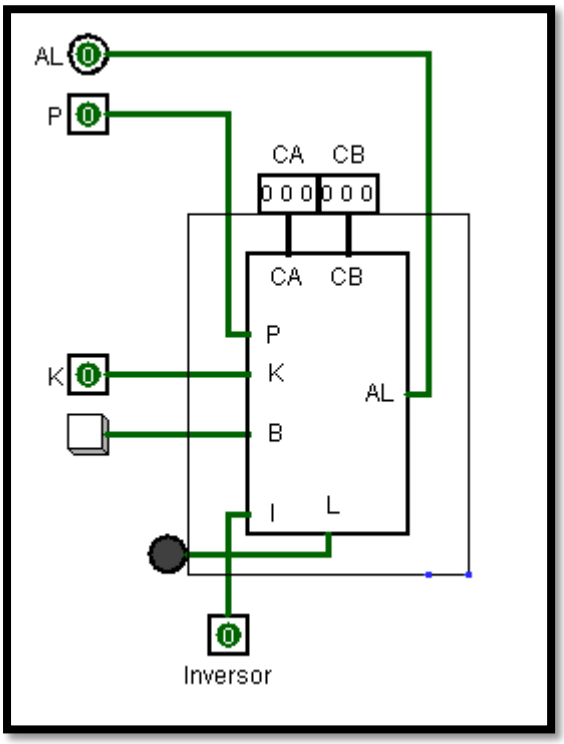
ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

*18/19i*

# Lógica e Sistemas Digitais

**Realizado pelo grupo 7:**
**Paulo Rosa – 44873**




Docente: Prof. José Antão

*28/10/18*

# 3.º Trabalho prático

Retoma-se o projecto do 1.º trabalho prático, cujo objectivo é desenvolver um sistema de alarme para um cofre, como mostra a figura 1.

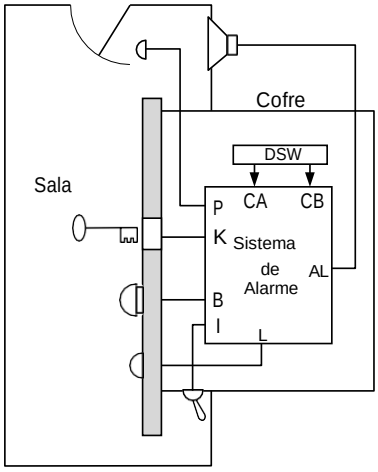



Figura 1 – Diagrama do geral do sistema

Mantém-se o mesmo comportamento no que diz respeito à porta, à chave e às exigências da utilização do inversor, mas adiciona-se segurança ao sistema no que diz respeito ao código de acesso. Para tal, substitui-se o módulo comparador por um novo módulo de validação.
A abertura da porta do cofre é mecânica e depende exclusivamente da manobra da chave **K**.

A estrutura do sistema de alarme será a que se apresentada na figura 2, onde se referem os vários sinais envolvidos:

- Um sensor **K** que indica a presença da chave;
- Um sensor **P**, activo quando a porta da sala está aberta;
- Um interruptor **I** que modifica o critério de validação do código;
- Dois **DIP *switches*** ocultos, **CA** e **CB** com três bits cada, para estabelecer o código secreto.
- Um botão de pressão **B**, dedicado à introdução do código de acesso;
- Um avisador luminoso **L**, para indicar o estado do sistema;
- Um sinal **V**, para informar o controlo que o código introduzido é **V**álido.

