



Laboratório 1 - Assembly MIPS e Verilog

Alunos:

- André Almeida - 12/0007100
- Arthur de Matos Beggs - 12/0111098
- Bruno Takashi Tengan - 12/0167263
- Gabriel Pires Iduarte - 13/0142166
- Guilherme Caetano - 13/0112925
- João Pedro Franch - 12/0060795
- Rafael Lima - 10/0131093

Parte A - Assembly MIPS

1) Simulador/Montador MARS

1.1) Dado o programa `sort.s` e o vetor: $V[10] = \{5,8,3,4,7,6,8,0,1,9\}$, ordená-lo em ordem crescente e contar o número de instruções (por tipo e por estatística) exigido pelo algoritmo. Qual o tamanho em bytes do código executável?

Para a ordenação crescente do vetor $V[10] = \{5,8,3,4,7,6,8,0,1,9\}$, foram contabilizadas 802 instruções.

Classificando as instruções por tipo se obtém:

- 255 instruções do tipo R;
- 472 instruções do tipo I;
- 75 instruções do tipo J.

Classificando as instruções por estatística se obtém:

- 308 instruções da ULA;
- 99 instruções de salto;
- 92 instruções de *branching*;

- 170 instruções de acesso à memória;
- 133 instruções de outros tipos.

O código executável se inicia no endereço de memória 0x00400000 e finaliza no endereço 0x00400124, ocupando 0x00000128 (296₁₀) bytes de memória.

1.2) Explique o que é o valor 589834 armazenado a partir do endereço 0x10010028. O emulador trabalha no formato Little Endian ou Big Endian?

O valor 589834₁₀ pode ser convertido para 0x0009000a. Os caracteres ASCII são codificados em 1 byte e podem ser representados por 2 dígitos hexadecimais, sendo 0x00 o caractere "\0" (NULL), 0x0a o caractere "\n" (NEW LINE) e 0x09 o caractere "\t" (TAB). Observando byte a byte os quatro primeiros bytes a partir do endereço 0x10010028, lê-se "\n\0\t\0". Como os bytes são salvos do menor para o maior endereço de memória, o emulador trabalha no formato Big Endian.

1.3) Considerando a execução em um processador MIPS com frequência de clock de 50MHz, que necessita 1 ciclo de clock para a execução de cada instrução (CPI=1).

(a) Calcule o tempo de execução necessário para o programa e dados do item 1.1).

$$t_{\text{exec}} = I * CPI * 1/f = 802 * 1 * 1/(50*10^6) = 1.604*10^{-2} \text{ ms}$$

(b) Para os vetores de entrada de n elementos já ordenados {1,2,3,4...n} e ordenados inversamente {n, n-1, n-2, ... ,2,1}, meça a o número de instruções necessário a execução para n = {1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 ,20 ,30 ,40 ,50 ,60 ,70 ,80 ,90 ,100} e plote esses dados em um mesmo gráfico n x I.

Utilizando o Matlab para armazenar os dados e gerar os gráficos, as duas matrizes geradas com os dados lidos foram:

Crescente = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100; 79, 114, 149, 184, 219, 254, 289, 324, 359, 394, 744, 1094, 1444, 1794, 2144, 2494, 2844, 3194, 3544];

Decrescente = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100; 79, 128, 197, 286, 395, 524, 673, 842, 1031, 1240, 4430, 9620, 16810, 26000, 37190, 50380, 65570, 82760, 101950];

Nessa representação, a primeira linha corresponde ao número de elementos no vetor, e a segunda linha corresponde ao número de instruções executadas. Com esses dados, o gráfico de Instruções por Tamanho do Vetor foi gerado para os vetores organizados de maneira crescente e decrescente.

