



FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Sistemas Lógicos (MiEI)

2017/2018 – 1º Semestre

Relatório Trabalho Final

Cálculo de expressão

$$((A \text{ AND } 11_2) - 1) + A \times 3_{10}$$

Docente:

[REMOVIDO]

Realizado por:

Cláudio [REMOVIDO] Pereira Nº[REMOVIDO]

Dina [REMOVIDO] Borrego Nº[REMOVIDO]

Matilde [REMOVIDO] Lucas Nº[REMOVIDO]

Índice

1	Introdução.....	3
1.1	Enunciação.....	3
1.2	Implementação.....	4
2	Análise.....	5
2.1	Descrição de funcionamento global.....	5
2.1.1	Parte de dados.....	6
2.1.2	Parte de controlo.....	6
2.1.3	Sistema.....	7
2.2	Situações geradores de resultados iguais a 0.....	11
3	Parte de Dados - descrição.....	12
3.1	Módulo ULA:.....	12
3.2	Módulo TEMP:.....	13
3.3	Módulo Z:.....	14
4	Síntese – Parte de Controlo.....	16
4.1	Diagrama de Estados.....	16
4.2	Tabela de Transição de Estados.....	17
4.3	Codificação de Estados.....	18
4.4	Tabela de Transição de Estados codificados, saídas e entradas dos Flip-flops.....	19
4.5	Expressões simplificadas das saídas por Mapas de Karnaugh.....	20
4.6	Expressões simplificadas das entradas dos Flip-flops por Mapas de Karnaugh.....	21
4.6.1	Flip-flops D.....	21
4.6.2	Flip-flops T.....	22
4.6.3	Flip-flops JK.....	23
4.6.4	Flip-flops escolhidos.....	24
4.7	Esquemático da Parte de Controlo.....	25
5	Implementação na FPGA Spartan 3E.....	26
6	Validação através de simulações.....	27
6.1	Definição das situações para teste.....	27
6.2	Resultados das simulações.....	28
6.3	Simulação externa.....	29
7	Resultados experimentais.....	32
8	Conclusões e Observações.....	33

1 Introdução

O trabalho dá-se em torno da implementação de uma expressão lógica num contexto físico.

1.1 Enunciação

A expressão lógica, é calculada por base nos nossos números de estudante:

[REMOVIDO]

Assim sendo, a expressão a implementar é:

$$((A \text{ AND } K_{\text{AND}}) \text{ DEC } K_{\text{DEC}}) + K_1 \times A$$

E a 2ª chave obtem-se:

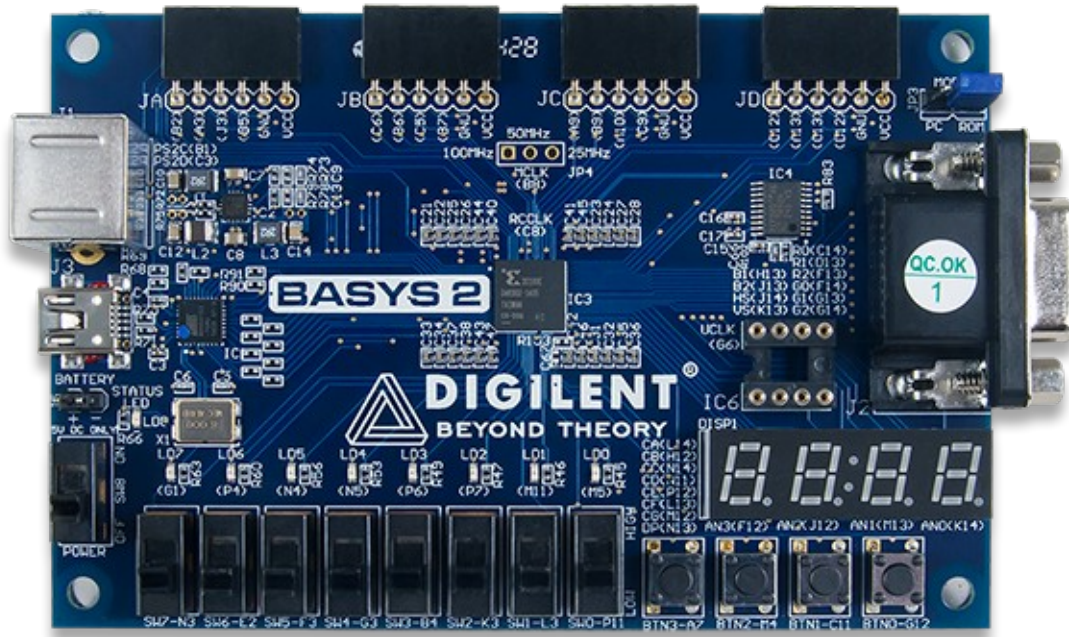
$$K_1 = 3_{10}, \quad K_{\text{DEC}} = 1, \quad K_{\text{AND}} = 3_{10}$$

Temos por tanto a seguinte expressão simplificada:

$$((A \text{ AND } 11_2) \text{ DEC } 1) + A \times 3_{10}$$

1.2 Implementação

Para a implementação foi-nos fornecida uma placa com um IC(circuito integrado) programável (FPGA, field-programmable gate array).



A (ou OPER1) é uma entrada manual na FPGA, obtida através dos seguintes switches:



Para a execução dos passos a tomar (descritos em detalhe nas próximas páginas) optamos por colocar o valor de K_{AND} embutido no programa. Para tal temos um bus (OPER2) que detém o valor codificado.