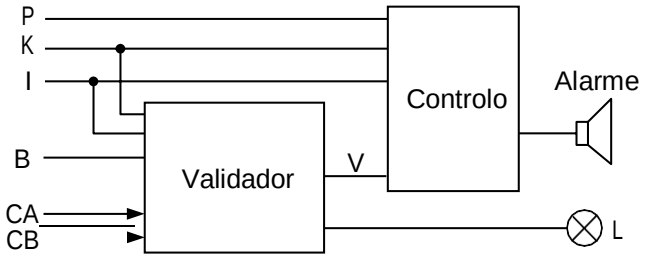



Figura 2 – Diagrama de blocos do sistema de alarme

**Operação**
Para que a chave possa ser inserida sem que o alarme soe, é necessário introduzir o código de acesso através do botão **B**. Para tal, deve premir-se o botão **B**, ao que o sistema responde acendendo a lâmpada **L** durante 5 s (±1 segundo). Neste intervalo de tempo são aceites sucessivas actuações (impulsos) sobre o botão **B**, as quais devem igualar "em número" o primeiro dígito a introduzir. Terminado aquele tempo, a lâmpada **L** apaga-se, durante 5 s, para que seja introduzido o segundo dígito. Consoante o valor da entrada **I**, o primeiro dígito a considerar será **CA** ou **CB** (**I**=0 implica **CA**).
A actuação de **B** que promove o início do processo também é contabilizada.

Finalizado o processo de introdução dos dois dígitos, caso estes coincidam com o código guardado nos DIP *switches*, o módulo de validação volta a acender a lâmpada **L** e a activar **V** durante 10 s, permitindo que neste intervalo de tempo a chave **K** seja inserida sem que o alarme soe.
Enquanto a chave **K** estiver presente, o sistema mantém a lâmpada acesa (e o sinal **V** activo). Retirada a chave, o sistema volta ao estado inicial.
Caso o código introduzido não coincida com o código guardado nos DIP *switches*, o sistema retoma o estado inicial, não chegando a acender a lâmpada **L**.
As arquitecturas propostas para o módulo validador, apresentadas na figura 3, são simples esquemas de princípio de funcionamento, podendo ser ajustadas ou até mesmo substituídas, mediante acordo, entre cada grupo de alunos e o docente da respectiva turma.

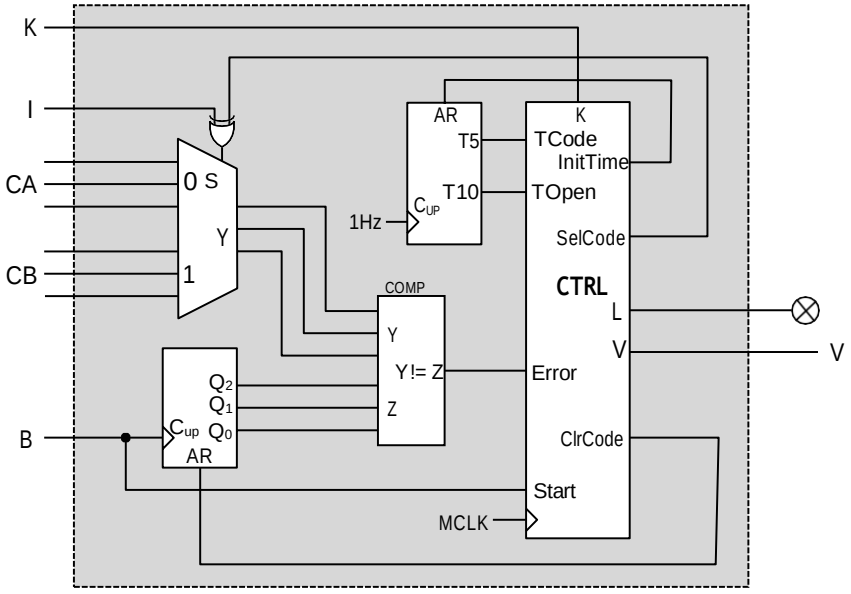



Figura 3 - Esquema de princípio de funcionamento do módulo Validador

De forma a melhor explorar o processo de síntese e teste, solicita-se a execução do trabalho em duas fases:
**1.ª fase**: Realize o módulo de controlo (CTRL) recorrendo a *flip-flops* do tipo D, para suporte à implementação da memória de estado. Utilize um dispositivo de lógica programável (PAL), para a implementação da lógica combinatória geradora de estado seguinte e dos sinais de saída. Nesta fase, o sinal MCLK deve ser produzido manualmente, através de um comutador. As entradas e saídas do módulo de controlo deverão ser simuladas, respectivamente, por comutadores e LEDs da base de ensaio (ATB). Note que na 1.ª fase, não se implementam os outros módulos representados no diagrama de blocos, mas o *ASM-chart* deve corresponder já aos requisitos necessários à implementação da 2.ª fase.
**2.ª fase**: Nesta fase, a implementação do módulo de controlo utiliza os *flip-flops* internos da PAL. Implementam-se agora, também sobre esse dispositivo de lógica programável, todos os outros módulos funcionais. Os sinais de *clock* (incluindo MCLK) deverão ser gerados por osciladores externos, disponíveis na base de ensaio.

