INSTITUTO POLITÉCNICO DE BEJA

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Sistemas Digitais

Trabalho de Grupo #2

Elaborado por:

João Faria, n.º 16145
Ricardo Madeira, n.º 16147
Luís Pratas, n.º 15970

Docente:

Miguel Tavares

13 de Janeiro de 2018

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

O presente relatório encontra-se organizado na seguinte forma: na secção 2.1 descreve-se a implementação da ALU, descrevendo em primeiro lugar o desenvolvimento dos seus componentes. A saber um bloco de adição de 5 bits com carry out e sinal de overflow, um bloco de subtracção de 5 bits com borrow out e sinal de overflow, um bloco de operação AND de 5 bits e por fim um bloco de operação xOR de 5 bits. Como extra foi também pedido o desenvolvimento de um bloco carry look-ahead, que permite acelerar as operações de soma e subtracção. Foi ainda pedido que criássemos uma ALU de 10 bits através da correcta ligação de duas ALUs de 5 bits.

Na secção 2.2 são exibidas figuras que representam os circuitos criados, bem como exemplos do correcto funcionamento das macros dos elementos.

Na secção 3 descreve-se o a implementação de um banco de registos, que contêm 2 x 8 registos de 5 bits cada um. O desenvolvimento deste bloco recorre a um decoder 3:8, flip-flop's D e MUX's 8:1. Cada um destes elementos foi criado para o efeito. Como extra foi pedido que criássemos um banco de registos de 10 bits através da correcta ligação de dois bancos de registos de 5 bits.

Na secção 3.2 são exibidas figuras que representam os circuitos criados, bem como exemplos do correcto funcionamento dos bancos de registos.

Na secção 4 são exibidas figuras dos circuitos finais que são compostos pela ALU de 5 bits ligada ao banco de registos de 5 bits e a ALU de 10 bits ligada ao banco de registos de 10 bits.

Finalmente na secção 5 apresentamos as nossas conclusões e opiniões sobre o trabalho pedido e comentamos o desenvolvimento do mesmo.

Na secção 6 está presente a bibliografia utilizada para este trabalho.

Em anexo a este relatório encontram-se os ficheiros correspondentes aos diversos circuitos, circuitos que construíram as macros, macros e circuitos implementados com macros implementados no DigitalWorks:

- **001. Adder_1bit_macro.dwm**
- **002. Subtrator_1bit_macro.dwm**
- **003. Adder_5bits_macro.dwm**
- **004. Subtrator_5bits_macro.dwm**
- **005. AND_5bits_macro.dwm**
- **006. xOR_5bits_macro.dwm**
- **007. MUX_4-1_macro.dwm**
- **008. ALU_5bits_macro.dwm**
- **009. Register_bank_macro.dwm**
- **011. Adder_1bit_circuit.dwm**
- **012. Subtrator_1bit_circuit.dwm**
- **013. Adder_5bits_circuit.dwm**
- **014. Subtrator_5bits_circuit.dwm**
- **015. AND_5bits_circuit.dwm**
- **016. xOR_5bits_circuit.dwm**
- **017. MUX_4-1_circuit.dwm**
- **018. ALU_5bits_circuito.dwm**
- **019. Register_bank_circuit.dwm**
- **101. Adder_1bit_circuit_with_macro.dwm**
- **102. Subtrator_1bit_circuit_with_macro.dwm**

- **103. Adder_5bits_circuit_with_macro.dwm**
- **104. Subtrator_5bits_circuit_with_macro.dwm**
- **105. AND_5bits_circuit_with_macro.dwm**
- **106. xOR_5bits_circuit_with_macro.dwm**
- **107. MUX_4-1_circuit_with_macro.dwm**
- **108. ALU_5bits_circuit_with_macro.dwm**
- **109. Register_bank_circuit_with_macro.dwm**
- **120. Register_bank_with_ALU_circuit.dwm**
- **201. Carry_look-head_macro.dwm**
- **202. ALU_10bits_macro.dwm**
- **203. Register_bank_10bits_macro.dwm**
- **211. Carry_look-head_circuit.dwm**
- **212. ALU_10bits_circuit.dwm**
- **213. Register_bank_10bits_circuit.dwm**
- **221. Carry_look-head_circuit_with_macro.dwm**
- **222. ALU_10bits_circuit_with_macro.dwm**
- **223. Register_bank_10bits_circuit_with_macro.dwm**
- **224. Register_bank_10bits_with_ALU_10bits_circuit.dwm**

2. Implementação da ALU

2.1. Tabelas de verdade e Mapas de Karnaugh

2.1.1. Somador de 1 bit com carry out

Para desenvolver o somador de 1 bit foi necessário escrever a tabela de verdades para a operação de soma, tendo em conta que existe um bit de carry in.

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela 1 – Tabela de verdades para somador de 1 bits com carry out

Escrevemos então os mapas de Karnaugh para os bits de soma e de carry out, simplificando as formulas o máximo possível.

A \ B, C_{in}	0, 0	0, 1	1, 1	1, 0
0	0	1	0	1
1	1	0	1	0

Tabela 2 - Mapa de Karnaugh para o bit de soma do somador

Simplificando a expressão que se obtêm do mapa de Karnaugh conseguimos uma expressão muito mais simplificada.

$$Sum = A\bar{B}\overline{C_{in}} + \bar{A}\bar{B}C_{in} + ABC_{in} + \bar{A}B\overline{C_{in}} \Leftrightarrow$$
$$\Leftrightarrow Sum = \overline{C_{in}}(A\bar{B} + \bar{A}B) + C_{in}(AB + \bar{A}\bar{B}) \Leftrightarrow$$
$$\Leftrightarrow Sum = A \oplus B \oplus C_{in}$$

Procedemos do mesmo modo para obter uma expressão simplificada para o bit de carry out.

A \ B, C_{in}	0, 0	0, 1	1, 1	1, 0
0	0	0	1	0
1	0	1	1	1

Tabela 3 - Mapa de Karnaugh para o bit de carry out do somador

$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\overline{C_{in}} + ABC_{in} \Leftrightarrow$$
$$\Leftrightarrow C_{out} = C_{in}(A\bar{B} + \bar{A}B) + AB(C_{in} + \overline{C_{in}}) \Leftrightarrow$$
$$\Leftrightarrow C_{out} = C_{in}(A \oplus B) + AB$$

2.1.1. Determinação de OVERFLOW para a soma

Por definição o overflow acontece quando a excedemos a capacidade de representação do número de bits que estamos a utilizar. Assim as condições para que haja overflow numa adição binária é que a soma de dois números positivos resulte num número negativo ou que a soma de dois números negativos resulte num número positivo. Colocando estes casos sobre a forma de tabela de verdades, com a adição dos bits de carry in e de carry out para ajudar a analise.

A	B	C_{in}	Sum	C_{out}	OVERFLOW
0	0	0	0	0	0
0	0	1	1	0	1
0	1	1	0	1	0
0	1	0	1	0	0
1	0	1	0	1	0
1	0	0	1	0	0
1	1	0	0	1	1
1	1	1	1	1	0

Tabela 4 - Tabela de verdades de overflow na adição

Se observarmos os bits de carry in e de carry out, conseguimos observar que apenas existe overflow quando os mesmos são diferentes, logo podemos escrever o seguinte mapa de Karnaugh.

