TÉCNICO LISBOA

Arquitetura de Computadores

2º Semestre 2016/2017

Unidade de Processamento

António Rouxinol Fragoso, nº 79116
Diogo Moura, nº86976

Primeira Parte - "Análise e caraterização da Unidade de Processamento"

1)

a) A Functional Unit apresentada no ficheiro VHDL do Vivado é constituída por 3 blocos essenciais que executam várias operações sobre as entradas A(15:0) e B(15:0) dependendo dos valores da variável de seleção FSUF(3:0).

Um dos blocos da Unidade Funcional é o bloco de deslocamento, convencionalmente chamado de Shifter. Este bloco tem como função receber uma entrada B(15:0) e realizar sobre ela várias operações como é o caso do Shift Left, deslocamento do valor 1 bit para a esquerda, permitindo realizar a multiplicação por 2, o Shift Right Logic, o Shift-Right Arithmetic, o Rotate Left 1 bit e, por fim, o Rotate Right 1 bit.

Do Shifter 2 saem flags que podem denunciar um incorreto funcionamento do bloco. Essas flags são: Carry-Out e Overflow. Para além das flags também sai deste bloco a saída D(15:0), agora alterada.

Um outro bloco pertencente à Functional Unit é o bloco aritmético, ou Arithmetic Unit, que engloba todas as operações de soma e de subtração. Neste bloco entram novamente A(15:0), B(15:0) e a entrada de seleção FSUF(1:0). Dependendo se a entrada B se encontrar ou não em complemento para 2, e se o Carry-in for 1 ou 0, as operações realizadas podem mudar, dado que as funções da entrada B e do *Carry in* do primeiro somador completo são:

```
Bn <= (B(15) and not FS(1)) or (not B(15) and FS(1));
C(0) <= FS(0);
```

FS	Y	Cin	OP
00	B	0	A+B
01	B	1	A+B+1
10	NB	0	A-B
11	NB+1	1	A-B+1

Tabela 1: Tabela de verdade da lógica de entrada B e do Carry in

Da unidade aritmética saem duas flags que, quando ativas, denunciam a ocorrência de fenómenos no bloco (Carry-Out e Overflow). À semelhança da unidade de deslocamento também deste bloco sai D(15:0) contendo todas as alterações efetuadas no mesmo.

O terceiro bloco pertencente à Functional Unit é o lógico, ou Logic Unit, que é responsável pelas operações a nível lógico efetuadas sobre as entradas A(15:0) e B(15:0) dependendo agora de uma variável de seleção mais extensa, FSUF(2:0).

Ao contrário do que ocorria nos blocos anteriores, neste segmento não existe qualquer flag, o que nos permite inferir que não será suposta a ocorrência de excedência da capacidade de processamento ou de outros erros. Desta unidade sai D(15:0) contendo todas as alterações efetuadas.

Seguidamente encontra-se um multiplexer com 3 variáveis de entrada, sendo elas as saídas D(15:0) proveniente do Shifter, Arithmetic Unit e Logic Unit, e uma variável de seleção FSUF(2:0) que determina o valor da entrada que é transportado para a saída O(15:0). Esta saída posteriormente dará origem à variável D(15:0).

Existe um outro multiplexer que tem como entradas as flags de Carry-Out do Shifter e do bloco Aritmético e como entrada de seleção a variável FSUF(2:0). Este multiplexer tem como função determinar qual é que é a flag que deve ser escolhida para a operação realizada (ou aritmética ou de Shift).

O terceiro e último multiplexer da Functional Unit tem como entradas as flags de Overflow do Shifter e do bloco Aritmético e como entrada de seleção a variável FSUF(2:0). Este multiplexer tem como função determinar qual é que é a flag que deve ser escolhida para a operação realizada (ou aritmética ou de Shift).

Por fim, existem dois buffers que recebem a saída do primeiro multiplexer, 0(15:0) e analisam a possibilidade de esta ser negativa ou nula. As saídas dos multiplexers ou buffers que determinam as flags encontra-se reunidas num único bus, FL(3:0).

Após detalhada observação do código VHDL, a fim de elaborar a tabela 1, é preciso constatar que:

- ✓ Numa primeira fase, são realizadas várias operações simultaneamente em cada um dos blocos da unidade funcional (AU, SH e LU). Seguidamente existe um multiplexer que seleciona o resultado final em virtude da entrada de seleção FSUF(2:0);
- ✓ Na unidade aritmética, através da variável de controlo FSUF(1:0), é selecionada a operação que se pretende realizar, através da lógica combinatória anterior, à entrada B(dependente de FS(1)), de cada full-adder e da entrada de Carry-in (dependente de FS(0)= do somador);
- ✓ Na unidade lógica, as operações de nível lógico entre os bits de A(15:0) e B(15:0) são selecionados pelos três bits menos significativos da variável de controlo FSUF(2:0) recorrendo a um multiplexer;
- ✓ Na unidade de deslocamento, as operações a realizar são controladas pelos três bits menos significativos de FS(2:0) que, ao atuar ao nível de um multiplexer, permitem efetuar as operações de shift e de rotate sobre o operando B(15:0).