Após a apresentação, cada grupo elabora um relatório sobre o trabalho, do qual conste:

- **o enunciado do trabalho**;
- a descrição sucinta dos métodos adoptados no projecto;
- em anexo os esquemas lógicos (ORCAD ou Logisim) e o código CUPL.

A apresentação do trabalho decorre no laboratório em data a combinar com o respectivo docente.

# Introdução

O objetivo deste trabalho é desenhar e construir, em hardware, um circuito que implemente as funcionalidades de um cofre com as características dadas no enunciado. Lido o enunciado, pode-se então prosseguir para a sua interpretação.

O circuito contém 10 entradas(sem contar com o MCLK e o CLK de 1hrtz):

P(Porta), se for 0, está fechada
K(Chave), se for 0, não está inserida
B, botão que forma um número de 3 bits
CA e CB, códigos em DIP switches, que juntos, definem o código para o cofre de 6 bits
I(Inversor), inverte a ordem da comparação dos códigos(A=0 implica CA primeiro)

E 2 saídas:

AL(Alarme), se for 1, é porque ocorreu alarme
Luz se for 1, é porque o primeira parte do código está a ser inserido ou o código inserido está certo para abrir o cofre

O cofre tem como código definido para o abrir, os bits de CA e do CB. E temos o botão B, que cada vez que é premido, forma um número de 3 por incremento com o valor que tinha anteriormente. É um contador ascendente. Logo, conta +1 quando é premido.

Por este contador registar e contar em 3 bits e o código para abrir o cofre ser de 6, é por isso que se tem este código separado em 2 conjuntos de 3 bits(CA e CB), para os podermos comparar de cada vez com o comparador

Essa transição de comparar um e outro é realizada pelo sinal que sai da máquina de estados (CTRL) que se chama SelCode. E podemos mudar a ordem de comparação dos códigos através do sinal I com o efeito da porta XOR. Cujo sinal afeta a saída do MUX 2x3 para ou sair os 3 bits de CA ou de CB para entrarem no comparador e que também recebe os bits do contador que forma o código inserido com o botão B. Se primeiro queremos inserir o código com o botão para corresponder ao CA e depois inserir o código para CB, o I terá que ser 0. Se for 1, as comparações serão ao contrário.

O outro contador de 4 bits inicia uma contagem de tempo sempre que se pretende usar o botão para definir o códico(num tempo de 5 segundos) e para restringir o tempo em que é possível inserir o chave depois de ter-se verificado que a comparação entre os dois B's e os C's é igual.

O alarme funciona varia acordo com as variáveis P, I e V(variável intermédia que representa B1+B2 = CA+CB).. Estas 3 primeiras variáveis formam 3 camadas de segurança para o sistema.

As imagens, código e desenhos estão em anexo.

# Realização

## Fase 1

Nesta fase pretende-se realizar apenas o CTRL. E não usar os flip flops da PAL ATF750C(v750c em CUPL), mas sim usar flipflops externos que vão ser 2 chips 74HC74D (2 flipflops dentro de cada um, ver anexo: Imagens 1 e 2). E dos 4 em total, vão ser usados 3, porque a minha máquina de estados vai ter 7 estados.

É importante notar que, embora não se vai propriamente usar os PIN's de entrada de reset e preset, deve-se meter ambos a 0(mas como são negados, liga-se diretamente a VCC) porque os flips flops não se comportam como desejado porque se não ligarmos este fio a algum sinal, as saídas do flip flop ficam a oscilar(alta impedância/high-Z).

### CTRL(ASM, Algorithmic state machine):

### Lógica do ASM:

No estado 000, há "out" de InitTime para que o counter de 4 bits começa a 0 quando se começar a inserção do código B. O sinal que indica isto é uma ligação direta ao botão. Se o botão foi premido, segue-se então para o estado 001.

