas possam ser executadas mais rapidamente, onde cada instrução executa apenas uma operação, que são todas do mesmo tamanho, tem poucos formatos, e todas as operações aritméticas devem ser executadas entre registradores[TI 2019].

2.3. Linguagem Assembly

Quando se trada de Assembly, se refere-se a uma linguagem de montagem. Ou seja, diferente da maioria das outras linguagens, que são compiladas e/ou interpretadas, programar em Assembly é escrever um código que é diretamente entendido pelo hardware[Unicamp 2019].

Dito isso, vale dizer, que a comunicação direta com um hardware eletrônico, se vale através do envio de sinais elétricos. Sendo que, os sinais mais fáceis para os computadores entenderem são ligados e desligados, e assim o alfabeto do computador tem apenas duas letras, sendo que os dois símbolos para essas duas letras são os números 0 e 1(vale o enfoco que pode-se referir a estas letras como bit binário).

Nesse sentido, tem que se acrescentar que os computadores são escravos de comandos, que são chamados de instruções. As instruções, que são apenas coleções de bits que o computador entende e obedece, podem ser consideradas números.

Dito isso, vale dizer que os primeiros programadores se comunicavam com computadores em números binários, mas isso era tão entediante que eles rapidamente inventaram novas notações que estavam mais próximas do modo como os humanos pensam.

No início, essas notações foram traduzidas para binário à mão, mas esse processo ainda era cansativo. Usando o computador para ajudar a programar o computador, os pioneiros inventaram o software para traduzir da notação simbólica para o binário. O primeiro desses programas foi chamado de assembler. Este programa traduz uma versão simbólica de uma instrução para a versão binária, vale dizer que essa versão simbólica aqui falada se trata da linguagem assembly[David A. Patterson 2017].

Isto posto, vale dizer que embora tivesses melhorias com o uso da linguagem assembly, isto esta longe das notações que um cientista gostaria de usar para simular o fluxo de fluidos ou que um contador poderia usar para equilibrar os livros. A linguagem de montagem requer que o programador escreva uma linha para cada instrução que o computador seguirá, forçando o programador a pensar como o computador[David A. Patterson 2017].

Para melhorar essa situação, vei a ideia que um programa poderia ser escrito para traduzir uma linguagem mais poderosa em instruções de computador, onde essas de linguagens de programação de alto nível e compiladores traduziriam programas em tais idiomas em instruções.

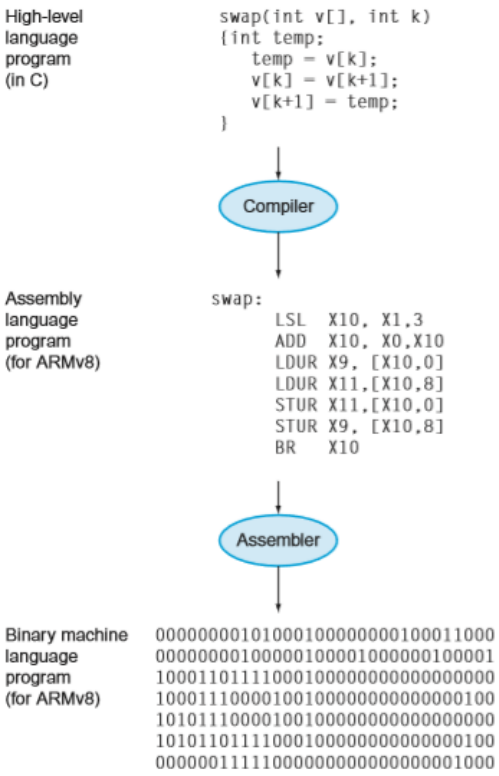


Figura 2. Programa C compilado em linguagem assembly e, em seguida, montado em linguagem de máquina binária.

Dito isso, vale dizer que essas linguagens de alto nível coma a linguagem C(Figura 2)[David A. Patterson 2017], por exemplo, oferece diversas vantagens, entre elas estão ser mais natural, usando palavras em inglês e notação algébrica, resultando em programas que se parecem muito mais com texto do que com tabelas de símbolos crípticos. Além disso, eles permitem que as linguagens sejam projetadas de acordo com o uso pretendido. Assim, Fortran foi projetado para computação científica, Cobol para processamento de dados corporativos, Lisp para manipulação de símbolos e assim por diante[David A. Patterson 2017].

Uma outra vantagem que deve ser destacada é a melhoria da produtividade do programador. Uma das poucas áreas em comum acordo no desenvolvimento de software é que leva menos tempo para desenvolver programas quando eles são escritos em linguagens que exigem menos linhas para expressar uma ideia[David A. Patterson 2017].