FSUF(3:0)	Arithmetic Unit	Cin	Operação		D(15:0)
000	B	0	AU1	A+B	A(15:0)+B(15:0)
001	B	1	AU1	A+B+1	A(15:0)+B(15:0)+1
010	$\bar{B}$	0	AU1	A+$\bar{B}$	A(15:0)+$\bar{B}$(15:0)
011	$\bar{B}$	1	AU1	-B	A(15:0)+$\bar{B}$(15:0)+1
100	B	0	LU1	A AND B	A(15:0) AND B(15:0)
101	B	1	LU1	A NAND B	A(15:0) NAND B(15:0)
110	$\bar{B}$	0	LU1	A OR B	A(15:0) OR B(15:0)
111	$\bar{B}$	1	LU1	A NOR B	A(15:0) NOR B(15:0)
1000	B	0	LU1	A XOR B	A(15:0) XOR B(15:0)
1001	B	1	LU1	A XNOR B	A(15:0) XNOR B(15:0)
1010	$\bar{B}$	0	SH1	Shift Left	B(14:0) & '0'
1011	$\bar{B}$	1	SH1	Shift Right Logic	'0' & B(15:1)
1100	B	0	SH1	Shift Left	B(14:0) & '0'
1101	B	1	SH1	Shift Right Aritmetic	B(15) & B(15:1)
1110	$\bar{B}$	0	SH1	Rotate Left 1 bit	B(14:0) & B(15)
1111	$\bar{B}$	1	SH1	Rotate Right 1 bit	B(0) & B(15:1)

b) No bloco aritmético, isto é, nas quatro primeiras operações, as flags relevantes são as de Zero, Negative, Overflow e Carry. A relevância das flags prende-se com o facto de este ser o bloco que gera somas e subtrações podendo estas incorrer em erros de excedência de capacidade ou de representação.

No bloco lógico, ou seja, nas seis operações seguintes, as flags mais importantes são a de Negative e Zero dado que as operações lógicas nunca geram valores fora do intervalo de representação nem carregam bits de transporte.

No bloco de deslocamento, que representa as últimas seis operações da tabela acima, as flags relevantes são as de Negative, Zero, Carry (pertencente a todas as operações utilizadas neste bloco dado que a saída de Overflow depende diretamente do último bit de FS) e as de Overflow (que pertencem às operações de Shift Left e Rotate Left 1 bit pois são selecionadas pelo Mux que deteta overflow).

2)

A unidade de armazenamento, ou Register File como é chamada no projeto de VHDL, é constituída por:

- ✓ Um descodificador de 16 saídas;
- ✓ 15 portas AND, que representam a lógica combinatória que permite ao utilizador escrever o sinal enviado pelo descodificador num registo sempre que o sinal de WR se encontre ativo;
- ✓ 15 registos que se encontram ligados a dois multiplexers, MA e MB;
- ✓ 2 multiplexers, MA e MB, que tem entradas às quais estão ligados os registos e que vão originar, na saída, A(15:0) e B(15:0), respetivamente.

3)

Sinais	Descrição e respetiva função
AA(3:0)	É um sinal de 4 bits que se apresenta à entrada do multiplexer MA, que, por sua vez, seleciona uma de 16 entradas(cada uma com 16 bits). A primeira entrada deste multiplexer encontra-se ligada à terra, isto é, ao GND, enquanto que as restantes estão ligadas aos registos. Dado que a primeira entrada se encontra ligada ao GND é relativamente fácil ligar qualquer dos registos ou o valor lógico 0 ao operando A.
BA(3:0)	É um sinal de 4 bits que se apresenta à entrada do multiplexer MB, que, por sua vez, seleciona uma de 16 entradas(cada uma com 16 bits). A primeira entrada deste multiplexer encontra-se ligada à terra, isto é, ao GND, enquanto que as restantes estão ligadas aos registos. Dado que a primeira entrada se encontra ligada ao GND é relativamente fácil ligar qualquer dos registos ou o valor lógico 0 ao operando B.
DA(15:0)	Sinal lógico que se encontra na entrada de um descodificador com 16 saídas. As saídas do descodificador encontram-se ligadas ao sinal de WR através de portas AND.
A(15:0)	Saída do multiplexer MA e que é selecionado como o operando A, sobre o qual poderão ou não recair as operações desejadas.
B(15:0)	Saída do multiplexer MB e que é selecionado como o operando B, sobre o qual poderão ou não recair as operações desejadas.
Data(15:0)	Sinal de 16 bits ligado aos registos cujo valor pode ser diretamente colocado na saída, a cada ciclo de relógio.
WR	Sinal de 1 bit que permite a escrita quando ativo a High (visto que se encontra combinado através de portas AND com as saídas do descodificador).

