Laboratório 9 (Trabalho de Avaliação)
Realização: Semanas 20 de Novembro a 15 de Dezembro
Apresentação/discussão: Semana 18 a 21 de Dezembro

Exercício 25 – Cálculo de uma expressão

Pretende-se desenvolver um sistema digital que calcule o resultado numérico de uma expressão, bem como ative um sinal de alarme quando o resultado for 0. A expressão conterá quatro operações, sendo três aritméticas (soma, multiplicação e decremento) e uma lógica (e).

Ao longo deste enunciado apresenta-se esboço de uma solução possível, que cada grupo deverá concretizar. O presente enunciado pode ser "melhorado" sempre que o grupo o considere justificado; sempre que considere que o enunciado não é explícito ou é ambíguo, poderá/deverá (o grupo) definir a melhor forma de superar essa limitação.

Avaliação e Prazos Gerais

Após a resolução e implementação bem-sucedidas do enunciado seguidamente apresentado, cada grupo deve preparar um relatório (sugere-se a utilização do "modelo" de relatório disponível na página moodle da disciplina). Todos os elementos do grupo devem participar na elaboração do relatório.

A entrega do relatório (em papel) deve ser realizada pessoalmente a um professor da disciplina (preferencialmente ao docente do turno respetivo ou ao professor responsável pela discussão do trabalho) e complementada com a submissão na página moodle da disciplina de um ficheiro "zipado" contendo o relatório (.pdf ou .doc/.docx) e o projeto realizado no ambiente ISE/WebPack da Xilinx.

Os prazos para entrega são os seguintes:

- relatório em papel: (algumas horas) antes do início da apresentação/discussão;
- ficheiro: submetido até 17 de Dezembro, 23:55.

As apresentações/discussões do trabalho, com análise do funcionamento do sistema desenvolvido, devem ser realizadas no laboratório 2.2-X direito/norte, deverão acontecer no período de 18 a 21 de Dezembro (de acordo com marcação prévia), ou em horário a combinar com os docentes das teóricas.

Como critérios principais na avaliação do trabalho referem-se: o relatório, a solução técnica encontrada (e demonstrada através de protótipo) e a discussão/defesa.

IMPORTANTE: Na discussão serão abordados temas associados à unidade lógica e aritmética (ULA) cuja implementação é disponibilizada na página moodle da UC, como mais à frente se refere.

Condicionantes laboratoriais

A implementação do sistema deverá ser baseada no kit de experimentação BASYS2 da Digilent em posse dos grupos de trabalho e disponível no laboratório e contendo um dispositivo lógico programável do tipo FPGA Spartan3E, considerando o ambiente de desenvolvimento WebPack/ISE da Xilinx.

O projeto deverá ser simulado no ambiente WebPack/ISE da Xilinx antes de se realizar a configuração da FPGA.

Objetivo

Pretende-se realizar um sistema que calcule o resultado de uma expressão e que ative uma saída sempre que o resultado for zero.

Cada grupo deverá identificar a expressão que necessita de realizar, de acordo com a seguinte tabela e o valor de OPCAO1, sendo OPCAO1 o resto da divisão inteira da soma dos números de estudante dos membros do grupo, por 6.

OPCAO1	Expressão
0	((K_1 * A) DEC K_{DEC}) AND K_{AND}
1	((K_1 * A) AND K_{AND}) DEC K_{DEC}
2	((A DEC K_{DEC}) + K_1 * A) AND K_{AND}
3	((A AND K_{AND}) + K_1 * A) DEC K_{DEC}
4	((A DEC K_{DEC}) AND K_{AND}) + K_1 * A
5	((A AND K_{AND}) DEC K_{DEC}) + K_1 * A

O operando de entrada, designado por A, é composto por N bits (o valor de N considerado é de 8).

Os valores para os parâmetros K_1, K_{DEC} e K_{AND} deverão ser determinados em cada grupo de trabalho de acordo com a seguinte tabela, em que OPCAO2 é o resto da divisão inteira da soma dos números de estudante dos membros do grupo, por 9.

OPCAO2	K_1	K_{DEC}	K_{AND}
0	1	3	7
1	2	2	5
2	3	1	3
3	1	3	6
4	2	2	3
5	3	1	6
6	1	3	3
7	2	2	5
8	3	1	7

Pretende-se que, no final, proceda à simulação no ambiente da Xilinx e à implementação numa FPGA Spartan 3E, verificando o seu correto funcionamento.

Descrição

O funcionamento do sistema pode ser descrito genericamente da seguinte forma. A atuação de um sinal de entrada (GO) provoca a computação da expressão e a sua comparação com 0. O algoritmo que se pretende utilizar para a realização da operação de multiplicação recorre ao método das somas sucessivas. *Nota: os grupos poderão prescindir da utilização da variável GO, permitindo que o sistema de alarme esteja sempre ativo (e não apenas após a ativação da variável GO).*

Análise

O diagrama de blocos que se propõe seguidamente deve ser encarado como uma sugestão; os grupos de trabalho dispõem de toda a liberdade para proporem e implementarem caracterizações alternativas (ou resultantes de otimizações à sugestão apresentada), desde que tenham presente a limitação do número de aulas para a sua realização.

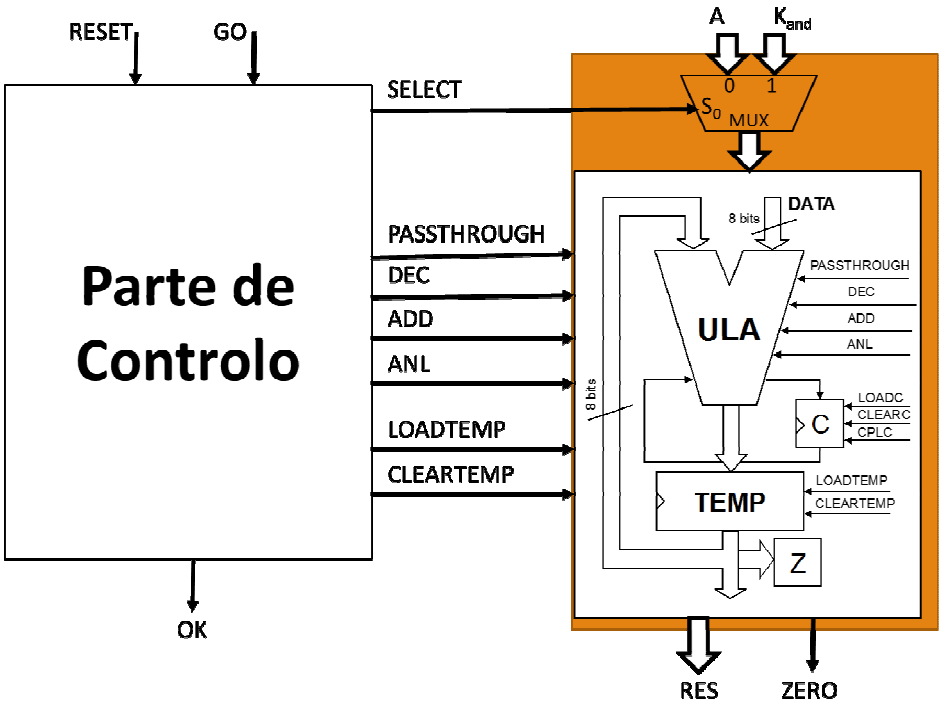
NOTA 1: O bloco de dados referido no diagrama de blocos proposto é disponibilizado na página moodle da UC e poderá ser integrado no projeto a desenvolver por cada grupo.