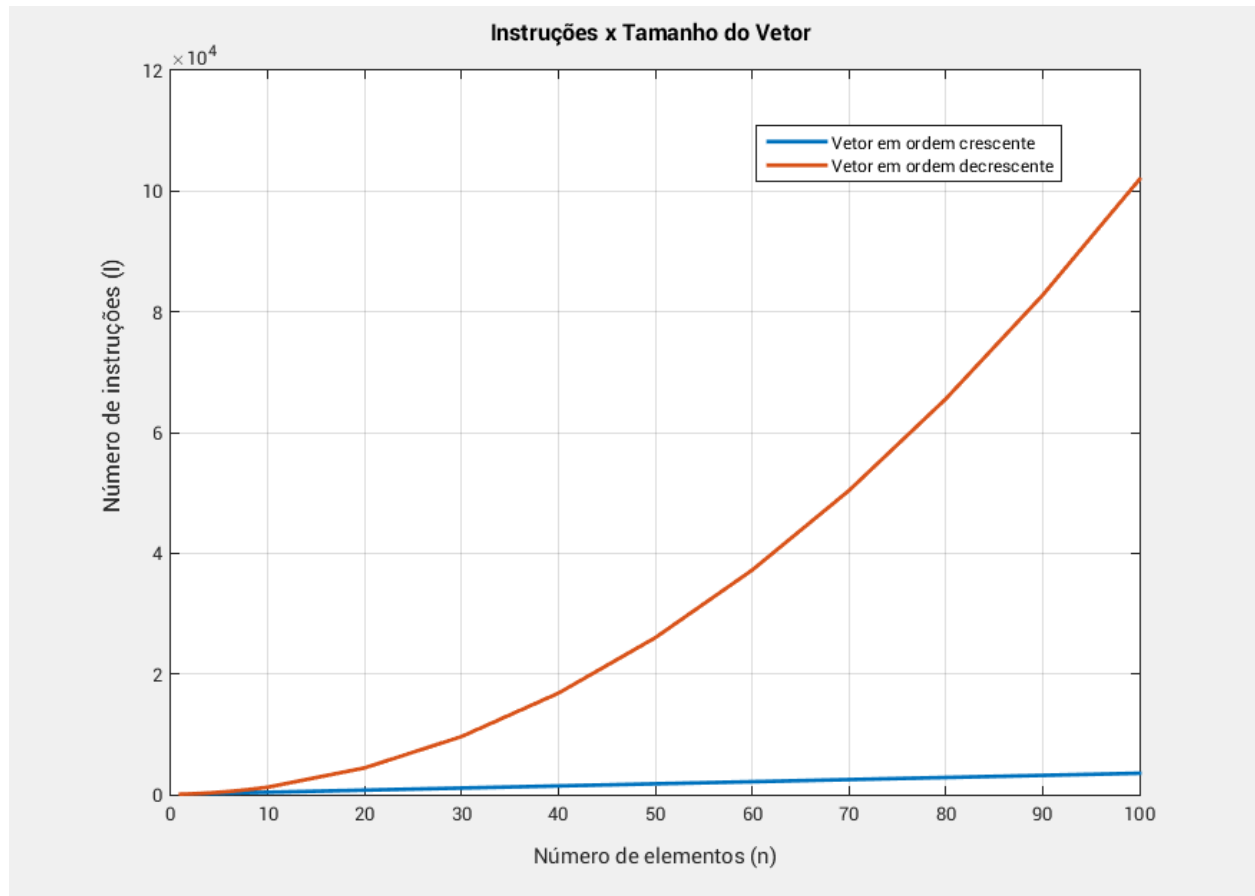


Figura 1 – Gráfico Número de instruções x Número de Elementos.

2) Compilador GCC

2.1) Dado o programa `sort.c`, compile-o e comente o código em Assembly obtido indicando a função de cada uma das diretivas do montador usadas no código Assembly (`.file` `.section` `.mdebug` `.previous` `.nan` `.gnu_attribute` `.globl` `.data` `.align` `.type` `.size` `.word` `.rdata` `.align` `.ascii` `.text` `.ent` `.frame` `.mask` `.fmask` `.set`).

O arquivo "`Lab1a/Pt2_GCC/sort_pt1.s`" contém os comentários indicando a função das diretivas do montador.

2.2) Indique as modificações necessárias no código Assembly gerado para poder ser executado corretamente no Mars.

- As diretivas `.set`, `.ent`, `.end`, `.ident`, `.frame`, `.mask`, `.fmask`, `.rdata`, `.gnu_attribute`, `.nan`, `.previous`, `.section` e `.file` foram retiradas (comentadas) por não serem reconhecidas pelo MARS;
- As diretivas `.align` após início do bloco de texto removidas (comentadas) pois o MARS não aceita a diretiva `.align` no segmento de texto;
- Os blocos de instruções `lui` e `addiu` foram trocados por `la`, uma vez que as instruções `lui` e `addiu` só aceitam inteiros como argumento no imediato;
- As instruções `j $31` foram trocadas pela instrução `jr $31`, uma vez que a instrução `j` não aceita um registrador como argumento;
- A rotina `main` foi transferida para o início do segmento de texto;
- `.ascii "%d\011\000"` foi alterado para `.asciiz "\t"` para facilitar a rotina `printf`;
- Foram criadas as rotinas `printf` e `putchar`;
- `jr $ra` da rotina `main` substituído por `j exit`, pois não há endereço de retorno;
- Foi criada a rotina `exit`.

O arquivo "`Lab1a/Pt2_GCC/sort_pt2.s`" contém as modificações realizadas e é executado no MARS conforme esperado.

2.3) Compile novamente o programa `sort.c` e compare o número de instruções executadas e o tamanho em bytes dos códigos obtidos com os dados do item 1.1) para cada diretiva de otimização da compilação `{-O0, -O1, -O2, -O3, -Os}`.

Os arquivos "`sort_pt2_O0.s`", "`sort_pt2_O1.s`", "`sort_pt2_O2.s`", "`sort_pt2_O3.s`" e "`sort_pt2_Os.s`" presentes na pasta "`Lab1a/Pt2_GCC/`" foram compilados com as diretivas `-O0`, `-O1`, `O2`, `-O3` e `-Os` respectivamente e modificados para executar no

MARS. O número de instruções e o tamanho em bytes de código de cada arquivo se encontram nas tabelas a seguir.

Arquivo	Instruções Tipo-R	Instruções Tipo-I	Instruções Tipo-J	Total de Instruções	Tamanho (bytes)
sort_pt1.s	255	472	75	802	296
sort_pt2_00.s	684	1259	60	2003	640
sort_pt2_01.s	363	478	48	889	476
sort_pt2_02.s	277	430	32	739	412
sort_pt2_03.s	277	430	32	739	412
sort_pt2_0s.s	413	532	95	1040	396

Tabela 1 – Instruções por tipo

Arquivo	Instruções da ULA	Instruções de Salto	Instruções de Branching	Instruções de Acesso à Memória	Outras instruções
sort_pt1.s	308	99	92	170	133
sort_pt2_00.s	519	106	61	892	425
sort_pt2_01.s	237	94	91	200	267
sort_pt2_02.s	199	55	82	142	261
sort_pt2_03.s	199	55	82	142	261
sort_pt2_0s.s	309	139	61	186	345

Tabela 2 – Instruções por estatística

3) Primos Gêmeos

3.1) Escreva um procedimento em Assembly MIPS que retorne o primeiro elemento do i-ésimo par de números primos gêmeos, int PG(int i).

O arquivo "Lab1a/Pt3_Primos/twinPrimes.s" contém o procedimento pedido. O procedimento recebe o valor do i-ésimo par desejado no registrador \$a0 e retorna o primeiro primo gêmeo do i-ésimo par no registrador \$v0.

3.2) Escreva um programa void main(void) que leia o valor de i do teclado e mostre na tela o resultado com a formatação printf("Primos Gêmeos(%d) = (%d,%d)\n",i ,P[i] ,P[i]+2).

O arquivo "Lab1a/Pt3_Primos/twinPrimes.s" contém o programa pedido.

3.3) Encontre o número imax, de forma que $P_{imax}+2$ seja o maior número Primo Gêmeo representável utilizando uma palavra de 32 bits sem sinal.

Usando a primeira conjectura de Hardy-Littlewood se obtêm a estimativa de que i_{MAX} é 12.739.550.

