Sistemas Lógicos (MiEI) – 2016/2017 – 1º Semestre

Relatório Trabalho Final

Cálculo de expressão

((A AND KAND) DEC KDEC) + K1 * A

Realizado por:

[João Fernandes – 49834

Francisco Silva – 50188

Diogo Silva – 50433]

[MiEI]

[T5]

Índice

1 Introdução

Usando os nossos números de aluno para calcular a expressão que iremos usar no resto do trabalho, na opção 1 obtivemos a expressão 5:

((A AND KAND) DEC KDEC) + K1 * A, dado que (49834+50188+50433)%6 = 5

Como (49834+50188+50433)%9 = 2, então os valores que iremos usar na expressão serão os seguintes:

Kand = 3, Kdec = 1, K1 = 3

Completando a expressão com os valores obtidos, temos que

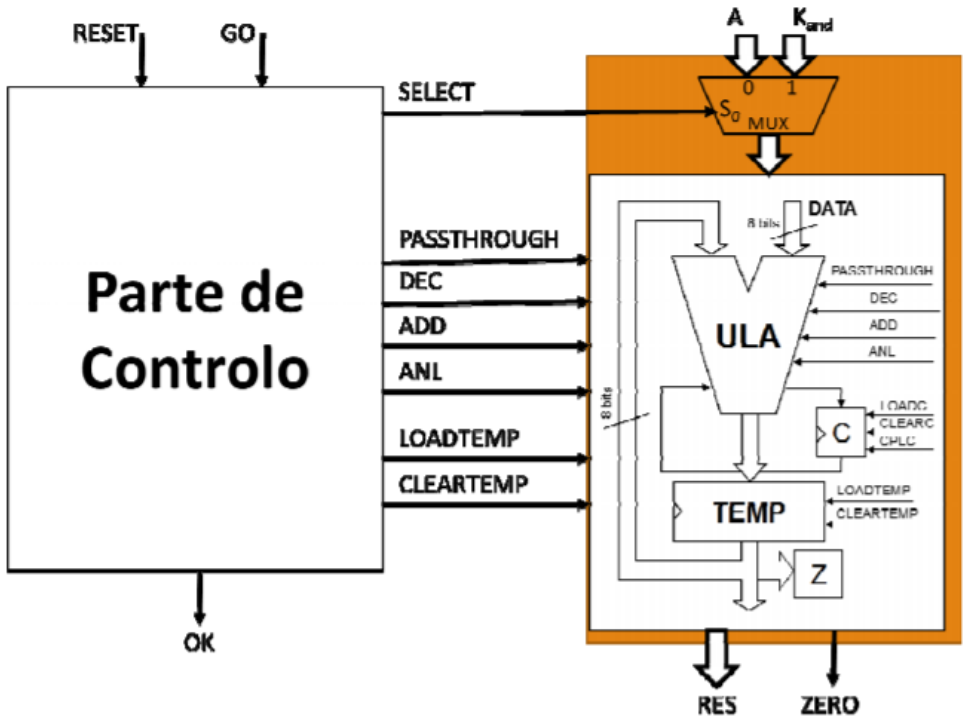
((A AND 3) DEC 1) + 3 * A, que será a expressão utilizada no trabalho.

O sistema implementado irá calcular o valor da expressão acima mencionada dado um determinado valor de A.

2 Análise

2.1 Descrição de funcionamento global

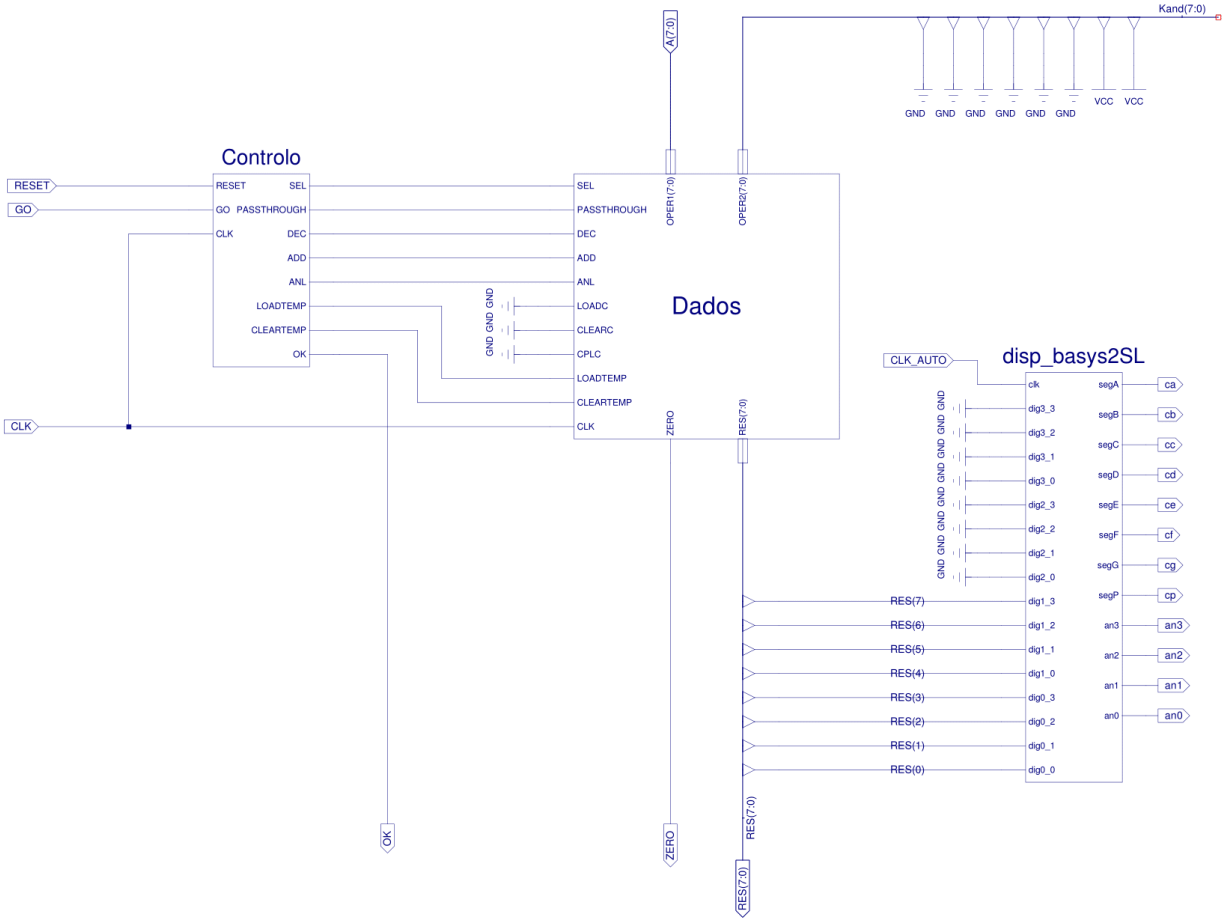
Este sistema está bipartido em duas partes, de controlo e de dados, o primeiro consiste numa máquina de estados que controla e comunica com a parte de dados, informando-a o tipo de operações a efetuar para completar a expressão dada. Esta comunicação é feita da forma abaixo descrita:



O GO ao ser detetado inicia a máquina de estados, que será introduzida em detalhe mais abaixo, que irá em fim, calcular a expressão e detetar caso este resultado seja igual a 0 (através da saída ZERO da parte de dados).

O RESET da parte de controlo está ligado ao CLEAR de cada um dos flip-flops D utilizados na máquina de estados, dado que, como o objetivo do RESET é retornar ao estado inicial, e este é codificado como “000”, ao fazer CLEAR de todos os flip-flops, iremos colocar os seus valores armazenados para 0, e por isso, retornamos ao estado inicial (S0 - 000).

Abaixo está o esquemático do sistema em completo, tal como foi implementado:



2.2 Situações geradores de resultados iguais a 0

Dado a expressão que obtivemos, não existem situações em que o resultado da expressão seja igual a 0.