Tabela 2- Estudo da interferência do sinal FSU na função de saída, tabela de verdade da lógica de entrada B de cada somador completo e da entrada de transporte

Foram requeridas uma série de micro operações a simular para o turno de sexta-feira:

Sinais de controlo										Valor Esperado		Resultado - obtido na simulação	
AA(3:0)	BA(3:0)	DA(3:0)	WR	MA	MB	KNS(15:0)	FSUF(3:0)	MD	MW	D(15:0)	Flags	D(15:0)	Flags
XXXX	XXXX	0001	1	1	0	0006h	0000	0	0	0006h	Não	0006h	Não
0000	XXXX	0010	1	0	1	239C	0000	0	0	239Ch	Não	239Ch	Não
XXXX	0000	0011	1	1	0	1B40	0000	0	0	1B40h	Não	1B40h	Não
XXXX	0011	0100	1	X	0	XXXXh	1111	0	0	0DA0h	Não	0DA0h	Não
0100	XXXX	0100	1	0	1	0004h	0001	0	0	0DA5h	Não	0DA5h	Não
0100	0010	0100	1	0	0	XXXXh	1000	0	0	2E39h	Não	2E39h	Não
0001	XXXX	0011	1	0	1	0019h	0011	0	0	FFEDh	N	FFEDh	N

Tabela 3- Através da análise detalhada das alterações a executar para o turno de sexta-feira, elaborou-se a presente tabela

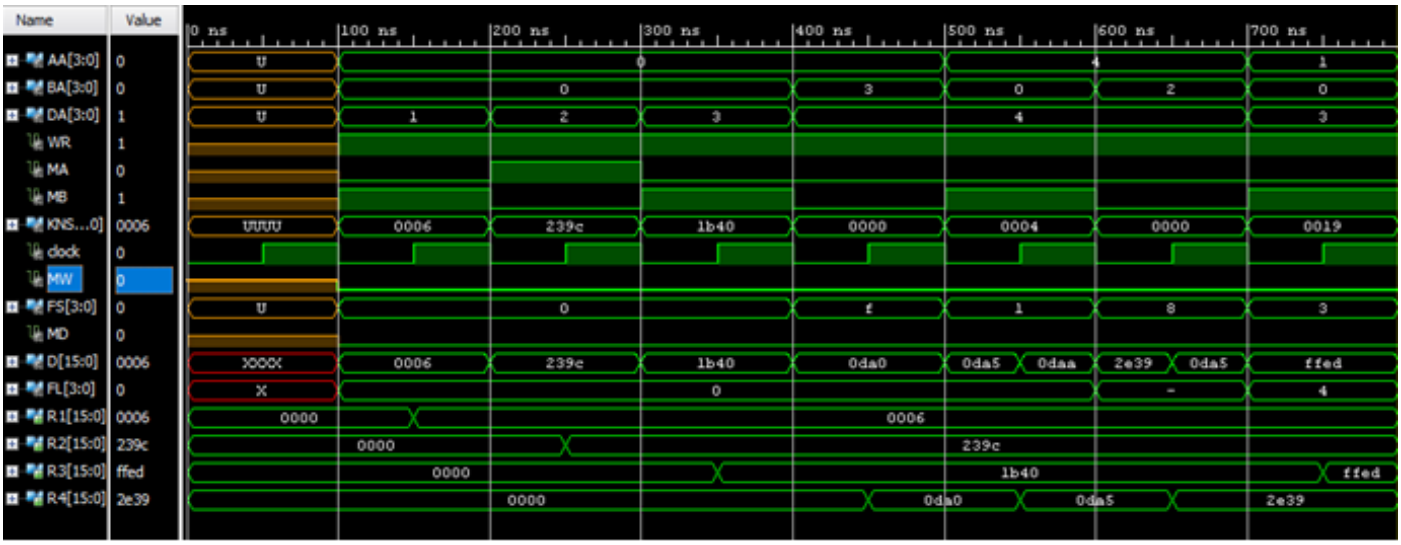


Figura 1 - Simulação Realizada

4)