Mesmo se usando um bom algoritmo de teste de primalidade e deixando o programa o mais otimizado o possível à medida das nossas capacidades, o programa levaria mais tempo que o prazo do trabalho para encontrar i_{MAX} exato. Executando por três dias ininterruptos, o programa atingiu somente 1/6 do índice máximo.

Utilizando artifícios de manipulação de registradores durante a execução do programa criado, foi encontrado $P_{imax}+2$.

$$P_{imax}+2 = 4294965841$$

3.4) Para o seu procedimento, determine uma equação $NInstr(i)$ que seja capaz de estimar o número de instruções necessárias ao cálculo de P_i+2 . Desenhe as curvas de $NInstr(i)$ e do número de instruções real em um gráfico para $i=\{1,2,3,...,100\}$.

Para se ter uma equação teórica da estimativa de número de instruções é necessário estimar o intervalo onde se tem o i-ésimo par de primos gêmeos. Dado os números de primos gêmeos de 1 a x, PG(X), tem se a equação 1.

$$PG(x) \gg x \frac{\log(\log(x))^2}{\log(x)^2}$$

Equação 1 – Estimativa da quantidade de números primos gêmeos.

O algoritmo usado para teste de primalidade consiste em dividir o número sendo testado, n, por todos os primos de 1 a raiz quadrada de n. Para se estimar a quantidade de primos de 1 a x a fórmula aperfeiçoada de Legendre, equação 2.

$$P(x) < \frac{x}{\ln(x) - 1.08366}$$

Equação 2 – Estimativa da quantidade de números primos.

Assim para cada i-ésimo par temos uma quantidade estimada de instruções conforme a equação 3, onde L é o número de instruções para se descartar um possível múltiplo de n.

$$\sum_{k=1}^{PG^{-1}(i)} \frac{\sqrt{k}}{\ln(\sqrt{k}) - 1.083} * L$$

Equação 3 – Estimativa teórica do número de instruções.

Para se ter uma equação empírica da estimativa de número de instruções se fez uma regressão polinomial de grau 4.

$$Ninst(i) = 0.0001003x^4 + -0.06287x^3 + 14.1x^2 + 90.77x^1 - 692.1x^0$$

Equação 4 – Estimativa empírica do número de instruções.

Usando a estimativa empírica tem-se a Figura 2.

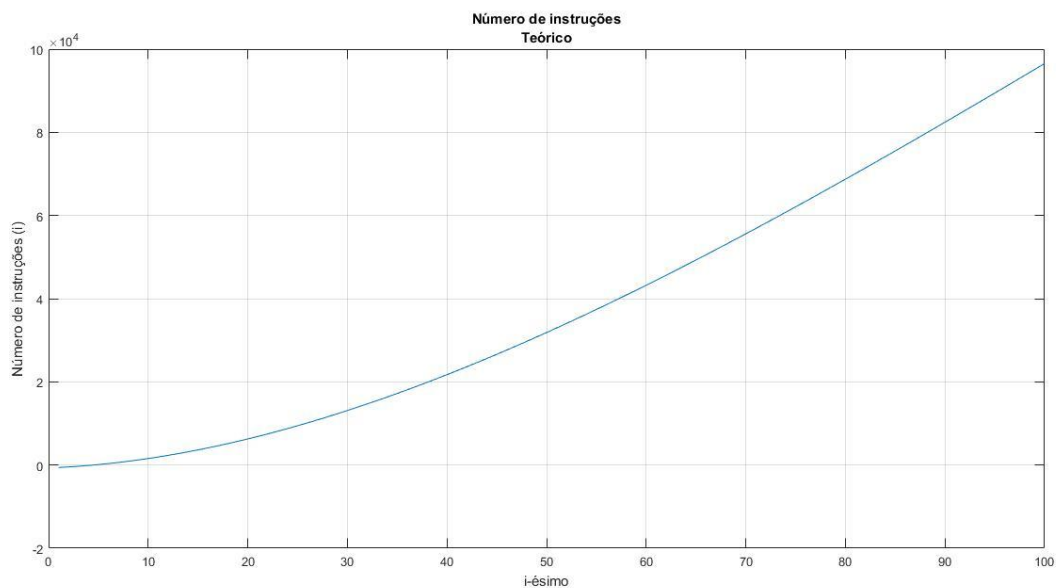


Figura 2 – Curva da função de estimativa do número de instruções.



Figura 3 – Curva do número real de instruções.

3.5) Considerando que vamos utilizar a implementação de um processador MIPS32 com CPI de 1 e frequência de clock de 50MHz, qual o tempo esperado para o cálculo do valor de $P_{i_{max}+2}$? Qual deveria ser a frequência deste processador para que o cálculo do $P_{i_{max}+2}$ se realizasse em 1 minuto?

A primeira conjectura de Hardy-Littlewood estima que o i_{MAX} representável em 32 bits é de 12739550. Usando a equação empírica, equação 4, do item 3.4 têm-se um número, estimado, de instruções em $2.6418 \cdot 10^{24}$.

Com uma CPI de 1 e frequência de clock de 50MHz, o tempo esperado para o cálculo do valor de $P_{i_{MAX}+2}$ é de:

$$t_{exc} = \frac{I * CPI * 1}{clock}$$

$$t_{exc} = 5.2836 * 10^{16} [s]$$

Para se ter t_{exc} em 1 minuto:

$$60 = \frac{2.6418 * 10^{24}}{clock}$$

$$clock = 4.4030 * 10^{22} Hz$$

$$clock = 44.030 ZHz$$

Parte B - Síntese de Hardware

2) Implementação do Somador de 4 bits

2.1) Implementação por desenho esquemático no Quartus II

- (a) Crie um novo diretório e projeto esquemático (somador_esquematico). Implemente os blocos esquemáticos do meio-somador de 1 bit (half_add.bsf), do somador completo de 1 bit (full_add.bsf) e do somador completo de 4 bits (add4.bsf), conforme as figuras em anexo (no roteiro).