2.4. Linguagem Verilog

A linguagem Verilog é uma linguagem de descrição de hardware usada para modelar sistemas eletrônicos ao nível de circuito. Essa ferramenta suporta a projeção, verificação e implementação de projetos analógicos, digitais e híbridos em vários níveis de abstração. Um dos principais atributos da modelagem de circuitos por linguagem descritiva frente à modelagem por captura de esquemático, é que desta maneira o projeto se torna independente da plataforma de desenvolvimento (IDE) na qual se está trabalhando[Wikipedia 2019]

2.5. Processador

Um processador também chamado de CPU (Central Processing Unit), é o componente de hardware responsável por processar dados e transformar em informação. Ele também transmite estas informações para a placa-mãe, que por sua vez as transmite para onde é necessário. A placa-mãe serve de ponte entre o processador e os outros componentes de hardware da máquina. Outras funções do processador são fazer cálculos e tomar decisões lógicas[Pacievitch 2019].

Em geral os processadores apresentam as seguintes características:

- Frequência de Processador (Velocidade, clock): medida em hertz, que define a capacidade do processador em processar informações ao mesmo tempo.
- Cores: o core é o núcleo do processador. Existem processadores core e multicore, ou seja, processadores com um núcleo e com vários núcleos na mesma peça.
- Cache: a memória Cache é um tipo de memória auxiliar, que faz diminuir o tempo de transmissão de informações entre o processador e outros componentes.

2.6. Display LCD

O display LCD é principal dispositivo de saída deste problema, nele será expresso todas as mensagens de saída. Vale dizer, que o mesmo é do modelo NHD-C0216CU-FSW-GBW-3V3, o qual foi embutido diretamente na placa do kit de desenvolvimento Mercúrio IV.

Dito isso, existem dois pontos que se deve destacar sobre este modelo, o primeiro é que ele precisa de um mapa de caracteres(esse não é em ASCII e sim definido pelo próprio do dispositivo) para definir que letra ou símbolo escrever(Figura 3). Além disso, é preciso ter noção de todos os comandos(Figura 3) necessários para manipulá-lo.

Figura 3. Mapa de caracteres.

Instruction	Instruction Code									Description	Instruction Execution Time		
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		OSC= 380KHz	OSC= 540kHz	OSC= 700KHz
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRAM. and set DDRAM address to "00H" from AC	1.08 ms	0.76 ms	0.59 ms
Return Home	0	0	0	0	0	0	0	0	1	x	Set DDRAM address to "00H" from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed.	1.08 ms	0.76 ms	0.59 ms
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	26.3 us	18.5 us	14.3 us
Display ON/OFF	0	0	0	0	0	0	1	D	C	B	D=1:entire display on C=1:cursor on B=1:cursor position on	26.3 us	18.5 us	14.3 us
Function Set	0	0	0	0	1	DL	N	DH	*0	IS	DL: interface data is 8/4 bits N: number of line is 2/1 DH: double height font IS: instruction table select	26.3 us	18.5 us	14.3 us
Set DDRAM address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address counter	26.3 us	18.5 us	14.3 us
Read Busy flag and address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read.	0	0	0
Write data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM/ICONRAM)	26.3 us	18.5 us	14.3 us
Read data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM/ICONRAM)	26.3 us	18.5 us	14.3 us

Figura 4. Algumas instruções.

No tocante a parte dos comandos(ou até mesmo para escrever um caractere), vale se acrescentar que a depender da faixa de clock que ele estiver operando vai ter um tempo de execução para realizar o comando. Existem tempos da ordem dos microssegundos como também dos mile segundos. Devido esse tempo de operação, ao qual, vale destacar que o maior é 1.08 ms, se fez necessário estabelecer um delay de 2 ms para que der tempo para operação ser encerrada.

Um último destaque sobre o display, e que as variáveis DB0-DB7 são informações referentes as instruções ou caracteres e a variável RS diferencia a instrução do caractere(sendo 1 para caractere e 0 para instrução).

3. Metodologia

O sistema desenvolvido teve suas bases de construção nas sessões PBL, onde foi discutido como lidar com cada problema em particular. As mesmas foram peça chave para os grupos discutirem suas ideias, dúvidas e propostas de implementação.

Dito isso, na primeira sessão houve a leitura do problema onde foi destacado diversos pontos. Um desses pontos, foi programar em Assembly o processador NIOS II, e que este é um softcore RISC de 32 bits com arquitetura Harvard. Um outro ponto importante abordado nesta sessão foi que se deveria desenvolver uma interface homem-máquina(IHM) que utiliza-se botoes como entrada, LEDs e um Display LCD como saída. Além desses pontos, um último que merece destaque tratado nesta sessão, foi que a descrição do processador deveria ser em Verilog e demais elementos utilizados para teste e validação do funcionamento do core.