NOTA 2: Uma descrição da unidade lógica e aritmética (ULA) cuja implementação se disponibiliza é apresentada no Anexo A a este enunciado.

NOTA 3: Sugere-se a leitura do Anexo B a este enunciado (como introdução aos procedimentos associados à decomposição de um sistema em parte de dados e parte de controlo), com a resolução de um multiplicador de 3*A, apresentado nas aulas teóricas.

O valor do resultado bem como do valor de A devem ser apresentados nos visualizadores de 7 segmentos.

Assim, considere-se, como ponto de partida, uma caracterização genérica contendo uma parte de controlo e uma parte de dados, em que a parte de controlo será realizada através de uma máquina de estados e a parte de dados através de uma arquitetura de transferência entre registos, tendo por base a Unidade Lógica e Aritmética fornecida. A seguinte figura ilustra a decomposição proposta para o sistema em parte de controlo e parte de dados, em que está implícito um sinal de relógio fornecido aos dois blocos (e onde não está representada a interligação aos visualizadores de 7 segmentos).

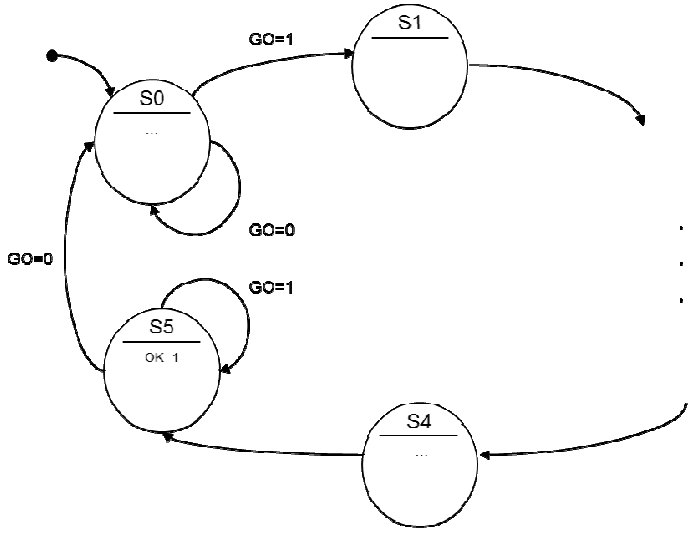


Sucintamente, a arquitetura que se sugere será controlada da seguinte forma:

- No início procede-se à inicialização da parte de controlo do sistema, através da ativação da entrada de RESET;
- A parte de controlo ficará à espera que o sinal de entrada GO seja ativado, após o que deve iniciar a sequência de ações associadas ao cálculo da expressão (como referido anteriormente, os grupos poderão optar por não utilizar a entrada GO, considerando, nesse caso, que está sempre ativa a intenção de fazer o cálculo);
- Realizado o cálculo da expressão, deve-se apresentar o resultado nos visualizadores de 7 segmentos disponíveis na placa de experimentação, bem como a saída de deteção de ZERO num led e evoluir para nova avaliação da expressão (após o sinal de GO ter sido desativado e ativado novamente, ou diretamente caso não se utilize a entrada GO).

Desta forma, recomenda-se que a parte de controlo seja uma máquina de estados, síncrona, arquitetura de Moore. A figura apresenta um esboço de possível diagrama de estados, coerente com a arquitetura de dados apresentada:

- O estado S0 é o estado inicial onde se espera pela ordem para realizar o cálculo;
- Os estados S1 a S4 realizam a sequência de cálculo;
- O estado S5 é o estado final (do cálculo) que consequentemente ativa a saída OK.



A presente máquina de estados é (obviamente) responsável pelo controlo das saídas da parte de controlo, a serem fornecidas à parte de dados.

Adicionalmente, deverá calcular (e apresentar no relatório) quais os valores de A (se existirem) que gerarão um resultado igual a 0 (que serão posteriormente verificados através de simulação e experimentação).

Teste e Validação

O teste e validação do sistema será realizado quer através de simulação no ambiente Xilinx ISim, quer por implementação no kit de experimentação Basys2 da Digilent disponibilizado a cada grupo de trabalho.

Configurações para Simulação com Xilinx ISim

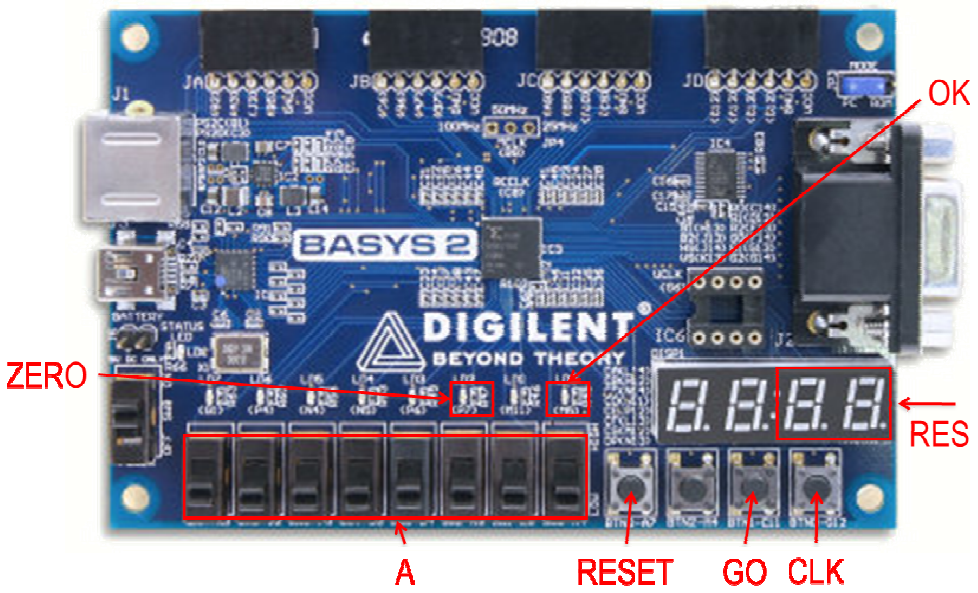
Para realizar o teste e validação do sistema, através de simulação, poderá considerar as seguintes definições de simulação (ficheiro VHDL Test Bench). Na prática, a especificação de simulação apresentada, introduz o decimal 15 na entrada de dados A e realiza o cálculo da expressão.

```
tb: PROCESS
BEGIN
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
END PROCESS;

A(7) <= '0'; A(6) <= '0'; A(5) <= '0'; A(4) <= '0';
A(3) <= '1'; A(2) <= '1'; A(1) <= '1'; A(0) <= '1';
RESET <= '1' , '0' after 30 ns;
GO <= '0', '1' after 50 ns, '0' after 210 ns;
```