Os esquemáticos foram todos implementados e se encontram na pasta Lab1>Lab1b>somador_esquematico, cada um com o seu devido nome. Caso for executar o projeto, coloque o arquivo "add4.bdf" como top-level.

- (b) Realize a simulação funcional completa do projeto do somador de 4 bits; Realize a simulação temporal com T=10ns para cada alteração de A. Explique o que ocorre.

A simulação funcional completa está toda guardada na pasta Lab1>Lab1b>somador_esquematico>waveforms. Para facilitar a visualização, imagens das simulações estão salvas na pasta "Imagens" que se encontra na mesma pasta onde estão as simulações. Nas simulações foi definido um valor fixo para B e fizemos A variar com o tempo do valor de 0 a F em hexadecimal. Sendo assim, os arquivos foram nomeados com um nome padrão seguido pelo valor de B que foi assumido na simulação. Para exemplificar, abaixo a imagem da simulação com B assumindo o valor 4 recebendo assim o nome "waveform21b4.jpg".

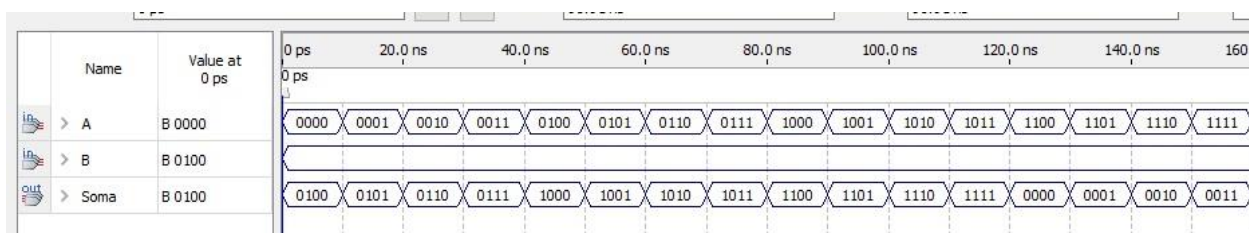


Figura 4 - Imagem da simulação funcional do somador de 4 bits com B igual a 4.

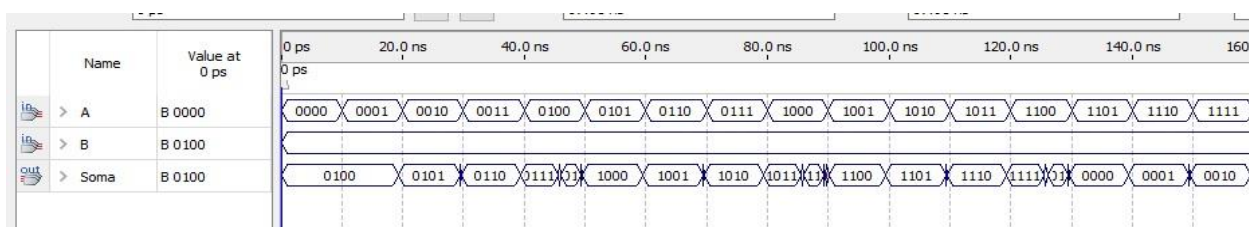


Figura 5 - imagem da simulação temporal do somador de 4 bits com B igual a 4.

Analisando a simulação temporal feita, com uma variação na entrada de dados igual a 10ns não é possível se obter um funcionamento satisfatório do circuito pois o intervalo de tempo não respeita o tempo necessário que o circuito precisa para processar os seus dados, gerando uma inconsistência nos seus resultados que piora nos momentos que ocorrem o pior caso de atraso de propagação no circuito.

- (c) Crie o arquivo de teste (teste.bdf) com as interconexões com a placa DE2-70, sintetize e teste.

O esquemático foi implementado e se encontra na pasta Lab1>Lab1b>somador_esquematico, cada um com o seu devido nome. Caso for executar o projeto, coloque o arquivo "teste.bdf" como top-level.

2.2) Descrição Verilog ao nível de portas lógicas no Quartus II

- (a) Crie um novo diretório e projeto Verilog (somador_portas). Implemente as descrições do meio-somador de 1 bit (half_addv.v), do somador completo de 1 bit (full_addv.v) e do somador completo de 8 bits (add8v.v), conforme as figuras (no roteiro).

Projeto "somador_portas" encontra-se na pasta Lab1>Lab1b>somador_portas. Para a execução do projeto desejado é necessário definir o arquivo de interesse como top-level.

- (b) Realize a simulação funcional completa do projeto do somador de 8 bits; Realize a simulação temporal com T=10ns para cada alteração de A. Explique o que ocorre.

Observa-se na simulação funcional (figura 4) a transição do valor de saída imediatamente após a alteração do valor de entrada. A variável de saída "oSUM" mostra os valores da soma de "iA" e "iB" variando de 0 a F hexadecimal. Para valores de soma maiores que F, o valor 1 é atribuído ao carry e o valor de "oSUM" é definido como o dígito hex menos significativo do valor da soma. A figura abaixo exemplifica o que foi feito, para acessar as demais imagens basta entrar em Lab1>Lab1b>somador_portas>waveforms>imagens. Os arquivos foram nomeados com um nome padrão seguido pelo valor de iB que foi assumido na simulação. Para exemplificar, abaixo a imagem da simulação com iB assumindo o valor 4 recebendo assim o nome "waveform21b4.jpg".

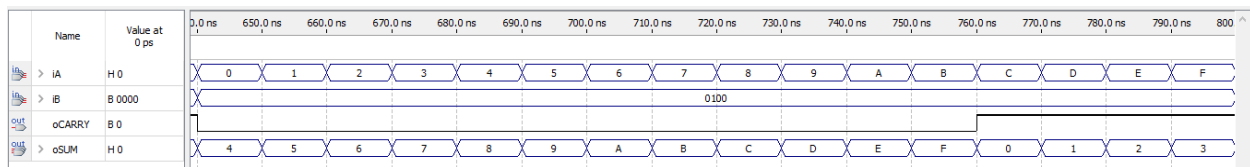


Figura 6 - Imagem da simulação funcional do somador em Verilog de 4 bits com B igual a 4.