Para chegar a tal conclusão, implementámos um programa em java cujo objetivo era calcular o resultado da expressão para todos os valores possíveis de A (entre 0 e 255).

O que verificámos foi que nenhum desses valores resultava num cálculo igual a 0. Além disso, quando A=0, o valor obtido seria -1, no entanto, como o sistema não foi projectado para lidar com resultados negativos, o -1 causa um overflow negativo do sistema, e o resultado aparenta ser FF (255). Pelo outro lado, quando A>85, o resultado será superior a 255 (o maior nº que se pode representar em 8 bits), e por isso não poderão ser correctamente apresentados usando este sistema. O que se verifica é que, de novo, ocorre um overflow positivo, que causa o sistema a mostrar o excedente entre o resultado e 255, ou seja, se o resultado for 256, irá ser mostrada a resposta 0, e caso seja, 257, será mostrada o resultado 1, e por conseguinte.

Código utilizado para chegar a esta conclusão:

```
public class Main {

	static int kAnd=3;
	static int kDec=1;
	static int k1=3;

	public static int[] decToBinary(int numb){
		int[] binary = new int[8];
		int dec = numb;
		for(int i=7;i>=0;i--){
			double value = Math.pow(2, i);
			if(value <= dec){
				dec -= value;
				binary[i]=1;
			} else{
				binary[i]=0;
			}
		}
		return binary;
	}

	public static double binaryToDec(int[] binary){
		double res = 0;
		for(int i=0;i<8;i++){
			double value = Math.pow(2, i);
			res += binary[i]*value;
		}
		return res;
	}

	public static double kand(int a,int b){
		int[] a2 = decToBinary(a);
		int[] b2 = decToBinary(b);
		int[] binary = new int[8];
		for(int i=0;i<8;i++){
			binary[i]=(a2[i] & b2[i]);
		}
		return binaryToDec(binary);
	}

	public static double calcExp(int A){
		return (kand(A,kAnd)-kDec)+k1*A;
	}

	public static void main(String[] args) {
		for(int i=0;i<256;i++){
			System.out.printf(i+": %.2f\n",calcExp(i));
		}
	}
}
```

Resultado da execução do código

0: -1,00

1: 3,00

2: 7,00

3: 11,00

4: 11,00

5: 15,00

6: 19,00

7: 23,00

8: 23,00

9: 27,00

10: 31,00

…

85: 255,00

86: 259,00 (após o valor 85, o valor apresentado não pode ser mostrado em 8 bits, o que irá causar um overflow do sistema)

87: 263,00

88: 263,00

89: 267,00

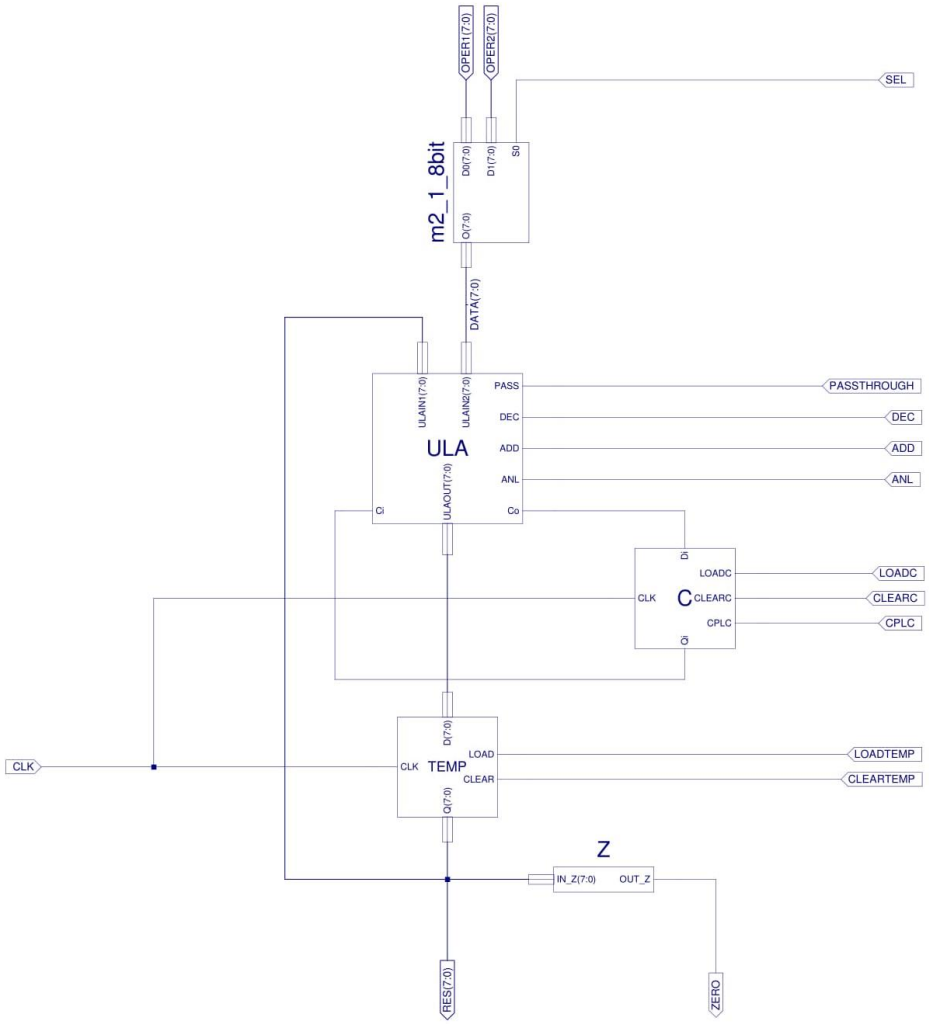
90: 271,00

…

(todos os valores seguintes são crescentes ou iguais ao anterior)

3 Parte de Dados - descrição

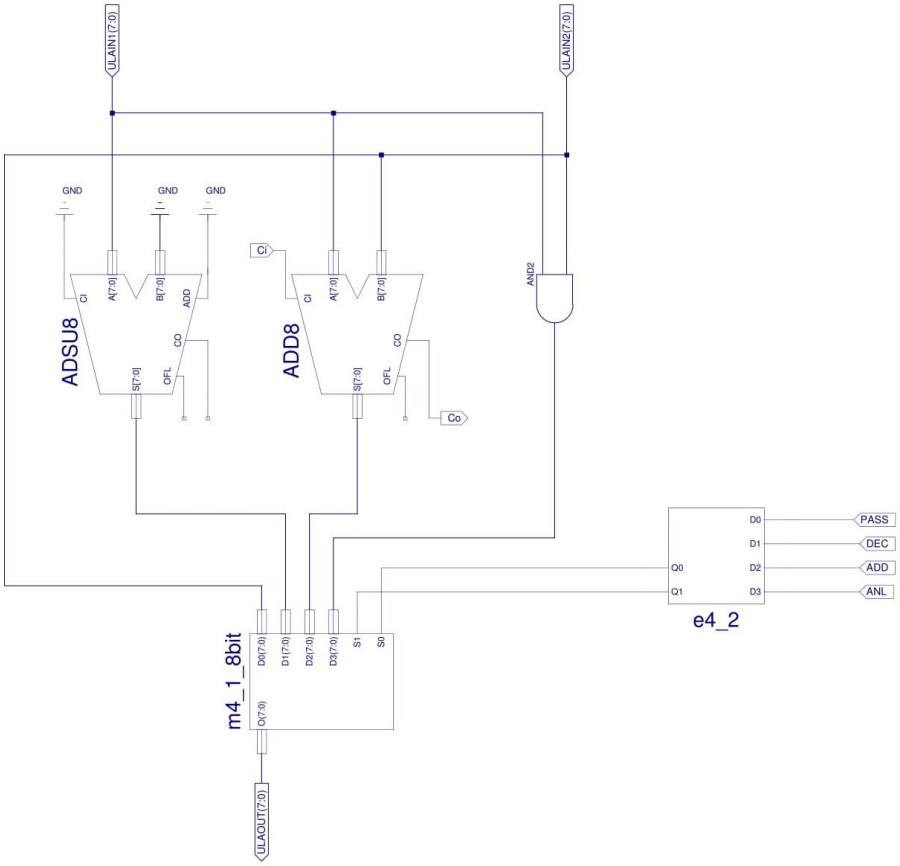
A parte de dados deste sistema é caracterizada pelo seguinte esquema:



A Parte de Dados é constituida por:

- Uma Unidade Lógica e Aritmética (ULA), capaz de realizar operações aritméticas (decremento e soma) e operações lógicas (‘e’ unicamente);
- As flags de C[arry] (Transporte de operações aritméticas);
- Módulo de Z[ero] (Comparação com Zero).