Configurações para Experimentação com kit Digilent Basys2

Para harmonizar a interface de experimentação dos vários sistemas no kit Digilent Basys2 sugere-se a configuração de implementação representada na figura seguinte. Desta forma será possível atribuir o valor de A através dos oito interruptores SW7..0, realizar uma (re-)inicialização do sistema através do botão de pressão BTN3 (correspondente ao RESET), arrancar o processo de cálculo através do botão de pressão BTN1 (correspondente ao GO) e dar *clocks* de forma manual através do botão de pressão BTN0 (sinal CLK); por outro lado, é possível visualizar a flag de Z no led LD7, o sinal OK de conclusão de cálculo no led LD0 e o resultado do cálculo nos visualizadores de 7 segmentos.



Para realizar a configuração acima, as definições (ficheiro UCF) que deverá utilizar são as apresentadas abaixo. Note-se que o *Clock* se encontra em modo manual (botão de pressão associado ao pino G12 no kit de experimentação); se desejar mudar para *Clock* para automático deverá apagar as duas entradas associadas ao sinal "CLK" e associar todos os sinais de *Clock* no esquemático de topo a uma mesma entrada de nome "CLK_AUTO".

```
net "CLK" loc = "G12";
net "CLK" CLOCK_DEDICATED_ROUTE = FALSE;
net "RESET" loc = "A7";
net "GO" loc = "C11";
net "A(7)" loc = "N3";
net "A(6)" loc = "E2";
net "A(5)" loc = "F3";
net "A(4)" loc = "G3";
net "A(3)" loc = "B4";
net "A(2)" loc = "K3";
net "A(1)" loc = "L3";
net "A(0)" loc = "P11";
net "ZERO" loc = "P7";
net "OK" loc = "M5";
```

```
#net list visualizador 7 segmentos
net "CLK_AUTO" loc="B8";
net "ca" loc="L14";
net "cb" loc="H12";
net "cc" loc="N14";
net "cd" loc="N11";
net "ce" loc="P12";
net "cf" loc="L13";
net "cg" loc="M12";
net "cp" loc="N13";
net "an3" loc="K14";
net "an2" loc="M13";
net "an1" loc="J12";
net "an0" loc="F12";
```

Bom trabalho!

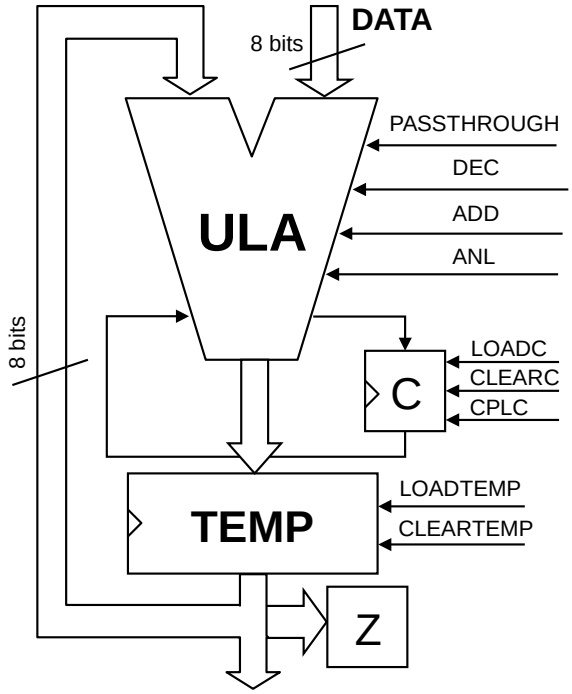
ANEXO A: Especificação e realização de uma Unidade Lógica e Aritmética

Objetivos

Realização de uma Unidade Lógica e Aritmética (ULA). Desenho e implementação de uma unidade, através de arquitetura uma transferência entre registos, capaz de realizar um pequeno conjuntos de operações aritméticas e operações lógicas. Verificação experimental do bom funcionamento da ULA.

Pequena Unidade Lógica e Aritmética (ULA)

Pretende-se criar uma pequena Unidade Lógica e Aritmética (ULA) capaz de realizar operações aritméticas (decremento e soma) e operações lógicas ('e' unicamente) e contendo as flags de C[arry] (Transporte de operações aritméticas) e Z[ero] (Comparação com Zero). A descrição sucinta da Unidade Lógica e Aritmética é a apresentada no diagrama de bloco em seguida:



A Unidade Lógica e Aritmética é composta pelos seguintes módulos:

- módulo ULA, um bloco de lógica combinatória, que realiza as operações aritméticas e lógicas a 8 bits. Contém duas entradas de dados de 8 bits e uma saída de dados de 8 bits, e ainda saída e entrada de 1 bit de transporte de operações aritméticas. Possui quatro entradas de seleção que permitem selecionar a operação desejada (PASSTHROUGH – passagem direta; DEC – Decremento; ADD – Soma; ANL – 'E' lógico bit-a-bit);
- módulo TEMP, um registo síncrono de 8 bits para guardar os dados (temporariamente) das operações aritméticas e lógicas realizadas pela ULA, possuindo entradas e saídas de dados de 8 bits, e entradas síncronas de controlo LOADTEMP (carregamento de dados à entrada no registo) e CLEARTEMP (limpar o registo);
- módulo C, um registo síncrono de 1 bit que implementa a flag de Carry, e que permite guardar o valor do transporte das operações aritméticas, possuindo uma entrada e uma saída de dados de 1 bits, e entradas síncronas de controlo LOADC (carregamento de dados à entrada no registo), CLEARTEMP (limpar o registo) e CPLC (complementar logicamente o registo);

