Sistemas Lógicos (MiEF+MiEB) – 2020/2021 – 2º Semestre

Relatório Trabalho Final

Sistema de Alarme

(1 * A + 2 * B) > 5

Realizado por:

60461 Rui Filipe MIEF P12

60583 Pedro Toscano MIEF P12

61340 Francisca Gomes MIEB P12

Índice

1 Introdução

Este relatório surge no âmbito do trabalho final de Sistemas Lógicos, “Sistema de Alarme”. O trabalho tem como objetivo implementar um sistema de alarme que é ativado quando uma determinada condição é verificada.

O sistema de alarme está dividido em duas partes, uma parte de controlo e uma parte de dados. A parte de controlo é composta por uma máquina de estados síncrona, arquitetura de Moore. A parte de dados é composta por um somador, que faz somas as sucessivas dos valores introduzidos (A ou B), cujos resultados são guardados num registo, que controla quantas vezes se fazem as somas dependendo dos valores de K1 e K2. Tem ainda um comparador, que compara o valor calculado com o valor que ativa o alarme. No caso de se verificar a igualdade, o alarme é ativado.

Todo o sistema descrito foi simulado utilizando o software Xilinx.

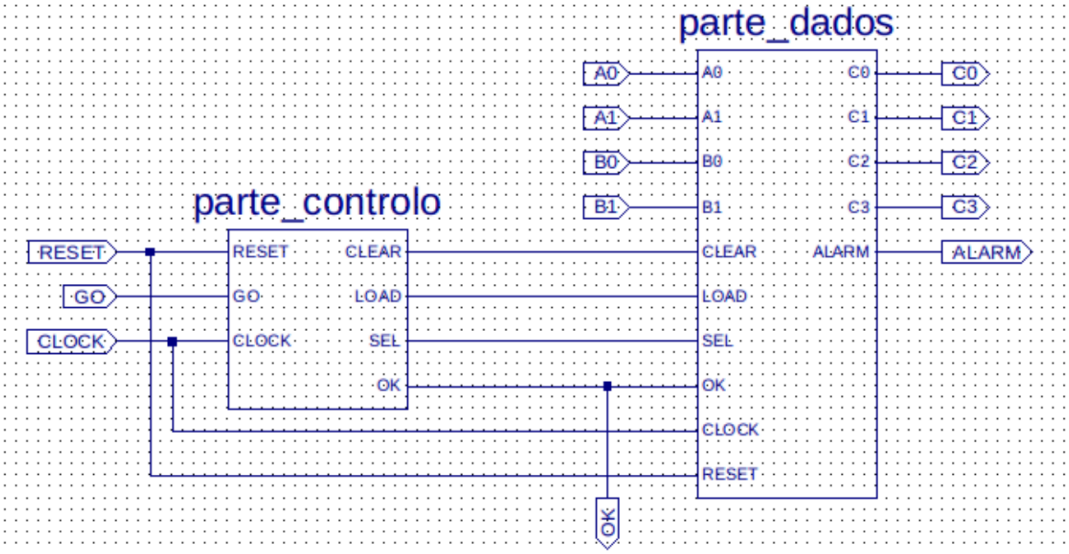
Durante a realização do trabalho foi necessário tomar algumas decisões, como por exemplo a escolha dos flip-flops a utilizar para implementar a parte de controlo. Foram feitas tabelas de verdade para todos os flip-flops e o critério de escolha foi utilizar o flip-flop que implica o menor número de portas a usar nas funções de entrada, assumindo que os flip-flops são módulos atómicos.

2 Análise

2.1 Descrição de funcionamento global

A implementação do nosso sistema de alarme seguiu a estrutura sugerida no enunciado. Considerámos que essa seria a melhor solução para a implementação pelo que não introduzimos nenhuma alteração.

Sendo assim, este é o aspeto do sistema de alarme.

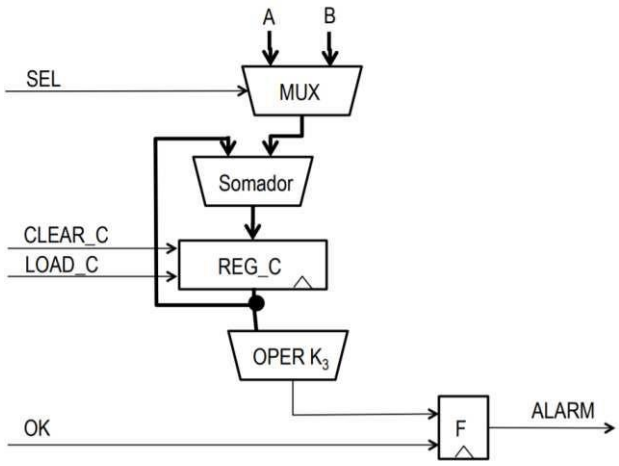


A atuação de um sinal de entrada (GO) provoca a computação de 1 * A + 2 * B e a sua comparação com 5. A saída de alarme é ativada sempre que a condição é verdadeira.

Parte de dados

A parte de dados é responsável por realizar o cálculo 1 * A + 2 * B, ou seja, 0 + A + B + B, e compará-lo com o número 5, sendo que o alarme será ativado se o resultado do cálculo efetuado for maior que 5. Nesta parte vão entrar as variáveis, A e B, cada uma com 2 bits.

A parte de dados tem o seguinte aspeto:

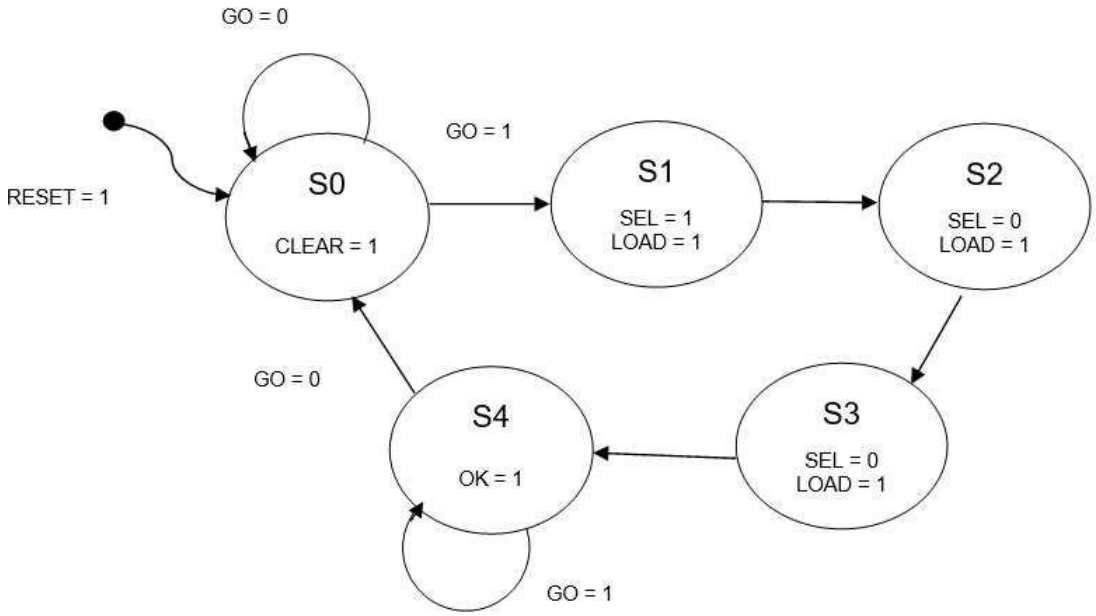


Para além das variáveis A e B entram também as variáveis CLEAR e LOAD, que são variáveis de saída da parte de controlo.

Parte de controlo

Este sistema é implementado recorrendo a uma máquina de estados síncrona, arquitetura de Moore. Esta máquina é responsável pelo controlo das saídas da parte de controlo, a serem fornecidas à parte de dados.

A nossa máquina de estados será composta por 5 estados e tem o seguinte aspeto.



Nesta máquina de estados temos variáveis que estão sob o controlo imediato do utilizador e não fazem ligação direta com a parte de dados – variáveis de entrada. Estas são:

● RESET – Quando RESET = 1 todo o sistema é colocado a zeros. Esta variável permite reiniciar o processo com novos valores para A e B, ou seja, garante o estado inicial da máquina (à excessão do estado anterior do alarme, que se mantém até ser atualizado no final do processo seguinte).

● GO – Quando GO = 1 inicia-se o processo, quando colocada a 0, a máquina de estados permanece no estado S0.

Existem também variáveis que não são controladas pelo utilizador – variáveis de saída. Estas são:

● LOAD – Quando LOAD = 1 é transportado para o somador da parte de dados o valor obtido na soma anterior, ou seja, atualiza o valor de C, para que se possam fazer os cálculos necessários através de somas sucessivas.

● CLEAR – Quando CLEAR = 1 todo o registo da parte de dados é limpo, colocando C com o valor 0.

● SEL – Como apenas é utilizado um somador na parte de dados, é preciso selecionar qual das variáveis (A ou B) vai entrar para ser somada. Essa seleção é feita por esta variável.

● OK – Quando o processo termina, a variável OK toma o valor 1, sinalizando que o cálculo terminou, impedindo assim a entrada de valores intermédios no comparador.