Módulo ULA:



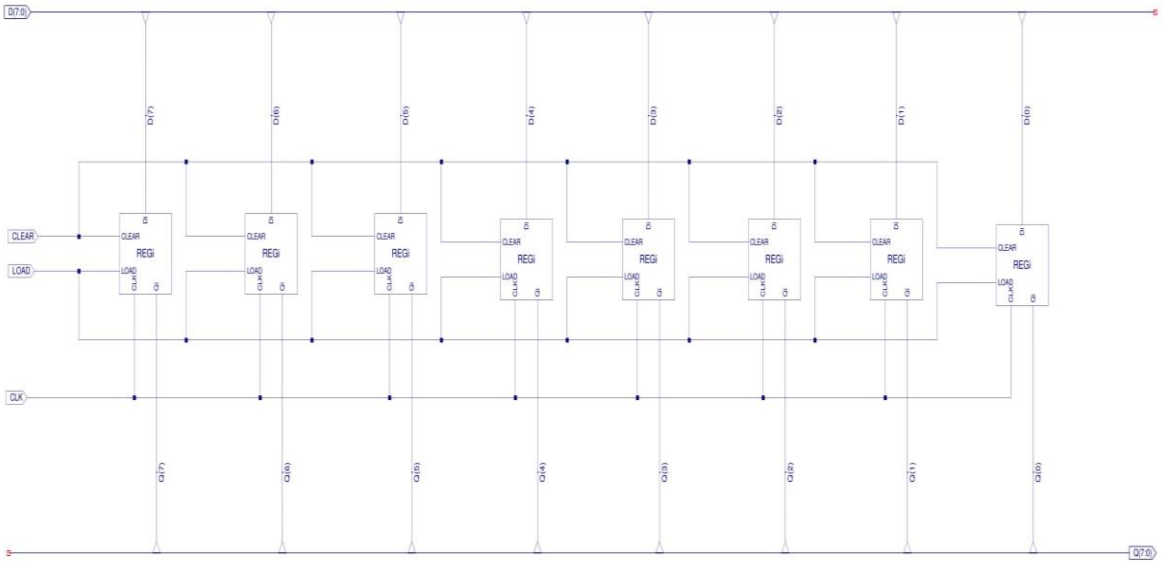
O módulo ULA caracteriza-se como um bloco de lógica combinatória que implementa as operações aritméticas e lógicas a 8 bits. O módulo ULA contém 2 entradas de dados, correspondendo a dois operandos de 8 bits a utilizar nas operações aritméticas e lógicas suportadas. Estes dois operandos dão entrada nas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

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

Este possui quatro entradas que permitem selecionar a operação pretendida, as entradas entram diretamente para um codificador (ENCODER) de 4-para-2 que define a 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 Co que representa o transporte gerado nas operações aritméticas. Analogamente, possui uma entrada de transporte Ci a afetar nas operações aritméticas a realizar pela Unidade Lógica e Aritmética.

Módulo Temp:



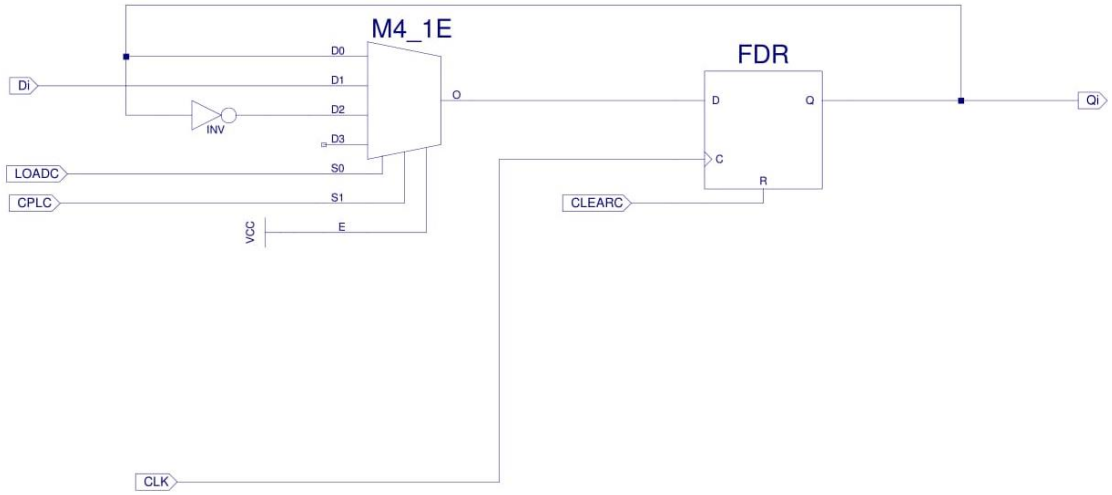
O Módulo TEMP é um registo síncrono de 8 bits com CLEAR e LOAD, que possui duas entradas síncronas de controlo:

• 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.

O registo TEMP de 8 bits tem como possível implementação a apresentada no esquema acima. O registo TEMP possui assim uma entrada de dados D de oito bits, uma saída de dados Q de oito bits, e os dois sinais de controlo (CLEAR e LOAD).

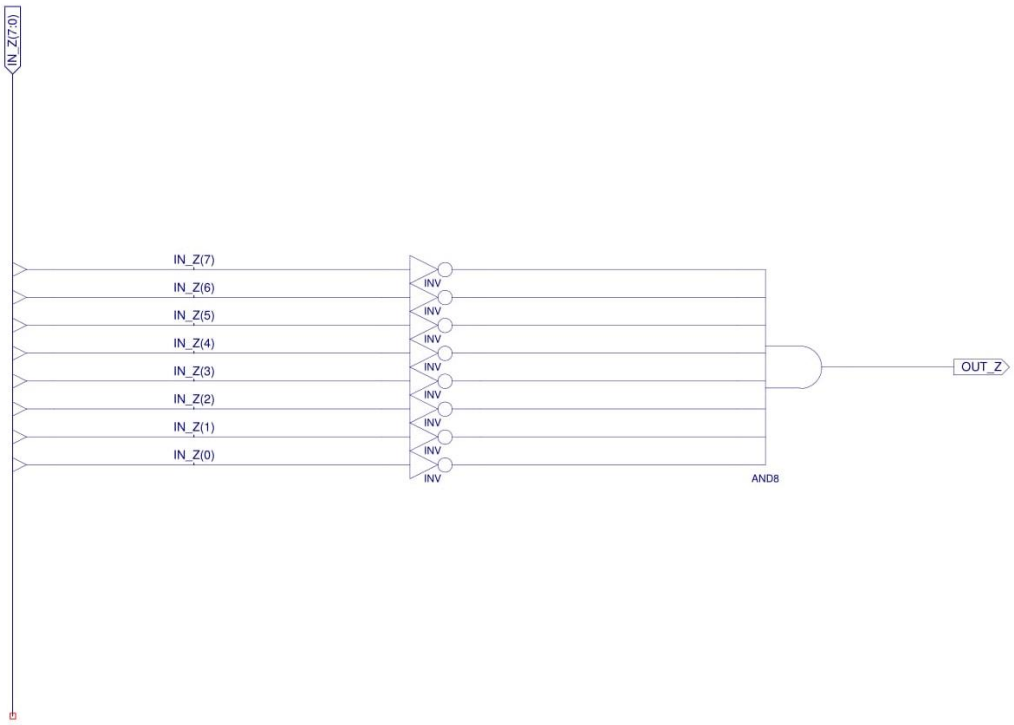
Módulo C:



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 - Apaga (sincronamente) o conteúdo do registo;
- LOADC - Carrega o registo com os dados à entrada;
- CPLC - Complementa o conteúdo do registo;

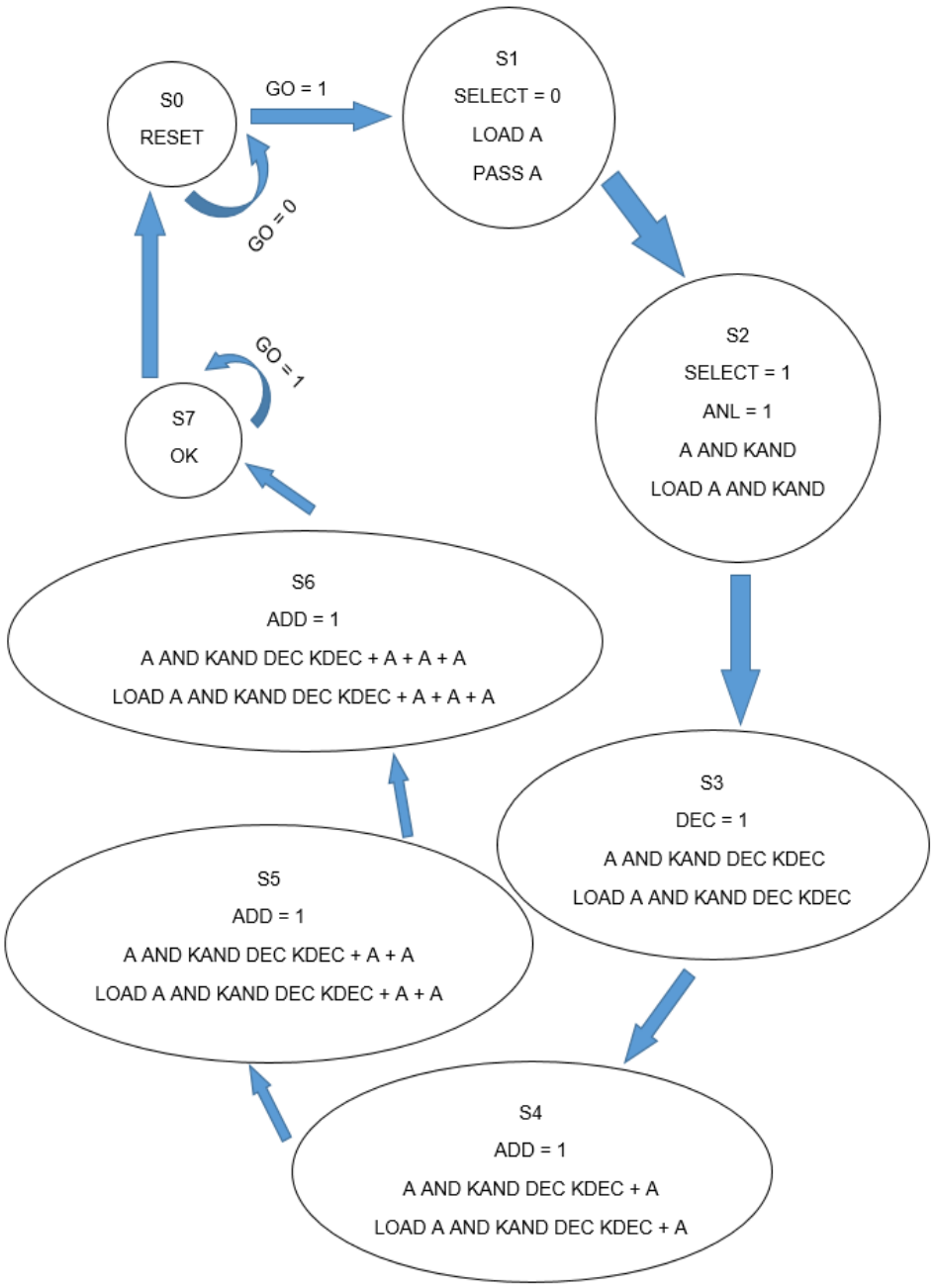
Módulo de Z:



O módulo Z está associado a uma saída de 1 bit (ZERO) que se apresenta ligada caso o resultado da operação seja igual a 0.

4 Síntese – Parte de Controlo

4.1 Diagrama de Estados



4.2 Tabela de Transição de Estados

Estado	G0 =0	GO = 1
S0	S0	S1
S1	S2	S2
S2	S3	S3
S3	S4	S4
S4	S5	S5
S5	S6	S6
S6	S7	S7
S7	S0	S7

4.3 Codificação de Estados

Estado	Codificação
S0	000
S1	001
S2	010
S3	011
S4	100
S5	101
S6	110
S7	111

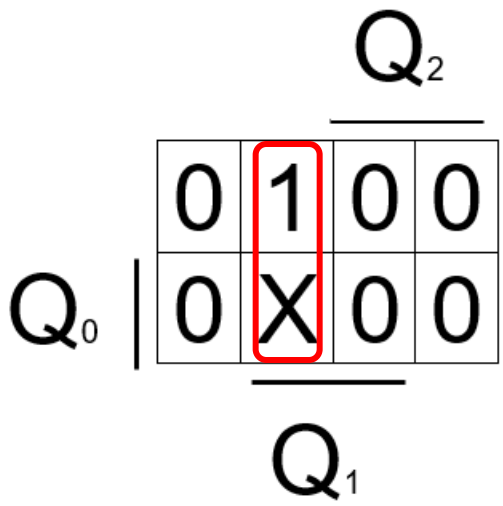
4.4 Tabela de Transição de Estados codificados, saídas e entradas dos Flip-flops.

Q2	Q1	Q0	GO	Q2'	Q1'	Q0'	D2	D1	D0
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	1	1	0	1	1	0
1	0	1	1	1	1	0	1	1	0
1	1	0	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1

4.5 Expressões simplificadas das saídas por Mapas de Karnaugh

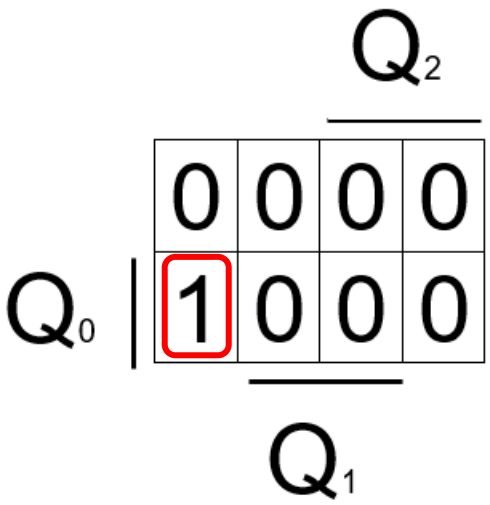
Q2	Q1	Q0	SEL	PASS	DEC	ADD	ANL	LOAD	CLEAR	OK
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	1	0	0
0	1	0	1	0	0	0	1	1	0	0
0	1	1	X	0	1	0	0	1	0	0
1	0	0	0	0	0	1	0	1	0	0
1	0	1	0	0	0	1	0	1	0	0
1	1	0	0	0	0	1	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1