C_{in} \ C_{out}	0	1
0	0	1
1	1	0

Tabela 5 - Mapa de verdades para para o overflow na adição

retirando imediatamente do mapa de Karnaugh podemos escreve a seguinte expressão.

$$Overflow = \overline{C_{in}}C_{out} + C_{in}\overline{C_{out}} \Leftrightarrow$$

$$\Leftrightarrow Overflow = C_{in} \oplus C_{out}$$

2.1.2. Subtractor de 1 bits com borrow out

De seguida escrevemos a tabela de verdade para a operação de subtracção, tendo em conta que existe um bit de borrow in.

A	B	B_{in}	Diff	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Tabela 6 - Tabela de verdades para subtractor de 1 bits com borrow out

Escrevemos então os mapas de Karnaugh para os bits de diferença e de borrow out, simplificando as formulas o máximo possível.

A \ B	0	0	1	1
A \ B_{in}	0	1	1	0
0	0	1	0	1
1	1	0	1	0

Tabela 7 - Mapa de Karnaugh para o bit de diferença do subtractor

Obtemos a seguinte expressão do mapa de Karnaugh que simplificamos.

$$Diff = A\bar{B}\overline{B_{in}} + \bar{A}\bar{B}B_{in} + AB B_{in} + \bar{A}B\overline{B_{in}} \Leftrightarrow$$
$$\Leftrightarrow Diff = \overline{B_{in}}(A\bar{B} + \bar{A}B) + B_{in}(AB + \bar{A}\bar{B}) \Leftrightarrow$$
$$\Leftrightarrow Diff = A \oplus B \oplus B_{in}$$

Procedemos do mesmo modo para obter uma expressão simplificada para o bit de borrow out.

A \ B	0	0	1	1
A \ B_{in}	0	1	1	0
0	0	1	1	1
1	0	0	1	0

Tabela 8 - Mapa de Karnaugh para o bit de borrow out do subtractor

$$B_{out} = \bar{A}\bar{B}B_{in} + \bar{A}BB_{in} + \bar{A}B\overline{B_{in}} + ABB_{in} \Leftrightarrow$$
$$\Leftrightarrow B_{out} = \bar{A}B(B_{in} + \overline{B_{in}}) + B_{in}(\bar{A}\bar{B} + AB) \Leftrightarrow$$
$$\Leftrightarrow B_{out} = \bar{A}B + B_{in}\overline{(A \oplus B)}$$

2.1.3. Determinação de OVERFLOW para a subtracção

As condições para que haja overflow numa subtração binária são que a subtracção de um número positivo com um número negativo resulte num número negativo ou que a subtração de um número negativo com um número positivo resulte num número positivo. Colocando estes casos sobre a forma de tabela de verdades, com a adição dos bits de borrow in e de borrow out para ajudar a analise.

A	B	B_{in}	Diff	B_{out}	OVERFLOW
0	0	0	0	0	0
0	0	1	1	1	0
0	1	1	0	1	0
0	1	0	1	1	1
1	0	1	0	0	1
1	0	0	1	0	0
1	1	0	0	0	0
1	1	1	1	1	0

Tabela 9 - Tabela de verdades de overflow na subtração

Se observarmos os bits de borrow in e de borrow out, conseguimos observar que apenas existe overflow quando os mesmos são diferentes, logo podemos escrever o seguinte mapa de Karnaugh.

B_{in} \ B_{out}	0	1
0	0	1
1	1	0

Tabela 10 - Mapa de Karnaugh para para o overflow na subtração

Retirando imediatamente do mapa de Karnaugh podemos escreve a seguinte expressão.

$$Overflow = \overline{B_{in}}B_{out} + B_{in}\overline{B_{out}} \Leftrightarrow$$
$$\Leftrightarrow Overflow = B_{in} \oplus B_{out}$$

2.1.4. Operação AND de 5 bits

Este bloco vai consistir um 5 portas AND ligadas a cada um dos pares de bits de entrada e fornecendo um bit de saída que equivale à operação AND entre esses bits. Sendo esta uma operação lógica não existe necessidade de um carry ou overflow. Assim a tabela de verdades vai ser igual à tabela de verdades da porta AND.

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 11 - Tabela de verdades para a operação AND

o mapa de Karnaugh para a mesma descreve o funcionamento da porta AND.

A \ B	0	1
0	0	0
1	0	1

Tabela 12 - Mapa de Karnaugh para a operação AND

Sem surpresas a expressão que se retira do mapa de Karnaugh é a seguinte.

$$AND = A \cdot B$$

2.1.5. Operação xOR de 5 bits

De igual modo este bloco vai consistir em 5 portas xOR ligadas a cada um dos pares de bits de entrada, fornecendo um bit de saída que equivale à operação xOR entre esses bits. Mais uma vez, sendo uma operação lógica não existem carrys ou overflow.

A tabela de verdades vai ser igual à tabela de verdades da porta xOR.

A	B	xOR
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 13 - Tabela de verdades para a operação xOR

O mapa de Karnaugh para a mesma descreve o funcionamento de uma porta xOR

A \ B	0	1
0	0	1
1	1	0

Tabela 14 - Mapa de Karnaugh para a operação xOR

A expressão que se retira do mapa de Karnaugh é a seguinte.

$$xOR = \bar{A}B + A\bar{B} \Leftrightarrow$$

$$\Leftrightarrow xOR = A \oplus B$$

2.1.6. Multiplexer de 4:1

O multiplexer é um circuito lógico combinatório desenhado para actuar com um interruptor que permite apenas o output de um sinal, de vários sinais de entrada. O multiplexer utilizado nesta ALU permite a entrada de 4 sinais, para controlar o output destes 4 sinais são necessários 2 bits que formarão um código que corresponde a cada uma das saídas. A seguinte tabela de verdades apresenta o funcionamento do multiplexer.

Input 0	Input 1	Input 2	Input 3	S1	S0	Output
1	X	X	X	0	0	1 (Input 0)
X	1	X	X	0	1	1 (Input 1)
X	X	1	X	1	0	1 (Input 2)
X	X	X	1	1	1	1 (Input 3)

Tabela 15 - Tabela de verdades para o multiplexer 4:1

Assim podemos escrever uma expressão simples que descreve o funcionamento do multiplexer.

$$Output = \overline{S_0}\overline{S_1}\,\text{Input } 0 + S_0\overline{S_1}\,\text{Input } 1 + \overline{S_0}S_1\,\text{Input } 2 + S_0S_1\,\text{Input } 3$$

2.1.7. ALU de 5 bits

Quando combinamos todos os blocos descritos anteriormente, podemos criar uma Unidade Lógica e Aritmética (ALU). A ALU é um circuito que realiza as principais operações aritméticas (soma, subtracção, multiplicação e divisão) e lógicas (AND, NOT, OR e xOR) para além de outras operações, como por exemplo deslocamentos de bits. No nosso caso a nossa ALU é bastante mais simples e apenas realiza soma, subtracção, AND e xOR. No entanto foi desenvolvida com o cuidado de respeitar o complemento para dois.

Para controlar a operação que está a realizar o output, foram utilizados MUX's 4:1. EDstes blocos permitem que todos os blocos de aritmética e lógica recebam os bits de imput, mas apenas um realize o output, através de dois bits que permitem um total de 4 combinações, ou seja o número de blocos de aritmética e lógica que estamos a utilizar.

Assim cada ALU de 5 bits irá receber 2 x 5 bits de input, 1 bit de carry in (que permite a modularidade do bloco) e irá fazer output de 5 bits de resultado, 1 bit de Carry out (que mais uma vez permite modularidade) e um sinal de overflow para assinalar quando a capacidade de representação é excedida.

Finalmente convêm salientar que devido à utilização do complemento para dois, os nossos cinco bits representam os números decimais de -16 a 15, criando assim frequentemente casos de overflow.