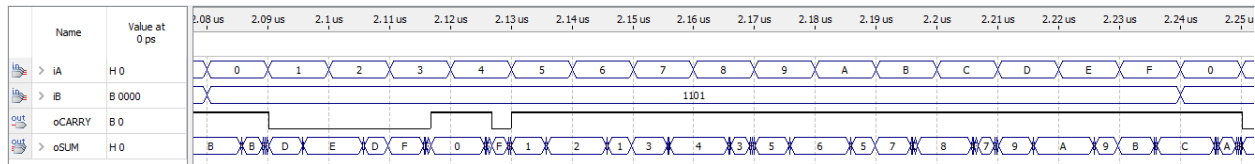


Figura 7 - Imagem da simulação temporal do somador em Verilog de 4 bits com B igual a 4.

Na simulação temporal percebe-se que o atraso para estabilização da saída varia muito dependendo das entradas, e até a estabilização os resultados não são confiáveis, tendo inclusive falsos valores carry out. Alguns valores de atraso são inclusive maiores que os 10ns entre uma variação e outra de "iA" fazendo com que a resposta não seja confiável em momento nenhum desses casos.

Table of Contents

Ignored Assignments

Incremental Compilation Section

Pin-Out File

Resource Section

- Resource Usage Summary
- Partition Statistics
- Input Pins
- Output Pins
- I/O Bank Usage
- All Package Pins
- Output Pin Default Load For Reported TCO
- Resource Utilization by Entity
- Delay Chain Summary
- Pad To Core Delay Chain Fanout
- Non-Global High Fan-Out Signals

Logic and Routing Section

Device Options

Operating Settings and Conditions

Delay Chain Summary

	Name	Pin Type	Pad to Core 0	Pad to Core 1
1	oSUM[0]	Output	--	--
2	oSUM[1]	Output	--	--
3	oSUM[2]	Output	--	--
4	oSUM[3]	Output	--	--
5	oCARRY	Output	--	--
6	iA[0]	Input	(0) 170 ps	(0) 170 ps
7	iB[0]	Input	(0) 170 ps	(0) 170 ps
8	iA[1]	Input	(6) 2514 ps	(6) 2514 ps
9	iB[1]	Input	(6) 2514 ps	(6) 2514 ps
10	iA[2]	Input	(6) 2514 ps	(6) 2514 ps
11	iB[2]	Input	(6) 2514 ps	(6) 2514 ps
12	iB[3]	Input	(6) 2514 ps	(6) 2514 ps
13	iA[3]	Input	(6) 2514 ps	(6) 2514 ps

Figura 8 – Delay Chain Summary (nível de portas).

O Delay Chain Summary mostra que na solução do Quartus II, alterações no nível menos significativo das entradas possuem pouco atraso total, mas alterações nos níveis mais significativos o atraso é maior. Isso mostra que a solução para o

circuito apresentada pelo Quartus II é diferente da projetada em código, seguindo o modelo do esquemático do roteiro.

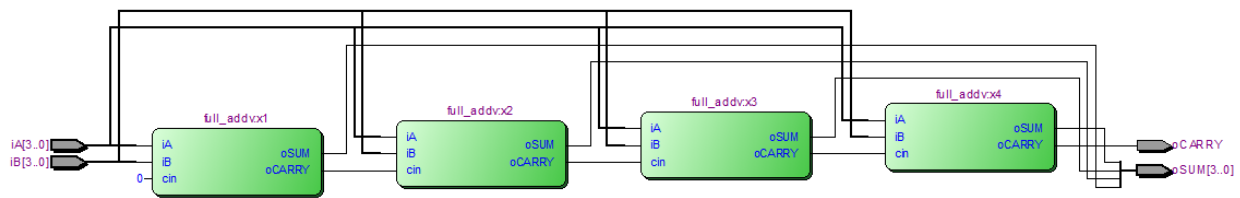


Figura 9 – Esquemático RTL Viewer – Modelo definido por código no projeto.

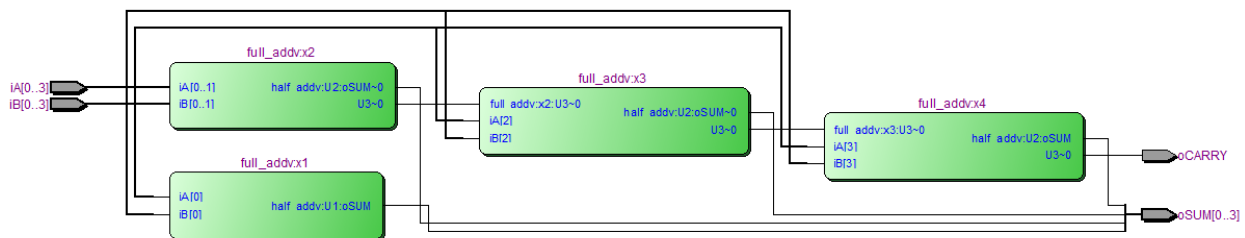


Figura 10 – Esquemático Post Mapping – Modelo implementado pelo Quartus II

Observando os circuitos das figuras 9 e 10, percebe-se que o motivo das diferenças nos tempos é devido ao fato de que o somador de 1 bit do nível menos significativo implementado pelo Quartus II tem alteração direta na resposta final, em vez de propagar a alteração por todo o circuito como no modelo projetado.

(c) Crie o arquivo de teste (teste.v) com as interconexões com a placa DE2-70, sintetize e teste.

O projeto foi implementado e se encontra na pasta Lab1>Lab1b>somador_portas. Caso for executar o projeto, coloque o arquivo "teste.v" como top-level.

2.3) Descrição Verilog ao nível comportamental no Quartus II

(a) Crie um novo diretório e projeto Verilog (somador_comportamental). Implemente um somador completo de 4 bits (add4cv.v).

Projeto "somador_comportamental" encontra-se na pasta Lab1>Lab1b>somador_comportamental. Para a execução do somador completo de 4 bits defina o arquivo "add4cv.v" como top-level.