A parti dos pontos que foram levantados, na primeira sessão começou-se a buscar soluções para desenvolver este problema. Dentre as maneiras que se buscou a solução, a primeira foi o trabalho de pesquisa que levou o conhecimento sobre o dispositivo do display LCD, o qual, possuí todo um mapeamento de caracteres e conjuntos de instruções para se possa manipulá-lo. Além disso, o trabalho de pesquisa também levou a adquirir todo conhecimento necessário sobre as instruções personalizadas como também do código assembly que seria empregado na solução.

Após esses levantamentos de informações através de pesquisa, foi então iniciado o processo de implementação da solução, no incio antes de fazê-la propriamente, foi feito

todo um conjunto de testes usando o JNIOSEmu, para realizar testes do código assembly. Além dessa ferramenta, foi usado o ModelSim, ao qual, foi possível realizar um teste funcional do código em Verilog da instrução personalizada. Depois ter feito todos esses testes com essas duas ferramentas, inciou-se a etapa de testes práticos, que em fim levaram a solução do problema.

4. Resultados e Discussões

4.1. Testes no ModelSIm

Os testes do ModelSim foram funcionais e buscavam garantir que a instrução customizável realizasse aquilo que foi programada para fazer, sendo que esse foi escrito em código Verilog. De tal modo, que foram realizados dois testes, sendo que foram usados três arquivos em Verilog para isso.

O primeiro arquivo, chamado lcd_controller.v é o arquivo da instrução customizada. Já o segundo se chama toplevel.v. Esses dois arquivos, estão diretamente ligados, pois o toplevel.v faz uso de uma instância de lcd_controller.v (Figura 5).

```
lcd_controller teste(dataa, datab, result, clk, clk_en, start, reset, done, bl, rs, rw, en, db);
initial
   begin
         dataa=32'b00000000000000000000000111111111;
         clk_en=1'b0;
         reset=1'b1;
         datab=32'b00000000000000000000000111111111;
         start=1'b0;
         #20 reset=1'b0;clk_en=1'b0;
         #20 start=1'b1;clk_en=1'b1;//20
         #20 start=1'b0;
        //#10 a=1'b0;b=1'b1;
        //#10 a=1'b1;b=1'b1;
        //#100 $finish;
   end
```

Figura 5. Trecho do código de toplevel.v onde existe uma instância de lcd_controller.

A parti dessa ligação, o código de toplevel.v faz todo um trabalho para inicializar valores para testar lcd_controller. Dentro esses valores, que foram inicializados, vale destacar o dataa e o dataab, pois são valores usados no código assembly deste projeto e que servem para passar as instruções e os dados para preencher caracteres no display LCD.

```
#usado para instrucoes do lcd
.macro instr databits
   custom 0, r0, r0, \databits
.endm
#usado para dados no lcd
.macro data databits
   movi r1, 1
   custom 0, r0, r1, \databits
.endm
```

Figura 6. Trecho do código que vai fazer referência ao dataa e o datab.

Dito isso, na imagem acima(Figura 6) mostra o trecho de código assembly, que vai fazer o link com o dataa e o datab, sendo que o r1(registrador) faz referência ao dataa e

databits(isso no código também será um registrador) ao datab. Abordado isso, vale chamar atenção para imagem abaixo, na qual, mostra um ponto do arquivo lcd_controller.v onde o dataa e o datab são usadas.

```
done <= 1'b0;
if (start) begin
    state <= working;
    rs <= dataa[0];
    db <= datab[7:0];
    contador <= 17'd0;
    en <= 1'b1;
end else begin
```

Figura 7. Trecho do código de lcd_controller.v em que dataa e datab são usados.

Esse trecho do código acima, mostra o ponto em que o valor rs(saída que é enviado para o display LCD que pode indicar se é uma instrução ou se trata de escrever um caractere) e db(saída que é enviada para o display LCD, que contem a informação sobre os dados de instrução ou caractere).

A parti dessa, explanação foi realizado o teste que faz uso deste dois arquivos, ao qual, o desfecho deste teste é mostrado nas duas imagens abaixo(Figura 8 e 9) . Sendo que, essas imagens mostram basicamente o resultado dos valores que foram inicializados, com as respectivas saídas desejadas.