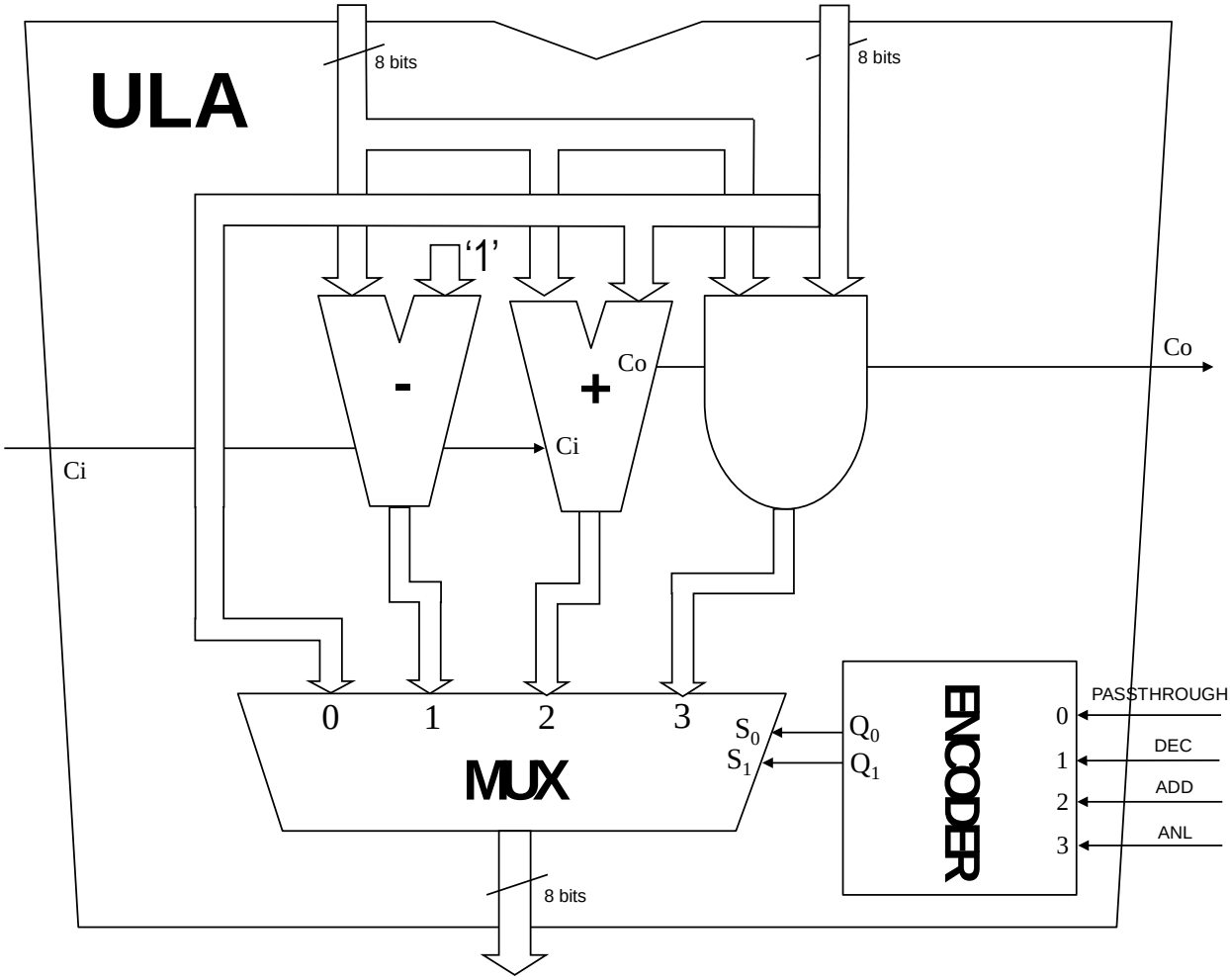
- módulo Z, um bloco de lógica combinatória, que implementa a flag de Zero e que verifica se o valor à saída do registo TEMP é igual a '0'.

Módulo ULA – Lógica combinatória que realiza operações aritméticas e lógicas

O módulo ULA propriamente dito caracteriza-se como um bloco de lógica combinatória que implementa as operações aritméticas e lógicas a 8 bits. O diagrama de blocos em anexo representa uma possível implementação da ULA que implementa as operações lógico-aritmética descritas anteriormente.

O módulo ULA contém 2 entradas de dados, correspondendo a dois operandos de 8 bits a utilizar nas operações aritméticas e lógicas suportadas. Estes dois operandos dão entrada nas diferentes funções específicas que implementam as várias operações lógico-aritmética suportadas pela ULA, a saber:

- Passagem direta (valor inalterado) do segundo operando;
- Decremento, realizado através de um subtrator do primeiro operando com valor decimal '1';
- Soma, realizado através da soma aritmética dos dois operandos;
- 'E' lógico bit-a-bit dos dois operandos, realizado por portas do tipo AND;



Neste sentido, o módulo ULA, possui quatro entradas que permitem selecionar a operação pretendida. Estas entradas entram diretamente para um codificador (ENCODER) de 4-para-2 que codifica em 2 bits de saída as 4 entradas de seleção referentes à operação lógico-aritmética pretendida. Este código de 2 bits segue depois como entrada de seleção num *multiplexer* (MUX) que permite definir qual a saída da ULA e que corresponde à operação lógico-aritmética selecionada.

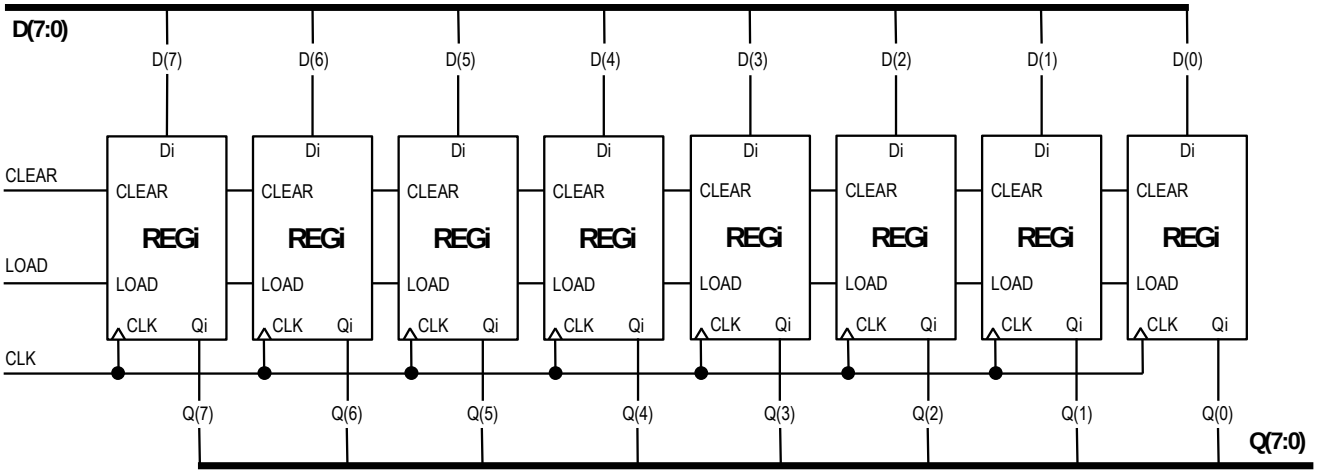
Finalmente, o módulo ULA possui um saída C_o que representa o transporte gerado nas operações aritméticas. Analogamente, possui uma entrada de transporte C_i a afetar nas operações aritméticas a realizar pela Unidade Lógica e Aritmética.