- (b) Realize a simulação funcional completa do projeto do somador de 4 bits; Realize a simulação temporal com $T=10\text{ns}$ para cada alteração de A. Explique o que ocorre.

A simulação funcional completa encontra-se em Lab1>Lab1b>somador_comportamental>waveforms>imagens. O mesmo método aplicado nas questões 2.1b e 2.2b foi seguido, fixando B e variando A de 0 a F hexadecimal. Os arquivos também seguem a mesma lógica de nomenclatura, waveform seguido do número da questão e o valor de B, exemplo: "waveform23b9".

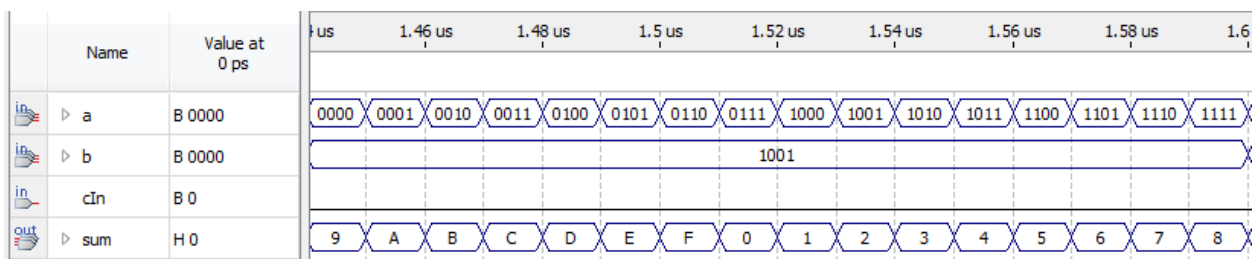


Figura 11 - Imagem da simulação funcional do somador em Verilog Comportamental de 4 bits com B igual a 9.

A simulação temporal foi realizada e notou-se que 10ns é um tempo inferior ao necessário para a estabilização do resultado, gerando resultados inconsistentes e não confiáveis. A figura 12 ilustra o problema.

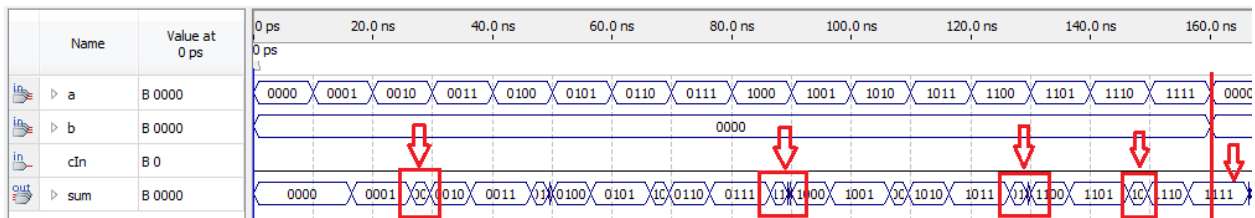


Figura 12 – Imagem da simulação temporal com somador Verilog Comportamental de 4 bits.

- (c) Modifique o arquivo de teste (teste.v) com as interconexões com a placa DE2-70, sintetize e teste; Apresente a verificação do projeto. Filme e narre os testes.

O projeto foi implementado e se encontra na pasta Lab1>Lab1b>somador_comportamental. Caso for executar o projeto, coloque o arquivo "teste.v" como top-level.

Link do YouTube: https://youtu.be/DE6yfg0s_3A

2.4) Análise comparativa das três sínteses do Somador de 4 bits

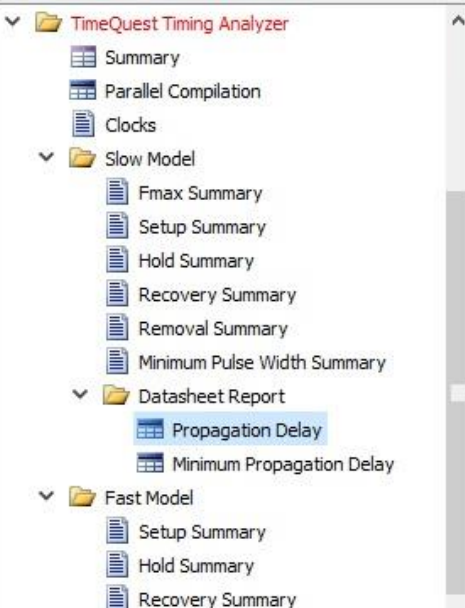
(a) Compare os números de Elementos Lógicos necessários e os maiores tempos de atraso de propagação (tpd) para cada implementação.

A tabela abaixo contém os dados para o somador de 4 bits para as implementações diferentes:

IMPLEMENTAÇÃO	Nº DE ELEMENTOS LÓGICOS	TEMPO MÁXIMO DE DELAY (ns)
ESQUEMÁTICO	6	11.020
VERILOG	7	10.834
VERILOG COMPORTAMENTAL	5	10.618

Tabela 3 – Comparação entre implementações para somador 4bits

Notem que o Verilog Comportamental tem menos elementos lógicos e um tempo máximo de delay menor. Abaixo as imagens para o tempo máximo de delay.



	Input Port	Output Port	RR	RF	FR	FF
1	A[0]	Soma[3]	11.020	11.020	11.020	11.020
2	B[0]	Soma[3]	10.867	10.867	10.867	10.867
3	B[1]	Soma[3]	10.833	10.833	10.833	10.833
4	A[0]	Soma[2]	10.566	10.566	10.566	10.566
5	A[1]	Soma[3]	10.508	10.508	10.508	10.508
6	B[0]	Soma[2]	10.413	10.413	10.413	10.413
7	B[1]	Soma[2]	10.379	10.379	10.379	10.379
8	A[3]	Soma[3]	10.192	10.192	10.192	10.192
9	A[1]	Soma[2]	10.054	10.054	10.054	10.054
10	B[2]	Soma[3]	9.986	9.986	9.986	9.986
11	A[0]	Soma[1]	9.879	9.879	9.879	9.879
12	B[0]	Soma[1]	9.733	9.733	9.733	9.733
13	B[1]	Soma[1]	9.689	9.689	9.689	9.689
14	B[2]	Soma[2]	9.529	9.529	9.529	9.529
15	B[0]	Soma[0]	9.510	9.510	9.510	9.510
16	A[0]	Soma[0]	9.386	9.386	9.386	9.386
17	A[1]	Soma[1]	9.364	9.364	9.364	9.364
18	A[2]	Soma[3]	6.025	6.025	6.025	6.025
19	A[2]	Soma[2]	5.571	5.571	5.571	5.571
20	B[3]	Soma[3]	5.350	5.350	5.350	5.350

Figura 13 – Tempo Máximo de delay para a implementação em esquemático.