2 Análise

2.1 Descrição de funcionamento global

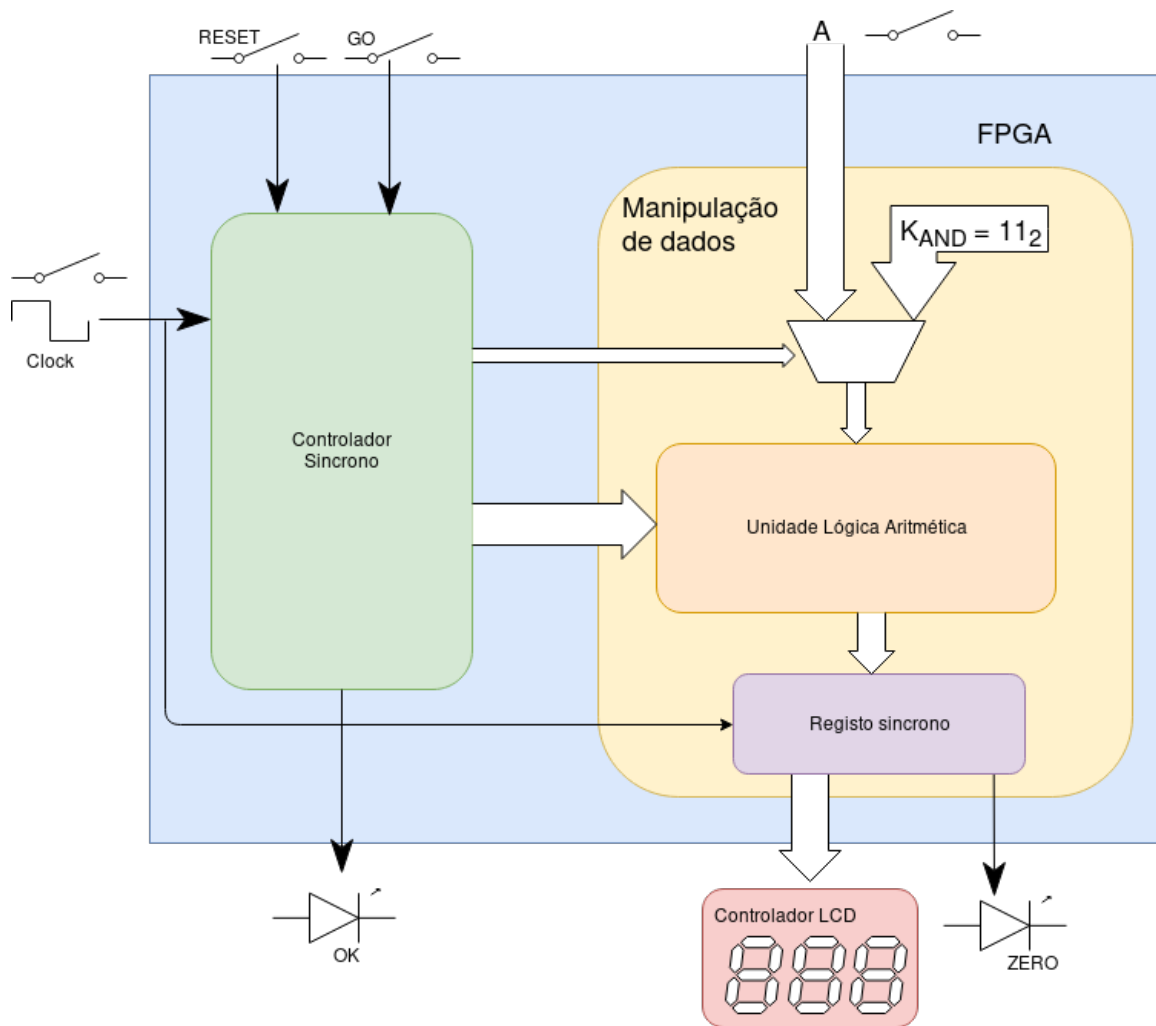


Diagrama de blocos de alto-nível da implementação efetuada

O sistema simulado pela FPGA pode ser descrito com recurso a duas partes principais distintas.

2.1.1 Parte de dados

A parte de dados (a amarelo na figura) é um aglomerado de lógica que opera por base em sinais booleanos que definem a operação a executar.

Os dados são fornecidos através de um barramento com 8 bits de largura (alternável entre duas fontes) que fornece os dados nos quais a operação é executada.

Após manipulados os dados são colocados num registo de 8 bits, aguardando consumo externo. (serão adquiridos pelo visualizador para amostragem).

O registo tem um funcionamento síncrono, e portanto está dependente de uma fonte de sinal de relógio (externa).

É ainda disposta uma saída lógica que indica se o resultado da execução foi o valor 0.

As fontes dos dados podem ser:

- Um valor pré-codificado na implementação (00000011_2)
- Um valor vindo de um bus externo

2.1.2 Parte de controlo

A parte de controlo (a verde na figura) manipula a parte de dados, de forma a que a mesma seja instruída a executar a expressão lógica desejada ($((A \text{ AND } 11_2) - 1) + A \times 11_2$).

Recebe dois sinais booleanos:

- **RESET** – Recomeçar o calculo da expressão
- **GO** – Progredir no calculo da expressão

Sendo síncrona, a parte de controlo depende de uma fonte de sinal de relógio, tendo sido utilizada a mesma que foi utilizada para a parte dos dados para sincronizar as operações entre blocos.

Quando o calculo da expressão está concluído é ativado um sinal booleano **OK**.

2.1.3 Sistema

Salienta-se que o nosso sistema enquanto que pode ser entendido pelo diagrama de blocos apresentado, sofreu modificações além das estritamente necessárias para melhorar a legibilidade tanto do esquemático como das leituras da placa.

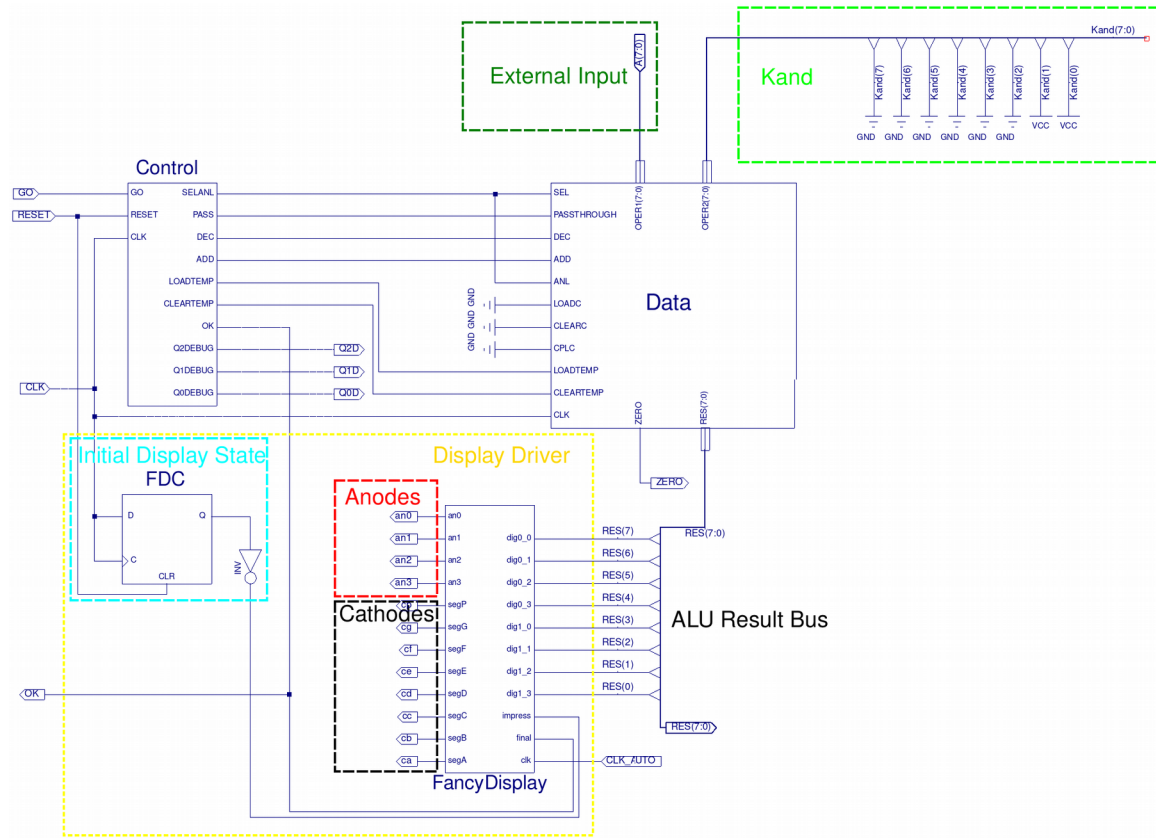
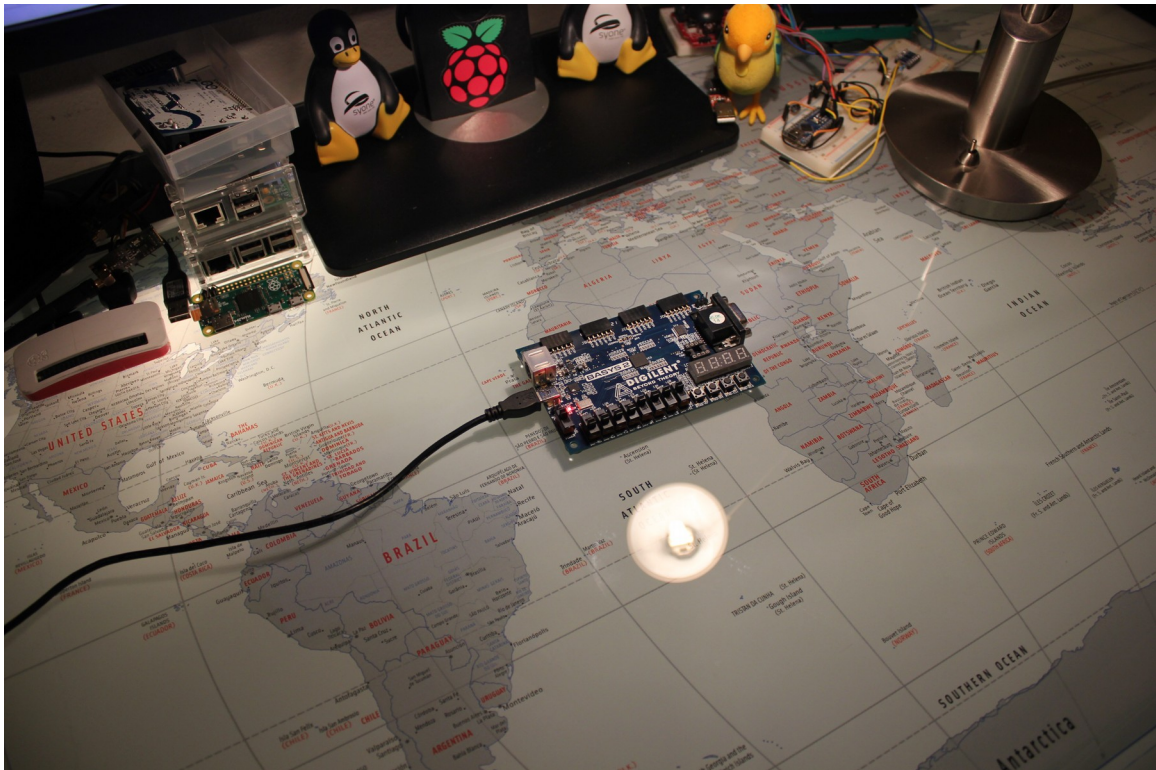