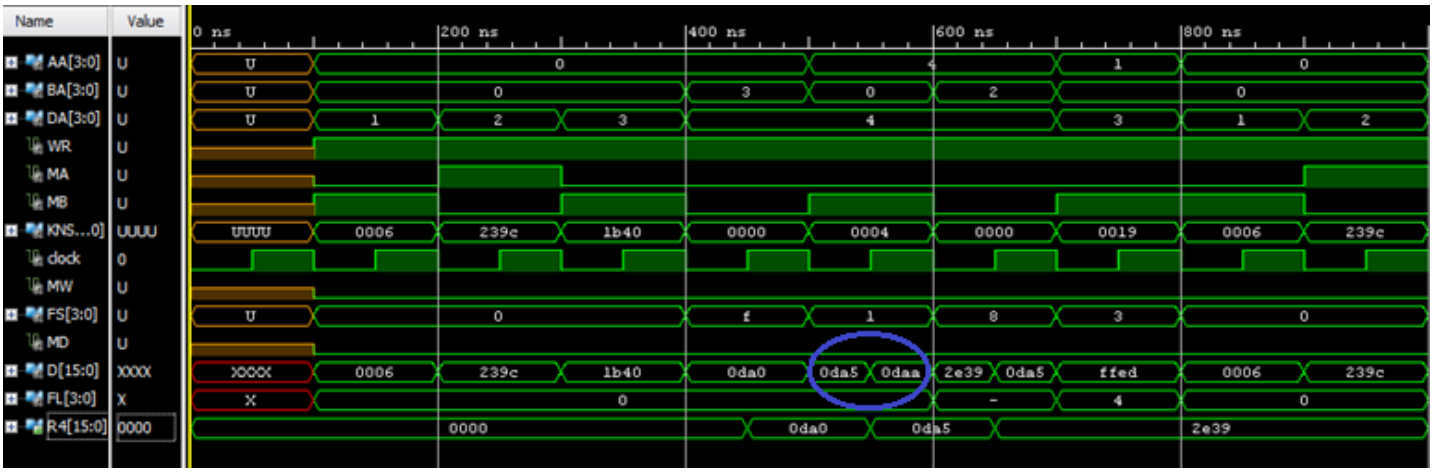


Figura 2 - Forma de Onda da simulação realizada

Após o flanco de relógio é guardado no registo o novo valor. Como esse registo está selecionado pelos *Multiplexers*, o novo valor contido no mesmo é de novo direcionado para a *Functional Unit*, que executa novamente uma operação de soma sobre o valor do registo.

A operação de soma especificada é executada duas vezes, mas apenas o primeiro valor é guardado no registo, dado que este está dependente do flanco de relógio.

Conclui-se que não se trata de qualquer tipo de erro.

Segunda Parte- "Síntese de novas funcionalidades para a Unidade de Processamento"

3)

Síntese as Funcionalidades

É requisitado que se altere o register file (RF) de forma a que este implemente uma nova operação que permite copiar valores entre registos sem recurso à Functional Unit (FU).

Criámos uma uma nova entrada do register file com designação FS e uma nova entrada no DataPath com designação FSRF que substituem as entradas WR.

FSRF	Função
00	Nenhuma operação
01	RD ← D
10	RD ← RA
11	RD ← RB

Tabela 4 – Funções do Register File alteradas

1. Introduzimos uma porta OR com FSRF(0) e FSRF(1) e ligámos a saída da mesmas a uma porta AND que também tem ligado o sinal DA(*destination adress). A saída dessa mesma porta AND é ligada ao enable* dos registos. Assim, sendo os registos apenas serão atualizados nas situações em FSRF é igual a "01" , "10" ou "11" e não serão atualizados no caso em que FSRF é igual a "00".

Código VHDL do Load de um dos registos:

```
L1 <= D1 and (FSRF(0)  or FSRF(1));
```

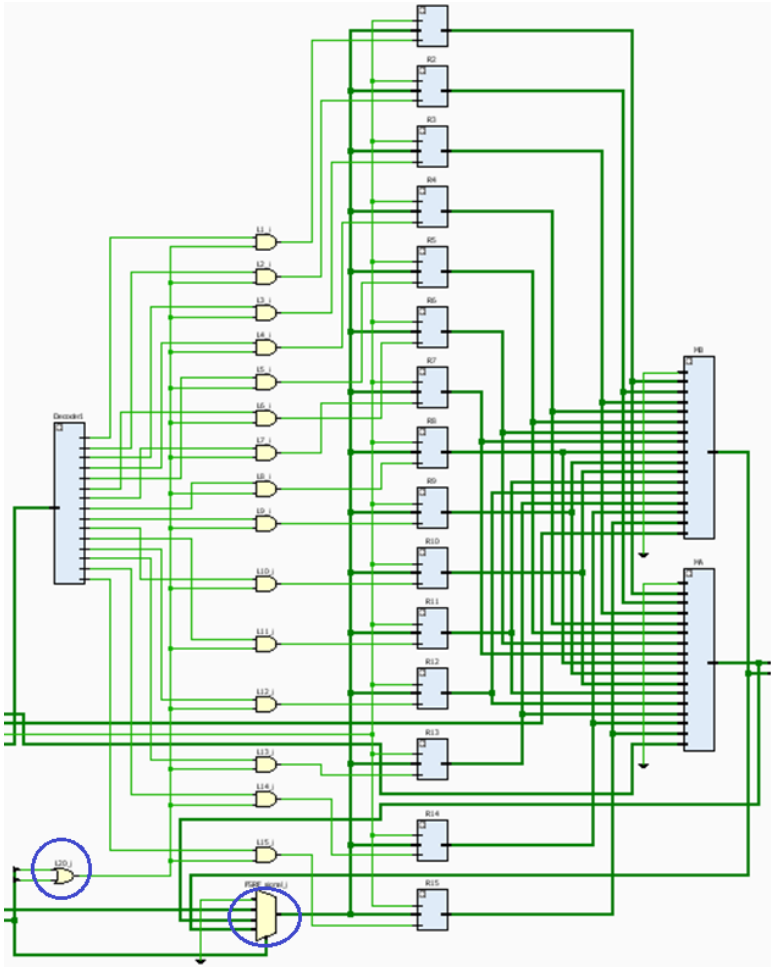
2. Introduzimos um MUX que:

- Tem com entrada de seleção FSRF;
- Nas entradas de Dados seleciona o valor Lógico '0' quando FSRF estiver a "00", seleciona a entrada Data quando FSRF estiver a "01", seleciona a saída do Multiplexer A quando FSRF estiver a "10", seleciona a saída do Multiplexer B;

Código VHDL:

```
with FSRF select FSRF_signal<= "0000000000000000" when "00",
                Data when "01",
                A when "10",
                B when "11",
                 (others =>'X')when others;
```

Análise do esquema



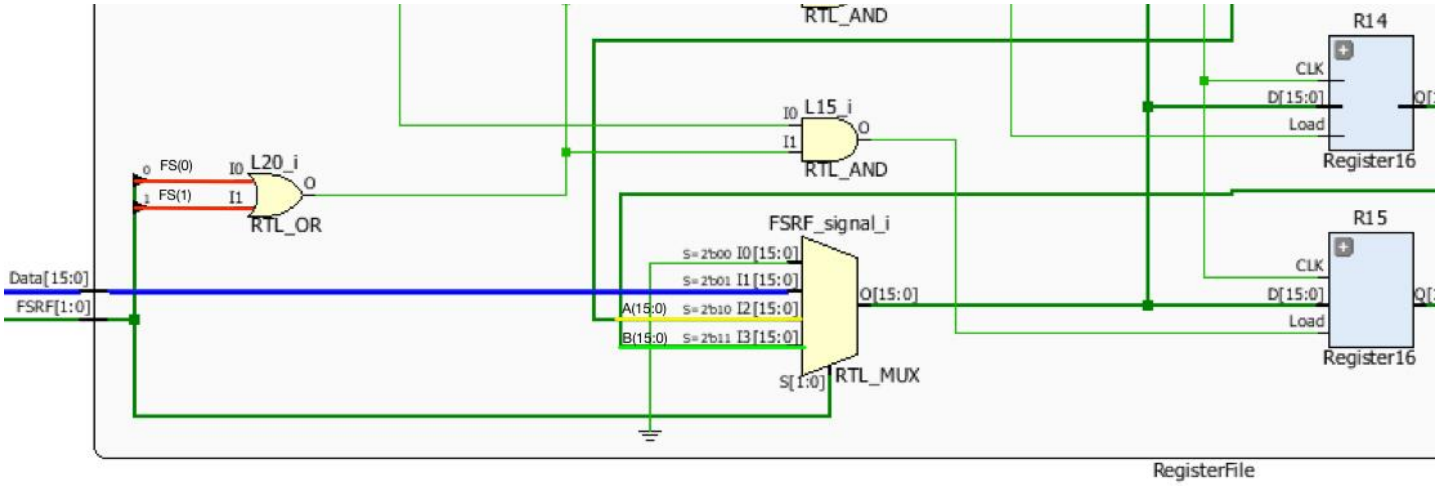


Figura 3: Esquema do RF com pormenores da porta OR, cuja saída liga ao enabe do contador e DO MuX, cuja saída liga à entrada de dados dos registos

Os Bits FSRF(0) e FRSF(1) estão ligados por uma porta OR que, por sua vez, tem a sua saída ligada a uma porta AND com o *Destination Adress* que faz o *Enable* do Registo. Deste modo, garante-se que o registo apenas será escrito quando FSRF for diferente de "00".

O MUX seleciona uma das entradas "Data", "A" ou "B" para a entrada de cada um dos registos, com a entrada de seleção FSRF(2:0) a respetivamente "01", "10" ou "11".