2.1.8. Carry look-ahead

Um bloco de carry look-ahead é utilizado para aumentar a velocidade a que a ALU realiza os cálculos aritméticos, uma vez que reduz o tempo necessário para determinar os bits de carry (soma) e de borrow (subtracção).

Para tal o bloco vais calcular (para cada posição) se a soma parcial desta posição vai criar um bit de carry out, caso lhe seja fornecido um bit de carry in. Com estes resultados calculados, o bloco vai deduzir se vai haver propagação de bits. Caso haja propagação, esta vai continuar a ser calculada até ao ultimo bit de carry out do bloco.

Assim para cada posição vai ser calculado o bit de geração (G) e o bit de propagação (P).

$$G = A \cdot B$$

$$P = A \oplus B$$

No nosso caso vamos ter os bits G_4 a G_0 e P_4 a P_0. No entanto com um pouco de pensamento lógico, facilmente constatamos que o bit de carry in da próxima posição é calculado pela operação OR entre G e (P AND C_i), uma vez que esse bit apenas pode existir se for gerado ou propagado pela posição anterior à que utiliza o carry in.

$$C_{i+1} = G + (P \cdot C_i)$$

Realizando os cálculos para os nossos cinco bits iremos ter que o nosso C_{out} pode ser calculado por:

$$C_{out} = G_4 + G_3 \cdot P_4 + G_2 \cdot P_4 \cdot P_3 + G_1 \cdot P_4 \cdot P_3 \cdot P_2 + G_0 \cdot P_4 \cdot P_3 \cdot P_2 \cdot P_1 + C_{in} \cdot P_4 \cdot P_3 \cdot P_2 \cdot P_1 \cdot P_0$$

Para calcular a soma de cada par dos cinco bits de input apenas temos que realizar a operação xOR entre o bit de propagação e o carry in e da soma parcial anterior. Assim no nosso caso, a soma de cada posição é dada pela expressão:

$$S_i = P_i \oplus C_{in\,i}$$

2.1.9. ALU de 10 bits

Para criar a ALU de 10 bits apenas tivemos que ter em atenção que uma ALU irá realizar os cálculos com os cinco bits menos significativos, fazer o output deste resultado para os cinco bits menos significativos do output e passar o bit de carry out para o bit de carry in da segunda ALU, que irá efectuar os cálculos correspondentes aos cinco bits mais significativos e passar os resultados para os cinco bits mais significativos do output. O overflow apenas ocorre na segunda ALU pelo que não é necessário utilizar o bit de sinalização da primeira ALU. O bit de carry out, do bloco inteiro, será o bit de carry out da segunda ALU, uma vez que é este que o final da operação com os 10 bits.

Por fim é importante esclarecer que os bits S_0 e S_1 devem ser comuns às duas ALUs para garantir que ambas estão a realizar a mesma operação.

2.2. Circuitos Realizados no DigitalWorks

2.2.1. Somador de 1 bit com sinal de carry out

De seguida apresentamos a figura 1 que apresenta o circuito criado e as figuras 2 e 3 que apresentam dois exemplos de funcionamento da macro criada.

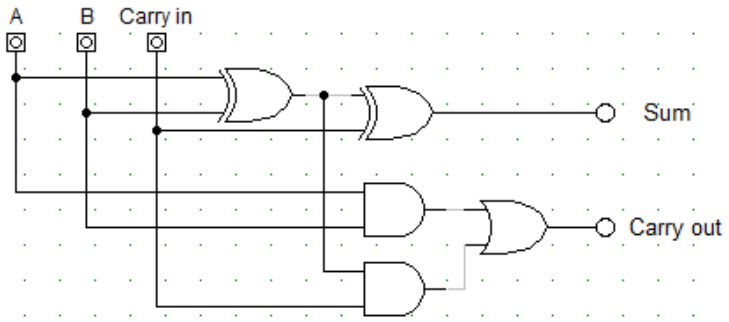

Figura 1 - Circuito combinatório que forma a macro do adder de 1 bit.

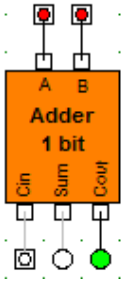

Figura 2 - Exemplo de funcionamento do adder de 1 bit para A = 1 e B = 1

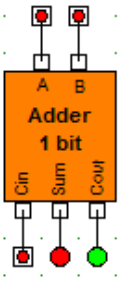

Figura 3 - exemplo de funcionamento do adder de 1 bit para A = 1 e B = 1 com carry in

2.2.2. Somador de 5 bits com sinal de overflow

De seguida apresentamos a figura 4 que apresenta o circuito criado e as figuras 5 e 6 que apresentam dois exemplos de funcionamento da macro criada.

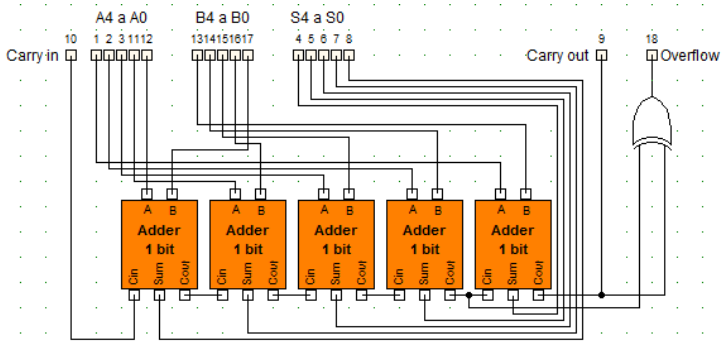

Figura 4 - Circuito combinatório que forma a macro do adder de 5 bits.

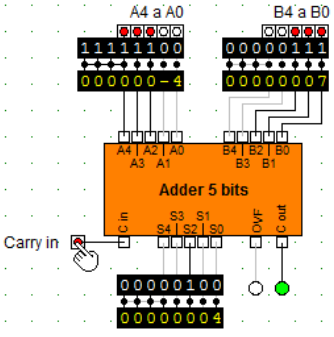

Figura 5 - Exemplo de funcionamento do adder de 5 bits para A = -4 e B= 7.

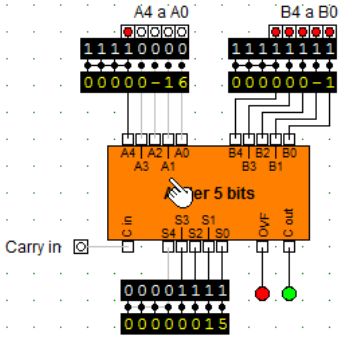

Figura 6 - Exemplo de funcionamento do adder de 5 bits para A =-16 e B= -1.

2.2.3. Subtractor de 1 bit com sinal de borrow out

De seguida apresentamos a figura 7 que apresenta o circuito criado e as figuras 8 e 9 que apresentam dois exemplos de funcionamento da macro criada.

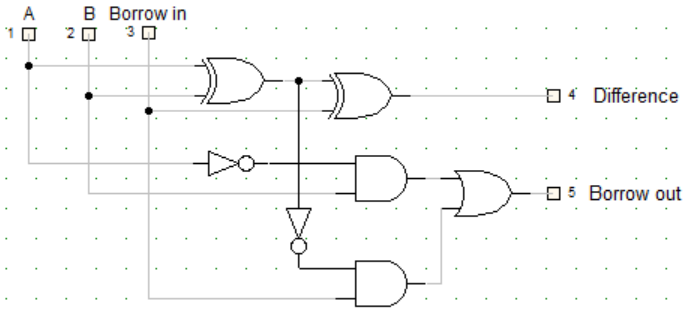

Figura 7 - Circuito combinatório que forma a macro do subtractor de 1 bit.