SEL



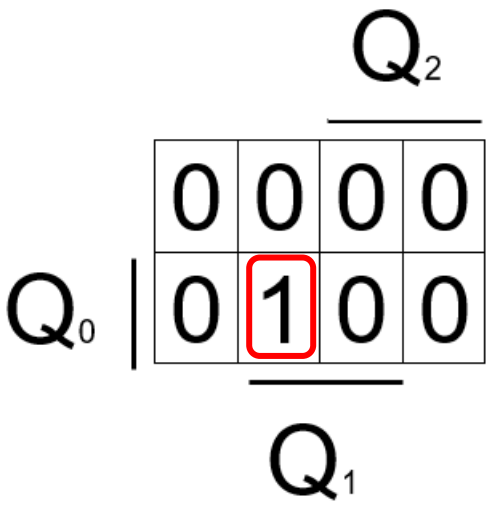
$$SEL = \overline{Q2} \cdot Q1$$

PASS



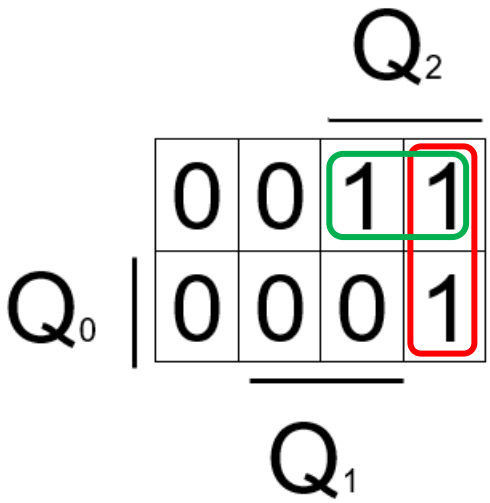
$$\text{PASS} = \overline{Q2} \cdot \overline{Q1} \cdot Q0$$

DEC



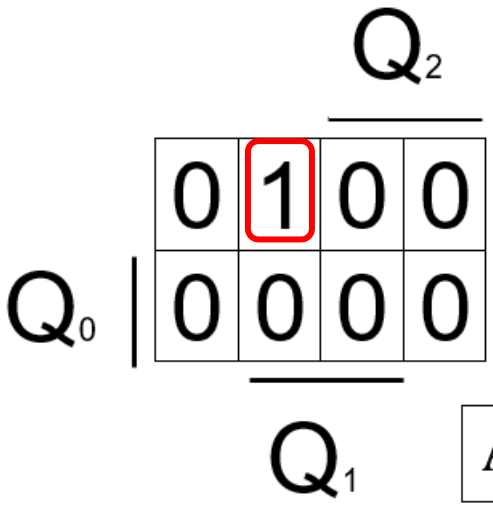
$$\text{DEC} = \overline{Q2} \cdot Q1 \cdot Q0$$

ADD



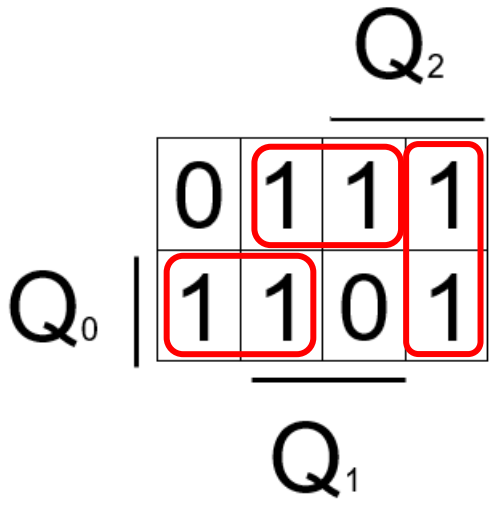
$$\text{ADD} = Q2 \cdot \overline{Q1} + Q2 \cdot \overline{Q0}$$

ANL



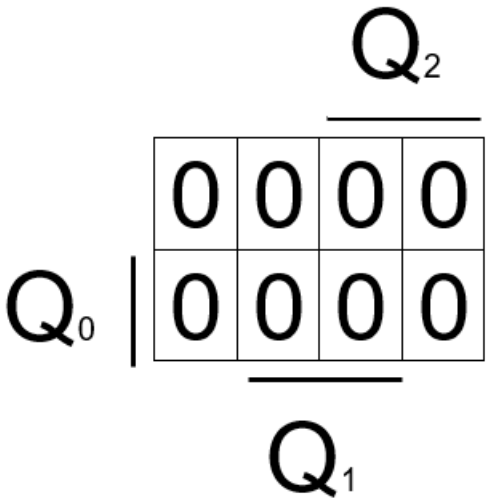
$$\text{ANL} = \overline{Q2} \cdot Q1 \cdot \overline{Q0}$$

LOAD



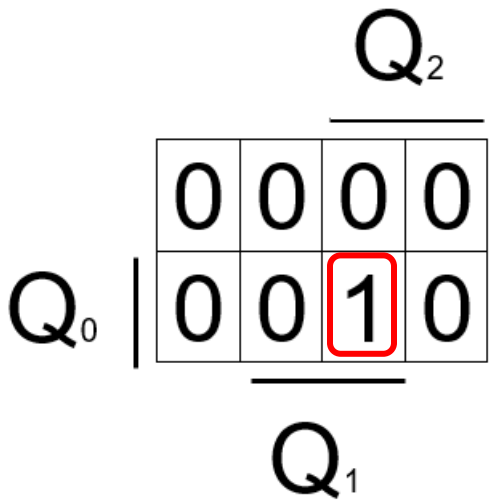
$$\text{LOAD} = \overline{Q2} \cdot Q0 + Q1 \cdot \overline{Q0} + Q2 \cdot \overline{Q1}$$

CLEAR



$$\text{CLEAR} = 0$$

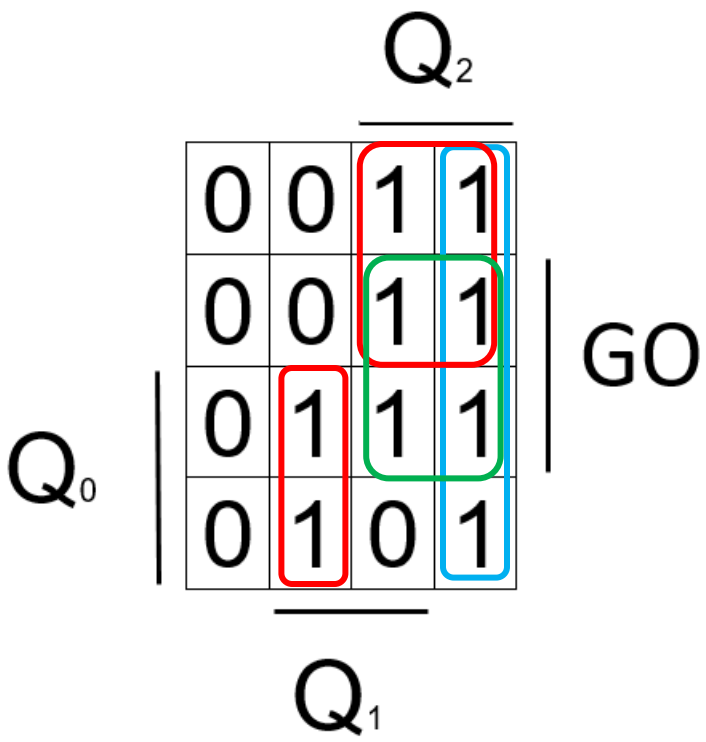
OK



$$\mathrm{OK} = Q2 \cdot Q1 \cdot Q0$$

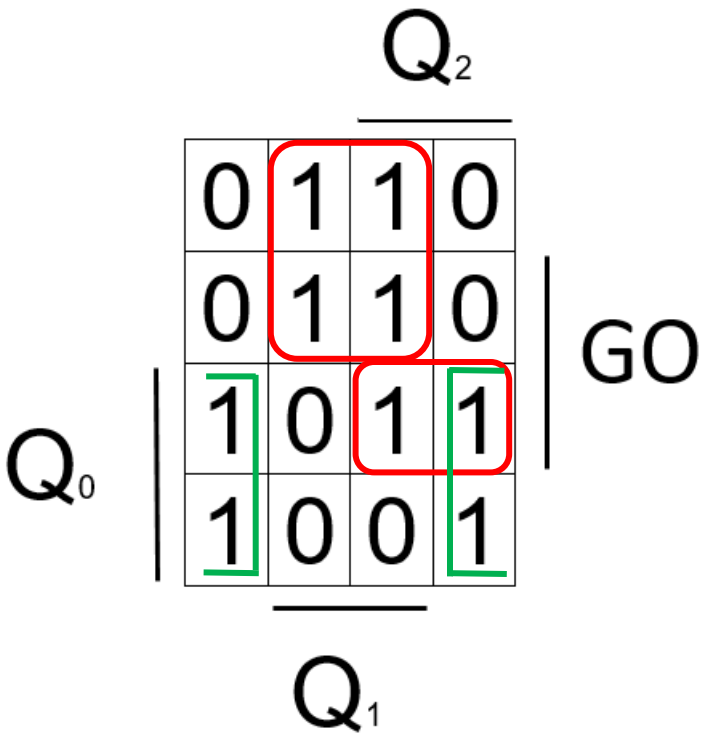
4.6 Expressões simplificadas das entradas dos Flip-flops por Mapas de Karnaugh