Teste à transferência entre Registos

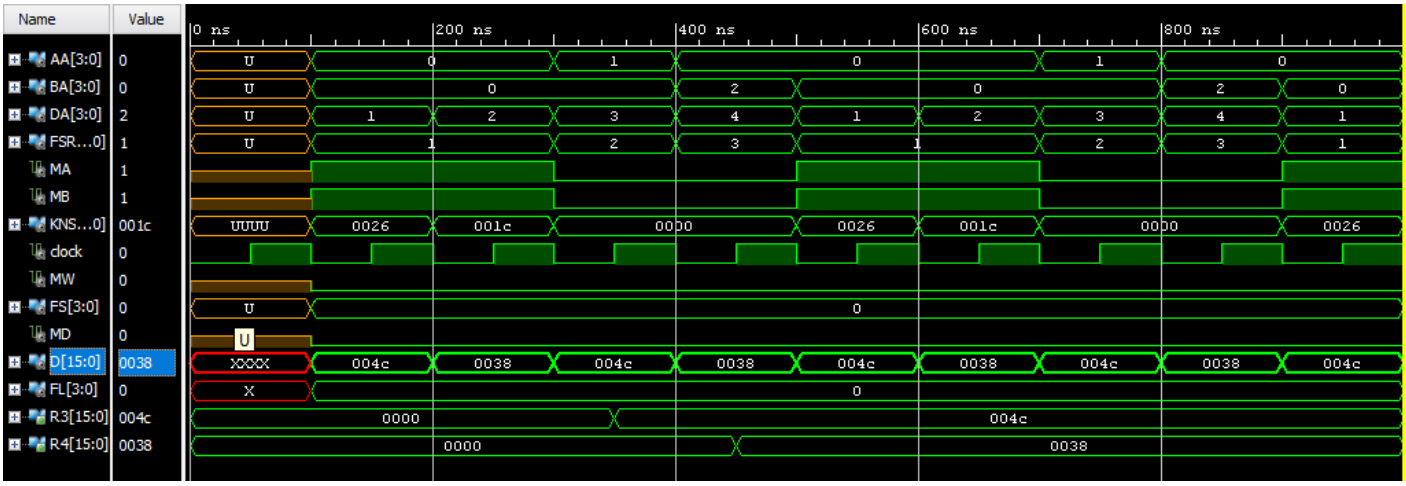


Figura 3 - Forma de Onda das Operações Realizadas

A simulação mostra que os valores guardados em R3 e R4 foram, respetivamente 004Ch e 0038h, conforme o esperado. Simulação efetuada com sucesso.

5)

Síntese das Funcionalidades

Foram-nos solicitadas as seguintes alterações:

1. *A unidade aritmética deve ser escolhida sempre que FS for da forma 0xxx*

Para efetuar esta alteração tivemos de alterar o MUX da *Functional Unit* que direciona os dados para a saída.

Código VHDL:

```
with FS(3 downto 1) select
        DI <=   DA when "000" | "001"|"010"|"011",
        DL when  "100",
        DS when "101" | "110" | "111",
        (others => 'X') when others;
```

2. *Sempre que o bit mais significativo do FS da unidade aritmética estiver a 1 devem ser efetuadas duas operações de 8 bits enquanto que quando está a zero devem-se manter operações de 16 bits pré-existentes.*

Criámos um novo sinal FS_2signal.
Utilizámos um MUX com o sinal de seleção FS(2) -o bit mais significativo do FS que entra na Unidade Aritmética. Este seleciona como saída o *carryout* do registo 7 quando estiver a 0 ou o bit FS(0) quando estiver a 1.

Código VHDL:

```
with FS(2) select FS_2signal <=   C(8) when '0',
                                  FS(0) when '1',
```

```
                                                    'X' when others;
```

Para efetuar as operações de 8 bits interrompemos a cadeia de transporte existente nos somadores completos (fulladders), substituindo o carry in do do somador 8 pelo valor anteriormente atribuído a FS_2signal.

Código VHDL:

```
aritGen: for I in 7 downto 0 generate
        AUi: ArithmeticUnitI port map (Ai=>A(I), Bi=>B(I), FS=>FS, CI=>C(I), Di=>Z(I), CO=>C(I+1));
end generate aritGen;

AUi: ArithmeticUnitI port map (Ai=>A(8), Bi=>B(8), FS=>FS, CI=>FS_2signal, Di=>Z(8), CO=>C(9));

aritGen2:for I in 15 downto 9 generate          AUi: ArithmeticUnitI port map (Ai=>A(I), Bi=>B(I),
FS=>FS, CI=>C(I), Di=>Z(I), CO=>C(I+1));
        end generate aritGen2;
    D <= Z;
```

3. Também tivemos de alterar o MUXs que escolhem as flags para cada operação, pois a saída depende agora de outros valores- a unidade aritmética será escolhida sempre que FS for da forma 0XXX.