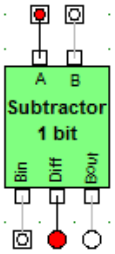

Figura 8 - Exemplo de funcionamento do subtractor de 1 bit para A = 1 e B = 0

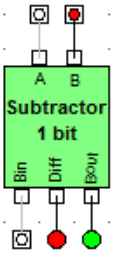

Figura 9 - exemplo de funcionamento do subtractor de 1 bit para A = 0 e B = 1

2.2.4. Subtractor de 5 bits com sinal de overflow

De seguida apresentamos a figura 10 que apresenta o circuito criado e as figuras 11 e 12 que apresentam dois exemplos de funcionamento da macro criada.

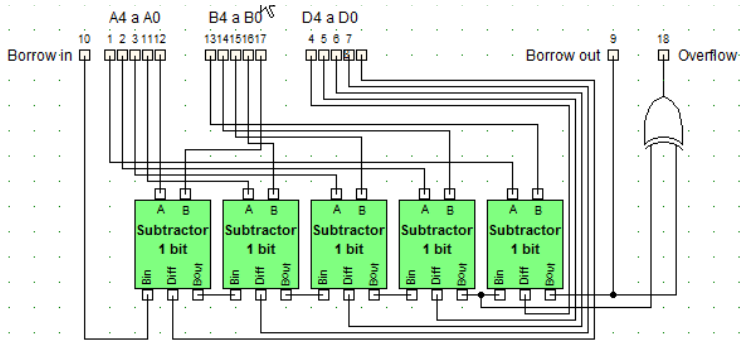

Figura 10 - Circuito combinatório que forma a macro do subtractor de 5 bits.

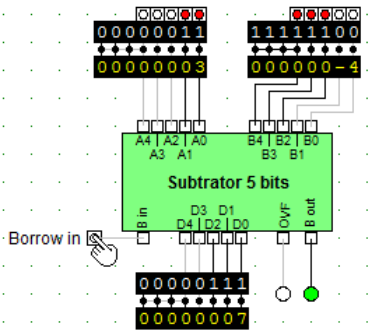

Figura 11 - Exemplo de funcionamento do subtractor de 5 bits para A = 3 e B= -4.

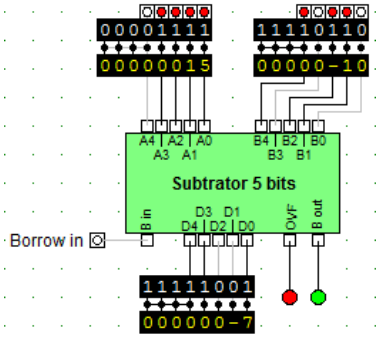

Figura 12 - Exemplo de funcionamento do subtractor de 5 bits para A = 15 e B= -10.

2.2.5. Operação AND de 5 bits

De seguida apresentamos a figura 13 que apresenta o circuito criado e as figuras 14 e 15 que apresentam dois exemplos de funcionamento da macro criada.

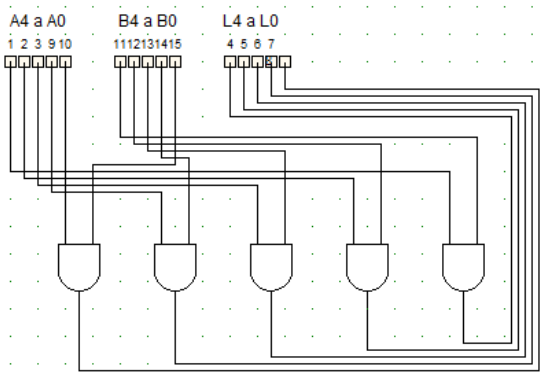

Figura 13 - Circuito combinatório que forma a macro da operação AND de 5 bits.

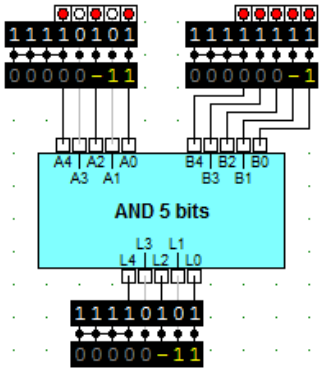

Figura 14 - Exemplo de funcionamento do AND de 5 bits para A = -11 e B= -1.

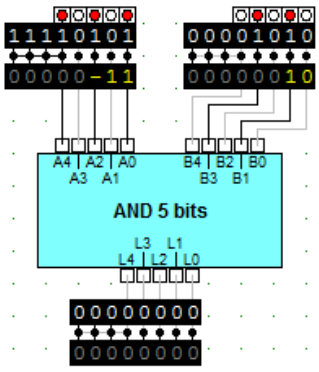

Figura 15 - Exemplo de funcionamento do AND de 5 bits para A = -11 e B= 10.

2.2.6. Operação xOR de 5 bits

De seguida apresentamos a figura 16 que apresenta o circuito criado e as figuras 17 e 18 que apresentam dois exemplos de funcionamento da macro criada.

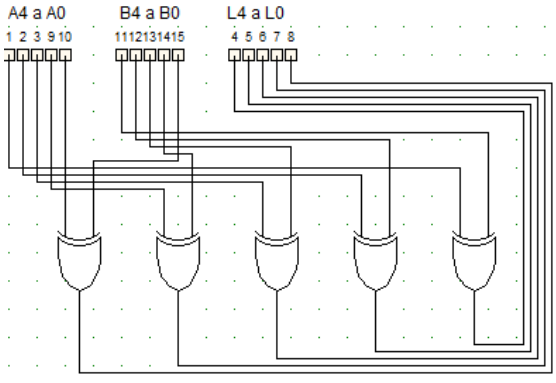

Figura 16 - Circuito combinatório que forma a macro da operação xOR de 5 bits.

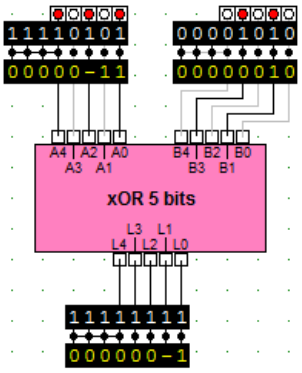

Figura 17 - Exemplo de funcionamento do xOR de 5 bits para A = -11 e B= 10.

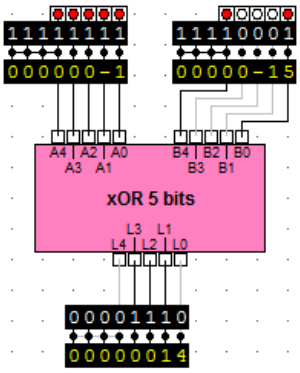

Figura 18 - Exemplo de funcionamento do xOR de 5 bits para A = -1 e B= 15.

2.2.7. ALU de 5 bits

De seguida apresentamos a figura 19 que apresenta o circuito criado e as figuras 20 e 21 que apresentam dois exemplos de funcionamento da macro criada.