Módulo TEMP – Registo síncrono 8 bits para guardar dados (temporariamente)

O Módulo TEMP é um registo síncrono de 8 bits com CLEAR e LOAD. O registo síncrono TEMP de 8 bits, possui duas entradas síncronas de controlo (CLEAR e LOAD) e que permitem os seguintes modos de funcionamento do registo:

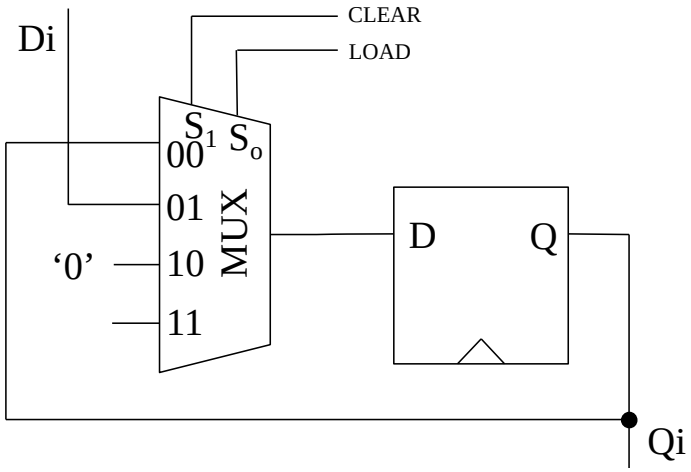
CLEAR	LOAD	Funcionamento do Registo
0	0	Mantém o conteúdo do registo inalterado
0	1	Carrega o registo com os dados à entrada (carregamento paralelo)
1	0	Apaga (sincronamente) o conteúdo do registo

O registo TEMP de 8 bits tem como possível implementação a apresentada no esquema abaixo, realizado por composição modular de oito registos de 1 bit com funcionalidades similares. O registo TEMP possui assim uma entrada de dados D de oito bits – *bus* D(7:0) –, uma saída de dados Q de oito bits – *bus* Q(7:0) –, e os dois sinais de controlo (CLEAR e LOAD).



Cada elemento (REGi) do registo REG (composto então por oito destes elementos, isto é, com i entre 0 e 7) dispõe de duas entradas síncronas para controlar o modo de funcionamento:

- CLEAR, que quando ativo, permite limpar o valor armazenado no elemento de memória (flip-flop);
- LOAD, que habita o carregamento do valor presente na entrada Di (Dados índice i) no flip-flop.

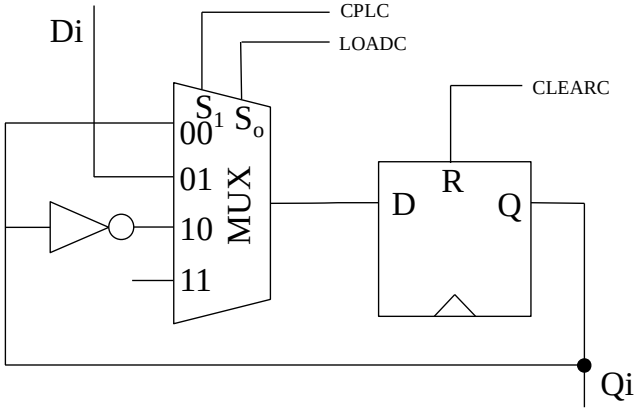


Módulo C – Registo síncrono de 1 bit que implementa a flag de Carry

O módulo C, implementa a flag de Carry da ULA, permitindo guardar o valor do transporte das operações aritméticas. O módulo C é realizado por um elemento de memória de 1 bit e contém as entradas síncronas de controlo CLEARC, LOADC e CPLC, que habilitam o seguinte funcionamento:

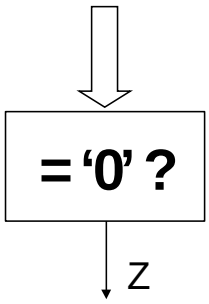
CLEARC	LOADC	CPLC	Funcionamento do Registo
1	X	X	Apaga (sincronamente) o conteúdo do registo
0	0	0	Mantém o conteúdo do registo inalterado
0	1	0	Carrega o registo com os dados à entrada
0	0	1	Complementa o conteúdo do registo

Uma possível implementação do registo C é aquela apresentada em seguida. Contém um elemento de memória de 1 bit (flip-flip) com entrada de R[eset] síncrona que permite apagar o seu conteúdo e que liga a entrada CLEARC do registo. Inclui ainda um *multiplexer* com 2 entradas de seleção para 4 entradas de dados, que determina genericamente os três restantes modos de funcionamento do registo C: 1) manter o valor do registo; 2) carregar o registo com o valor do transporte à entrada; e 3) complementar o conteúdo do registo.



Módulo Z – Lógica combinatória que implementa a flag de Z(ero)

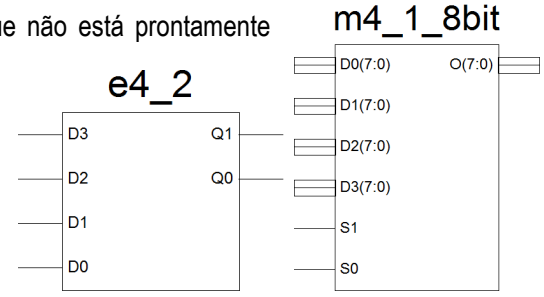
O módulo Z implementa a flag de Z(ero) da Unidade Lógica e Aritmética. É realizado à custa de simples lógica combinatória que verifica se o valor à entrada do módulo (e à saída do registo TEMP) é igual a '0'. Uma possível implementação do módulo Z é através de um comparador aritmético.



Notas sobre a implementação disponibilizada

O ambiente ISE WebPack não disponibiliza *multiplexers* com entradas de dados acima de 1bit e, para a implementação sugerida do módulo ULA, é necessário um *multiplexer* de oito entradas de dados de 8 bits cada, com duas entradas de seleção, e saída de dados de 8 bits. Também não disponibiliza módulos descodificadores. Assim, estão disponíveis os módulos m4_1_8bit (*multiplexer* de 8 bits, com duas entradas de seleção para quatro entradas de dados) e e4_2 (codificador de 4-para-2).

Para implementar a função AND a 8 bits, módulo que não está prontamente disponíveis na biblioteca de símbolos do ISE WebPack, foi utilizada a funcionalidade de iteração de instâncias. Para tal, deverá selecionar um qualquer símbolo elementar, fazer 'Edit' –> 'Rename' –> 'Rename Selected Instance...', selecionar 'Iterated instance name' e definir um 'Starting Value' e um 'Ending Value', por exemplo, 'Starting Value'=7 e um 'Ending Value'=0 para iterar 8 bits!