2.2 Situações geradores de situações de alarme

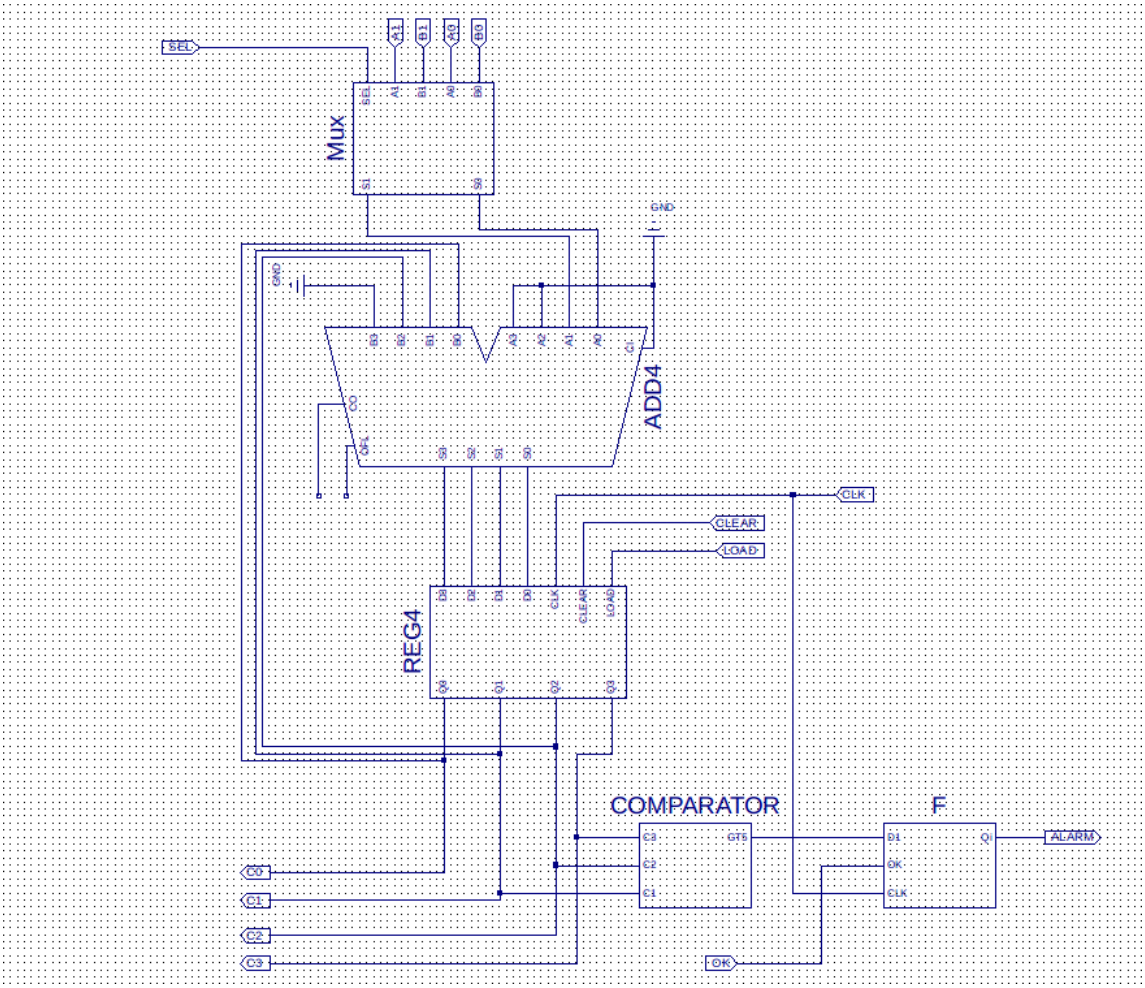
O alarme é ativado quando as variáveis A e B, de dois bits, correspondem aos valores que fazem com que a condição (1 * A + 2 * B) > 5 seja verdadeira. Para efeitos de uma visualização mais clara, representámos as situações geradoras de alarme atribuindo todos os valores possíveis para A e B, em base decimal.

A	B	ALARM
0	0	0
0	1	0
0	2	0
0	3	1
1	0	0
1	1	0
1	2	0
1	3	1
2	0	0
2	1	0
2	2	1
2	3	1
3	0	0
3	1	0
3	2	1
3	3	1

Estes valores serão confirmados mais à frente com as simulações e testes realizados.

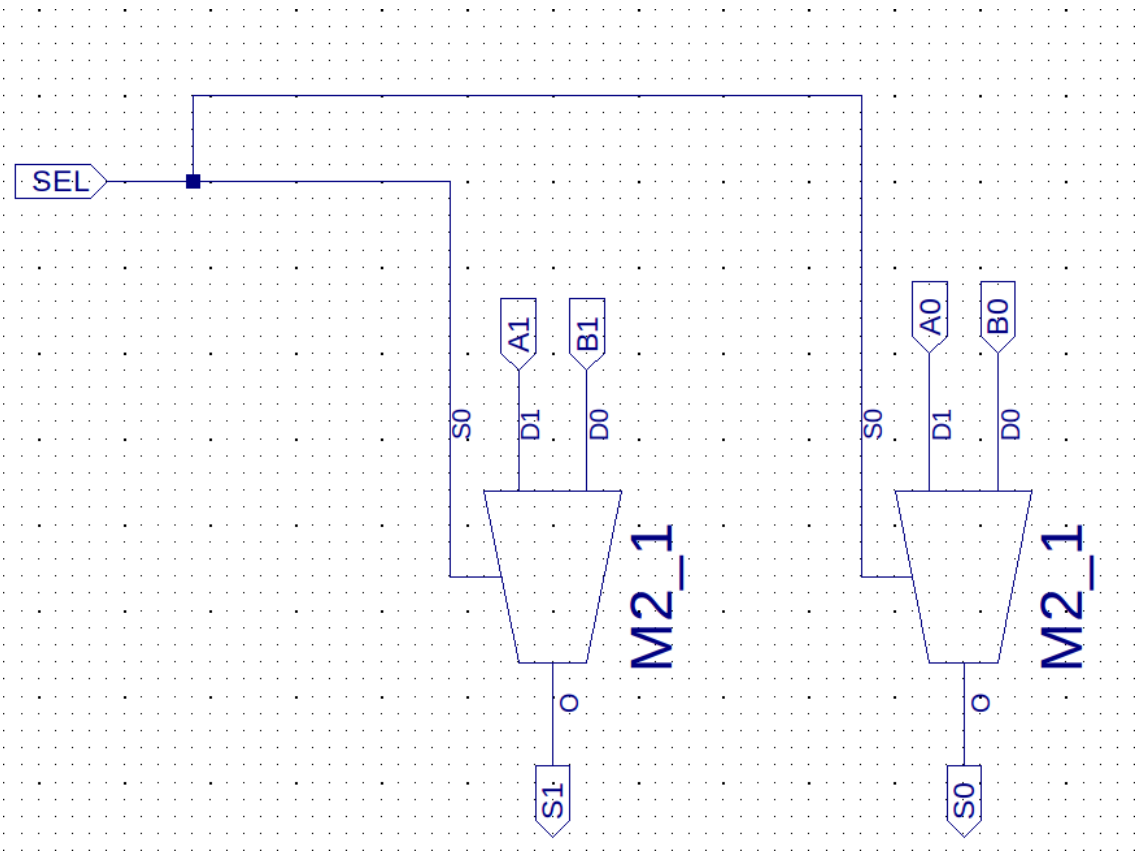
3 Síntese – Parte de Dados

A parte de dados é responsável por, primeiramente, selecionar a variável que vai ser somada sucessivamente e realizar essa mesma soma. Esta parte está intimamente ligada à parte de controlo pois tanto a seleção da variável como as somas são feitas de acordo com variáveis que provêm desta. Seguidamente, é feita a comparação com o valor que ativa o alarme e respetiva ativação do mesmo, se for caso disso. Todos os módulos que constituem a parte de dados serão explicados ao longo desta parte do relatório.

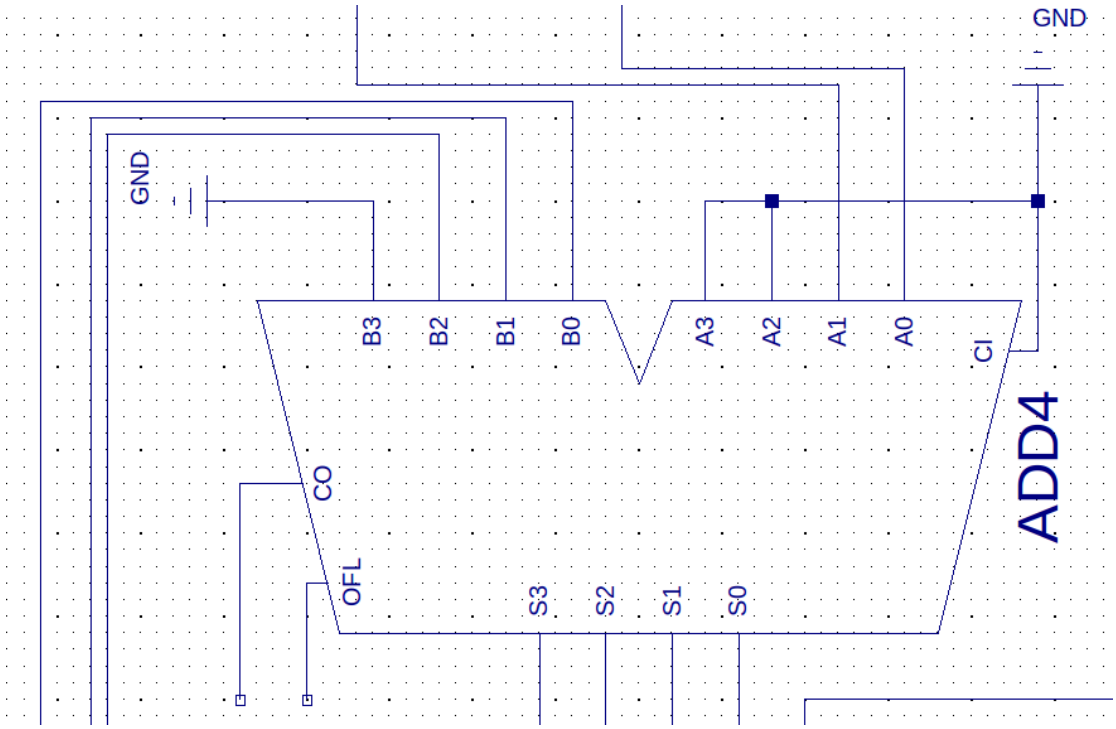


3.1 Módulo MUX

Este módulo é composto por dois multiplexers com uma entrada de seleção, um dos multiplexers para os bits menos significativos (A0 e B0) e o outro para os bits mais significativos (A1 e B1). Tem a função de fornecer o valor correto, entre A e B, para a realização das operações necessárias para obter C, consoante o sinal SEL, fornecido pela parte de controlo.