D2



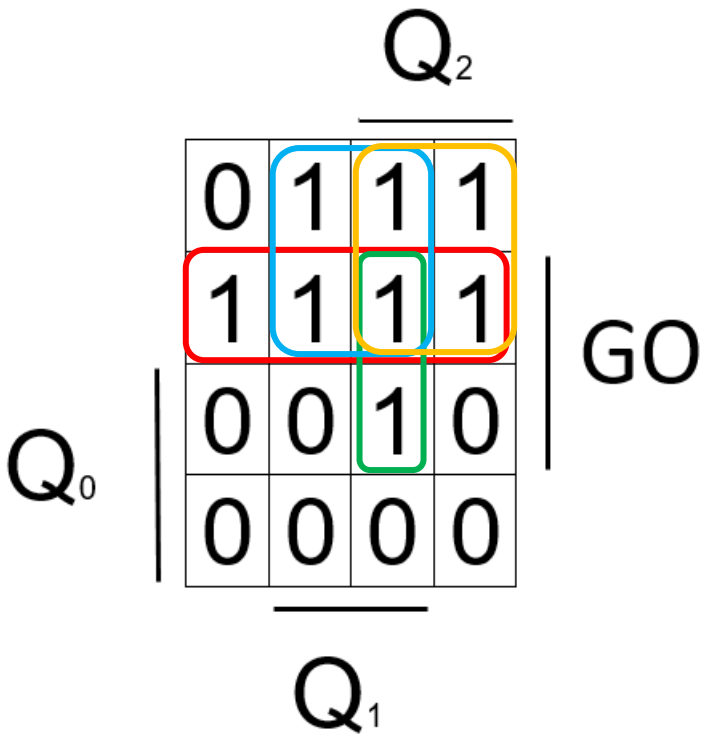
$$D2 = Q2 \cdot \overline{Q1} + Q2 \cdot \overline{Q0} + Q2 \cdot G0 + \overline{Q2} \cdot Q1 \cdot Q0$$

D1



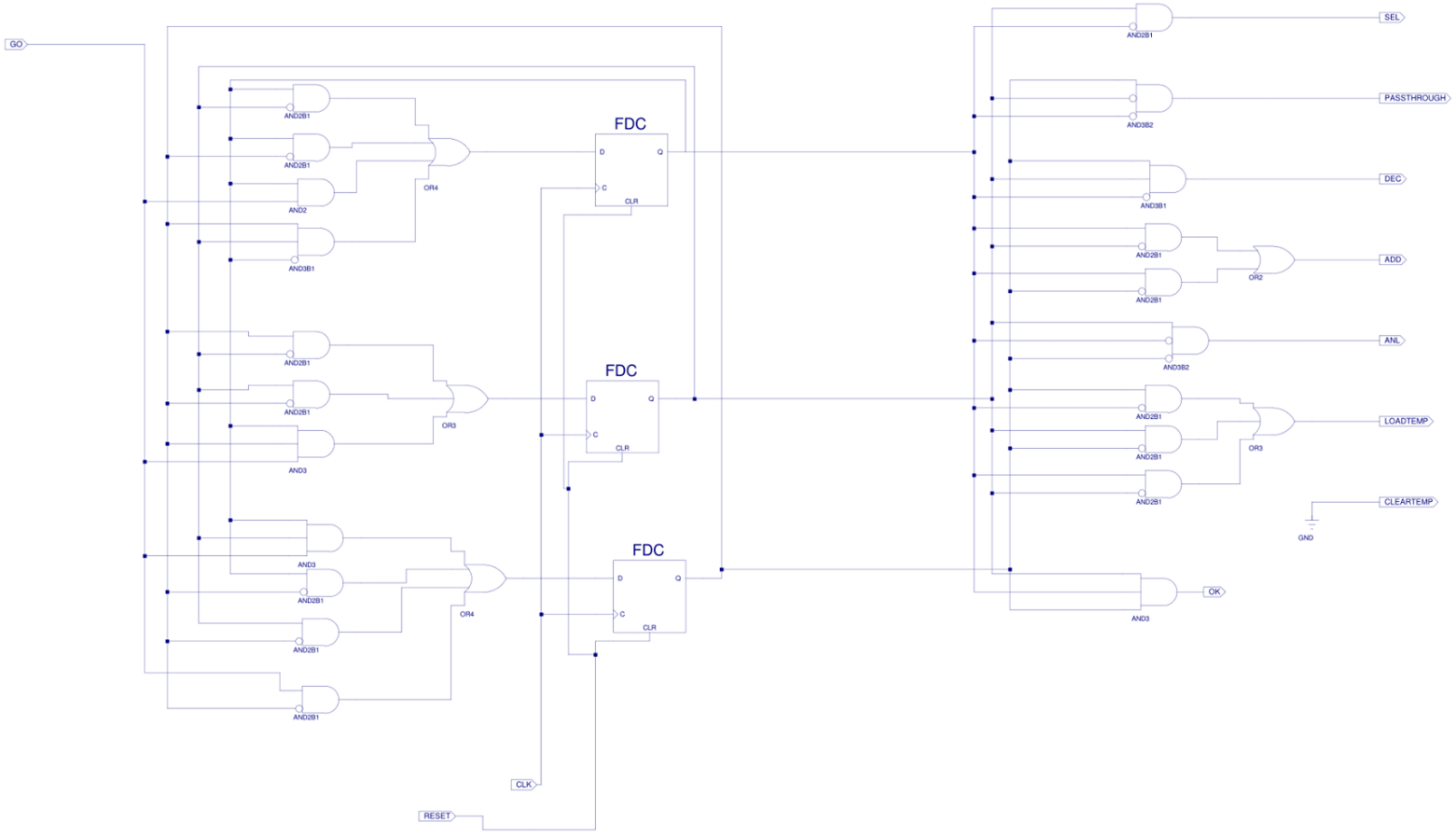
$$D1 = \overline{Q1} \cdot Q0 + Q1 \cdot \overline{Q0} + Q2 \cdot Q0 \cdot GO$$

D0



$$D0 = \overline{Q0} \cdot GO + Q1 \cdot \overline{Q0} + Q2 \cdot \overline{Q0} + Q2 \cdot Q1 \cdot GO$$

4.7 Esquemático da Parte de Controlo

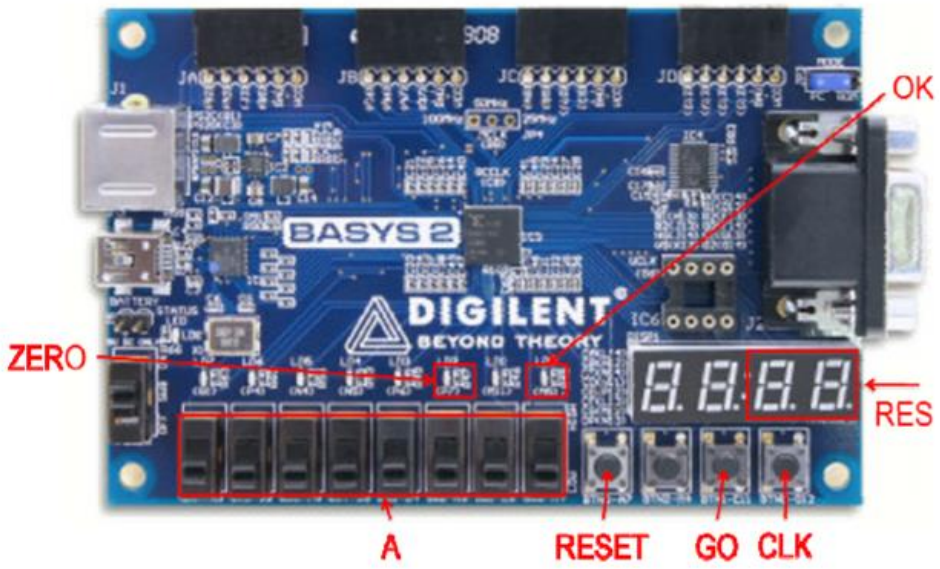


5 Implementação na FPGA Spartan 3E

Ao programar a FPGA usou-se o seguinte conteúdo:

```
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";
```

Como consequência tem-se:



ZERO – LED que está ativo quando a operação é igual a 0

A – O valor de A introduzido, sendo o switch mais significativo aquele mais à esquerda.