Simulação

Poderá proceder à simulação da ULA para validar a sua implementação (e perceber alguns detalhes do seu funcionamento). Para tal, poderá, se desejar, utilizar as definições de simulação abaixo. Basicamente, é definindo o valor decimal 7 no bus de entrada DATA que é guardado no registo TEMP. Seguidamente é decrementado o valor do registo TEMP (de 7 para 6). Depois é somado o valor no bus de entrada DATA (decimal 7) ao valor presente no registo TEMP (decimal 6), sendo o resultado guardado no registo TEMP (decimal 13). Finalmente, é realizado um AND bit-a-bit entre o valor no bus de dados ('00000111') e o valor no registo TEMP ('00001101') sendo guardado no registo TEMP ('00000101').

```
tb : PROCESS
  BEGIN
  CLEARTEMP <= '0'; LOADTEMP <= '0';
  LOADC <= '0'; CLEARC <= '0'; CPLC <= '0';
  PASSTHROUGH <= '0'; DEC <= '0'; ADD <= '0'; ANL <= '0';
  DATA(7)<= '0'; DATA(6) <= '0'; DATA(5) <= '0'; DATA(4) <= '0';
  DATA(3)<= '0'; DATA(2) <= '1'; DATA(1) <= '1'; DATA(0) <= '1';
  CLK <= '0'; wait for 50 ns;
  CLEARTEMP <= '1';
  CLEARC <= '1';
  CLK <= '1'; wait for 50ns;
  CLEARTEMP <= '0'; PASSTHROUGH <= '1'; LOADTEMP <= '1';
  CLK <= '0'; wait for 50ns; CLK <= '1'; wait for 50ns;
  PASSTHROUGH <= '0'; DEC <= '1'; LOADTEMP <= '1';
  CLK <= '0'; wait for 50ns; CLK <= '1'; wait for 50ns;
  DEC <= '0'; ADD <= '1'; LOADTEMP <= '1'; LOADC <= '1';
  CLK <= '0'; wait for 50ns; CLK <= '1'; wait for 50ns;
  ADD <= '0'; ANL <= '1'; LOADTEMP <= '1'; LOADC <= '0';
  CLK <= '0'; wait for 50ns; CLK <= '1';
  WAIT;
END PROCESS;
```

ANEXO B: Realização de um multiplicador (muito) simples

Objetivos

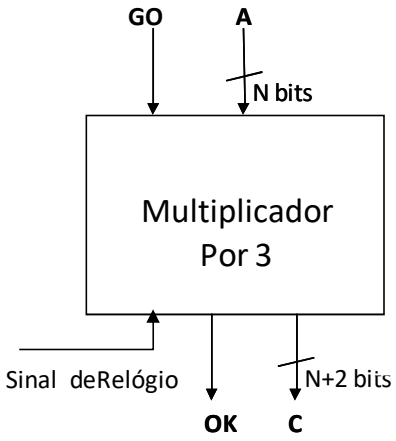
Verificação de funcionamento de sistema digital realizado através de uma arquitetura de transferência entre registos e decomposto logicamente em parte de dados e parte de controlo.

Multiplicador 3*A

Descrição

O funcionamento do sistema que se pretende desenvolver pode ser descrito genericamente da seguinte forma: o utilizador atualiza o valor do operando de entrada A, após o que atua um sinal de entrada (GO) que provoca a realização da operação de multiplicação obtendo-se 3*A na saída. Um sinal de fim de operação (OK) será ativado quando o resultado estiver disponível na saída.

O algoritmo que se pretende analisar utiliza somas sucessivas para proceder ao cálculo da multiplicação. Desta forma, a operação 3*A será realizada através da realização de 3 somas de A, isto é, 0+A+A+A.

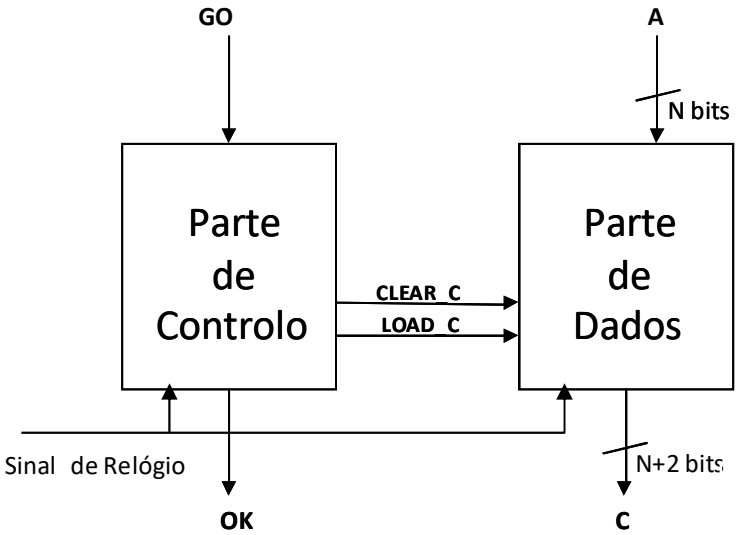


Análise

Considere-se, como ponto de partida, uma caracterização genérica contendo uma parte de controlo e uma parte de dados:

- a parte de dados dispõe de entradas de dados (contendo o operando A), de saídas de dados (contendo o resultado da operação C), e de 2 entradas para interligação à parte de controlo.
- a parte de controlo, a implementar através de um circuito sequencial síncrono, sob controlo de um sinal de relógio, dispõe de uma entrada, o sinal de início de operação GO, e de uma saída OK, permitindo sinalizar o fim de operação, bem como de 2 saídas que permitirão interatuar adequadamente com a parte de dados;

A figura seguinte ilustra a decomposição proposta para o sistema.



Especificação de implementação da parte de dados

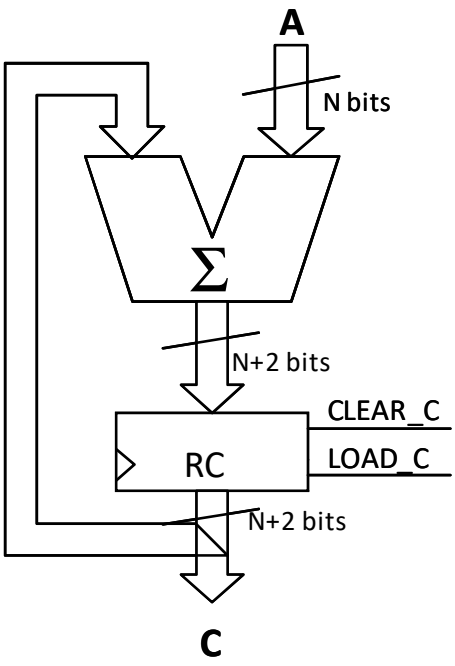
Quanto à parte de dados, é necessário considerar uma arquitectura que suporte a operação básica do nosso algoritmo: a utilização de um registo acumulador, que irá memorizando as somas parciais obtidas durante o processo e um somador do valor acumulado com A.

Uma arquitectura possível é a que se apresenta (ao lado) através de um diagrama de blocos, contendo um somador e um registo com duas entradas de controlo.