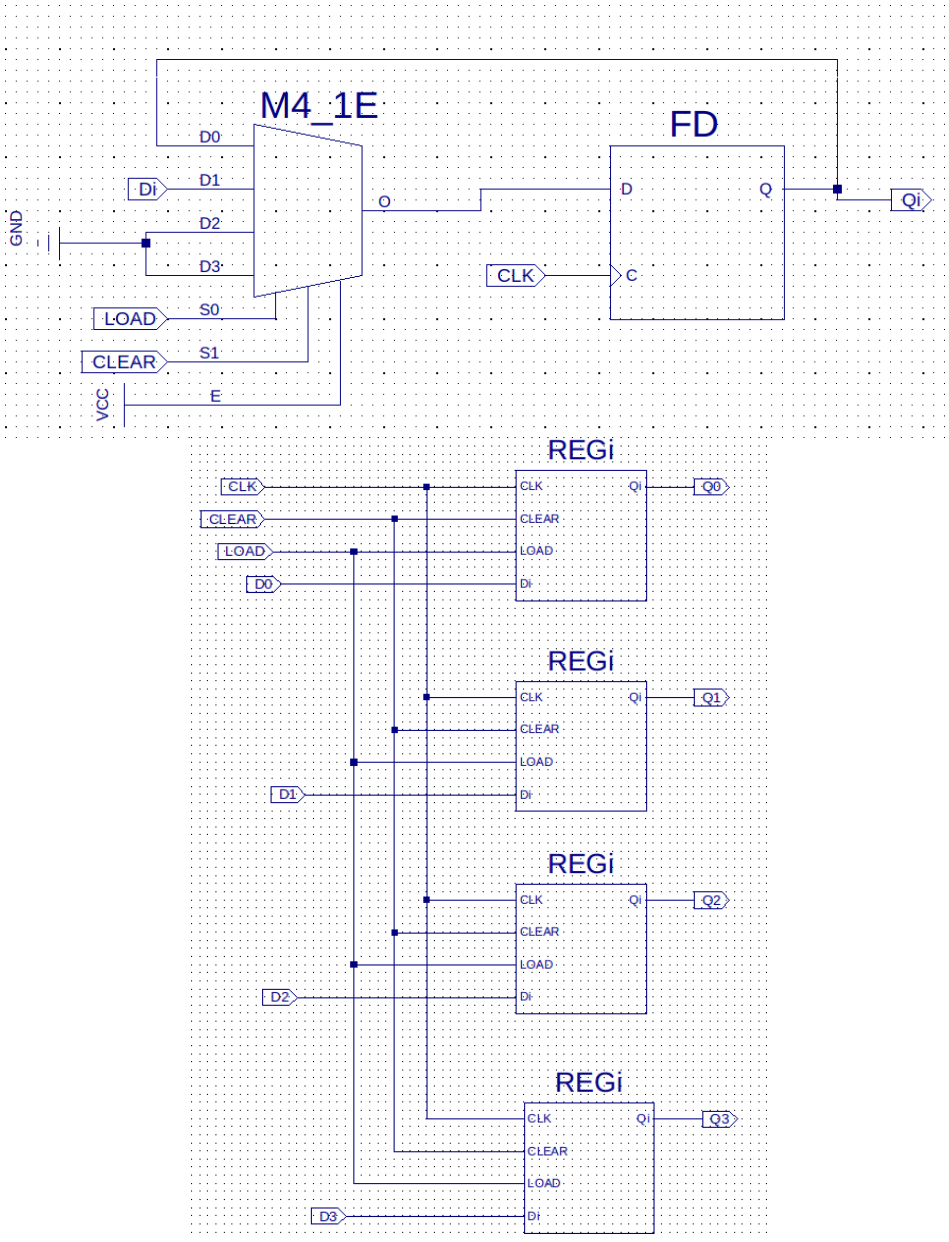
3.2 Módulo Somador



O módulo somador recebe a variável selecionada no módulo MUX e o valor guardado no registo, somando os dois. Os valores de A e B são introduzidos nas entradas A0 e A1, ficando as entradas A3, A2 e Transporte de Entrada ligadas ao GROUND, enquanto os valores guardados no registo são introduzidos nas entradas B2, B1 e B0. Como o maior número possível resultante da operação (1 * A + 2 * B) é 9, apenas são necessários 4 bits, não existindo transporte de saída. Como o resultado da penúltima soma (0 + A + B) é, no máximo, 6, que é o último valor que vai do registo para as entradas B do somador, então a entrada B3 nunca vai estar ativa, visto que 6 é um número de 3 bits, assim podemos ligar a mesma ao GROUND. O sinal OFL também não é necessário para este sistema.

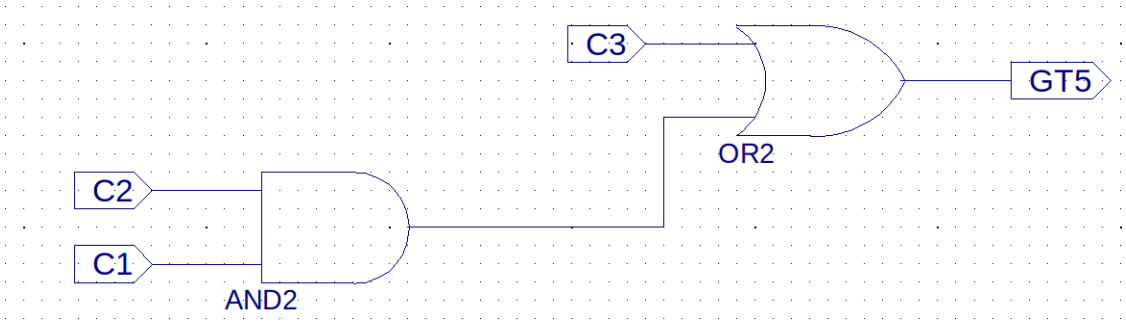
3.3 Módulo REG_4

O módulo de registo tem a função de guardar as sucessivas atualizações do valor C, à medida que os produtos e somas de A e B são executadas, e reenviá-lo para o somador no caso de ser ainda necessário efetuar alguma soma. Este módulo utiliza 4 módulos de registo de um bit (REGi), um para cada bit do valor máximo possível de C. Assim, o módulo REG4 receberá os quatro bits resultantes da soma e guardá-los-á recorrendo às capacidades do flip-flop D, podendo depois sair para o somador ou para o próximo módulo, o comparador.



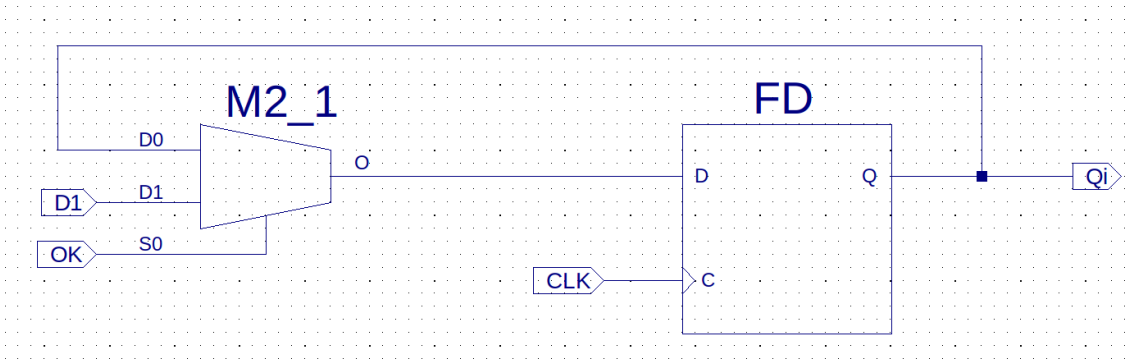
3.4 Módulo GT5

O módulo de comparação tem a função de verificar se a saída é maior que 5.



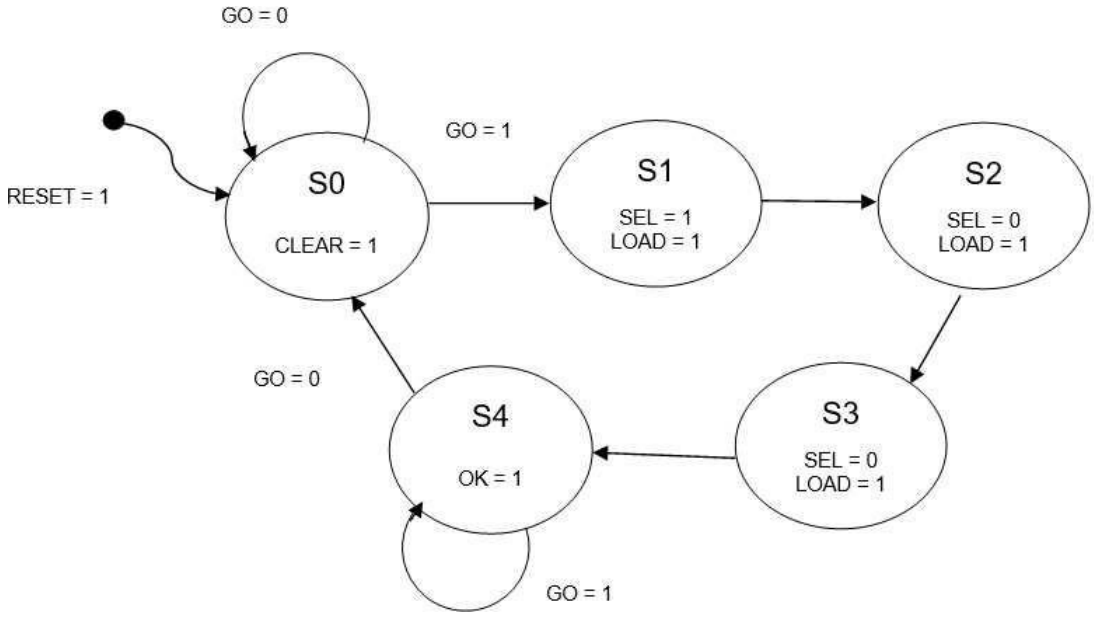
3.5 Módulo F

O módulo F é responsável por ativar e guardar o sinal de alarme, mesmo quando o sinal de entrada OK é igual a 0. Também garante que o alarme só é ativado quando o cálculo terminar, apesar de, no nosso caso, essa função não ser crucial, visto que a partir do momento que um dos resultados das somas é maior que 5, o alarme pode logo ser ativado.



4 Síntese – Parte de Controlo

4.1 Diagrama de Estados



Na base desta máquina de estados está a necessidade de somar sucessivamente um determinado valor e a necessidade de escolher qual de dois valores introduzidos somar. Deste modo considerámos que SEL = 1 significa que a variável selecionada para entrar no somador é A e, consequentemente, quando SEL = 0, a variável selecionada é B.

4.2 Tabela de Transição de Estados

Estado Atual	Estado Seguinte	
	GO = 0	GO = 1
S0	S0	S1
S1	S2	S2
S2	S3	S3
S3	S4	S4
S4	S0	S4

4.3 Codificação de Estados

Estado Atual	Codificação Q2 Q1 Q0
S0	0 0 0
S1	0 0 1
S2	0 1 0
S3	0 1 1
S4	1 0 0

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

4.4.1 Tabela de transição de estados codificados e saídas.

Na seguinte tabela é possível observar a codificação escolhida para os estados (binário natural) e a transição de estados de acordo com a variável GO. É também possível ver quais os valores das saídas CLEAR, LOAD, OK e SEL, correspondentes a cada estado.

Estado atual	**Estado seguinte**		**CLEAR**	**LOAD**	**OK**	**SEL**
Q2 Q1 Q0	GO = 0 Q2' Q1' Q0'	GO = 1 Q2' Q1' Q0'				
0 0 0	0 0 0	0 0 1	1	0	0	0
0 0 1	0 1 0	0 1 0	0	1	0	1
0 1 0	0 1 1	0 1 1	0	1	0	0
0 1 1	1 0 0	1 0 0	0	1	0	0
1 0 0	0 0 0	1 0 0	1	0	1	0
1 0 1	X X X	X X X	X	X	X	X
1 1 0	X X X	X X X	X	X	X	X
1 1 1	X X X	X X X	X	X	X	X

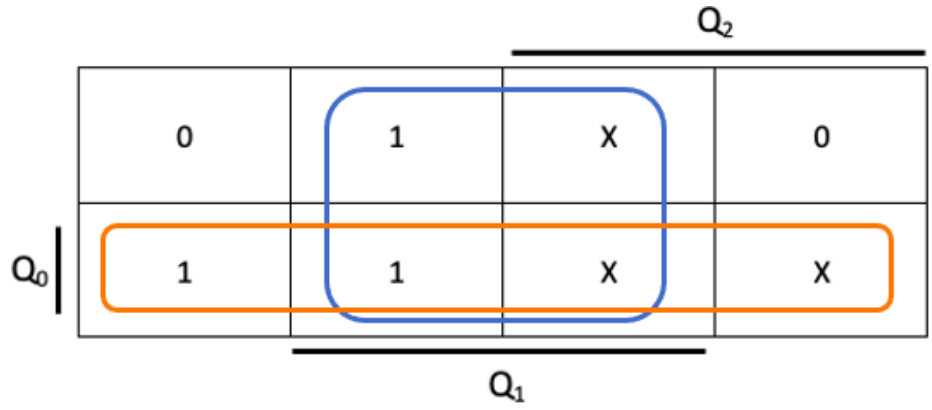
4.4.2 Tabela de entradas dos flip-flops.