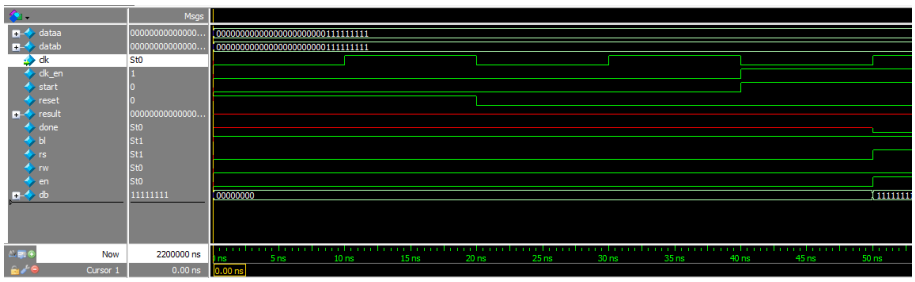


Figura 8. Resultado do teste com os dois arquivos.

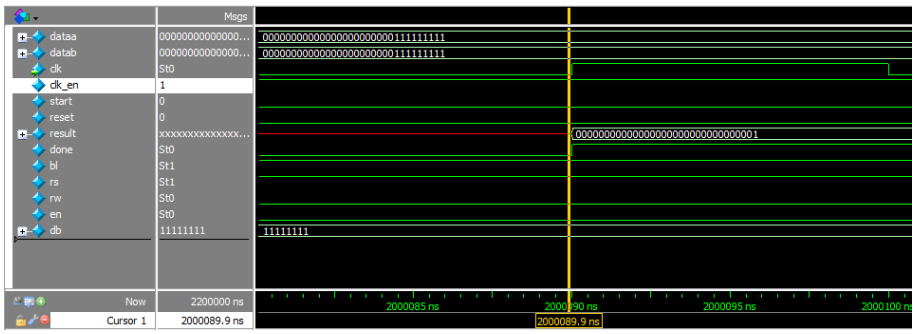


Figura 9. Resultado do teste com os dois arquivos.

No tocante as saídas desejadas, as que mais vale destaque é a done(essa saída indica que instrução customizada terminou sua tarefa) a qual as imagens mostram ela iniciando com zero e depois terminam com um(isso indica fim).Além da done, outro destaque é a db(informação enviada ao display LCD referente instrução ou caractere), que começou com os dados todo em zero e depois passou a emitir os valores todos em um.

Explanado esses pontos do teste um, já se referindo ao teste dois, ao qual, faz uso do arquivo chamado lcd_controller2.v. Esse arquivo, nada mais é do que uma cópia do arquivo o lcd_controller.v. O que difere um do outro é no lcd_controller2.v existe um conjunto de valores de entrada que são inicializados dentro deste arquivo.

Dito isso, a imagem abaixo mostra o trecho de código onde é feito a inicialização dos valores de entrada, inclusive vale destacar que esses valores são semelhantes ao do teste do arquivo toplevel.v, contudo os resultados são mais detalhados.

```
initial
   begin
    dataa=32'b00000000000000000000000111111111;
    clk_en=1'b0;
    reset=1'b1;
    datab=32'b00000000000000000000000111111111;
    start=1'b0;
    #20 reset=1'b0;clk_en=1'b0;
    #20 start=1'b1;clk_en=1'b1;//20
    #20 start=1'b0;
   end
```

Figura 10. Entradas inicializadas.

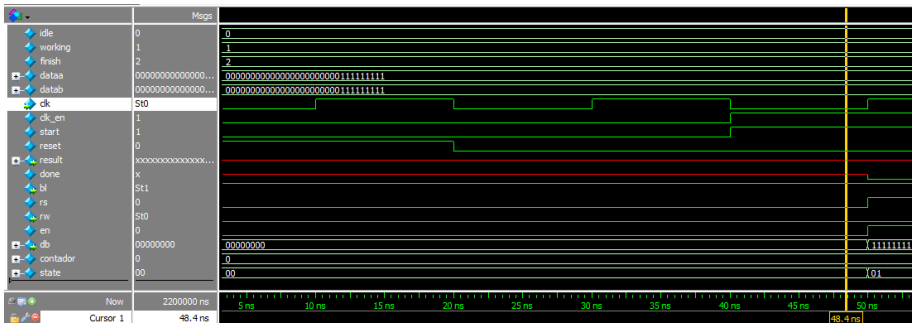


Figura 11. Resultado do teste do arquivo lcd_controller2.v.