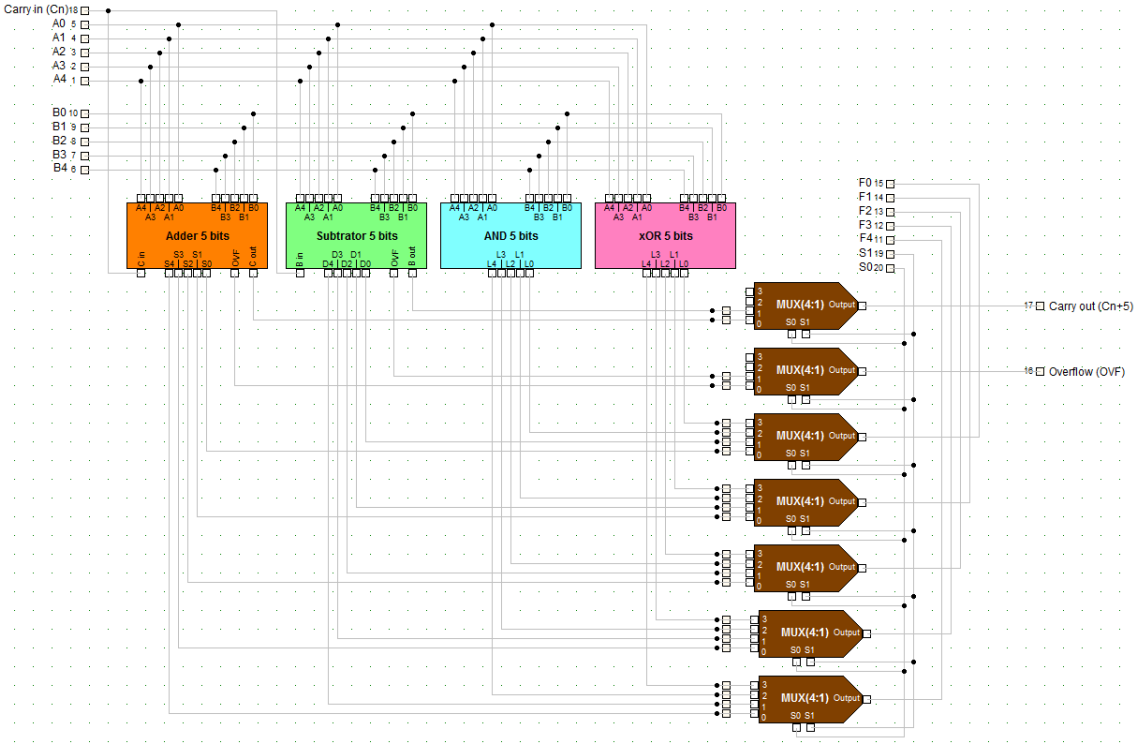

Figura 19 - Circuito combinatório que forma a macro da ALU de 5 bits.

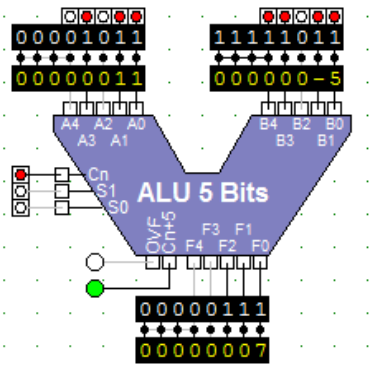

Figura 20 - Exemplo de funcionamento da ALU de 5 bits para A = 11 e B= -5 e operação de soma.

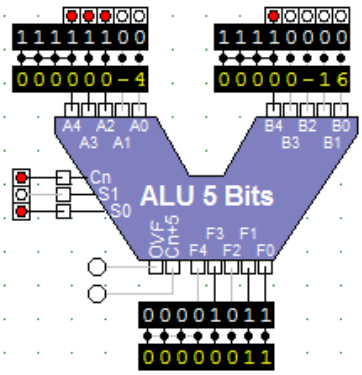

Figura 21 - Exemplo de funcionamento da ALU de 5 bits para A = -4 e B= -16 e operação de subtracção.

2.2.8. Carry look-ahead

De seguida apresentamos a figura 22 que apresenta o circuito criado e as figuras 23 e 24 que apresentam dois exemplos de funcionamento da macro criada.

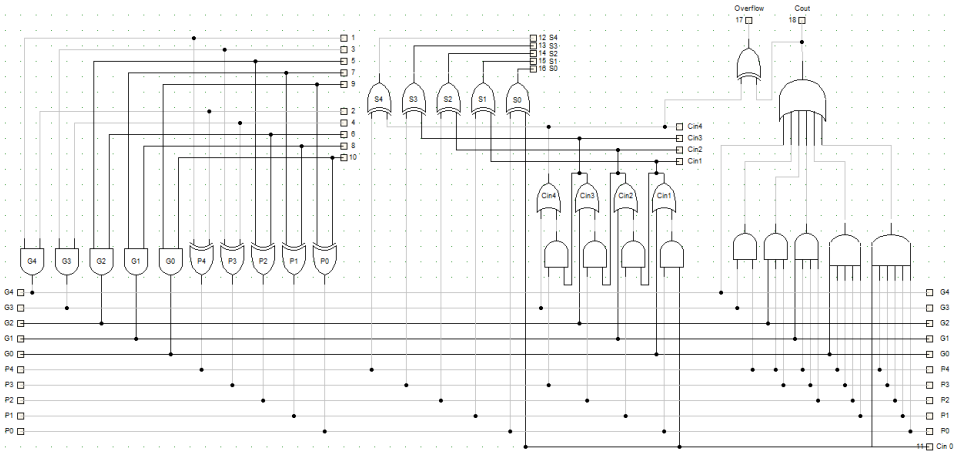

Figura 22 - Circuito combinatório que forma a macro do adder de 5 bits com carry look-ahead.

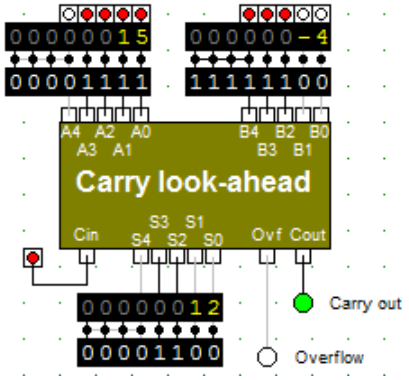

Figura 23 - Exemplo de funcionamento do adder de 5 bits com carry look-ahead para A = 15 e B= -4.

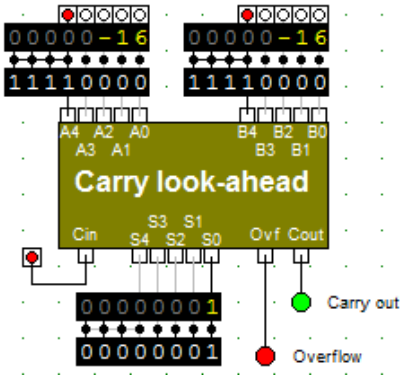

Figura 24 - Exemplo de funcionamento do adder de 5 bits com carry look-ahead para A = 15 e B= -16.

2.2.9. ALU de 10 bits

De seguida apresentamos a figura 25 que apresenta o circuito criado e as figuras 26 e 27 que apresentam dois exemplos de funcionamento da macro criada.

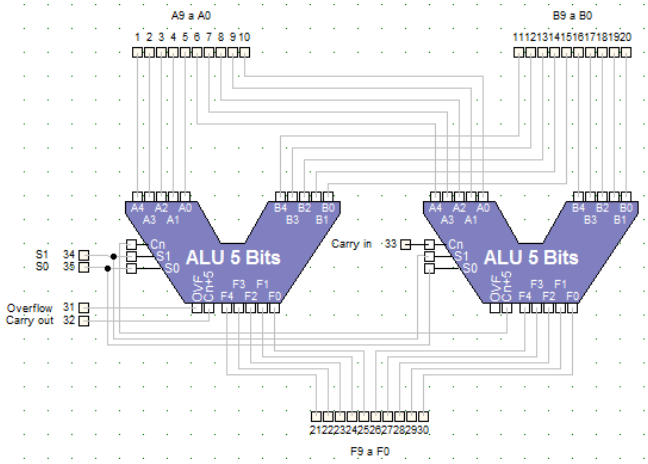

Figura 25 - Circuito combinatório que forma a macro da ALU de 10 bits.