Código VHDL:

```
- - flag  carry
   with FS(3 downto 1) select
   C <=    COA when "000" | "001" |"010" | "011",
           '-' when "100",
           COS when "101" | "110" | "111",
           'X' when others;

   - - flag overflow
   with FS(3 downto 1) select
   O <=    OVA when "000" | "001" |"010" | "011",
           '-' when  "100",
           OVS when "101" | "110" | "111",
           'X' when others;
```

Análise do esquema

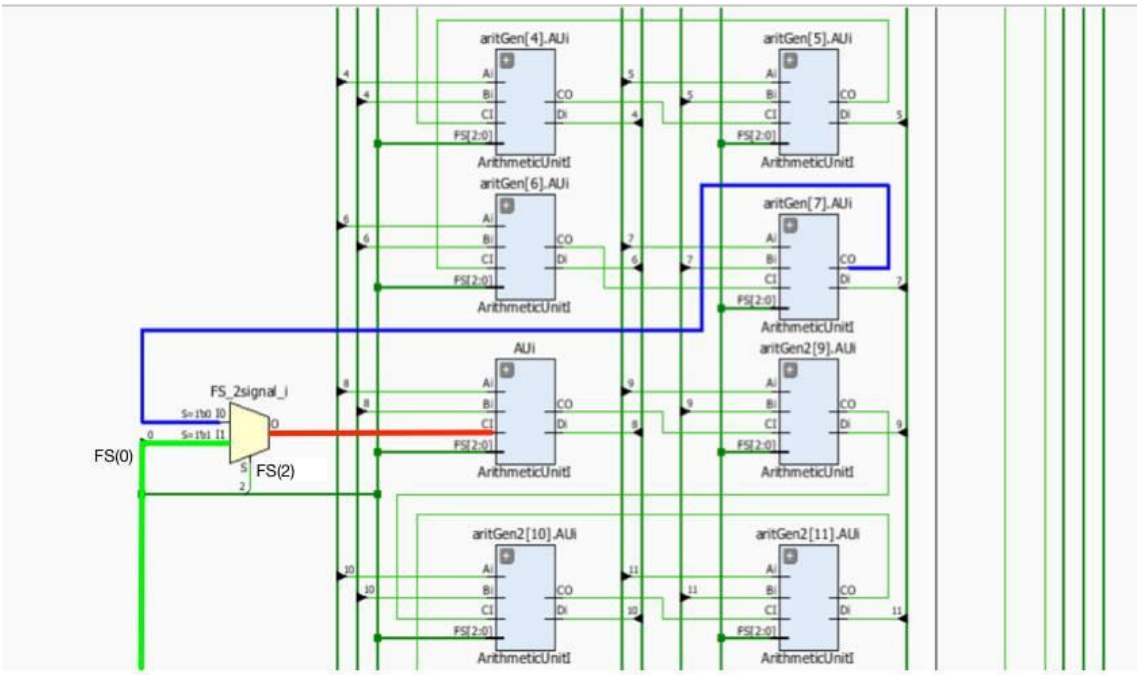


Figura 4 - Mux criado e entradas de dados do MUX CO(7) e FS(0)

Repare-se que foi interrompida a cadeia de transporte entre os somadores completos.
A entrada FS(2)- a vermelho- seleciona para a saída do MUX (que está ligada ao Carry in do somador completo 8) o CARRY OUT do Somador Completo 7 , quando está a 0, ou o bit FS(0) quando está a 1.

Teste à Unidade Aritmética

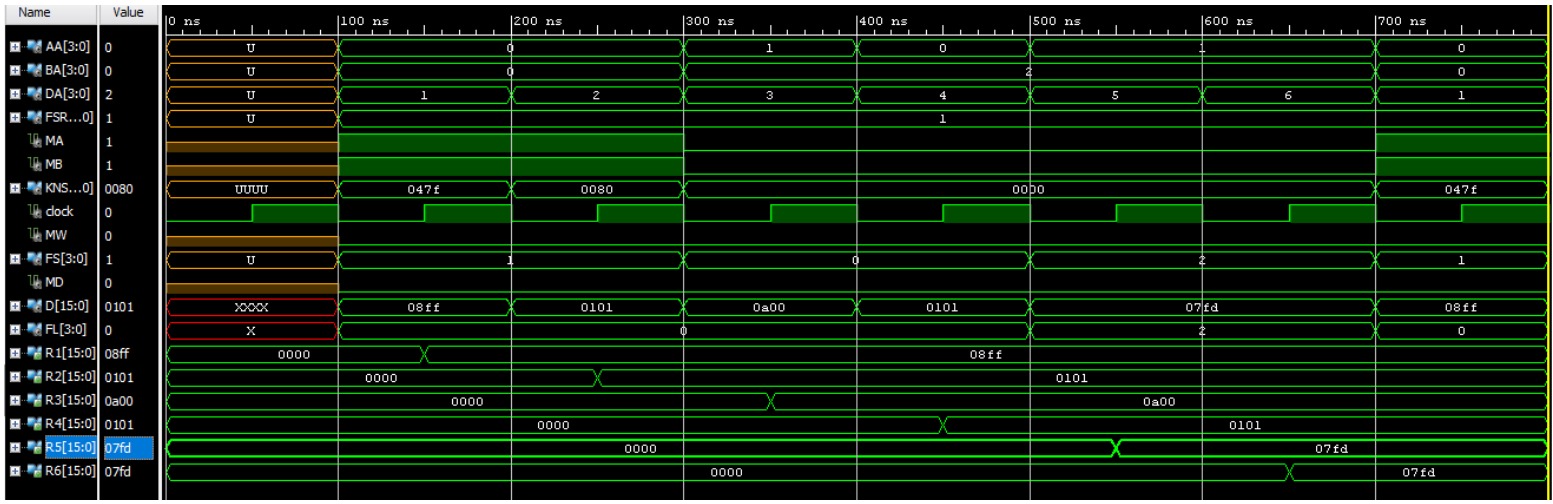


Figura 6 -Teste à Unidade Aritmética

O teste decorreu conforme o esperado, dado que os valores guardados nos registos foram os pretendidos.