Como foi dito anteriormente, foi necessário escolher que tipo de flip-flops usar. O nosso grupo decidiu fazer tabelas de verdade e mapas de Karnaugh para todos os tipos de flip-flops que conhecemos (tipo D, tipo T e tipo JK), para depois poder analisar qual o flip-flop que necessita de menos portas lógicas nas funções de entrada.

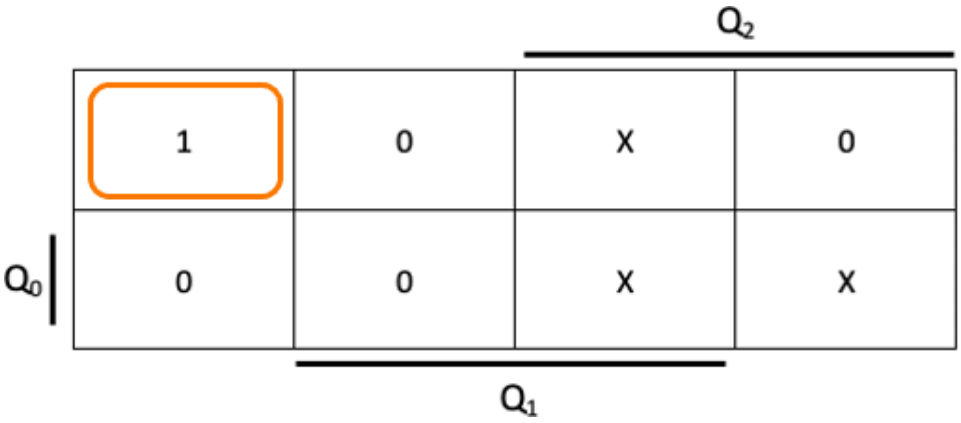
Q2	Q1	Q0	G0	Q2'	Q1'	Q0'	D2	D1	D0	T2	T1	T0	J2K2	J1K1	J0K0
0	0	0	0	0	0	0	0	0	0	0	0	0	0 X	0 X	0 X
0	0	0	1	0	0	1	0	0	1	0	0	1	0 X	0 X	1 X
0	0	1	0	0	1	0	0	1	0	0	1	1	0 X	1 X	X 1
0	0	1	1	0	1	0	0	1	0	0	1	1	0 X	1 X	X 1
0	1	0	0	0	1	1	0	1	1	0	0	1	0 X	X 0	1 X
0	1	0	1	0	1	1	0	1	1	0	0	1	0 X	X 0	1 X
0	1	1	0	1	0	0	1	0	0	1	1	1	1 X	X 1	X 1
0	1	1	1	1	0	0	1	0	0	1	1	1	1 X	X 1	X 1
1	0	0	0	0	0	0	0	0	0	1	0	0	X 1	0 X	0 X
1	0	0	1	1	0	0	1	0	0	1	0	0	X 0	0 X	0 X
1	0	1	0	X	X	X	X	X	X	X	X	X	XX	XX	XX
1	0	1	1	X	X	X	X	X	X	X	X	X	XX	XX	XX
1	1	0	0	X	X	X	X	X	X	X	X	X	XX	XX	XX
1	1	0	1	X	X	X	X	X	X	X	X	X	XX	XX	XX
1	1	1	0	X	X	X	X	X	X	X	X	X	XX	XX	XX
1	1	1	1	X	X	X	X	X	X	X	X	X	XX	XX	XX