OK – LED que fica ativo no fim de uma operação

RES – Resultado da operação apresentado em Hexadecimal

RESET – Botão que reinicia a operação

GO – Botão que muda o estado de GO para iniciar a operação

CLK – Botão que gera pulsos do clock para efetuar as mudanças de estado

6 Validação através de simulações

6.1 Definição das situações para teste

Para os testes decidimos testar as situações A = 0, A = 45, A = 85 e A = 86

```
    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) <= '0'; A(2) <= '0'; A(1) <= '0'; A(0) <= '0';
    RESET <= '1' , '0' after 30 ns;
    GO <= '0' , '1' after 50 ns, '0' after 210 ns;
-- *** End Test Bench - User Defined Section ***

END;
```

Caso A = 0 (00000000)

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

    A(7) <= '0'; A(6) <= '1'; A(5) <= '0'; A(4) <= '1';
    A(3) <= '0'; A(2) <= '1'; A(1) <= '0'; A(0) <= '1';
    RESET <= '1' , '0' after 30 ns;
    GO <= '0' , '1' after 50 ns, '0' after 210 ns;
-- *** End Test Bench - User Defined Section ***

END;
```

Caso A = 85 (01010101)

```
    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) <= '1'; A(4) <= '0';
    A(3) <= '1'; A(2) <= '1'; A(1) <= '0'; A(0) <= '1';
    RESET <= '1' , '0' after 30 ns;
    GO <= '0' , '1' after 50 ns, '0' after 210 ns;
-- *** End Test Bench - User Defined Section ***

END;
```

Caso A = 45 (00101101)

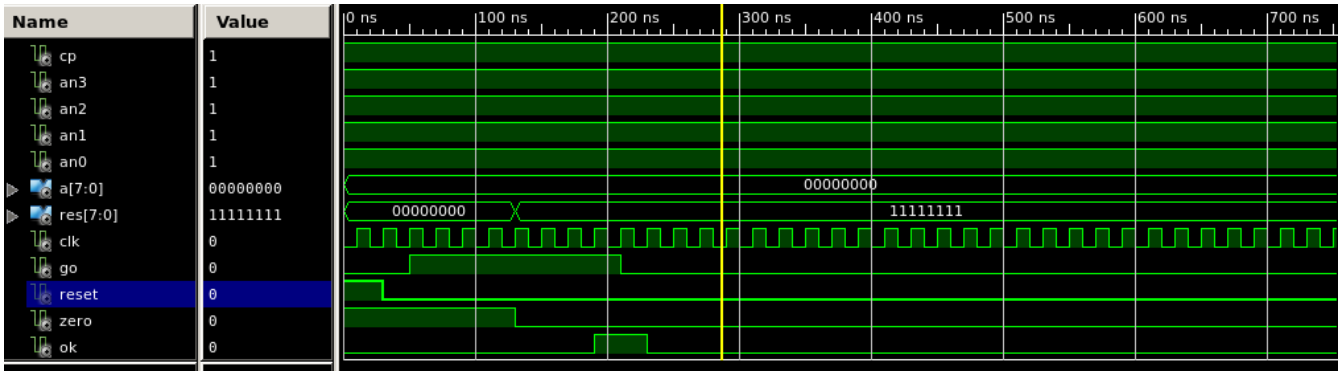
```
    tb : PROCESS
    BEGIN
       clk <= '0';
       wait for 10 ns;
       clk <= '1';
       wait for 10 ns;
    END PROCESS;

    A(7) <= '0'; A(6) <= '1'; A(5) <= '0'; A(4) <= '1';
    A(3) <= '0'; A(2) <= '1'; A(1) <= '1'; A(0) <= '0';
    RESET <= '1' , '0' after 30 ns;
    GO <= '0' , '1' after 50 ns, '0' after 210 ns;
-- *** End Test Bench - User Defined Section ***