No estado 001, há "out" de Luz, porque sinaliza que está ser inserido o primeiro valor de B. Agora usa-se o valor de TCode, que vem do counter que acabou de começar a contar na entrada deste estado(porque o sinal InitTime baixou), para formar os 5s para definir o valor que se vai definir com o botão. Assim que terminar este tempo, vê-se se houve erro. Se houve erro segue-se para o estado 011.

No estado 011, só é usado na primeira tentativa de acertar uma porção do código. O propósito deste estado é de que quando a pessoa errar na primeira porção do código, não o saber. Porque se pessoa errar e voltar para o estado 000 e ao tentar usar o botão a luz vai acender e vai saber que voltou ao início e que a primeira porção do códico está errada. Logo, aqui e com este conhecimento, consegue-se cortar em metade a probabilidade de descobrir a primeira parte do códico. Quando se vem para este estado, o counter está com os bits que correspondem aos 5s, e em vez de fazer reset do counter, vamos esperar que chegue aos 10s que vai corresponder os mais 5s de quando se insere o B2 para tentar acertar na outra porção do código. Mas vai ser fútil qualquer tentativa de qualquer forma porque já se errou na primeira porção do código e depois acabado os 10s, segue-se para o estado 000(e há out de ClearCode para que quando chegue a este estado, o valor de B esteja 0 e se mantenha a 0 até ao estado 000).

Caso não haja erro na primeira parte do código, vai-se para o estado 010 que faz "out" de ClrCode para limpar o códico armazenado no counter do botão B(para facilitar a inserção do próximo código, e começar a acrescentar +1 apartir do 0). E segue sem condições para o próximo estado.

No estado 100, faz-se out do SelCode para se selecionar a outra porção do código que vai ser avaliada. Novamente, a pessoa mete o valor que quer no B(que pode ser interpretado como o segundo B ou B2), enquanto TOpen for falso. E quando for verdadeiro(é porque já se passaram 5s, da passagem dos 5, contados anteriormente, para 10) vê-se se há Error. Se houver, vai-se para o estado 010, e que de seguida vai para o estado 000, porque TOpen está verdadeiro. Isto é possível porque a frequência do MCLOCK é superior ao clock do Timer. Se não houver erro, segue para o estado 101.

No estado 101, faz-se out de InitTime, para se contar 10s no próximo estado. Segue-se sem condições para o próximo estado: 110.

No estado 110, como não houve erro nas 2 porções do código, há "out" de V e L. E dá-se 10s para meter a chave do cofre.

E finalmente, chegou-se ao estado 111. Se se chega a este estado(e ao anterior também) é porque conseguiu-se acertar no código. Unicamente e imediatamente é confirmado se

se tem a chave. Portanto, se se meteu no estado anterior, o cofre fica aberto e fica-se neste estado(e bloqueado nesta abertura enquanto houver chave). Há "out" de V,L e ClearCode(para quando se vá para o estado 000, o B estar a 0).

(Ver anexo: Imagem 4)

## Raciocínio da implementação do ASM:

Se estamos no estado 000, a única opção que queremos é que vá para o estado 001, por isso, as entradas para os mux's (e consequentemente, flip flops) vão ser 0 para S2 e S1 e o S0 só vai ser 1 quando Start for verdadeiro. E usamos as próprias variáveis deste estado para pudermos fazer "out" de InitTime

$$S0 = Start$$
$$S1 = S2 = 0$$
$$InitTime = \overline{D2} \bullet \overline{D1} \bullet \overline{D0}$$

Se estamos no estado 001, S0 será igual a 1 apenas enquanto TCode for falso ou houver erro. S1 só é possível ser 1 quando TCode for verdadeiro. E S2 continua a 0.

$$S0 = TCode + \overline{Error}$$
$$S1 = TCode$$
$$S2 = 0$$
$$L = \overline{D2} \bullet \overline{D1} \bullet D0$$

Se estamos no estado 011, apenas quando TOpen for verdadeiro, se vai para o estado 000.