4.5 Expressões simplificadas das saídas por Mapas de Karnaugh

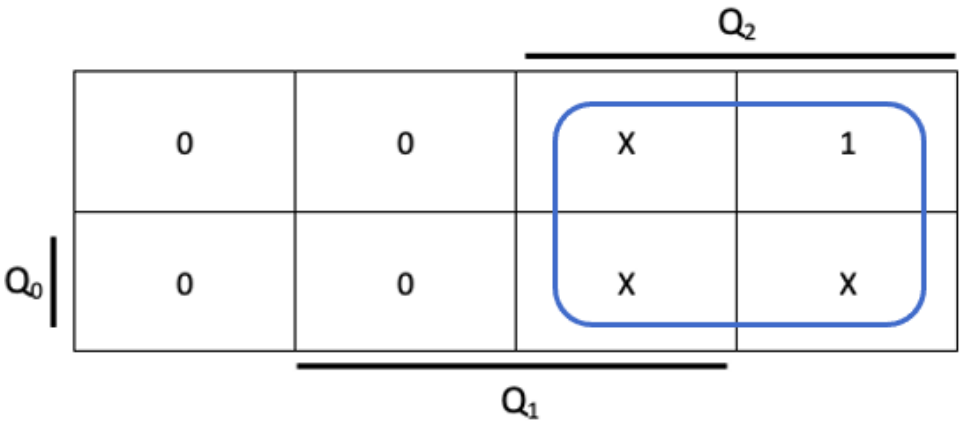
LOAD: LOAD = Q1 + Q0



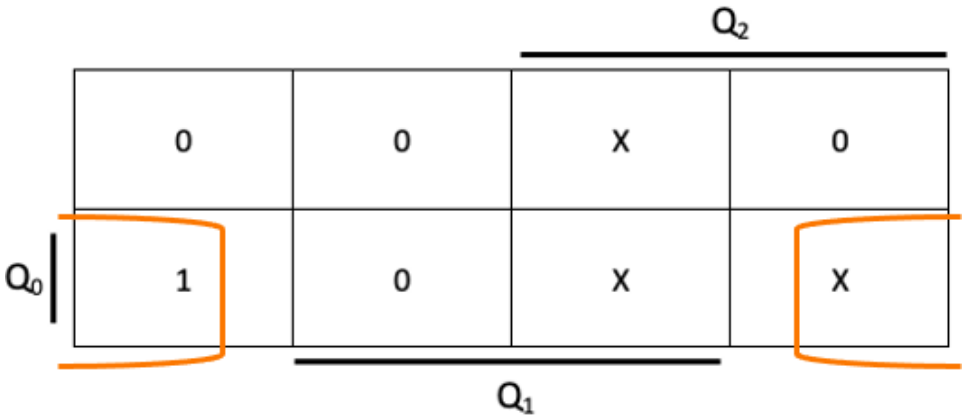
CLEAR: CLEAR = /Q0./Q1./Q2



OK: OK = Q2

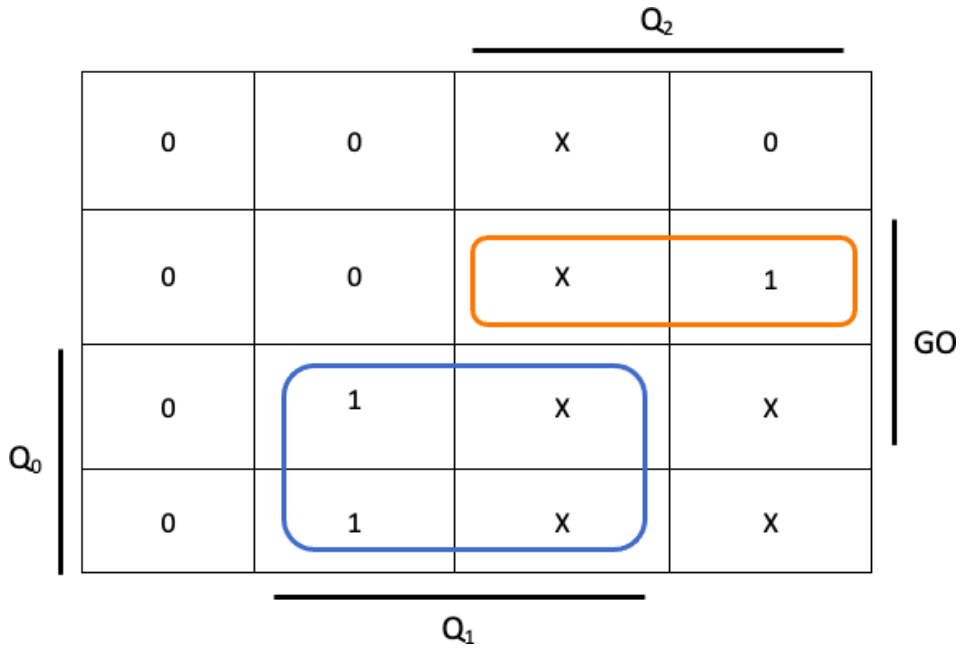


SEL: SEL = Q0./Q1

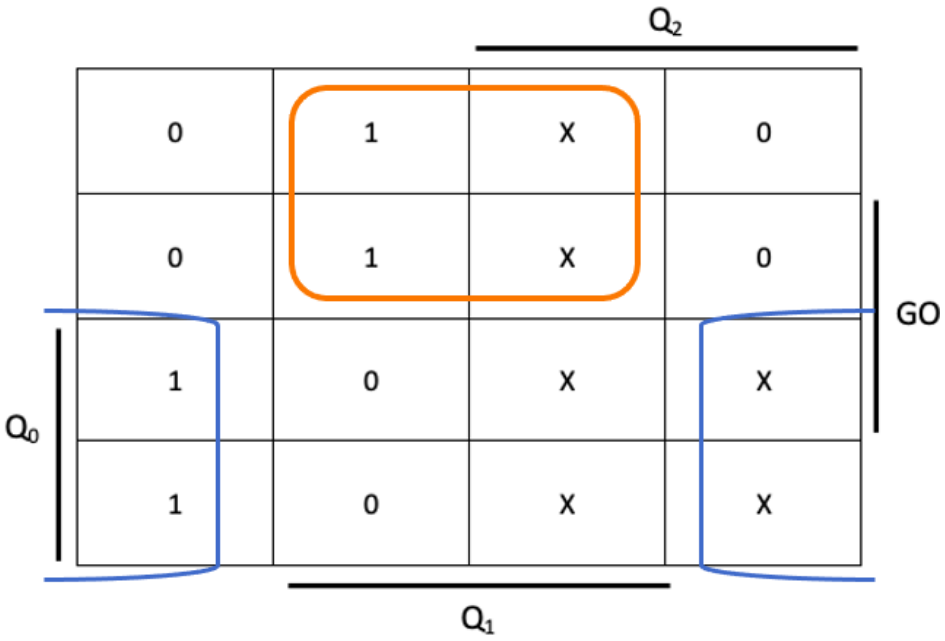


4.6 Expressões simplificadas das entradas dos Flip-flops por Mapas de Karnaugh

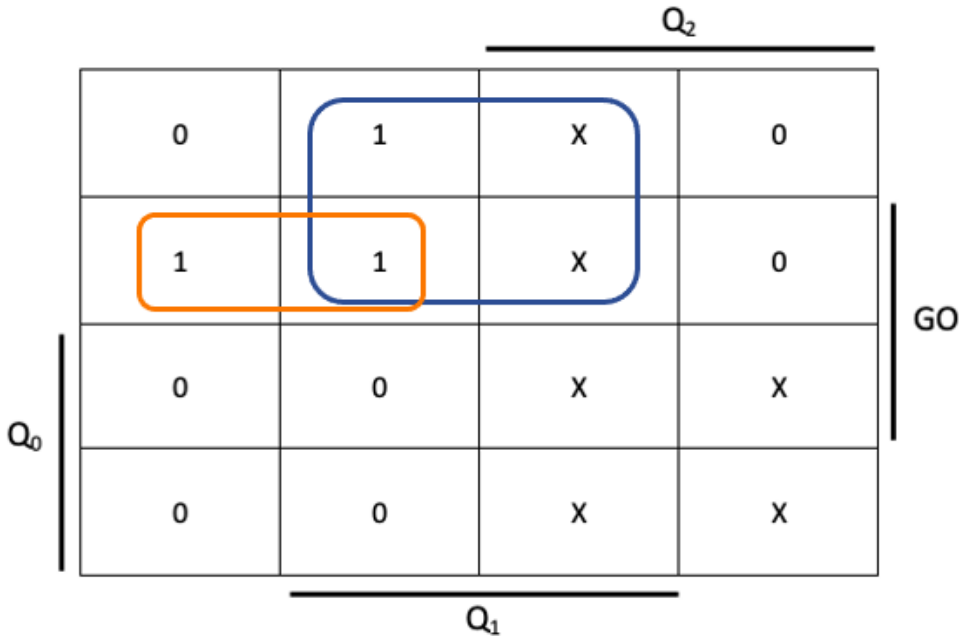
D2: D2 = Q1.Q0 + GO.Q2./Q0



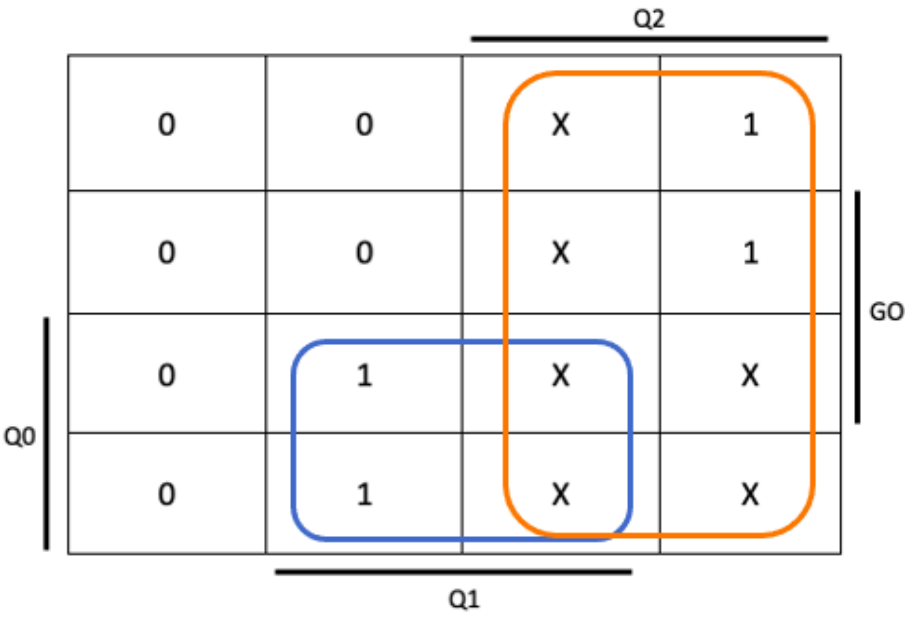
D1: D1 = Q1./Q0 + Q0./Q1 = Q1 Q0



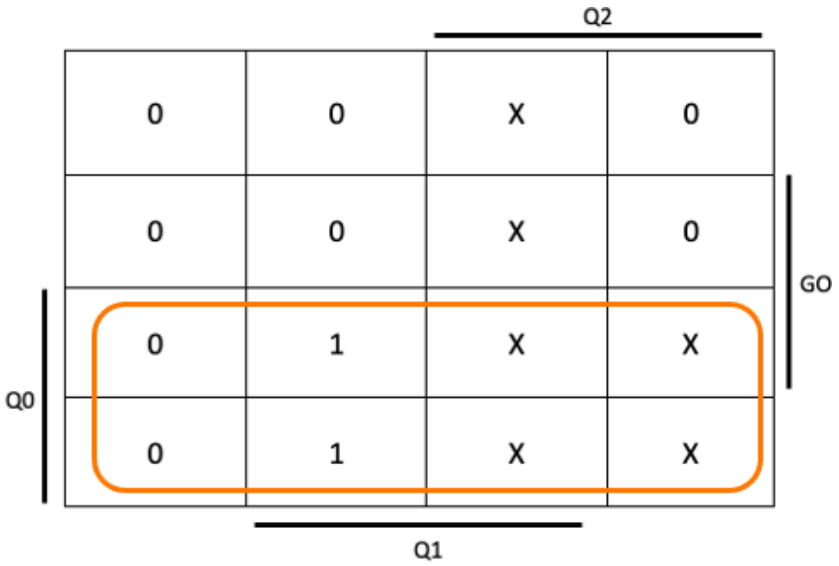
D0: D0 = Q1./Q0 + /Q2./Q0.GO



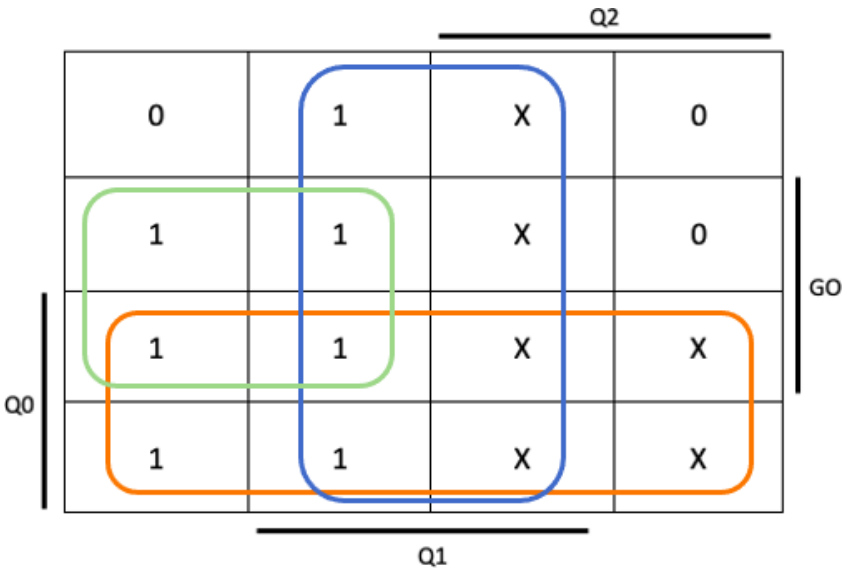
T2: T2 = Q2 + Q1.Q0



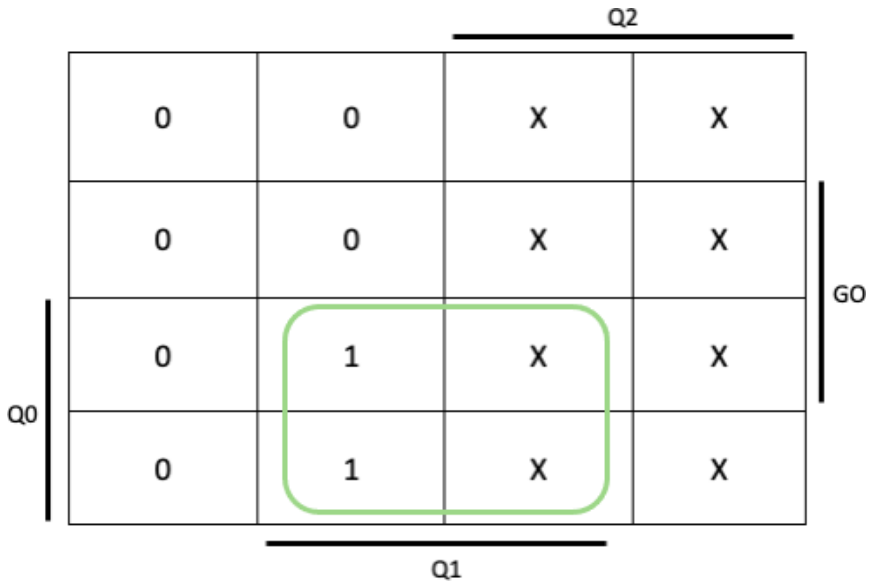
T1: T1 = Q0



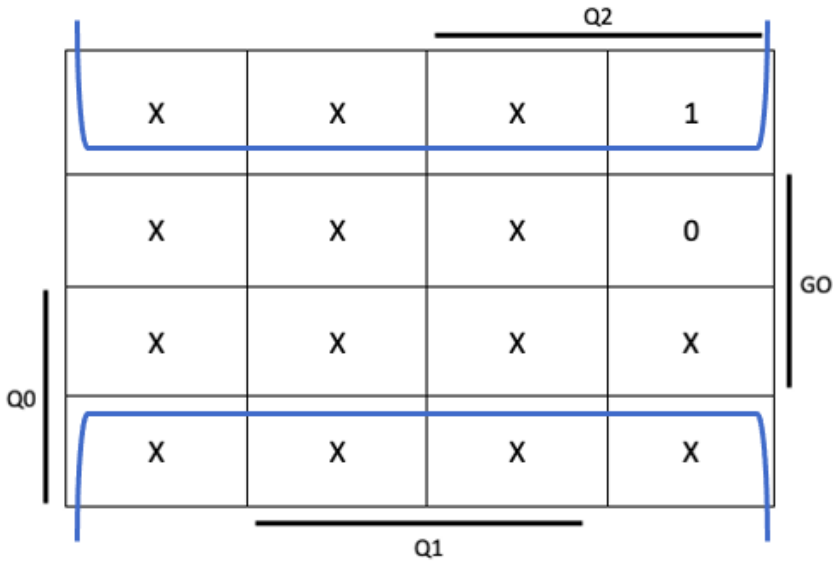
T0: T0 = Q1 + GO./Q2 + Q0



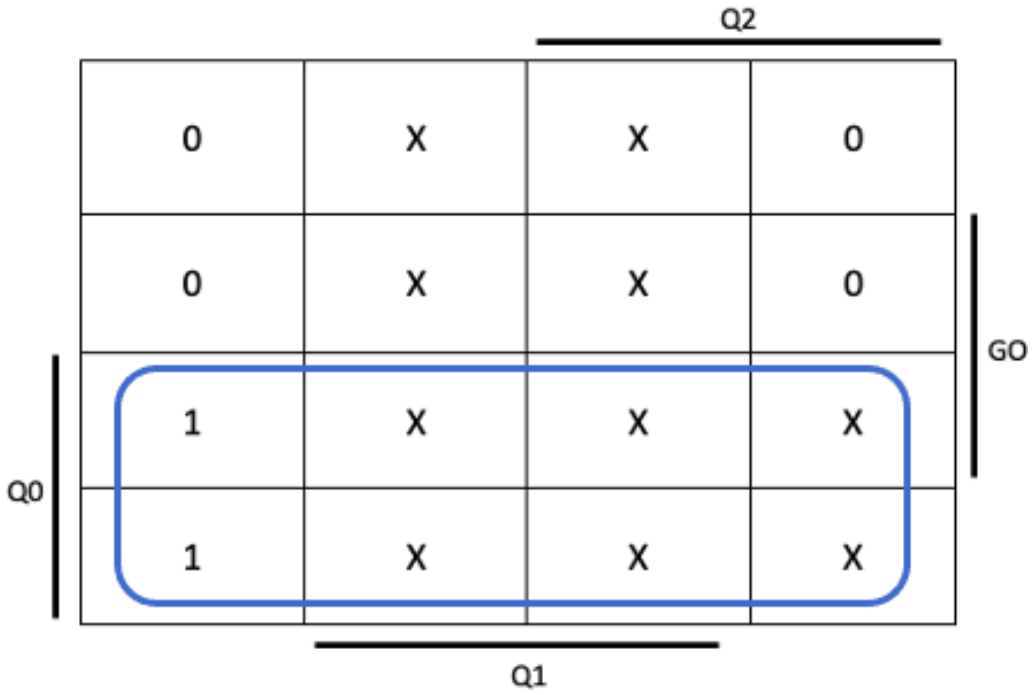
J2: J2 = Q1.Q0



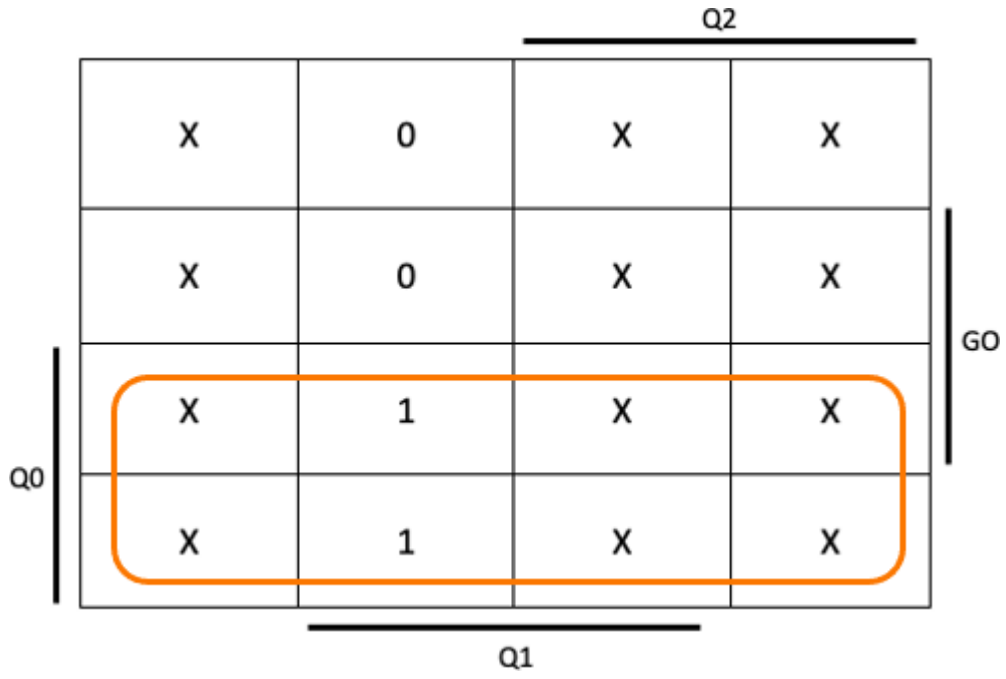
K2: K2 = /GO



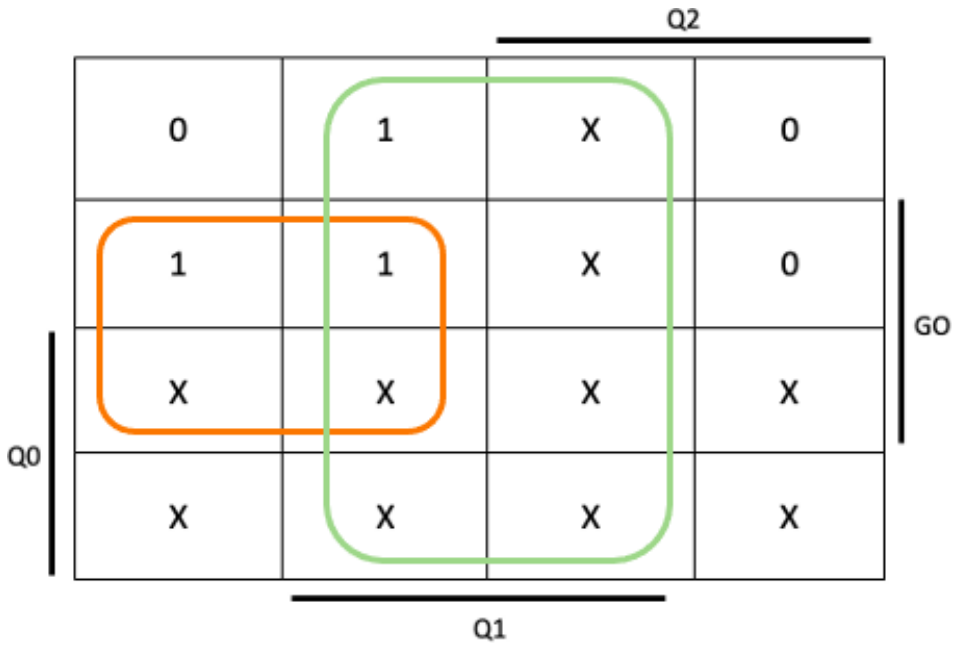
J1: J1 = Q0



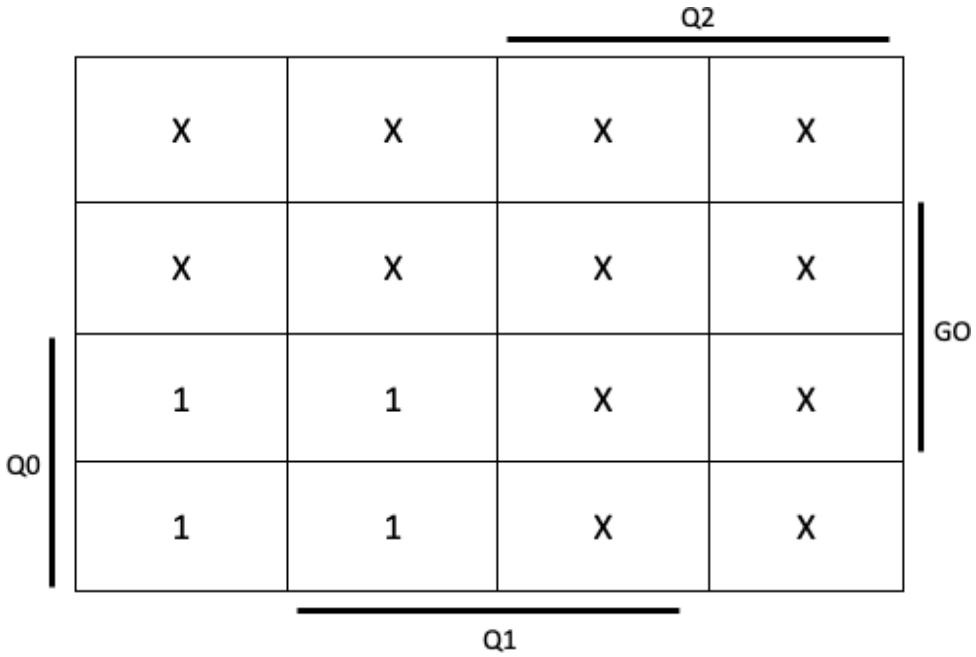
K1: K1 = Q0



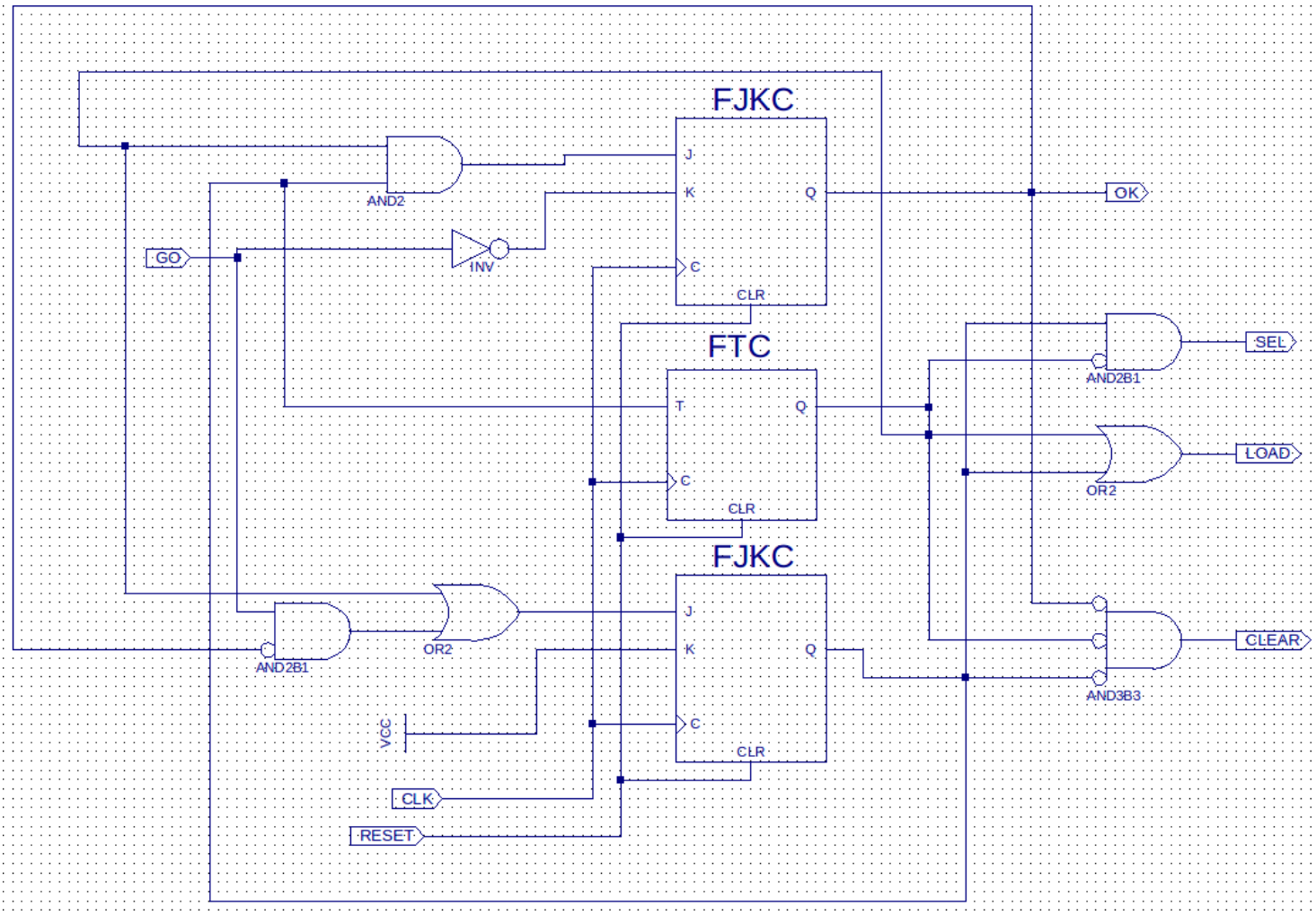
J0: J0 = Q1 + /Q2.GO



K0: K0 = 1



4.7 Esquemático da Parte de Controlo



5 Implementação na FPGA Spartan 3E

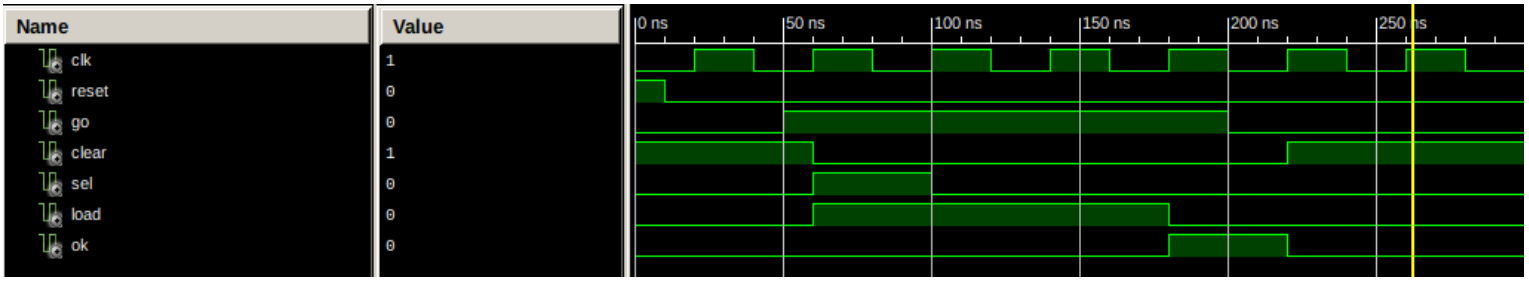
```
NET "CLK" LOC = "G12";
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;
NET "RESET" LOC = "A7";
NET "GO" LOC = "N3";