Datapath

Conseguimos o correto funcionamento da Unidade de Processamento e a mesma está pronta a ser sintetizada. Neste esquema apresentamos o panorama geral do resultado do nosso projeto.

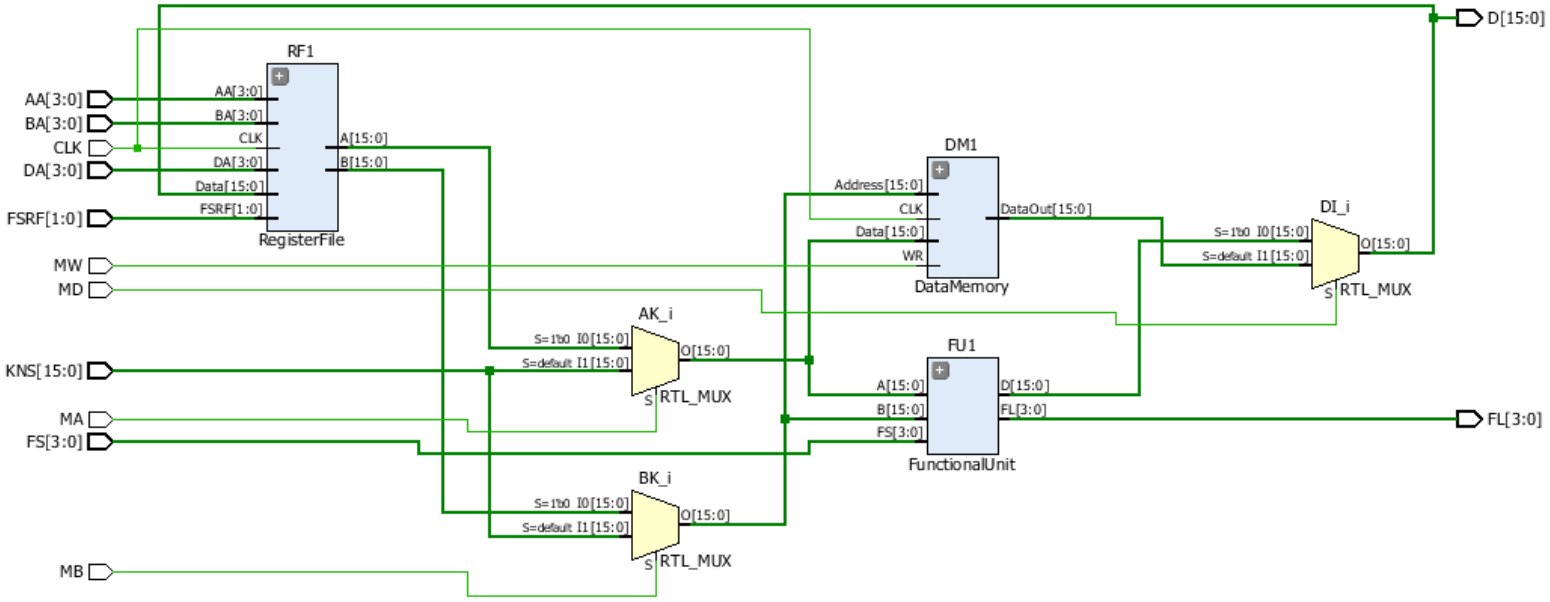


Fig.7: Esquema do DATAPATH elaborado no Vivado

Conclusões

No decorrer deste trabalho aquilo que nos suscitou maiores dificuldades foi, no ínicio, a interpretação do funcionamento da Unidade Funcional. Ainda assim, conseguimos ultrapassar as nossas dificuldades nomeadamente recorrendo ao *Schematich* elaborado pelo *Vivado* o que nos permitiu ter uma visão mais abrangente das conexão entra cada unidade do DATAPATH. Recorrendo à funcionalidade *go to source code* , conseguimos mais facilmente perceber o papel de cada unidade na funcionalidade geral.

No que toca à Síntese de Novas Funcionalidades o mais difícil foi interpretar o enunciado corretamente e a partir dele extrair informação correta das especificações funcionais pedidas para implementar.

Acreditamos que este projeto foi importante para desenvolvermos competências relativas à Unidade de Processamento , sua síntese, implementação e teste.