Diagrama da implementação do sistema

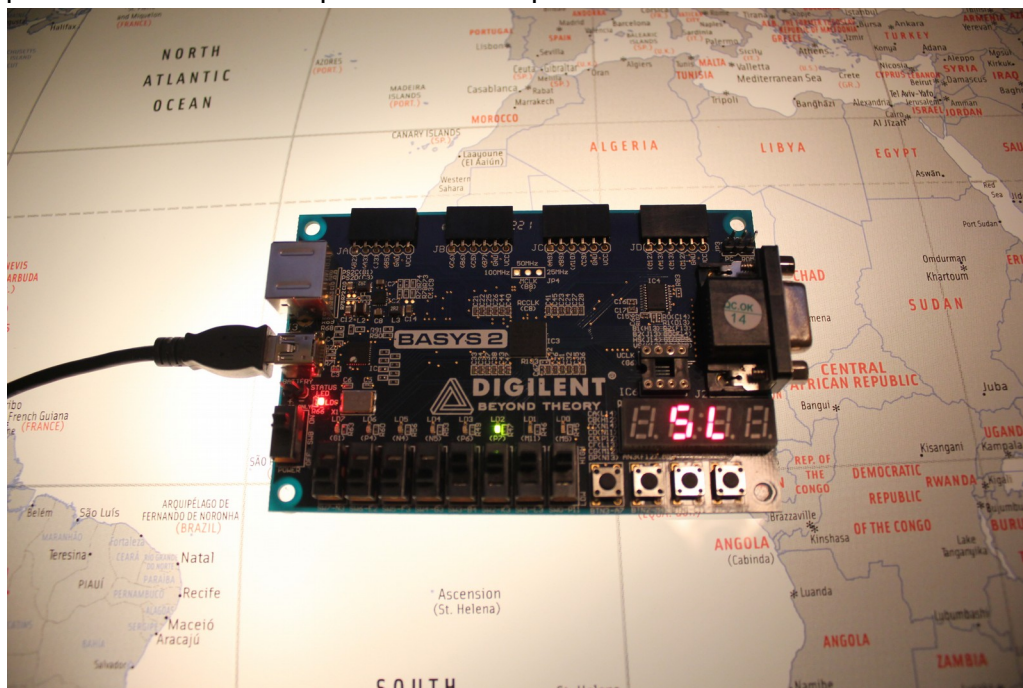
Foram efetuadas modificações na driver do controlador LCD com a finalidade de exibir alguns estados.

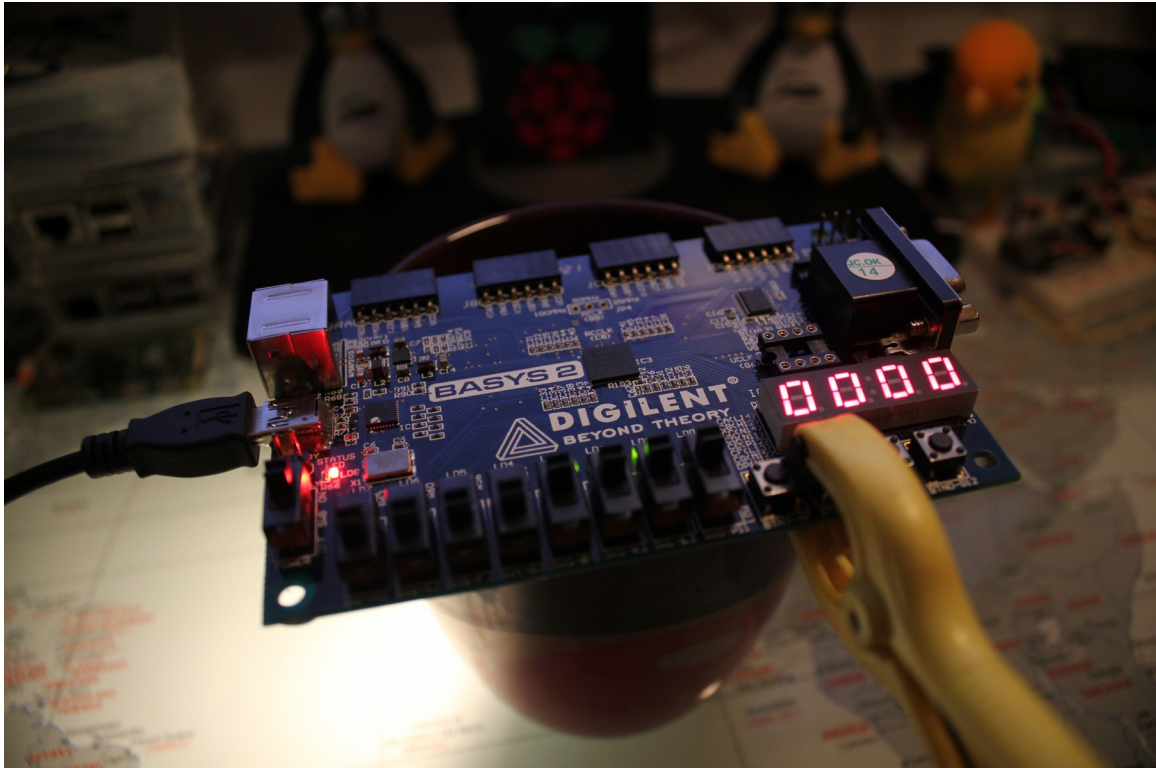
Existe um flip-flop adicional ("Initial display State" na figura) que conserva a saída "1" durante apenas o primeiro ciclo de clock.



Placa após ligação ao computador

Assim que a placa é programada e começa a executar a expressão, a primeira vez que está no estado 0 após um reset apresenta-se:





Após o primeiro ciclo de clock (com ou sem alteração de estado)

Por fim, como no ultimo estado só podem ser utilizados os ultimos 2 digitos, os primeiros foram programados para mostrarem “E.=” (*Expression equals*).



2.2 Situações geradores de resultados iguais a 0

Se decomposermos as operações de $((A \text{ AND } 11_2) - 1) + A \times 3_{10}$ obtemos a seguinte comparação lógica:

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
AND	0	0	0	0	0	0	1	1
<hr/>								
	0	0	0	0	0	0	A_1	A_0

É observável que apenas os últimos 2 bits de A importam após a operação $A \text{ AND } 11_2$.