$$S0 = \overline{TOpen}$$
$$S1 = \overline{TOpen}$$
$$S2 = 0$$
$$ClearCode = \overline{D2} \bullet D1 \bullet D0$$

Se estamos no estado 010, faz-se "out" de ClearCode e vai-se sem condições para o estado 100

$$S0 = S1 = 0$$
$$S2 = 1$$
$$ClearCode = \overline{D2} \bullet D1 \bullet \overline{D0}$$

Se estamos no estado 100, há "out" de SelCode. Enquando o TOpen for falso, S0 é 0, e quando verdadeiro, S0 vai ser igual a 1 nas duas ocasiões em que ou vai para 011 ou para 101. S1 só vai ser igual a 1 quando for para o estado 011, logo, quando TOpen for verdadeiro e houver Error. E S2 mantem-se a 1 enquanto TOpen for falso ou houver Error.

$$S0 = TOpen$$
$$S1 = TOpen \bullet \overline{Error}$$
$$S2 = \overline{TOpen} + \overline{Error}$$
$$SelCode = D2 \bullet \overline{D1} \bullet \overline{D0}$$

Se estamos no estado 101, há "out" de InitTime para podermos contar os 10s no próximo estado. Segue-se sem condições para o estado 110.

$$S2 = S1 = 1$$
$$S0 = 0$$
$$ClearCode = D2 \bullet \overline{D1} \bullet D0$$

Se estamos no estado 110, há "out" de V e L. E apenas quando TOpen for verdadeiro, vai-se para o estado 111.

$$S0 = S1 = 1$$
$$S2 = TOpen$$
$$V = L = D2 \bullet D1$$

Se estamos no estado 111, apenas quando K for falso se vai para o estado 000 e se for verdadeiro permanece neste estado. Há "out" de de ClearCode, V e L.

$$S2 = S1 = S1 = K$$
$$ClearCode = V = L = D2 \bullet D1$$

(Ver anexo o código CUPL 1-3
E desenhos dos circuitos em Logisim 1
E fotografia da fase 1: Imagem 3)

# Fase 2

Nesta fase, implementam-se o counter up do botão de 3 bits, o counter up do timer de 4 bits, o MUX 2X3 e o comparador de 3 bits. E usamos os flip flops do v750c.

## Mux 2x3:

$$S = SelCode \oplus I$$
$$Y0 = CA0 \bullet \bar{S} + CB0 \bullet S$$
$$Y1 = CA1 \bullet \bar{S} + CB1 \bullet S$$
$$Y2 = CA2 \bullet \bar{S} + CB2 \bullet S$$

(ver em anexo: Logisim 2)

## Comparador:

Ocorre erro quando em qualquer porção/bit do código não corresponda ao outro, logo:

$$Error = (Z0 \oplus Y0) + (Z1 \oplus Y1) + (Z2 \oplus Y2)$$

(ver em anexo: Logisim 3)

## Counter do botão:

Foram usados flipflops do tipo T para simplificar a resolução. O "Clock" vai ser o próprio botão e este counter up é de 3 bits. O primeiro flip flop (Q0) tem como entrada um sinal fixo de 1. E os próximos flipflops tem como entrada, e consecutivamente, uma porta AND da saída do flipflop anterior com a entrada dele. (Ver em anexo: Logisim 4).

$$Z0 = 1$$
$$Z1 = Z0$$
$$Z2 = Z0 \bullet Z1$$

## Counter do timer:

Este counter é praticamente igual ao anterior. A diferenças é que este é de mais um bit(por isso tem mais um flipflop) e tem 2 portas AND tal que quando os bits correspondem ao número 5 e 10, é ativado o output TCode e TOpen, respetivamente. (Ver em anexo: Logisim 5)

$$D0 = 1$$
$$D1 = D0$$
$$D2 = D0 \bullet D1$$
$$D3 = D0 \bullet D1 \bullet D2$$

$$TCode = \overline{D3} \bullet D2 \bullet \overline{D1} \bullet D0$$
$$TOpen = D3 \bullet \overline{D2} \bullet D1 \bullet \overline{D0}$$

Ver em anexo(Logisim 6 e CUPL 4, 5 e 6) estes conjuntos todos juntos e que formam o bloco "Validador".