As funções a implementar por cada um dos blocos são:

- o somador implementa um somador de N+2 bits com N+2 bits; deverá ser tido em consideração o facto da parte superior da entrada associada a A conter zeros (isto é, considerando N=5 por exemplo, a entrada do somador tem oito bits, 3 dos quais estão a zero e o resto vem do exterior);

- o registo RC dispõe de duas entradas síncronas: CLEAR_C, que quando activo permite limpar o valor armazenado no registo e LOAD_C que permite o carregamento do valor presente na entrada do registo.



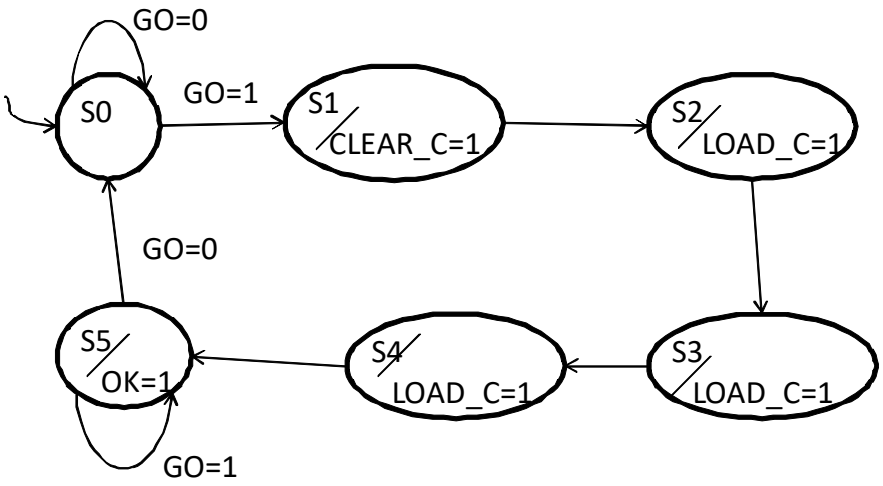
Especificação de implementação da parte de controlo

Quanto à parte de controlo, recomenda-se que seja uma máquina de estados, síncrona, arquitectura de Moore. A figura seguinte apresenta um diagrama de estados, coerente com a arquitectura de dados apresentada:

- os estados S0 e S5 são os estados inicial e final, sensíveis ao sinal de entrada GO;

- o estado S1 é o responsável pela inicialização da arquitectura, em particular por limpar RC;

- os estado S2, S3 e S4 são responsáveis pelas somas sucessivas.

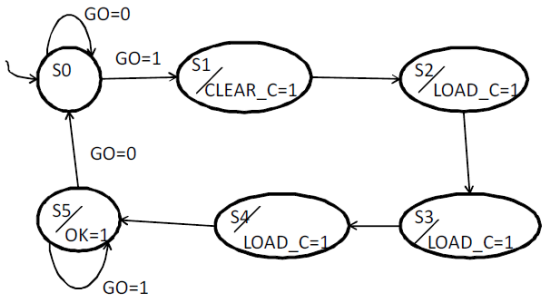


Resolução

a) Parte de Controlo

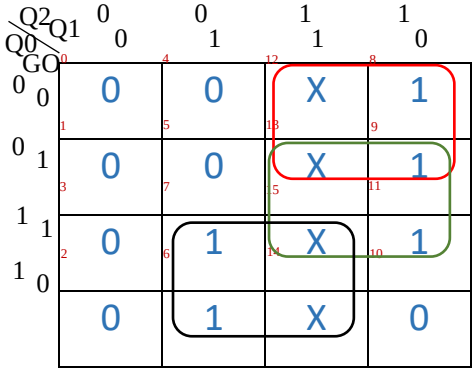
Como só temos 6 estados apenas precisamos de 3 bits para codifica-los.

Estado	*Codificação*
S_0	000
S_1	001
S_2	010
S_3	011
S_4	100
S_5	101

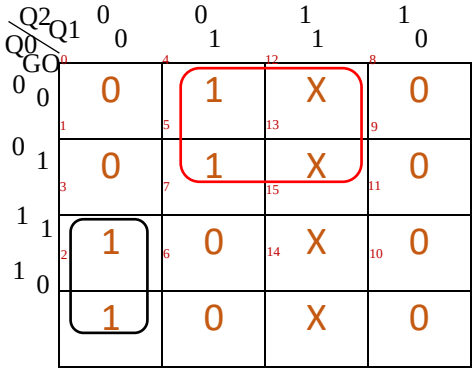


Q_2	Q_1	Q_0	GO	Q'_2	Q'_1	Q'_0	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0
0	0	1	1	0	1	0	0	1	0
0	1	0	0	0	1	1	0	1	1
0	1	0	1	0	1	1	0	1	1
0	1	1	0	1	0	0	1	0	0
0	1	1	1	1	0	0	1	0	0
1	0	0	0	1	0	1	1	0	1
1	0	0	1	1	0	1	1	0	1
1	0	1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1	0	1
1	1	0	0	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

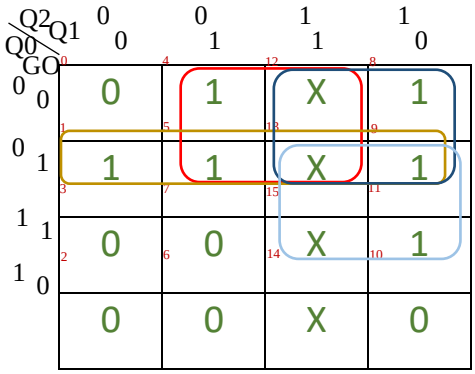
Tabela de transição de estados e entradas de flip-flops, considerando que se escolhem flip-flops D (Nesta tabela, temos de incluir a dependência do GO, porque esta variável influencia a transição de estados):



$$D_2 = Q_2.\overline{Q_0} + Q_2.GO + Q_1.Q_0$$



$$D_1 = Q_1.\overline{Q_0} + \overline{Q_2}.\overline{Q_1}.Q_0$$

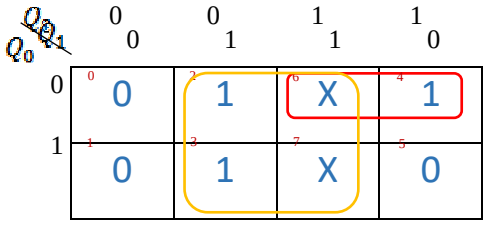


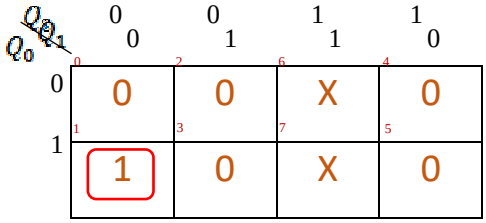
$$D_0 = Q_1.\overline{Q_0} + Q_2.\overline{Q_0} + \overline{Q_0}.GO + Q_2.GO$$

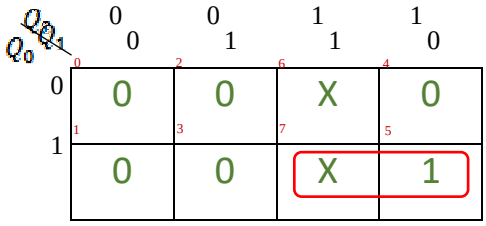
Para a codificação das saídas, apenas precisamos do estado em que a máquina se encontra, uma vez que numa máquina de Moore as saídas ficam ativas durante os estados.