Aplicando a segunda operação $((A \text{ AND } 11_2) - 1)$ existem 3 resultados possíveis:

	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	
DEC	0	0	0	0	0	0	0	1	
<hr/>									
	X	X	X	X	X	X	X	X	Se $A_1 = 0 \wedge A_0 = 0$ (Negativo)
<hr/>									
	0	0	0	0	0	0	A_1	0	Se $A_0 = 1$
<hr/>									
	0	0	0	0	0	0	0	1	Se $A_1 = 1 \vee A_0 = 0$

Sabendo que A tem de ser positivo e que de $A \times 3$ só podem resultar múltiplos de 3, o único valor de A que possibilita uma nulificação do resultado é o próprio $A=0$.

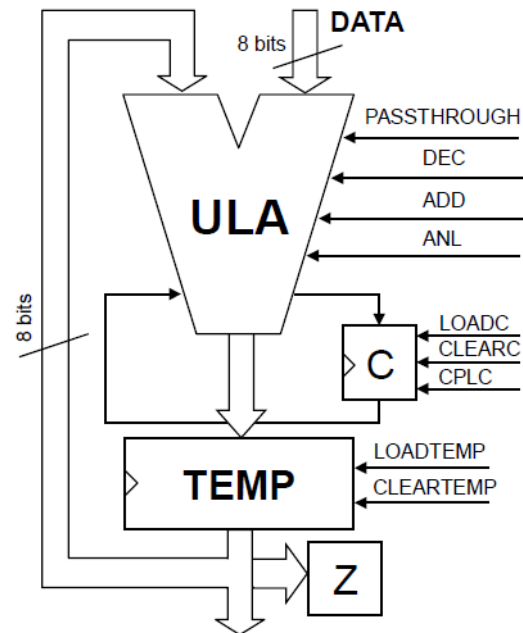
Se $A=0$, então A_1 também é 0, e nesse caso a soma de $((A \text{ AND } 11_2) - 1)$ com $A \times 3$ é zero.

3 Parte de Dados - descrição

A parte de dados consiste numa Unidade Lógica e Aritmética (referida como *ULA* ou *ALU*), que é capaz de realizar operações lógicas e aritméticas através de uma transferência de registos.

A ULA é constituída por quatro módulos:

- Módulo ULA;
- Módulo TEMP;
- Módulo Z;
- Módulo C.



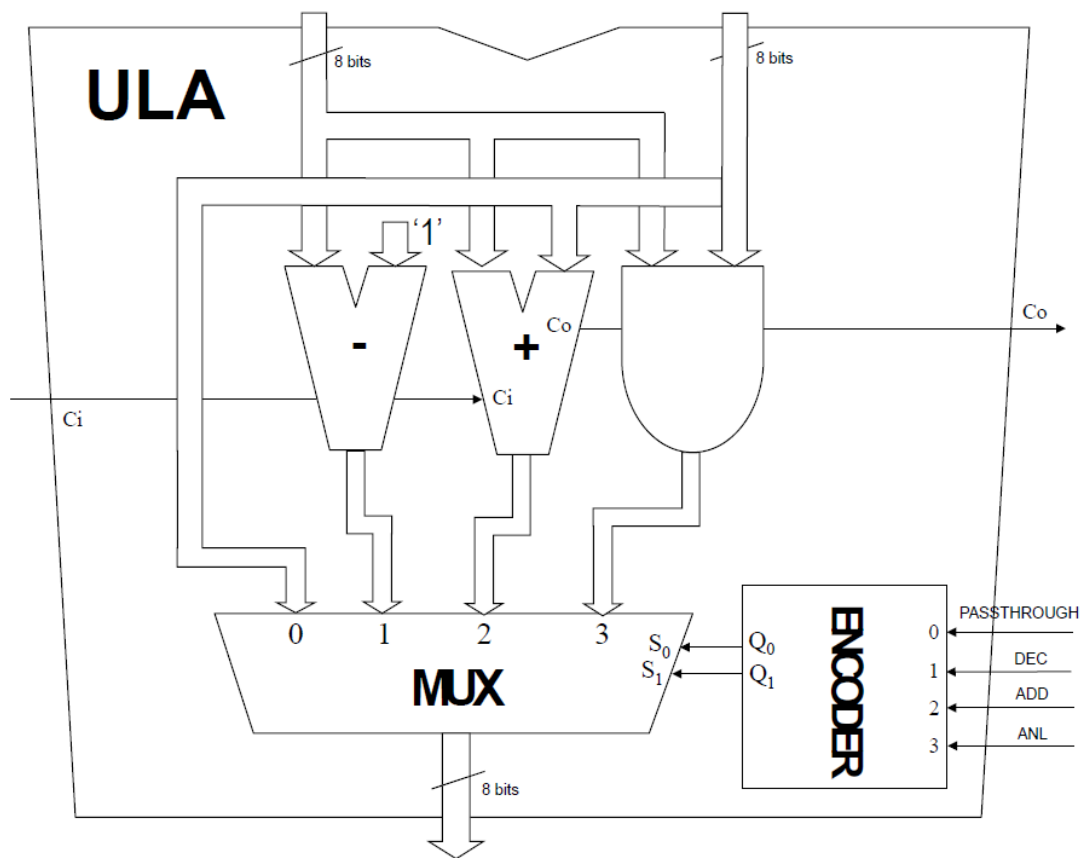
3.1 Módulo ULA:

O módulo *ULA* executa operações lógicas e aritméticas a 8 bits, sendo um bloco de lógica combinatória. Possui duas entradas e uma saída de dados, todas com 8 bits, para além de uma saída e entrada de 1 bit de transporte de operações aritméticas.

Tem ainda quatro entradas de seleção que são responsáveis pela seleção da operação pretendida.

As entradas de seleção são:

- **PASSTHROUGH** – o input passa diretamente para o registo, independentemente do seu valor.
- **DEC** – decremento, executado através de um subtrator do primeiro operando com o valor decimal “1”;
- **ADD** – soma, executada através da soma aritmética dos dois operandos;
- **ANL** – representa “&” lógico bit-a-bit do operando com o valor constante de K_{AND} .



As entradas de seleção estão diretamente ligadas a um codificador 4-para-2, que serve para codificar as 4 entradas de seleção em 2 bits de saída, que por sua vez são ligadas a um *multiplexador* (*MUX*) que define qual a saída da *ULA*, ou seja, a saída correspondente à operação lógico-aritmética selecionada.

Este módulo possui ainda uma saída C_o que representa o transporte gerado nas operações aritméticas e também uma entrada de transporte C_i que interfere nas operações aritméticas que se realizam na *ULA*.

Estas correspondem ao módulo *C* (*Carry*), o qual não foi utilizado no nosso trabalho.

3.2 Módulo TEMP:

O módulo *TEMP* é um registo síncrono de 8 bits, que é responsável por guardar temporariamente os dados das operações lógico-aritméticas efetuadas na *ULA*. Possui duas entradas de controlo:

- **CLEAR** – quando está ativo limpa o valor armazenado no flip-flop (célula de memória);
- **LOAD** – quando está ativo o registo é carregado com os dados que vieram das operações lógico-aritméticas.

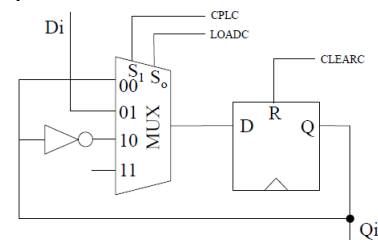
CLEAR	LOAD	Funcionamento do Registo
0	0	Mantém o conteúdo do registo inalterado
0	1	Carrega o registo com os dados à entrada (carregamento paralelo)
1	0	Apaga (sincronamente) o conteúdo do registo

Para tornar as operações pretendidas possíveis é necessário quer o módulo *TEMP* esteja vazio, pelo que se recorre à ativação da entrada **CLEAR** (que limpa o registo).

3.3 Módulo C:

O módulo *C* implementa a *flag Carry* da *ULA* o que permite guardar o valor do transporte das operações aritméticas. É constituído por um elemento de memória de 1 bit e por 3 entradas de controlo:

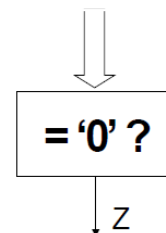
- **CREARC** – limpa o registo;
- **LOADC** – carrega dados à entrada do registo;
- **CPLC** – complementa logicamente o registo.