NET "A1" LOC = "B4";
NET "A0" LOC = "K3";
NET "B1" LOC = "L3";
NET "B0" LOC = "P11";
NET "ALARM" LOC = "G1";
NET "OK" LOC = "P4";
NET "C3" LOC = "P6";
NET "C2" LOC = "P7";
NET "C1" LOC = "M11";
NET "C0" LOC = "M5";
```

6 Validação através de simulações

6.1 Testes unitários

6.1.1 Teste da parte de controlo

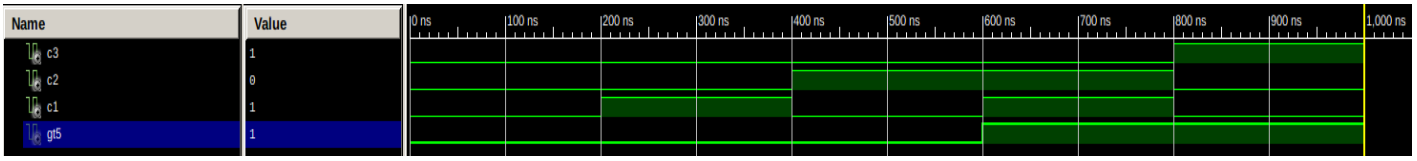
Para verificar o bom funcionamento da parte de controlo, temos de verificar que: quando RESET toma o valor 1, a máquina vai para o estado S0 (CLEAR = 1); quando GO toma o valor 1, inicia-se o sistema, ou seja, a máquina vai para o estado S1 (SEL=1, LOAD=1), desse para o S2 (SEL = 0, LOAD = 1), desse para o S3 (SEL=0, LOAD=1), e finalmente para o S4 (OK=1), permanecendo nesse estado até que GO tome o valor 0, que faz com que a máquina volte para o estado S0 (CLEAR=1) e permaneça nesse estado. É precisamente isso que se denota quando se simula a parte de controlo:




```
tb : PROCESS
BEGIN
        CLK <= '0' ; wait for 20 ns;
        CLK <= '1' ; wait for 20 ns;
END PROCESS;
        RESET <= '1', '0' AFTER 10 ns;
        GO <= '0', '1' after 50 ns, '0' after 200 ns;
```

6.1.2 Teste comparador

O comparador deve devolver o valor lógico 1 quando a variável de entrada (C) é maior que 5, e o valor lógico 0 quando é menor ou igual a 5. É exatamente isso que se verifica:



C3 <= '0', '1' after 800 ns;

C2 <= '0', '1' after 400 ns, '0' after 800 ns, '1' after 1200 ns;

C1 <= '0', '1' after 200 ns, '0' after 400 ns, '1' after 600 ns, '0' after 800 ns, '1' after 1000 ns, '0' after 1200 ns, '1' after 1400 ns;

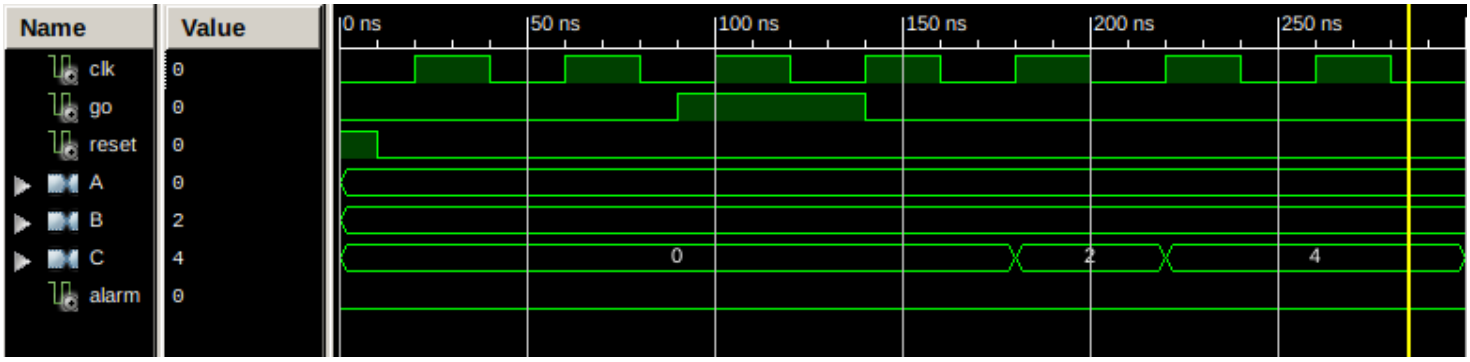
6.2 Testes de sistema

Para testar a implementação deste sistema considerámos as seguintes situações de teste:

Teste	A	B	ALARM
1	0	2	0
2	0	3	1
3	3	1	0
4	1	3	1

6.3 Teste 1

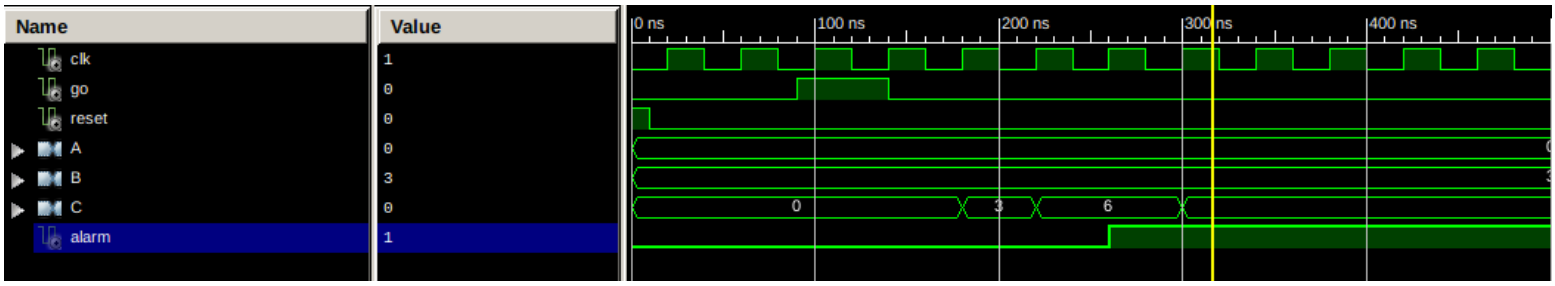
Teste de valores para A e B (A = 00, B = 10) que garantem que C = 4 <= 5 e, consequentemente, a não ativação do alarme.




```
tb : PROCESS
BEGIN
        CLK <= '0' ; wait for 20 ns;
        CLK <= '1' ; wait for 20 ns;
END PROCESS;
        GO <= '0', '1' after 90 ns, '0' after 140 ns;
        A0 <= '0';
        A1 <= '0';
        B1 <= '1';
        B0 <= '0';
        RESET <= '1','0' after 10 ns;
```

6.4 Teste 2

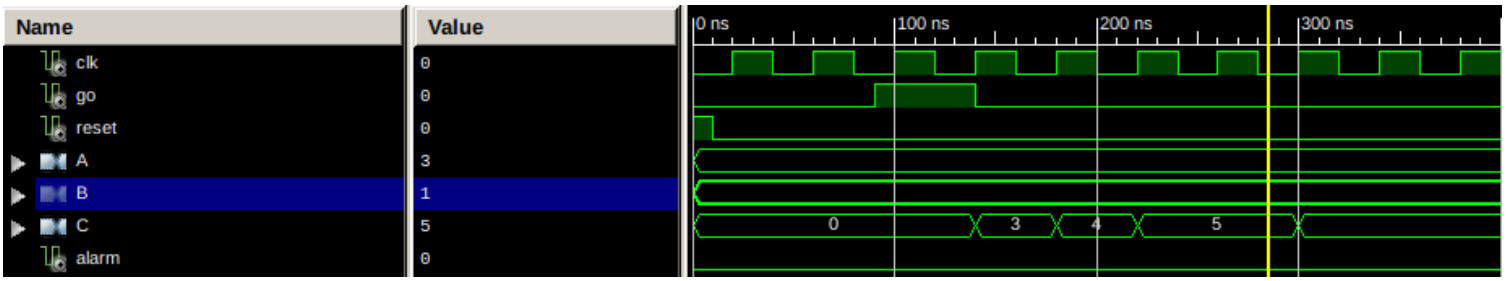
Teste de valores para A e B (A = 00, B = 11) que garantem que C = 6 > 5 e, consequentemente, a ativação do alarme.




```
tb : PROCESS
BEGIN
        CLK <= '0' ; wait for 20 ns;
        CLK <= '1' ; wait for 20 ns;
END PROCESS;
        GO <= '0', '1' after 90 ns, '0' after 140 ns;
        A0 <= '0';
        A1 <= '0';
        B1 <= '1';
        B0 <= '1';
        RESET <= '1','0' after 10 ns;
```