Tabela de codificação de saídas:

Q_2	Q_1	Q_0	$Load_C$	$Clear_C$	OK
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	X	X	X
1	1	1	X	X	X

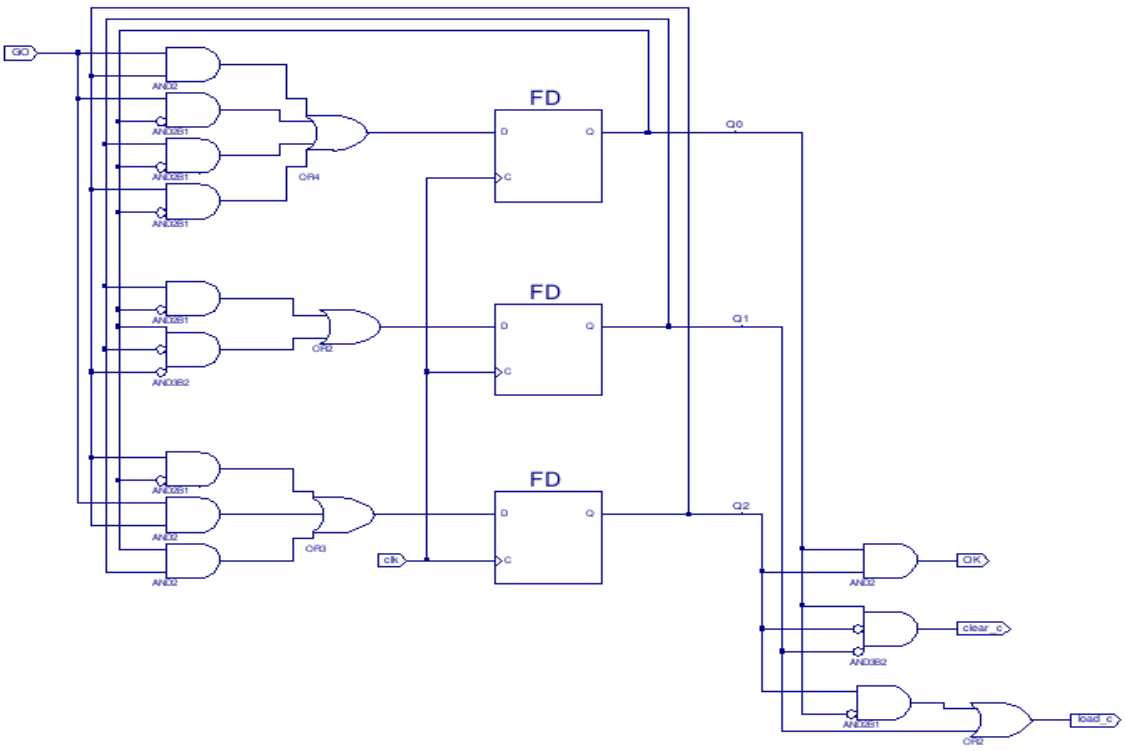






$$Load_C = Q_2.\overline{Q_0} + Q_1$$
$$Clear_C = \overline{Q_2}.\overline{Q_1}.Q_0$$
$$OK = Q_2.Q_0$$

Após a criação das tabelas e da obtenção das funções através de Mapas de Karnaugh, conseguimos chegar ao seguinte esquemático:

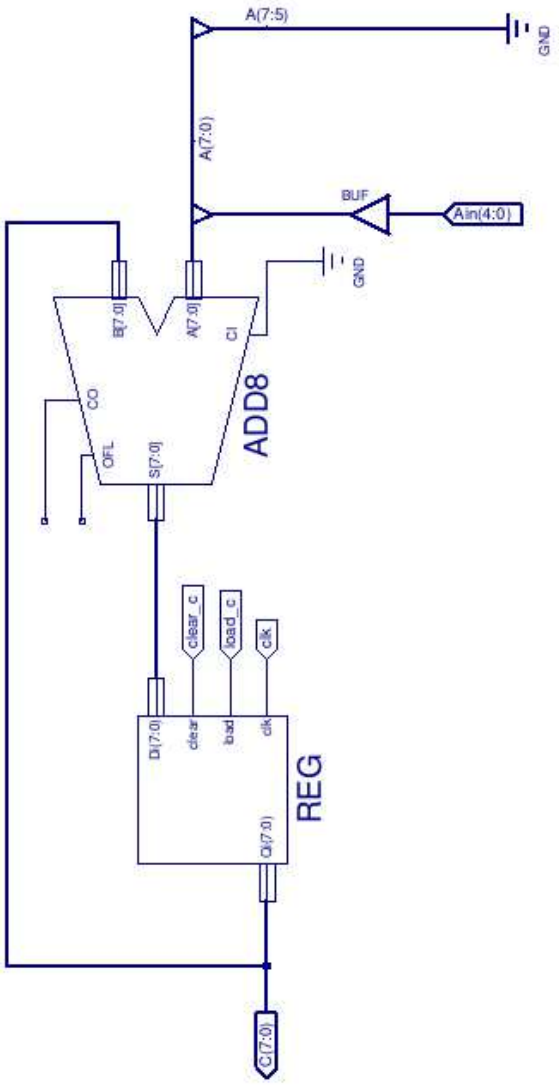


Criámos as funções de "next state" através das entradas dos flip-flops (D0, D1 e D2) e implementámos as saídas através das suas saídas (Q0, Q1 e Q2).

b) Parte de Dados

Utilizando um registo com controlo de CLEAR e LOAD, um somador de 8 bits e barramentos para interligação, a parte de dados é implementada. Os "BUF" utilizados no esquemático que se apresenta são para poder colocar apenas alguns bits de um bus com valores obtidos através de inputs (Ain0-Ain4, onde vamos colocar o valor a multiplicar).

Para implementar o esquemático da imagem em baixo, tem de se iterar o "gnd" (ground) da seguinte forma. Carrega-se com o botão do lado direito do rato no símbolo "GND" → Symbol → Rename Selected Instance, coloca-se um certo na caixa que diz "Iterated Instance Name" e itera-se para 3 bits (o starting value e o ending value têm de perfazer os 3 bits necessários para o bus).



c) Final

Após a criação das partes de dados e de controlo, cria-se um símbolo para cada parte, unindo-os num esquemático final. Neste final, ligam-se as entradas de CLOCK, GO (para iniciar o processo de multiplicação) e o valor de A (valor a multiplicar).

Como saídas, ter-se-á o OK, que nos avisa quando a multiplicação terminou e o C, o valor final obtido da multiplicação.