END;
```

Caso A = 86 (01010110)

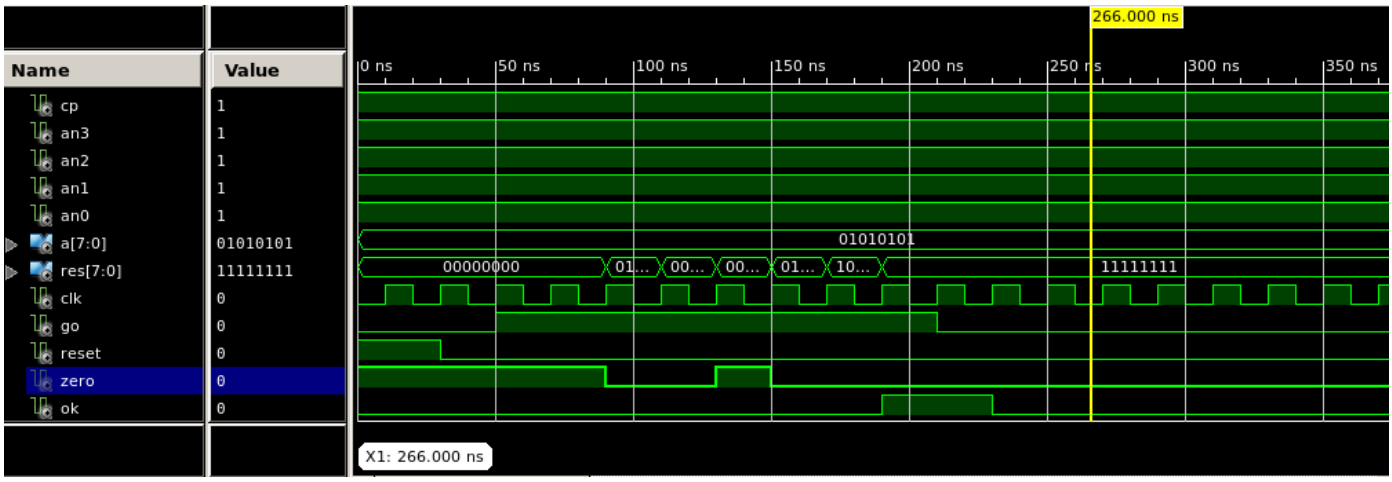
6.2 Resultados das simulações



Caso A = 0 (00000000)

Resultado esperado = -1

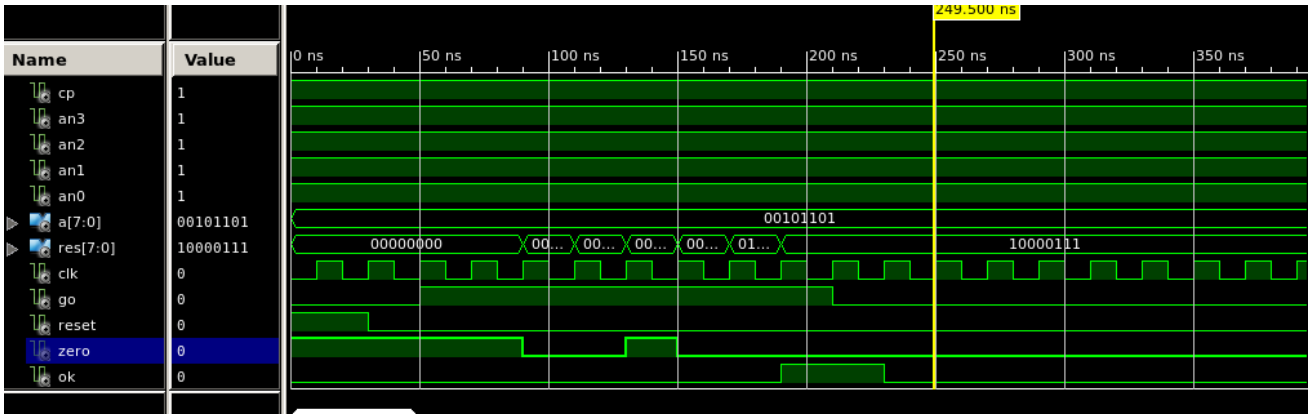
Resultado obtido = 255 (incapacidade de lidar com nºs negativos)



Caso A = 85 (01010101)

Resultado esperado = 255 (11111111)

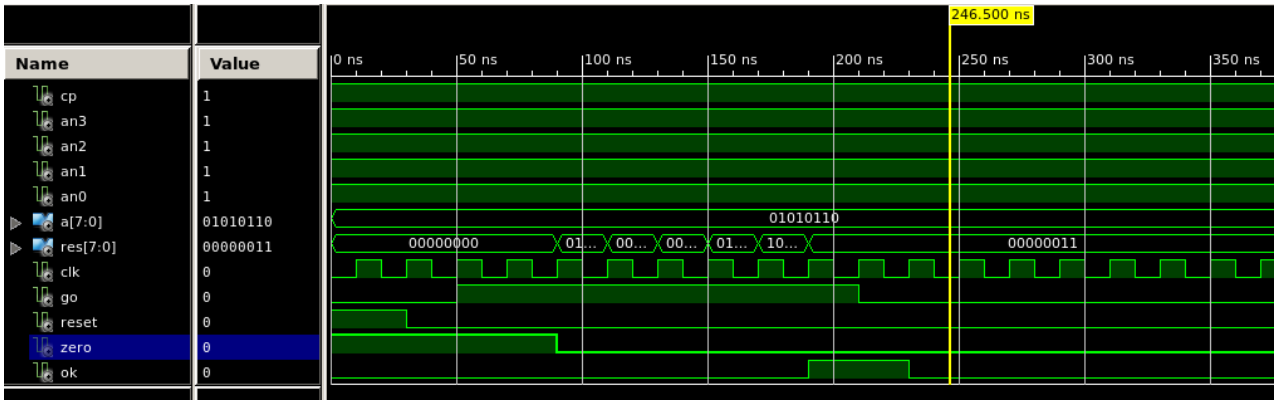
Resultado obtido = 255 (11111111)



Caso A = 45 (00101101)

Resultado esperado = 135 (10000111)

Resultado obtido = 135 (10000111)



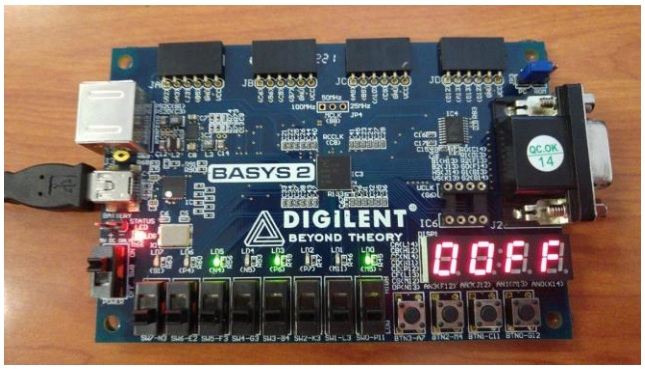
Caso A = 86 (01010110)

Resultado esperado = 259 (não pode ser representado em 8 bits)

Resultado obtido = 3 (00000011 – pois cria um overflow e a contagem volta a 0 após 255)

7 Resultados experimentais

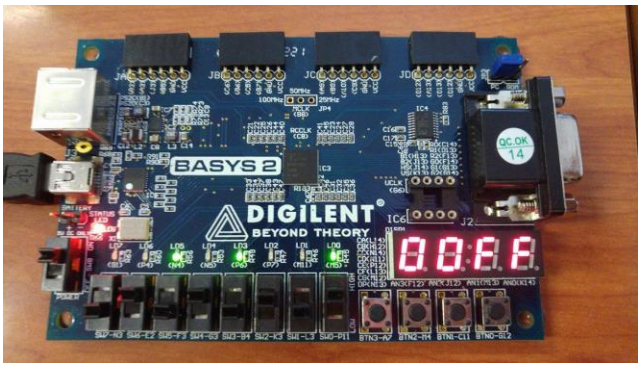
Realizando os mesmos testes das simulações no FPGA, obtivemos os seguintes resultados:



Caso A = 0 (00000000)

Resultado da simulação = 255 (FF – base 16)

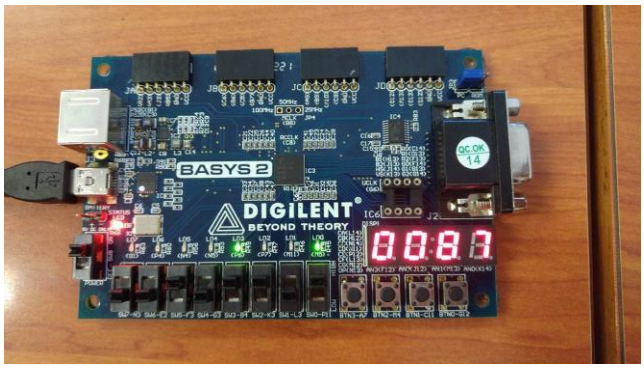
Resultado obtido = 255 (FF – base 16)



Caso A = 85 (01010101)

Resultado da simulação = 255 (FF – base 16)

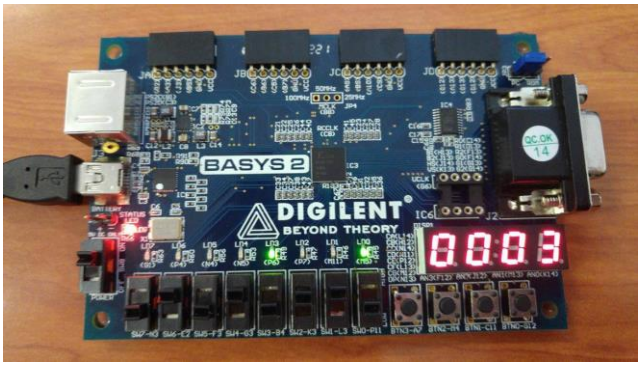
Resultado obtido = 255 (FF – base 16)



Caso A = 45 (00101101)

Resultado da simulação = 135 (87 – base 16)

Resultado obtido = 135 (87 – base 16)



Caso A = 86 (01010110)

Resultado da simulação = 3 (3 – base 16)

Resultado obtido = 3 (3 – base 16)

8 Conclusões e Observações

Pensamos que a carga horária fornecida para a realização do trabalho foi suficiente para completar o trabalho, no entanto, como tivemos alguns percalços e atrasos na realização do trabalho, a eficiência do método usado (flip-flops D) pode não ter sido a melhor, dado que não foram analisadas outras possíveis opções.

A dificuldade a utilizar o Xillinx para implementar o sistema, atrasou significativamente o desenvolvimento do trabalho, no entanto o tempo fornecido foi suficiente para compensar estes atrasos. Além disso, a ajuda indispensável dos professores (tanto das aulas práticas como das teóricas), ajudou a concluir o trabalho e a implementação dele no FPGA fornecido de forma mais rápida e simples.

No final, podemos concluir que a elaboração deste trabalho, desenvolveu a nossa capacidade a esquematizar e implementar sistemas mais complexos, e forneceu-nos com experiência prática com ferramentas como o Xillinx e o FPGA.