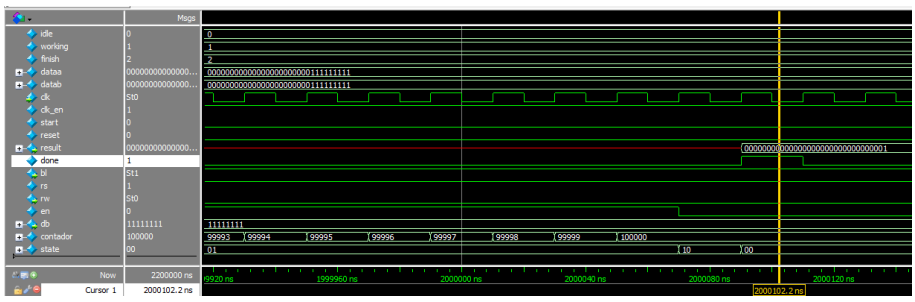
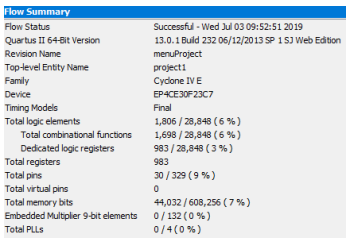


Figura 12. Resultado do teste do arquivo lcd_controller2.v.

As imagens acima(Figura 11 e 12), mostra o que o acontece em detalhes no tocante a instrução customizada, vale destacar aqui a variável contador(essa é do tipo reg), ao qual, ela inicia em zero e chega a 100000, fazendo se tenha como resultado um delay muito importante, que está ligado ao que foi abordado na seção Fundamentação Teórica sobre o display LCD.

4.2. Características Técnicas

No software Quartus, cada projeto compilado gera testes e informação triviais para demonstrar o desempenho do codigo quando for programado na FPGA, onde ele define modos de operações próximos de um circuito embarcado. Utilizando a função Chip Planner do Quartus, observa-se gráficamente a quantidade de LABs (Logic Array Block), mémoria, e entradas/saídas do chip usado no problema. Pode se, utilizar o resumo criado ápos a compilação do projeto.



Flow Summary	
Flow Status	Successful - Wed Jul 03 09:52:51 2019
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	menuProject
Top-level Entity Name	project1
Family	Cyclone IV E
Device	EP4CE30F23C7
Timing Models	Final
Total logic elements	1,806 / 28,848 (6 %)
Total combinational functions	1,698 / 28,848 (6 %)
Dedicated logic registers	983 / 28,848 (3 %)
Total registers	983
Total pins	30 / 329 (9 %)
Total virtual pins	0
Total memory bits	44,032 / 608,256 (7 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Figura 13. Sumário de compilação

Os LABs são um conjunto de elementos lógicos, e são representados pelo bloco em azul, onde azul claro não estam sendo utilizados e os mais escuros estão. Quanto mais escuro a cor, mais elementos estão sendo utilizados. As memórias representadas pelos blocos em cinza, entre os LABs, e quando ocupado, em verde. E em cinza nas bordas, as Entradas/Saidas.

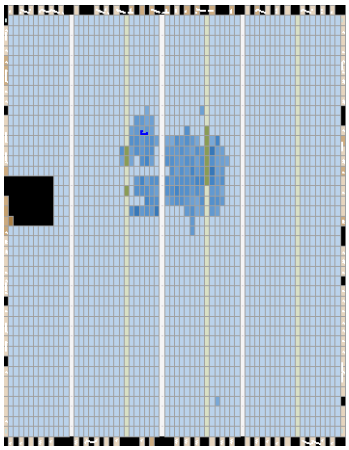


Figura 14. Área ocupada pelo projeto na FPGA

5. Conclusão

O produto desenvolvido atende a todos os requisitos determinados. Implementações ou upgrades poderiam dar solidez ao projeto, utilizando o display de sete segmentos em vez

dos leds melhorando a visualização de qual índice foi selecionado no menu. Também a prototipação de uma máquina de estados para o menu, utilizando a Custom Instruction, na qual foi implementado em Software, melhorando o desempenho e organizando o código fonte.

Referências

David A. Patterson, J. L. H. (2017). *Computer Organization and Design The Hardware/Software Interface ARM Edition*. Elsevier.

Pacievitch, Y. (2011 (acessado 29 de junho , 2019)). *Processador*.

Prado, A. C. (2014 (acessado 29 de junho , 2019)). *FPGA*.

Souza, A. C. ((acessado 29 de junho , 2019)). *RISC X CISC - Pipeline*.

TI, C. (2016 (acessado 29 de junho , 2019)). *Arquitetura de Computadores – CISC X RISC – Definição, aplicação e diferenças*.

Unicamp ((acessado 29 de junho , 2019)). *Arquitetura ARM Linguagem Assembly*.

Wikipedia (2017 (acessado 29 de junho , 2019)). *Verilog*.