## Controlo:

E finalmente, o controlo que já tinha sido calculado no 1º trabalho prático:

$$AL = K \bullet (\overline{V} + (\overline{I} \bullet \overline{P}))$$

Ver em anexo Logisim 7 e 8

# Conclusão

Em suma, todos os objetivos foram realizados. Foram feitas provas e testes para chegar à resolução final, que mostra ser coerente com a lógica do problema segundo o enunciado do trabalho. Foi apreciada e foi interessante a realização deste trabalho, porque representa um circuito que se pode aplicar no mundo real e que, neste caso, se aplica ao funcionamento de um cofre e foi planeado usando os conhecimentos desta cadeira de lógica e sistemas digitais.

# Anexos

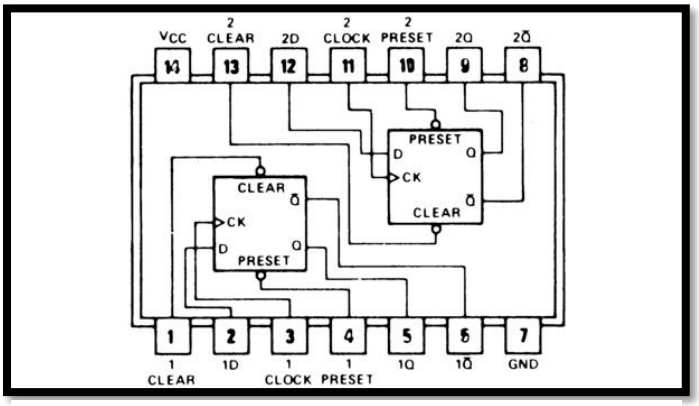



*Imagem 1, 74HC74*

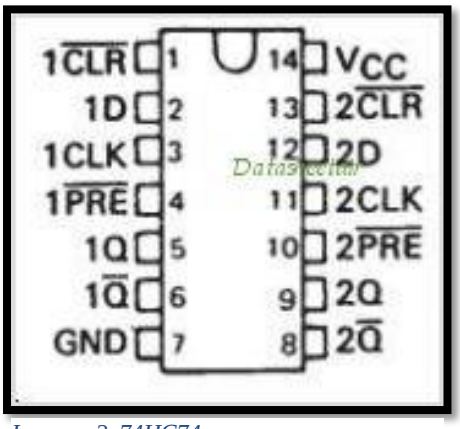



*Imagem 2, 74HC74*

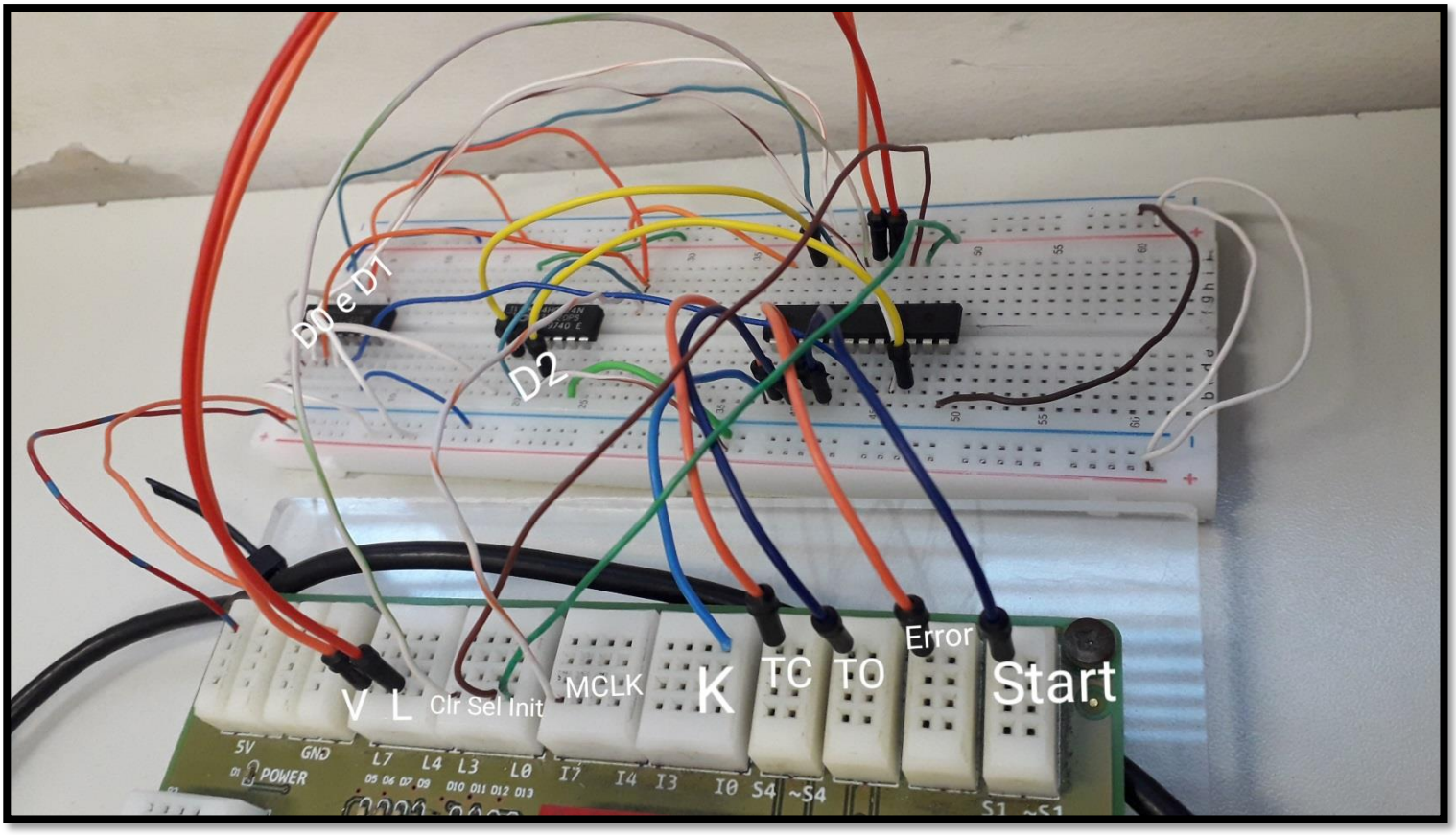
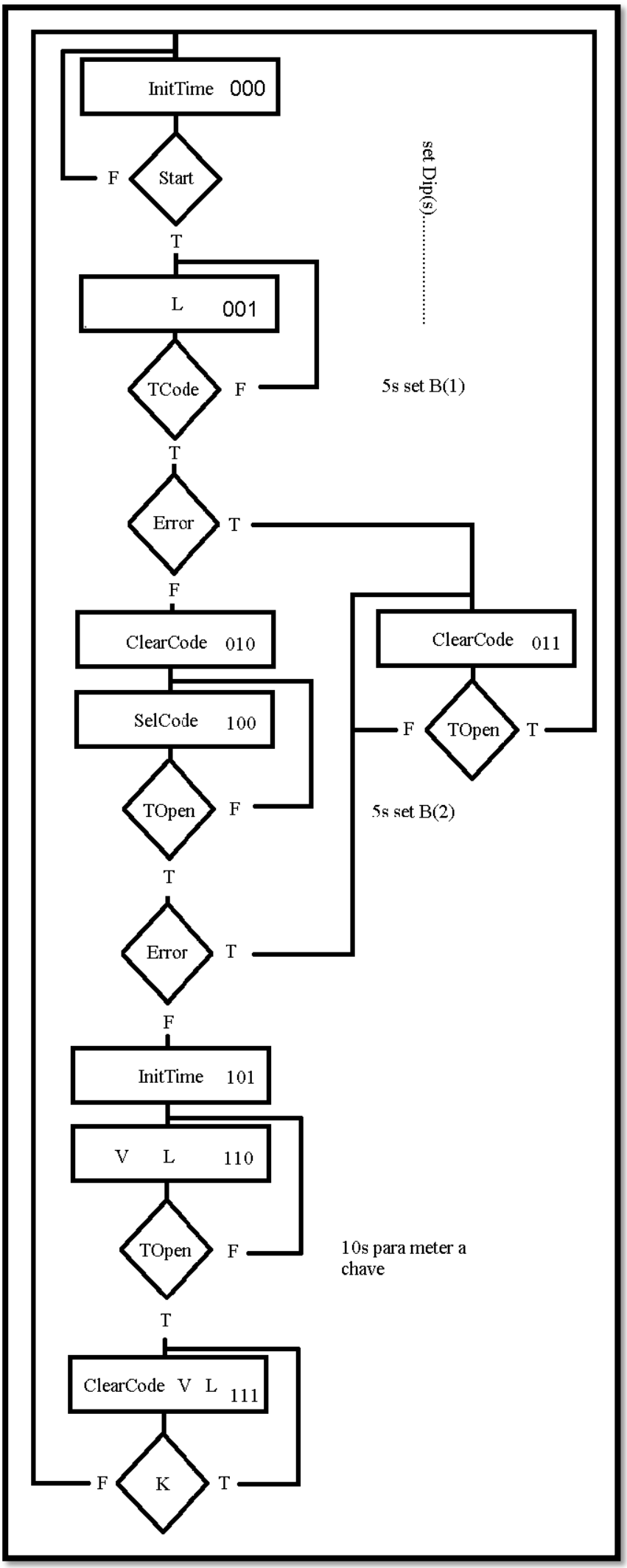