Compilation Report - somador

add4v.v

Table of Contents

Flow Summary

Flow Settings

Flow Non-Default Global Settings

Flow Elapsed Time

Flow OS Summary

Flow Log

Analysis & Synthesis

Fitter

Assembler

TimeQuest Timing Analyzer

Summary

Parallel Compilation

Clocks

Slow Model

Fmax Summary

Setup Summary

Hold Summary

Recovery Summary

Removal Summary

Minimum Pulse Width Summary

Datasheet Report

Propagation Delay

Minimum Propagation Delay

Fast Model

Multicorner Timing Analysis Summary

Minimum Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	iB[1]	oSUM[3]	10.834	10.834	10.834	10.834
2	iB[1]	oCARRY	10.818			10.818
3	iB[1]	oSUM[2]	10.421	10.421	10.421	10.421
4	iA[1]	oSUM[3]	10.390	10.390	10.390	10.390
5	iA[1]	oCARRY	10.374			10.374
6	iA[1]	oSUM[2]	9.977	9.977	9.977	9.977
7	iB[1]	oSUM[1]	9.943	9.943	9.943	9.943
8	iB[2]	oSUM[3]	9.930	9.930	9.930	9.930
9	iB[2]	oCARRY	9.914			9.914
10	iB[3]	oSUM[3]	9.720	9.720	9.720	9.720
11	iB[3]	oCARRY	9.701			9.701
12	iA[2]	oSUM[3]	9.552	9.552	9.552	9.552
13	iA[2]	oCARRY	9.536			9.536
14	iB[2]	oSUM[2]	9.522	9.522	9.522	9.522
15	iA[1]	oSUM[1]	9.498	9.498	9.498	9.498
16	iA[3]	oSUM[3]	9.432	9.432	9.432	9.432
17	iA[3]	oCARRY	9.417			9.417
18	iA[2]	oSUM[2]	9.135	9.135	9.135	9.135
19	iB[0]	oSUM[3]	6.891	6.891	6.891	6.891
20	iB[0]	oCARRY	6.875			6.875
21	iA[0]	oSUM[3]	6.590	6.590	6.590	6.590
22	iA[0]	oCARRY	6.574			6.574
23	iB[0]	oSUM[2]	6.478	6.478	6.478	6.478
24	iA[0]	oSUM[2]	6.177	6.177	6.177	6.177
25	iB[0]	oSUM[1]	6.000	6.000	6.000	6.000
26	iA[0]	oSUM[0]	5.932	5.932	5.932	5.932
27	iA[0]	oSUM[1]	5.698	5.698	5.698	5.698
28	iB[0]	oSUM[0]	5.688	5.688	5.688	5.688

Figura 14 – Tempo Máximo de delay para a implementação em Verilog.

add4cv.v

Compilation Report

add4cv

Table of Contents

Flow Log

Analysis & Synthesis

Fitter

Assembler

TimeQuest Timing Analyzer

Summary

Parallel Compilation

Clocks

Slow Model

Fmax Summary

Setup Summary

Hold Summary

Recovery Summary

Removal Summary

Minimum Pulse Width Summa

Datasheet Report

Propagation Delay

Minimum Propagation De

Fast Model

Multicorner Timing Analysis Sumn

Multicorner Datasheet Report Su

Propagation Delay

	Input Port	Output Port	RR	RF	FR	FF
1	cIn	sum[3]	10.618	10.618	10.618	10.618
2	cIn	sum[2]	10.546	10.546	10.546	10.546
3	a[2]	sum[3]	10.494	10.494	10.494	10.494
4	b[1]	sum[3]	10.489	10.489	10.489	10.489
5	b[1]	sum[2]	10.417	10.417	10.417	10.417
6	cIn	sum[1]	10.405	10.405	10.405	10.405
7	a[1]	sum[3]	10.335	10.335	10.335	10.335
8	cIn	sum[0]	10.295	10.295	10.295	10.295
9	a[1]	sum[2]	10.263	10.263	10.263	10.263
10	b[2]	sum[3]	10.257	10.257	10.257	10.257
11	a[2]	sum[2]	10.106	10.106	10.106	10.106
12	b[1]	sum[1]	9.959	9.959	9.959	9.959
13	b[2]	sum[2]	9.870	9.870	9.870	9.870
14	a[1]	sum[1]	9.806	9.806	9.806	9.806
15	a[3]	sum[3]	9.795	9.795	9.795	9.795
16	b[3]	sum[3]	9.653	9.653	9.653	9.653
17	a[0]	sum[3]	6.281	6.281	6.281	6.281
18	b[0]	sum[3]	6.243	6.243	6.243	6.243
19	a[0]	sum[2]	6.209	6.209	6.209	6.209
20	b[0]	sum[2]	6.171	6.171	6.171	6.171
21	a[0]	sum[1]	6.068	6.068	6.068	6.068
22	b[0]	sum[1]	6.030	6.030	6.030	6.030
23	a[0]	sum[0]	5.643	5.643	5.643	5.643
24	b[0]	sum[0]	5.604	5.604	5.604	5.604

Figura 15 – Tempo Máximo de delay para a implementação em Verilog Comportamental.

(b) Implemente as correspondentes versões síncronas. Quais as maiores frequências de clock utilizáveis?