6.5 Teste 3

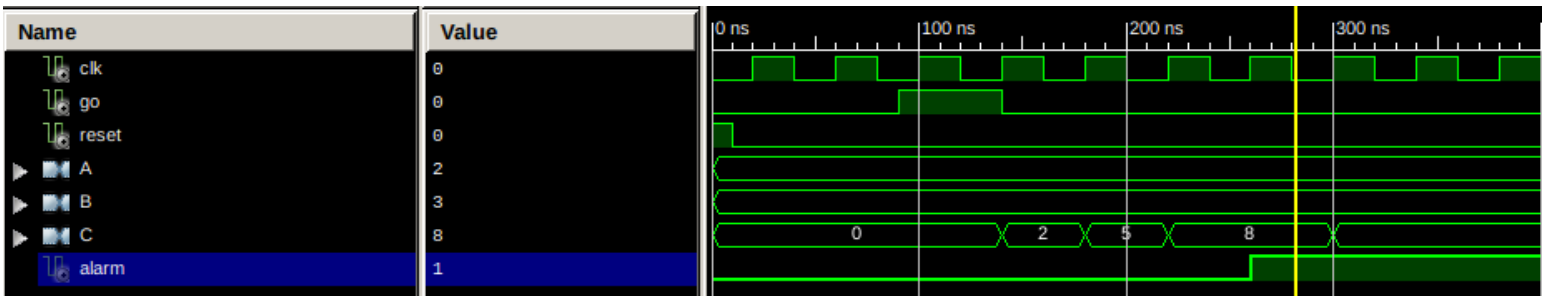
Teste de valores para A e B (A = 11, B = 01) que garantem que C = 5 <= 5 e, consequentemente, a não ativação do alarme.




```
tb : PROCESS
BEGIN
        CLK <= '0' ; wait for 20 ns;
        CLK <= '1' ; wait for 20 ns;
END PROCESS;
        GO <= '0', '1' after 90 ns, '0' after 140 ns;
        A0 <= '1';
        A1 <= '1';
        B1 <= '0';
        B0 <= '1';
        RESET <= '1','0' after 10 ns;
```

6.6 Teste 4

Teste de valores para A e B (A = 01, B = 11) que garantem que C = 8 > 5 e, consequentemente, a ativação do alarme.

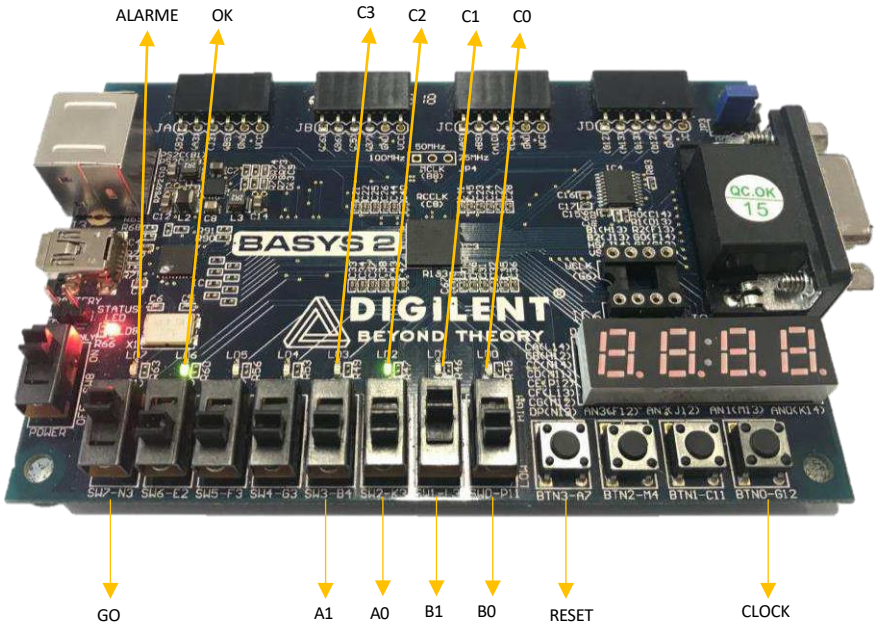



```
tb : PROCESS
BEGIN
        CLK <= '0' ; wait for 20 ns;
        CLK <= '1' ; wait for 20 ns;
END PROCESS;
        GO <= '0', '1' after 90 ns, '0' after 140 ns;
        A0 <= '0';
        A1 <= '1';
        B1 <= '1';
        B0 <= '1';
        RESET <= '1','0' after 10 ns;
```

7 Resultados experimentais

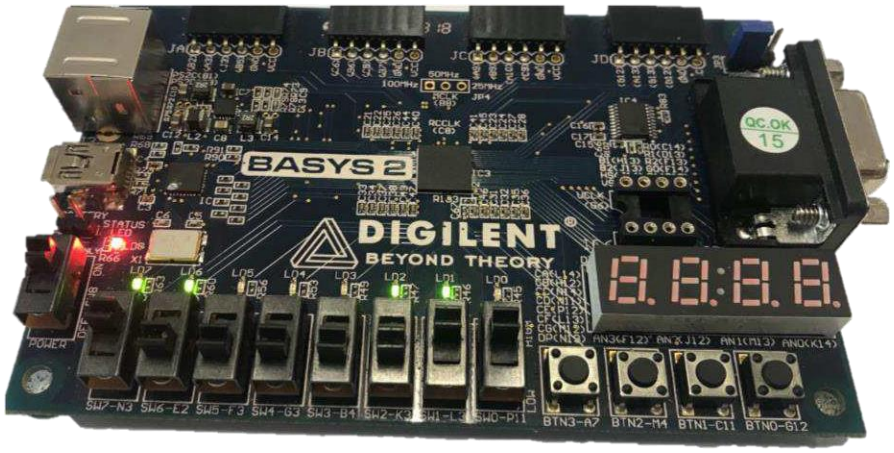
Teste 1

Operação: 1 * 0 + 2 * 2 > 5, ou seja, 4 > 5. Como a inequação não se verifica, o led associado ao alarme permanece desligado.



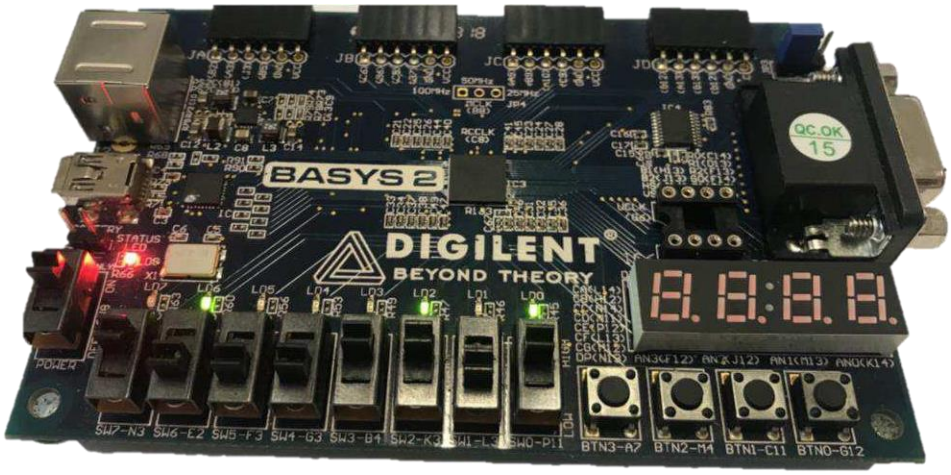
Teste 2

Operação: 1 * 0 + 2 * 3 > 5, ou seja, 6 > 5. Como a inequação se verifica, o led associado ao alarme fica ligado.



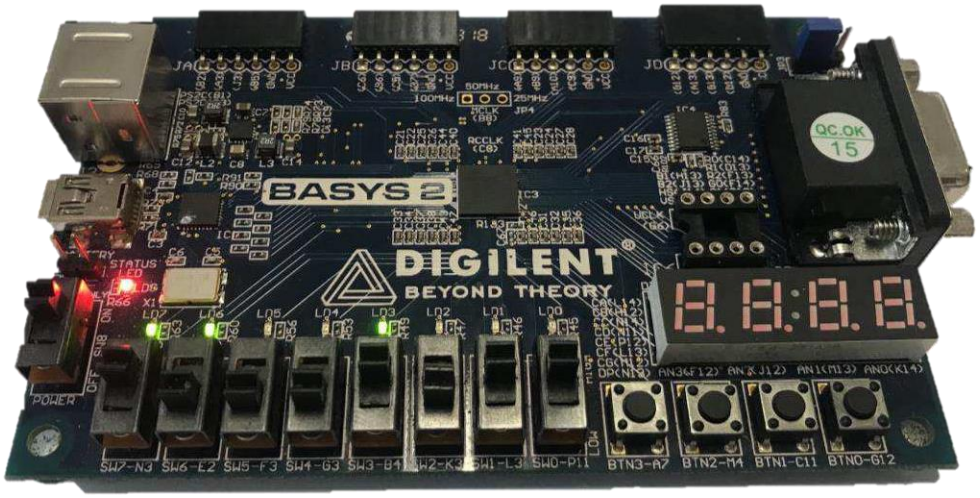
Teste 3

Operação: 1 * 3 + 2 * 1 > 4, ou seja, 5 > 5. Como a inequação não se verifica, o led associado ao alarme permanece desligado.



Teste 4

Operação: 1 * 2 + 2 * 3 > 5, ou seja, 8 > 5. Como a inequação se verifica, o led associado ao alarme fica ligado.



8 Conclusões e Observações

Concluindo, a resolução do trabalho foi muito satisfatória. Conseguimos minimizar os recursos usados criando o nosso próprio módulo comparador, em vez de usar um módulo do xilinx (que requeria muitas mais portas), e também fazendo mapas de Karnaught para todo o tipo de flip-flops que conhecemos, de modo a escolher aqueles que minimizassem o número de portas nas funções de entrada. As simulações foram bem sucedidas, logo consideramos este projeto um sucesso. A carga horária utilizada rondou as 12 horas, sendo que a maior parte foi feita em aula. Pensamos que foi um trabalho muito interessante, pois foi possível usar todos os conhecimentos que adquirimos ao longo do semestre e colocá-los em prática, inclusive ver os resultados na placa FPGA, que foi muito gratificante.