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

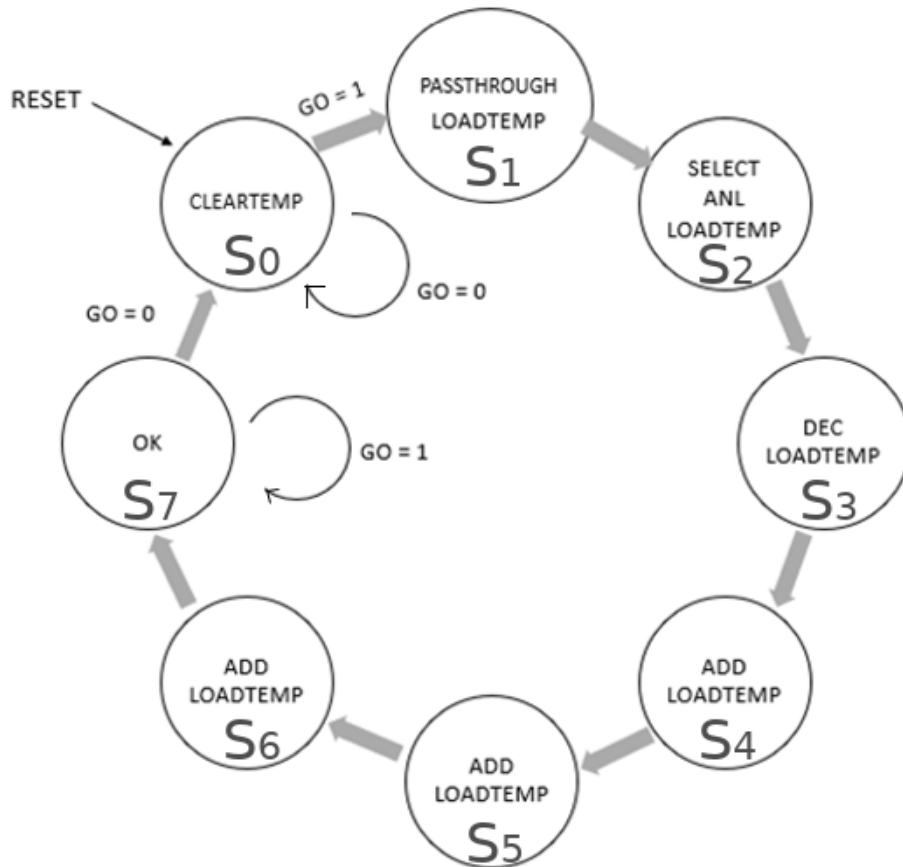
3.4 Módulo Z:

O módulo *Z* é responsável pela comparação do valor final (valor da saída do registo *TEMP*) com 0, através de lógica combinatória. É possível utilizar um comparador aritmético como forma de implementação deste módulo.



4 Síntese – Parte de Controlo

4.1 Diagrama de Estados



4.2 Tabela de Transição de Estados

A transição de estados ocorre do seguinte modo:

	GO	Q_2^*	Q_1^*	Q_0^*	Q_2	Q_1	Q_0
S_0	0	0	0	0	0	0	0
S_1	0	0	0	1	0	1	0
S_2	0	0	1	0	0	1	1
S_3	0	0	1	1	1	0	0
S_4	0	1	0	0	1	0	1
S_5	0	1	0	1	1	1	0
S_6	0	1	1	0	1	1	1
S_7	0	1	1	1	0	0	0
S_0	1	0	0	0	0	0	1
S_1	1	0	0	1	0	1	0
S_2	1	0	1	0	0	1	1
S_3	1	0	1	1	1	0	0
S_4	1	1	0	0	1	0	1
S_5	1	1	0	1	1	1	0
S_6	1	1	1	0	1	1	1
S_7	1	1	1	1	1	1	1

De notar que a variável GO restringe a capacidade de passar do primeiro estado e a de reter o último.

4.3 Codificação de Estados

Para satisfazer os 8 estados necessários para a implementação do nosso trabalho, foi necessária a utilização de 3 bits distintos ($2^3=8$).

A codificação dos mesmos é a seguinte:

Estado	Codigo	CLEAR	PASS	SEL	ANL	DEC	ADD	LOAD	OK
S ₀	000								
S ₁	001								
S ₂	010								
S ₃	011								
S ₄	100								
S ₅	101								
S ₆	110								
S ₇	111								

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

	GO	Q ₂	Q ₁	Q ₀	D ₂	D ₁	D ₀	T ₂	T ₁	T ₀	J ₂	K ₂	J ₁	K ₁	J ₀	K ₀
S ₀	0	0	0	0	0	0	0	0	0	0	0	X	0	X	0	X
S ₁	0	0	0	1	0	1	0	0	1	1	0	X	1	X	X	1
S ₂	0	0	1	0	0	1	1	0	0	1	0	X	X	0	1	X
S ₃	0	0	1	1	1	0	0	1	1	1	0	X	X	1	X	1
S ₄	0	1	0	0	1	0	1	0	0	1	X	0	0	X	1	X
S ₅	0	1	0	1	1	1	0	0	1	1	X	0	1	X	X	1
S ₆	0	1	1	0	1	1	1	0	0	1	X	0	X	0	1	X
S ₇	0	1	1	1	0	0	0	1	1	1	X	1	X	1	X	1
S ₀	1	0	0	0	0	0	1	0	0	1	0	X	0	X	1	X
S ₁	1	0	0	1	0	1	0	0	1	1	0	X	1	X	X	1
S ₂	1	0	1	0	0	1	1	0	0	1	0	X	X	0	1	X
S ₃	1	0	1	1	1	0	0	1	1	1	1	X	X	1	X	1
S ₄	1	1	0	0	1	0	1	0	0	1	X	0	0	X	1	X
S ₅	1	1	0	1	1	1	0	0	1	1	X	0	1	X	X	1
S ₆	1	1	1	0	1	1	1	0	0	1	X	0	X	0	1	X
S ₇	1	1	1	1	1	1	1	0	0	0	X	0	X	0	X	0

4.5 Expressões simplificadas das saídas por Mapas de Karnaugh

As seguintes saídas apenas tem um único estado em que ocorrem e não consideram a variável GO (com ou sem a mesma, tem o mesmo funcionamento)

$$\text{PASSTHROUGH} = \neg Q_2 \cdot \neg Q_1 \cdot Q_0$$

$$\text{SELECT} = \neg Q_2 \cdot Q_1 \cdot \neg Q_0$$

$$\text{CLEARTEMP} = \neg Q_2 \cdot \neg Q_1 \cdot \neg Q_0$$

$$\text{ANL} = \neg Q_2 \cdot Q_1 \cdot \neg Q_0$$

$$\text{DEC} = \neg Q_2 \cdot Q_1 \cdot Q_0$$

$$\text{OK} = Q_2 \cdot Q_1 \cdot Q_0$$

As saídas ativas em mais de um estado são:

L	0	0	1	1
	0	1	1	0
0 0	1	1	0	0
0 1	1	1	1	1
1 1	1	0	0	1
1 0	1	1	0	0

A	0	0	1	1
	0	1	1	0
0 0	0	1	1	0
0 1	0	1	1	0
1 1	0	0	0	0
1 0	0	1	1	0

$$\text{LOADTEMP} = \neg Q_2 \cdot Q_0 + \neg GO \cdot Q_0 + \neg Q_1 \cdot Q_0 \text{ (Dois quadrados e uma linha)}$$

$$\text{ADD} = Q_2 \cdot \neg Q_0 + Q_2 \cdot \neg Q_1 \text{ (Dois quadrados)}$$

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

Os mapas de Karnaugh consideram a ordem das variáveis como sendo:

GO, Q_2, Q_1, Q_0 , ou seja dos valores da linha sombreada GO é o primeiro e Q_1 o segundo, já da coluna sombreada Q_1 é o primeiro e Q_0 o segundo.