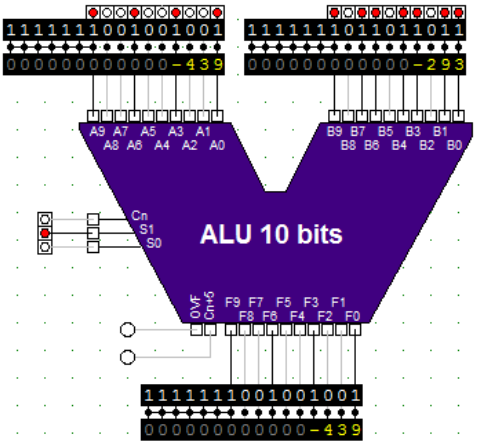

Figura 26 - Exemplo de funcionamento da ALU de 10 bits para A = -439 e B= -293 e operação AND.

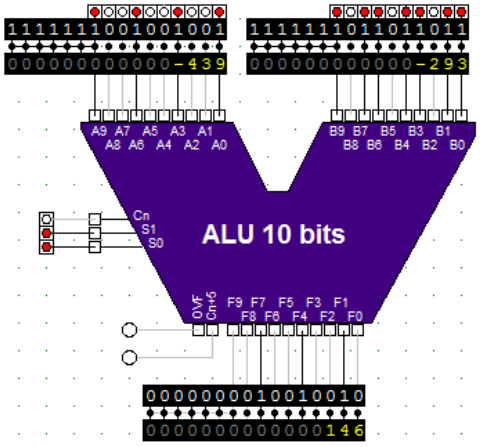

Figura 27 - Exemplo de funcionamento da ALU de 10 bits para A = -439 e B= -293 e operação xOR.

3. Implementação do banco de registos

3.1. Descodificador

O descodificador é um circuito combinatório que converte um código binário de entrada de N bits (no nosso caso 3 bits) na activação de uma linha de saída, quando um bit de escrita é activo. Assim por exemplo no nosso descodificador, que possuía 3 bits de entrada e 8 linhas de saída, ao fornecermos o código binário 011 (número decimal 3) e activarmos o bit de escrita, este iria activar a linha nº3 com o valor 1. No nosso caso a activação desta linha vai ter o efeito de activar o bit de escrita de um flip-flop D.

O nosso decoder funciona com base em portas AND. Cada porta vai representar uma linha de saída, em que recebe os três bits do código numérico. Sendo que todos os bits que são representados por 0, são negados antes de entrar na porta AND. Assim quando seleccionamos a linha 3 (código binário 011) o que vai entrar na porta AND será a negação do 0, 1 e 1. Quando activamos o bit de escrita, que também entra na porta AND, todas os bits de entrada passam a 1, fazendo que a saída dessa porta seja 1 e consequentemente a activação dessa mesma linha.

Nas figuras 28 e 29 apresentamos o nosso circuito e a macro que criamos para facilitar a montagem dos circuitos.

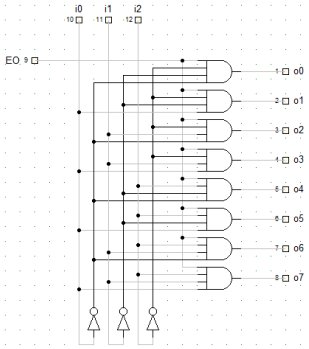

Figura 28 - Circuito combinatório do descodificador.

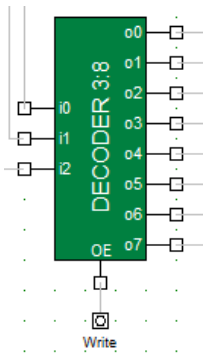

Figura 29 - Macro do descodificador.

3.2. Banco de registos de 5 bits

O nosso banco possui as funções de escrita e de leitura. para realizar a escrita de um valor de 5 bits é necessário indicar qual o endereço em que o mesmo será guardado, através de um código de 3 bits (que representará os endereços 0 a 7), indicar qual o valor de 5 bits a ser escrito e finalmente activar o bit de escrita. O código de 3 bits vai activar uma linha de um decoder 3:8. A linha que foi activada vai ligar ao bit de escrita de uma macro que é simplesmente 5 flip-flop's D agrupados, tal como pode ser observado nas figuras 30 e 31.

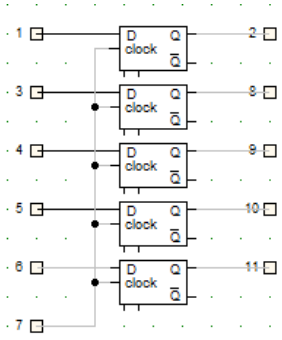

Figura 30 - Circuito da macro de 5 Flip-Flop's D

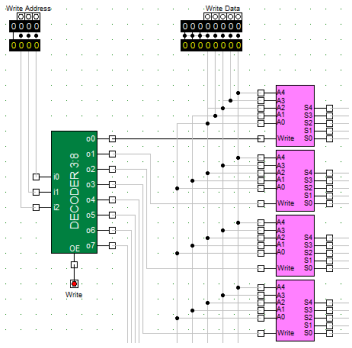

Figura 31 - Detalhe da linha de escrita do endereço 0 activada

Assim, com pode também ser observado na figura ??, os bits que se pretendem escrever são fornecidos pelas ligações A0 a A4, sendo que cada uma destas ligações corresponde à ligação D do filp-flop. Assim o estado que corresponde ao bit fica guardado no flip-flop. Para cada valor de 5 bits são necessários 5 flip-flop's D, que "guardam" um registo. Assim para acomodar os 8 registos necessitamos de 5x8 flip-flop's D, ou seja 40. Para tornar o esquema mais simples de interpretar optamos por criar macros dos conjuntos de flip-Flop's.

Neste momento já compreendemos como se conseguem "guardar" os valores. A segunda função do banco é ser capaz de ler esses registos guardados. Para tal é necessário seleccionarmos qual o output que vai ser apresentado. É importante perceber que no nosso circuito todos os flip flop's estão a fazer output, ou seja o nosso problema não é conseguir apresentar um output, mas sim apresentar apenas um output. Para tal utilizamos multiplexers 8:1, que nos vão permitir seleccionar quais os outputs dos flip-flop's a apresentar.

Os multiplexers estão divididos em dois conjuntos de 5, pois são necessários 5 para apresentar os cinco bits do valor "guardado". Vamos considerar apenas um conjunto de cinco, uma vez que o os dois conjuntos funcionam de maneira semelhante, variando apenas o input que lhes é fornecido.

Cada multiplexer recebe um código de 3 bits (0 a 7) que representa o endereço a ser lido, e isto fará que, dos oito inputs recebidos, seja feito output daquele correspondente ao endereço que pretendemos ler. Cada um dos multiplexers recebe apenas um bit dos oito registos guardados. Ou seja, o primeiro multiplexer recebe os bits mais significativos dos oitos registos, o segundo multiplexer recebe os seguintes bits mais significativos e assim por diante até que o quinto multiplexer recebe os bits menos significativos. Podemos observar estas ligações na figura 32, onde é possível observar 2 macros de flip-flops ligadas ao primeiro conjunto de 5 multiplexers.