Para o somador esquemático foi criado um novo arquivo esquemático "add4-sincrono.bdf" que se encontra na pasta Lab1>Lab1b>somador_esquemático. Para tornar o somador esquemático síncrono foi adicionado flipflops JK nas saídas de cada um dos somadores "full_add", tendo uma de suas saídas entrando no set do flipflop, outra saída negada entrando no reset do flipflop e colocando um clock como o enable, assim as suas saídas tornaram-se síncronas com o clock. No circuito o clock teve que ser negado por conta da lógica do flipflop do Quartus II para que funcionassem no pos-edge.

Para o somador em Verilog a nível de portas lógicas, a versão síncrona foi adicionada à raiz do projeto como "add4v_s". Ele se encontra em Lab1>Lab1b>somador_portas. Para tornar o somador síncrono, uma condicional de leitura foi imposta ao somador para que as entradas só variem no posedge do clock. Assim como a condicional de entrada, uma condicional foi imposta à escrita do resultado nos displays de sete segmentos, como pode ser observado nos arquivos "display7_s" e "show_carry_s", ambos presentes na mesma pasta de "add4v_s".

Para o somador em Verilog Comportamental, foi adicionado uma variável chamada clk (clock). Na chamada da rotina foi permitido que só executasse a soma com um post edge do clk. Tornando o somador síncrono, afinal ele só executará a rotina na borda de descida de clk. O arquivo encontra-se em Lab1>Lab1b>somador_comportamental com o nome de "add4cvs.v".

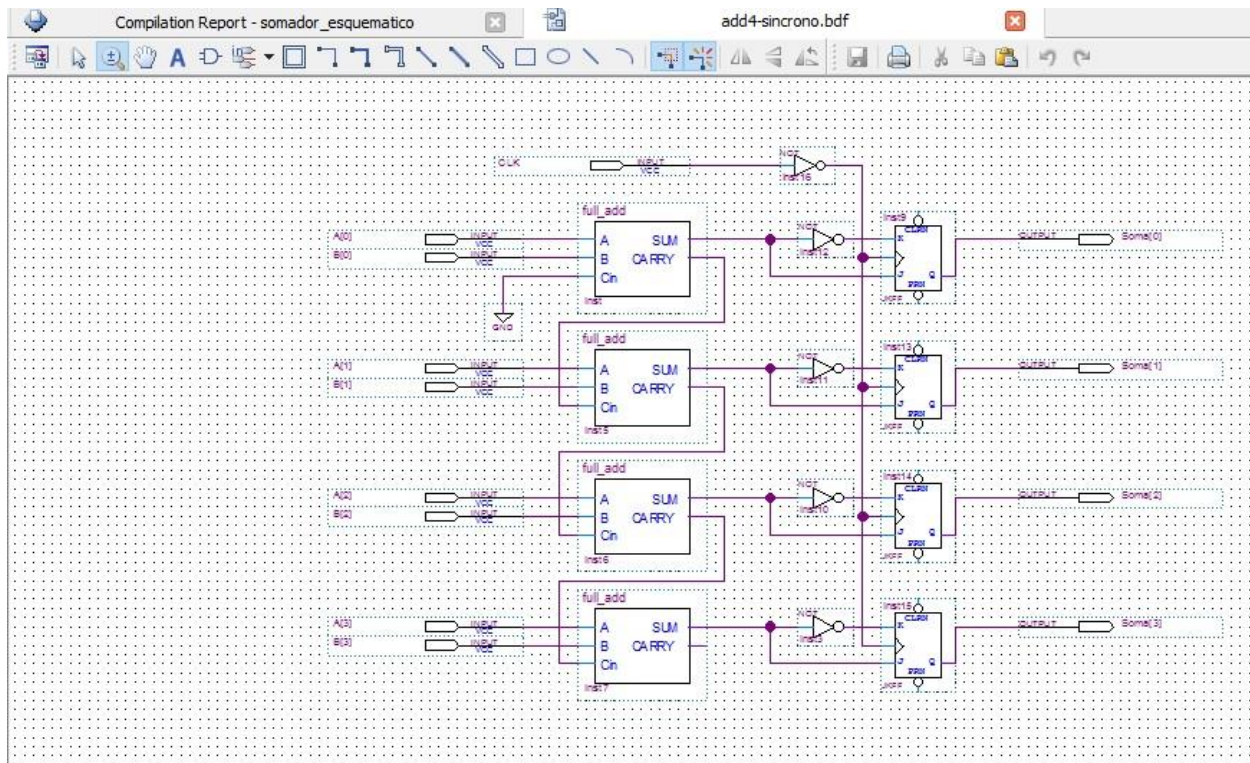


Figura 16 – Somador de 4 Bits Síncrono.

A máxima frequência de clock utilizável pelos somadores deve respeitar o maior tempo de atraso de propagação (tempo máximo de delay) obtido no item 2.4.a. Sabendo quem a fórmula da frequência é $1/T$, sendo T o período, neste caso T é o tempo máximo de delay, logo as frequências máximas de cada implementação do somador é:

IMPLEMENTAÇÃO	FREQUENCIA MÁXIMA (MHz)
ESQUEMÁTICO	90.744
VERILOG	92.302
VERILOG COMPORTAMENTAL	94.179

Tabela 4 – Frequência Máxima de clock para cada implementação de somador 4bits

3) Análise comparativa das sínteses do Somador de 32 bits

- (a) Compare os números de Elementos Lógicos necessários e os maiores tempos de atraso de propagação (tpd) para cada implementação.

IMPLEMENTAÇÃO	Nº DE ELEMENTOS LÓGICOS	TEMPO MÁXIMO DE DELAY (ns)
ESQUEMÁTICO	77	43.950
VERILOG	78	37.636
VERILOG COMPORTAMENTAL	33	14.102

Tabela 5 – Comparação entre as implementações para somador de 32bits

- (b) Implemente as correspondentes versões síncronas. Quais as maiores frequências de clock utilizáveis?

IMPLEMENTAÇÃO	FREQUENCIA MÁXIMA (MHz)
ESQUEMÁTICO	22.753
VERILOG	26.570
VERILOG COMPORTAMENTAL	70.911

Tabela 6 – Frequência Máxima de clock para cada implementação de somador 32bits