4.6.1 Flip-flops D

D_2	0	0	1	1
	0	1	1	0
00	0	1	1	0
01	0	1	1	0
11	1	0	1	1
10	0	1	1	0

D_1	0	0	1	1
	0	1	1	0
00	0	0	0	0
01	1	1	1	1
11	0	0	1	0
10	1	1	1	1

D_0	0	0	1	1
	0	1	1	0
00	0	1	1	1
01	0	0	0	0
11	0	0	1	0
10	1	1	1	1

$$D_2 = \neg Q_1 \cdot Q_2 + \neg Q_0 \cdot Q_2 + GO \cdot Q_1 \cdot Q_0 + \neg Q_2 \cdot Q_1 \cdot Q_0$$

(Dois quadrados e duas meias linhas)

$$D_1 = Q_1 \cdot \neg Q_0 + \neg Q_1 \cdot Q_0 + GO \cdot Q_2 \cdot Q_0 \text{ (Duas linhas e meia coluna)}$$

$$D_0 = Q_1 \cdot \neg Q_0 + \neg Q_1 \cdot \neg Q_0 \cdot Q_2 + \neg Q_1 \cdot \neg Q_0 \cdot GO + Q_1 \cdot Q_2 \cdot GO$$

(Uma linha, uma meia coluna e duas meias linhas)

4.6.2 Flip-flops T

T_2	0	0	1	1
	0	1	1	0
00	0	0	0	0
01	0	0	0	0
11	1	1	0	1
10	0	0	0	0

T_1	0	0	1	1
	0	1	1	0
00	0	0	0	0
01	1	1	1	1
11	1	1	0	1
10	0	0	0	0

T_0	0	0	1	1
	0	1	1	0
00	0	1	1	1
01	1	1	1	1
11	1	1	0	1
10	1	1	1	1

$$T_2 = Q_1 \cdot Q_0 \cdot \neg(GO + Q_2) \text{ (Uma linha e não uma coluna)}$$

$$T_1 = \neg GO \cdot Q_0 + \neg Q_1 \cdot Q_0 + \neg Q_2 \cdot Q_0 \text{ (Dois quadrados e uma linha)}$$

$$T_0 = (\neg GO + \neg Q_2 + \neg Q_1 + \neg Q_0) \cdot (GO + Q_2 + Q_1 + Q_0) \text{ (Não dois valores)}$$

4.6.3 Flip-flops JK

J_2	0	0	1	1
	0	1	1	0
00	0	X	0	X
01	0	X	0	X
11	1	X	1	X
10	0	X	0	X

J_1	0	0	1	1
	0	1	1	0
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	X	X	X	X

J_0	0	0	1	1
	0	1	1	0
00	0	1	1	1
01	X	X	X	X
11	X	X	X	X
10	1	1	1	1

K_2	0	0	1	1
	0	1	1	0
00	X	0	0	X
01	X	0	0	X
11	X	1	0	X
10	X	0	0	X

K_1	0	0	1	1
	0	1	1	0
00	X	X	X	X
01	X	X	X	X
11	1	1	0	1
10	0	0	0	0

K_0	0	0	1	1
	0	1	1	0
00	X	X	1	X
01	1	1	X	1
11	1	1	0	1
10	X	X	X	X

$$J_2 = Q_0 \cdot Q_1 \text{ (Uma linha)}$$

$$K_2 = \neg GO \cdot Q_0 \cdot Q_1 \text{ (Meia linha)}$$

$$J_1 = \neg Q_1 \cdot Q_0 \text{ (Uma linha)}$$

$$K_1 = Q_0 \cdot \neg GO + Q_0 \cdot \neg Q_2 \text{ (Dois quadrados)}$$

$$J_0 = Q_0 + Q_2 + GO \text{ (Par de linhas e dois pares de duas colunas)}$$

$$K_0 = \neg GO + \neg Q_1 + \neg Q_2 \text{ (Duas linhas e dois pares de duas colunas)}$$

4.6.4 Flip-flops escolhidos

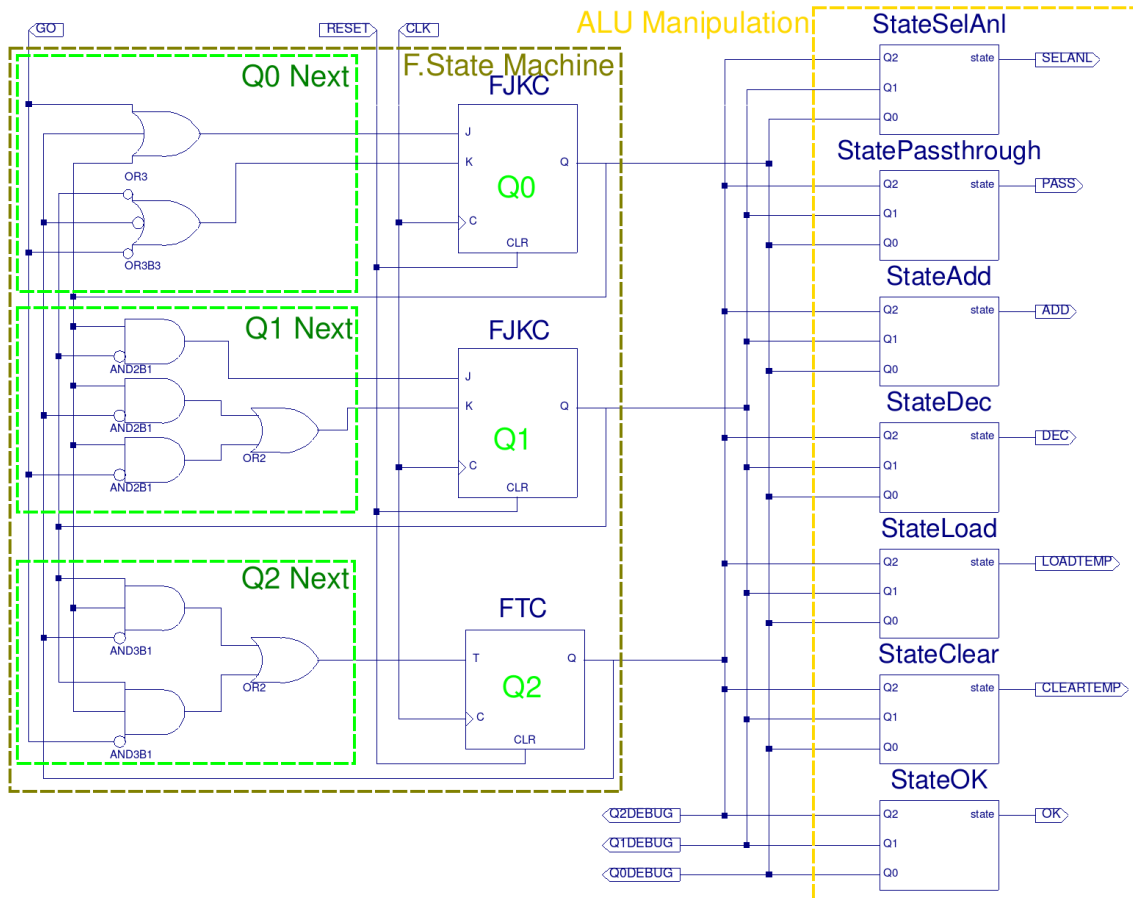
Os flip-flops mais simples para a maquina de estados em questão são:

$Q_2 \rightarrow T_2$ (3 portas lógicas, excluindo *NOT's*)

$Q_1 \rightarrow JK_1$ (2 portas lógicas, excluindo *NOT's*)

$Q_0 \rightarrow JK_0$ (4 portas lógicas, excluindo *NOT's*)

4.7 Esquemático da Parte de Controlo



O conteúdo dos chips **State*** são as expressões calculadas na secção 4.5.