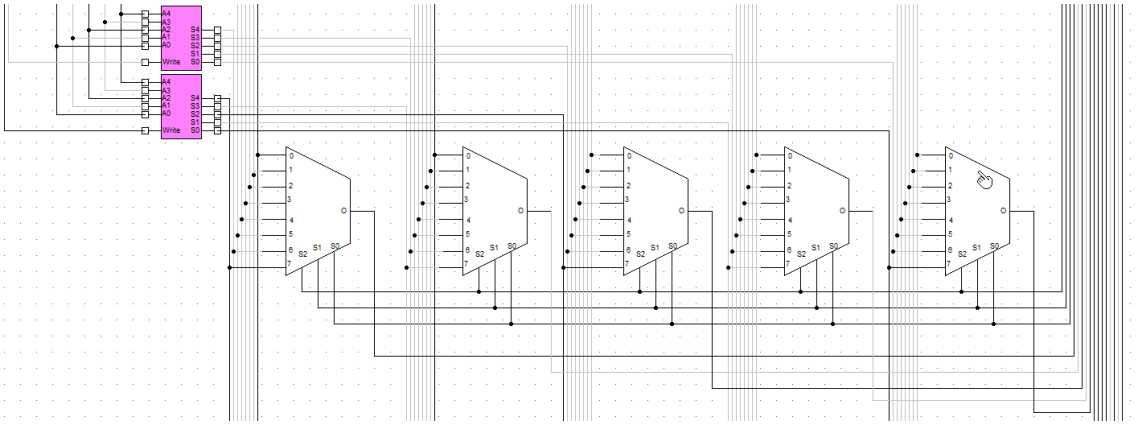

Figura 32 - Detalhe das ligações das macros dos flip-flops aos multiplexers.

Como cada multiplexer recebe o mesmo código de 3 bits do endereço, todos vão apresentar apenas os bits que correspondem a esse endereço. Assim é possível limitar o output dos nossos 40 flip-flops a apenas 5 que correspondem a um endereço. Na figura 33 podemos ver o detalhe do output dos multiplexers, que estão a o endereço 111 (decimal 7) e a fazer um output de 10101 (-11 em complemento de 2).

Figura 33 - Detalhe da leitura dos outputs dos multiplexers.

Podemos ainda verificar na figura 33 que está presente um segundo conjunto de interruptores e displays que controlam o segundo conjunto de 5 multiplexers e que trabalham independentemente. Neste exemplo esse conjunto apresenta o endereço 000 (decimal 0) e fazem um output de 00011 (3 em complemento para 2).

Por fim criamos uma macro, que simplificou a implementação de um circuito tão grande. A mesma pode ser observada na figura 34.

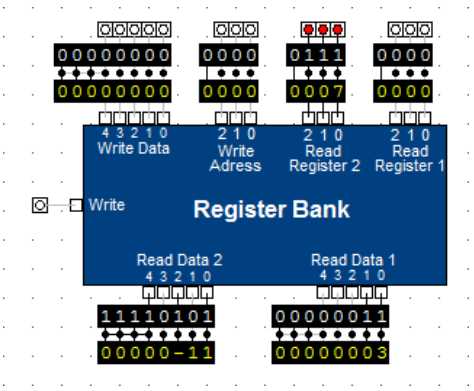

Figura 34 - Macro do banco de registos de 5 bits.

3.3. Banco de registos de 10 bits

Na implementação do banco de registos de 10 bits foi apenas necessário ter em atenção que quando ligamos 2 bancos de registos de 5 bits, cada um deles deve receber todos os sinais de input de igual modo (write, write address, etc…), excepto os 10 bits que vão ser guardados que devem ser repartidos em blocos de 5 bits, sendo que um bloco representa os bits mais significativos e o bloco seguinte os bits menos significativos. A figura 35 apresenta o circuito que utilizamos para a implementação deste banco de registos.

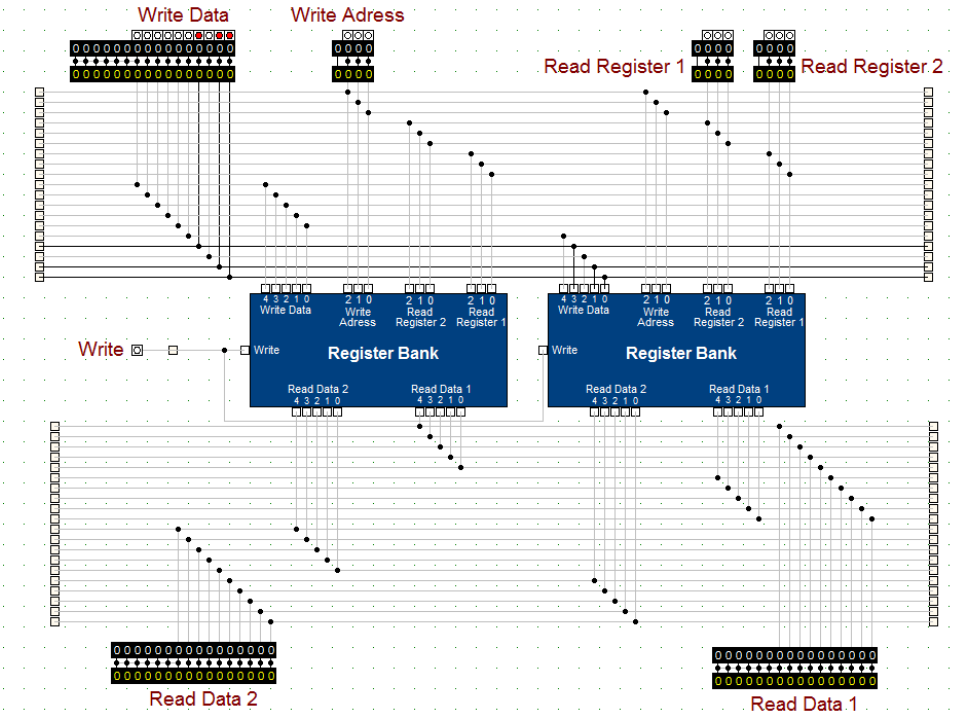

Figura 35 - Circuito para a implementação do banco de registos de 10 bits

Finalmente criamos uma macro que corresponde a este circuito, tal como pode ser observada na figura 36.

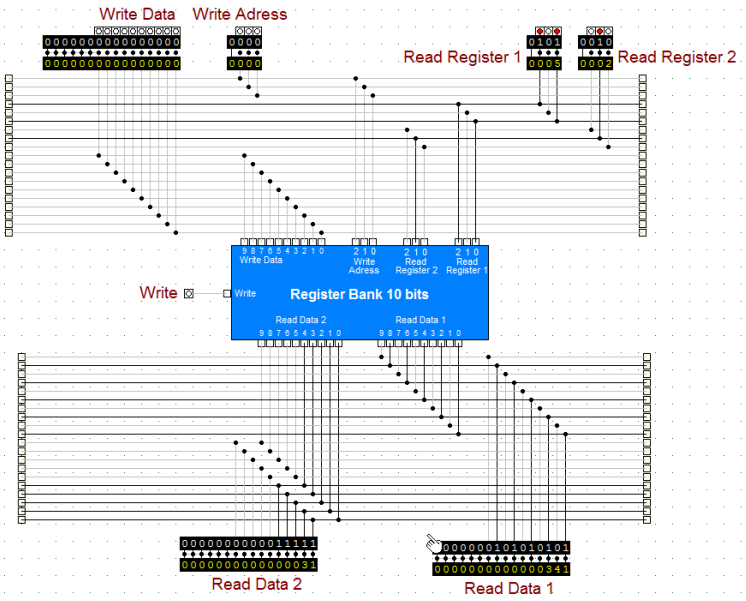

Figura 36 - Implementação da macro de 10 bits

4. Implementação do banco de registos com ALU

4.1. Implementação do banco de registos com ALU de 5 bits

A implementação deste circuito final baseou-se simplesmente na correcta ligação entre as macros, tal como é apresentado na figura 37.