As saídas de debug permitem que se verifique o estado atual da maquina de estados finita.

5 Implementação na FPGA Spartan 3E

Foi utilizado o seguinte mapeamento entre o hardware e a nossa implementação lógica:

```
#Input Signals
net "CLK" loc = "G12";
net "CLK" CLOCK_DEDICATED_ROUTE = FALSE;
net "RESET" loc = "A7";
net "G0" loc = "M4";
#Input Variable
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";
#Signal LEDs
net "ZERO" loc = "P7";
net "OK" loc = "M5";
#Automatic clock (for the display)
net "CLK_AUTO" loc="B8";
#Display cathodes
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";
#Display anodes
net "an3" loc="K14";
net "an2" loc="M13";
net "an1" loc="J12";
net "an0" loc="F12";
```

6 Validação através de simulações

6.1 Definição das situações para teste

A simulação do da parte de controlo foi efetuada com as seguintes variáveis e parâmetros temporais:

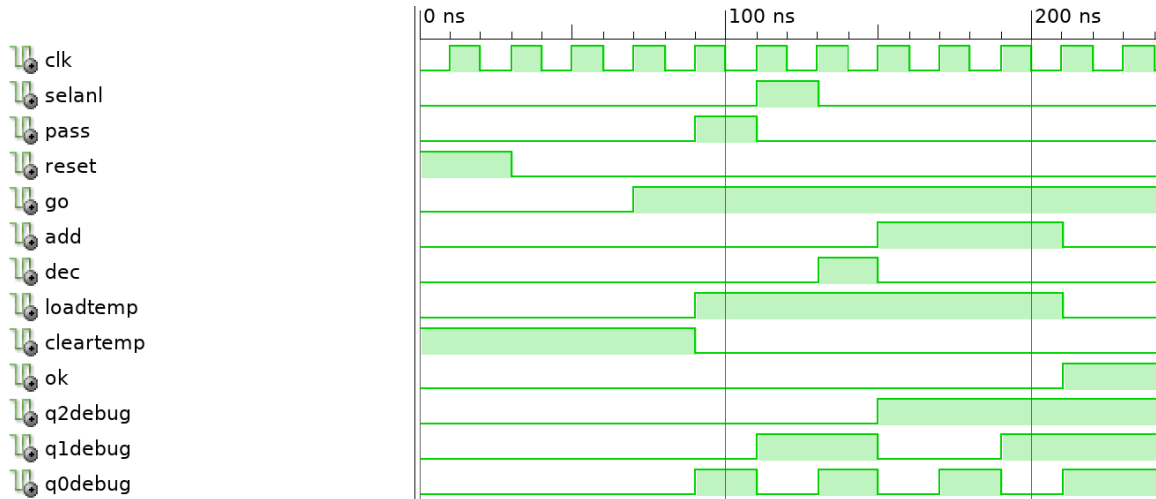
```
tb: PROCESS
BEGIN
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
END PROCESS;
reset <= '1', '0' after 30 ns;
go <= '0', '1' after 70 ns;
```

Enquanto que a simulação ao sistema foi efetuada com:

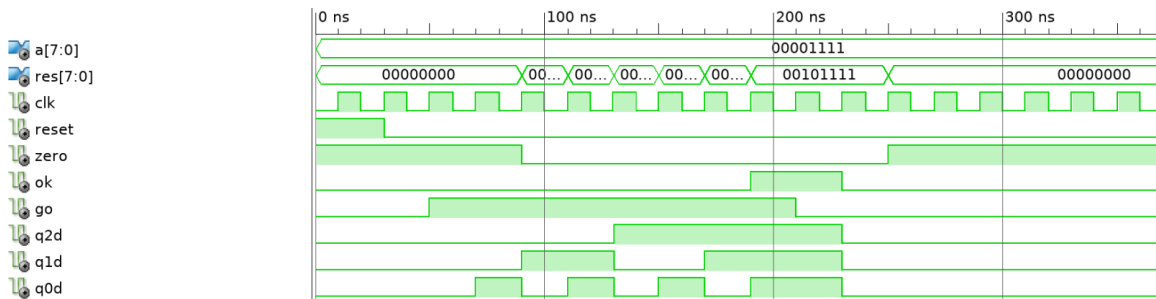
```
tb: PROCESS
BEGIN
    clk <= '0';
    wait for 10 ns;
    clk <= '1';
    wait for 10 ns;
END PROCESS;
A(7) <= '0'; A(6) <= '0'; A(5) <= '0'; A(4) <= '0';
A(3) <= '1'; A(2) <= '1'; A(1) <= '1'; A(0) <= '1';
RESET <= '1', '0' after 30 ns;
GO <= '0', '1' after 50 ns, '0' after 210 ns;
```

6.2 Resultados das simulações

A simulação do circuito de controlo tem o seguinte diagrama temporal resultante:



A simulação do circuito do sistema tem o seguinte diagrama temporal resultante:



A alteração dos parâmetros de simulação (*A*, *reset*, *go*) demonstra resultados consistentes com o observado para os valores apresentados.

6.3 Simulação externa

Foi ainda efetuada uma simulação através de um programa externo por nós concebido.

Segue abaixo o código fonte (em Python) do programa em questão:

```
class Data:
    def __init__(self, a):
        self.memory = self.sel = 0
        self.k_and = 3
        self.input_bus = self.a = a

    def clock_cycle(self, sel=0, op=None, load=False, clear=False):
        self.sel = sel

        if clear:
            self.memory = 0

        if sel:
            self.input_bus = self.k_and
        else:
            self.input_but = a

        result = None
        if op is "add":
            result = self.add()
        elif op is "dec":
            result = self.dec()
        elif op is "pass":
            result = self.passthrough()
        elif op is "anl":
            result = self.anl()

        if load:
            self.memory = result

        self.display()
        if self.memory == 0:
            print("*blink* Zero LED *blink*")

    def change_input(self, a):
        self.a = a

    def dec(self):
        return self.input_bus-1 if self.input_bus>0 else 0

    def add(self):
        return self.input_bus + self.memory

    def passthrough(self):
        return self.input_bus
```

```

def anl(self):
    return self.input_bus & self.memory

def display(self):
    print("LED segment:" + format(self.memory, '02x'))

class Controller:
    def __init__(self):
        self.state = 0
        self.data_module = Data(0)

    def clock_cycle(self, reset=False, go=False):
        if self.state in range(1,8):
            self.state += 1

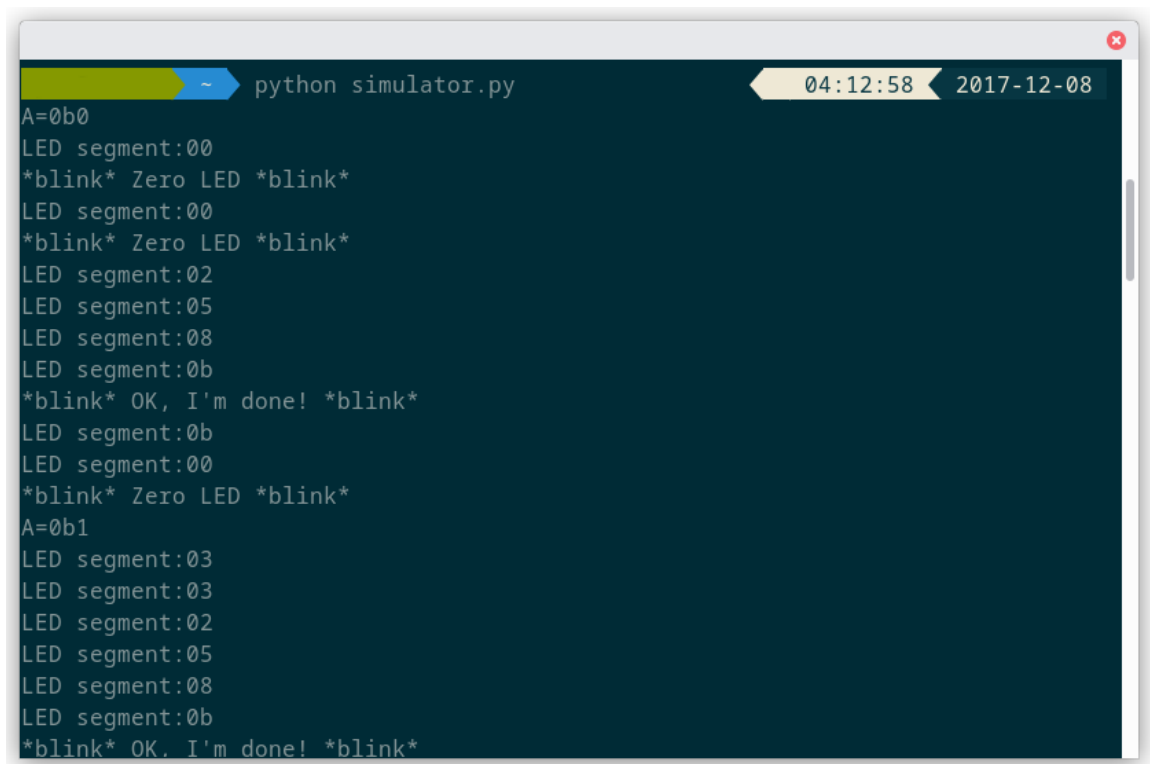
        if go and self.state == 0:
            self.state = 1

        if reset:
            self.state = 0

        if self.state == 0:
            self.data_module.clock_cycle(clear=True)
        elif self.state == 1:
            self.data_module.clock_cycle(op="pass", load=True)
        elif self.state == 2:
            self.data_module.clock_cycle(sel=True, op="anl", load=True)
        elif self.state == 3:
            self.data_module.clock_cycle(op="dec", load=True)
        elif self.state == 4:
            self.data_module.clock_cycle(op="add", load=True)
        elif self.state == 5:
            self.data_module.clock_cycle(op="add", load=True)
        elif self.state == 6:
            self.data_module.clock_cycle(op="add", load=True)
        elif self.state == 7:
            if go:
                print("*blink* OK, I'm done! *blink*")
                self.data_module.clock_cycle()
            else:
                self.state = 0
                self.data_module.clock_cycle(clear=True)

controller = Controller()
for a in range(0, 10):
    print("A=" + bin(a))
    controller.data_module.change_input(a)
    for cycle in range(0, 7):
        controller.clock_cycle(go=True)
    controller.clock_cycle(go=False, reset=True)

```

A terminal window with a dark blue background and white text. The window title bar shows a red close button, a yellow bar with a blue arrow pointing right, and the text "python simulator.py". On the right side of the title bar, there is a yellow bar with the time "04:12:58" and a date "2017-12-08". The terminal output consists of two blocks of text. The first block starts with "A=0b0" and ends with "*blink* OK, I'm done! *blink*". The second block starts with "A=0b1" and ends with "*blink* OK, I'm done! *blink*".

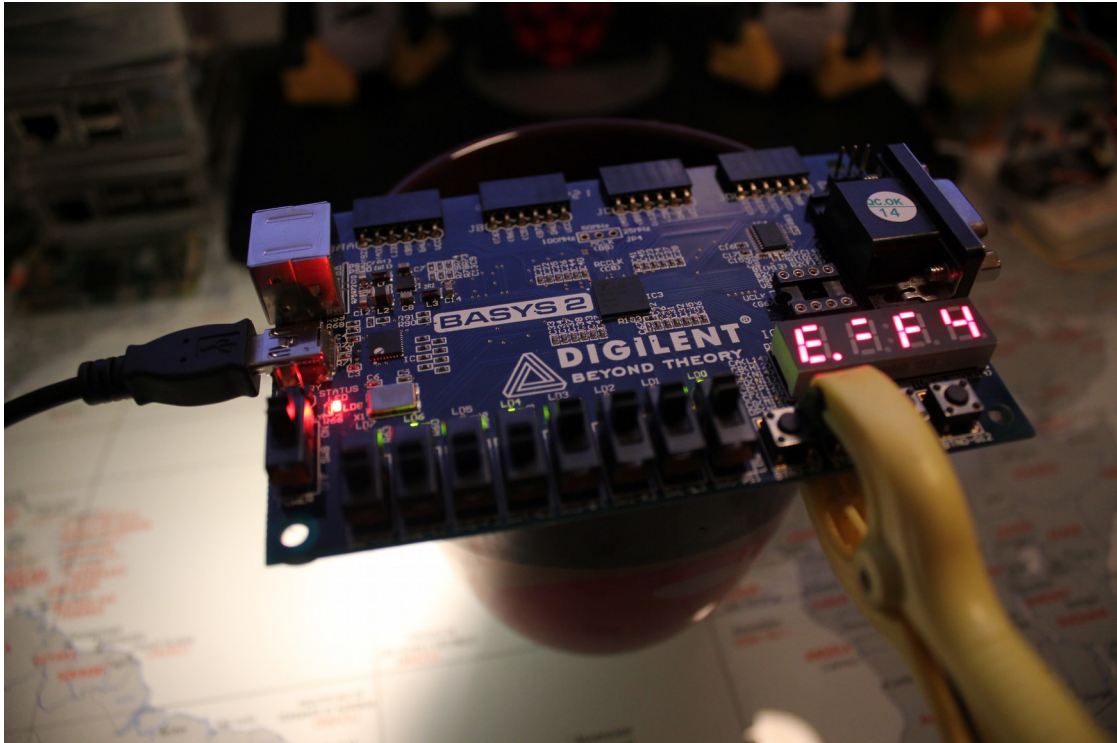
```
A=0b0
LED segment:00
*blink* Zero LED *blink*
LED segment:00
*blink* Zero LED *blink*
LED segment:02
LED segment:05
LED segment:08
LED segment:0b
*blink* OK, I'm done! *blink*
LED segment:0b
LED segment:00
*blink* Zero LED *blink*
A=0b1
LED segment:03
LED segment:03
LED segment:02
LED segment:05
LED segment:08
LED segment:0b
*blink* OK, I'm done! *blink*
```

Decorrer da simulação

7 Resultados experimentais

A experimentação do nosso circuito lógico na placa deu exatamente o resultado pretendido.

Foi possível verificar todos os estados a serem percorridos, efetuar *resets* devidamente, e aguardar pela intenção de avançar (sinal *GO*).



8 Conclusões e Observações

Foi-nos possível criar um sistema digital que ainda que básico comparativamente aos sistemas ditos modernos, deteve um enorme papel na nossa aprendizagem e compreensão da cadeia.

O fato de termos a possibilidade de sair do simulador para algo concreto, real, físico é uma boa experiência, ver-se o nosso trabalho a produzir resultados.

Apesar de inúmeras complicações com o software Xilinx ISE o mesmo acabou por se mostrar capaz à execução

Pensamos que não ficou nada por fazer. Os únicos pontos menos claros (lidar com *overflows* e *underflows*) pensamos que procedemos bem ao não os considerar.

A carga horária necessária à execução deste projeto foi de cerca de 20 horas (coletivas) por semana, sensivelmente 100 horas totais, e ainda que de início tenhamos tido bastante dificuldade a avançar com o trabalho, no fim revelou-se muito mais simples ao ponto de nos termos esforçado em mais do que o solicitado (indicações nos esquemáticos, simulações adicionais e reprogramação do LCD).

A única crítica significativa que temos é o software de desenvolvimento. Quer em máquina virtual como nativo, o Xilinx ISE frequentemente funciona de forma diferente ao expectável e apresenta resultados inconsistentes. O mesmo ocupa imenso espaço e é lento para certas tarefas pouco complexas.

Concluindo, damos-nos por satisfeitos. Pensamos ter sido sucedidos e agradecemos aos docentes por tal feito.