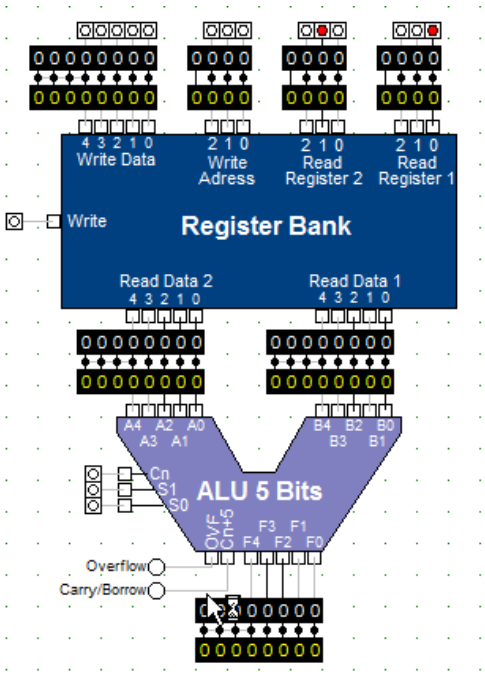

Figura 37 - Implementação do circuito de banco de registos + ALU para 5bits

4.2. Implementação do banco de registos com ALU de 10 bits

Tal como o circuito anterior, também neste apenas foi necessário fazer as ligações correctas entre as macros, tal como é apresentado na figura 38.

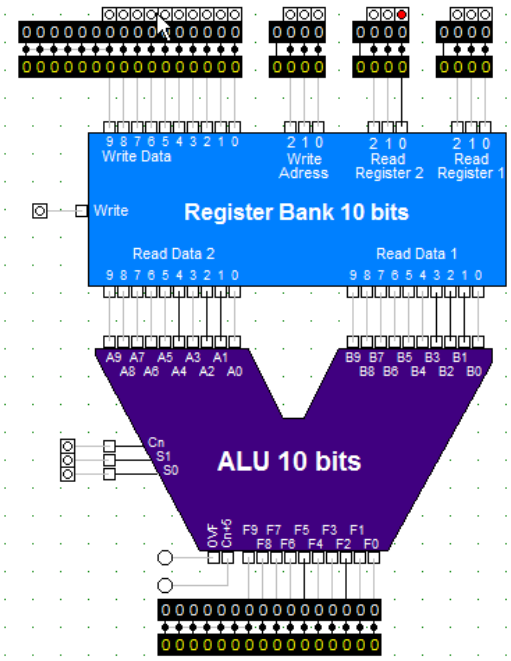

Figura 38 - Implementação do circuito de banco de registos + ALU para 10 bits

5. Conclusões

Tal como já tínhamos referido no trabalho anterior, colocamo-nos numa posição diferente aos outros alunos na disciplina, uma vez que já assistimos as aulas de Arquitectura de computadores e sabemos que estes conceitos serão importantes no semestre seguinte. Assim muitas das nossas considerações serão realizadas também com a luz dos conhecimentos e dificuldades que nos foram apresentadas em Arquitectura de Computadores.

O primeiro aspecto que gostaríamos de salientar nesta conclusão é que nunca compreendemos verdadeiramente a complexidade de uma ALU. A projecção e implementação desta ALU (que consideramos simples pois apenas realiza 4 operações simples e com um número muito pequeno de bits) tornou-se em certas partes problemática, uma vez que era necessária uma implementação do simples para o complexo. Numa primeira tentativa não optamos por essa abordagem, o que provocou um aumento na complexidade do circuito que não conseguimos administrar. Optamos assim por reiniciar o trabalho com uma filosofia de blocos criando todos os pequenos blocos das operações, projectando depois em cima destes, até atingirmos todas as operações pedidas por o nosso enunciado.

O segundo aspecto prende-se com a complexidade técnica do banco de registos. Não sendo um circuito particularmente complexo é no entanto bastante extenso e difícil de gerir em termos de ligações. No entanto cremos que realizamos um bom trabalho e que fomos capazes de apresentar um circuito bastante limpo e de fácil leitura.

Tal como no trabalho anterior gostaríamos de salientar as dificuldades em utilizar o Digital Works, uma vez que tal como nos foi informado no inicio da disciplina é um software antiquado e que não possui opções de roolback nem de auto-saving. A ausência destas opções pode causar verdadeiras perdas catastróficas no trabalho realizado se não houver um bom planeamento e boas práticas de utilização do software.

Outra complexidade sentida a nível do software foi a criação de macros com dimensões, ou ligações, suficientes para que os circuitos pudessem ser limpos e de fácil leitura. Mais do que uma vez as ligações entre duas macros não estavam alinhadas entre si, obrigando a que não fosse possível a ligação com uma simples recta, mas necessário um caminho sinuoso que tornava o circuito confuso. Por esta razão refizemos alguns circuitos mais do que uma vez, apenas conseguindo atingir a exactidão apresentada quando optamos por normalizar as distâncias entre as tags das macros criadas.

Foi relativamente difícil interligar o trabalho realizado por os três, uma vez que é muito complicado conseguir um “estilo” de desenho de circuito no Digital Works, mas pensamos que o conseguimos.

Por fim gostaríamos de salientar que a excelente execução que realizamos dependeu em grande parte do facto de termos despendido muito tempo para a realização do mesmo, termos tido a vontade de recomeçar a fazer certas partes do trabalho desde zero quando encontrávamos dificuldades que podiam trazer problemas mais tarde e ainda de termos iniciado o planeamento e execução muito cedo dentro da nossa “janela temporal” para realização do mesmo.

É ainda de referir que o enunciado deste trabalho era bastante claro e estruturado, facilitando aos alunos a compreensão do que era pedido. Esta pode parecer uma conclusão muito

simplista, no entanto no nosso ponto de vista de alunos do segundo ano já obtivemos enunciados e propostas de trabalhos muito confusos que tornaram o nosso trabalho muito difícil. Achamos assim que também devemos agradecer o tempo despendido na criação do enunciado, pois foi graças a ele que conseguimos um percurso para a nossa execução do mesmo. Agradecemos também ao professor Tavares pela ajuda e esclarecimentos prestados, sem os quais não teria sido possível a realização deste trabalho

Concluímos, pois, que este foi um trabalho que nos deu gozo realizar pois estava dentro dos nossos conhecimentos e capacidades, demonstrando-se também algo desafiante. E que contribuiu para alargarmos os nossos conhecimentos em relação ao funcionamento dos sistemas digitais.

6. Bibliografia

Binary Adder Half and Full Adder. (18 de 12 de 2017). Obtido de www.electrical4u.com: https://www.electrical4u.com/binary-adder-half-and-full-adder/

Binary Substractor. (18 de 12 de 2017). Obtido de www.electrical4u.com: https://www.electrical4u.com/binary-substractor/

Overflow Detection . (22 de 12 de 2017). Obtido de CIS-77 Home: http://www.c-jump.com/CIS77/CPU/Overflow/lecture.html