*Imagem 3, 1º fase montagem*

*Imagem 4, ASM, CTRL*



# Códigos CUPL:

```
/* *************** INPUT PINS *********************/
PIN  1  =   K;
PIN  2  =   TCode;
PIN  3  =   TOpen;
PIN  4  =   Error;
PIN  5  =   Start;

PIN  8  =   D0;
PIN  9  =   D1;
PIN  10 =   D2;

/* *************** OUTPUT PINS *********************/
PIN  15 =   InitTime;
PIN  16 =   SelCode;
PIN  17 =   L;
PIN  18 =   V;
PIN  19 =   ClrCode;
PIN  20 =   M0;
PIN  21 =   M1;
PIN  22 =   M2;
```

*CUPL 1, fase 1*

```
                 _________________
           |     Cofre3      |
     K x---|1              24|---x Vcc
TCode x---|2              23|---x
TOpen x---|3              22|---x M2
Error x---|4              21|---x M1
Start x---|5              20|---x M0
      x---|6              19|---x ClrCode
      x---|7              18|---x V
   D0 x---|8              17|---x L
   D1 x---|9              16|---x SelCode
   D2 x---|10             15|---x InitTime
      x---|11             14|---x
  GND x---|12             13|---x
          |_________________|
```

*CUPL 2, fase 1*

```
M0 = !D2&!D1&!D0 & Start     #   !D2&!D1&D0 & (!TCode # Error)   #   !D2&D1&D0 & !TOpen #
D2&!D1&!D0 & TOpen   #   D2&D1&!D0 & TOpen    #   D2&D1&D0 & K;

M1 = !D2&!D1&D0 & TCode  #   !D2&D1&D0 & !TOpen   #   D2&!D1&!D0 & (TOpen & Error)   #
D2&!D1&D0  #   D2&D1&!D0   #   D2&D1&D0 & K;

M2 = !D2&D1&!D0  #   D2&!D1&!D0 & (!TCode # !Error)   #   D2&!D1&D0  #   D2&D1&!D0   #
D2&D1&D0 & K;

ClrCode = !D2&D1&!D0 #  D1&D0;
InitTime = !D2&!D1&D0 # D2&!D1&D0;
SelCode = D2&!D1&!D0;
L = !D2&!D1&D0 # D2&D1;
V = D2&D1;
```

*CUPL 3, fase 1*

```
//* *************** INPUT PINS *********************/
PIN 1    = MCLK                         ;
PIN 2    = UMHRTZ                       ;
PIN 3    = CA2                            ;
PIN 4    = CA1                           ;
PIN 5    = CA0                           ;
PIN 6    = CB2                           ;
PIN 7    = CB1                         ;
PIN 8    = CB0                           ;
PIN 9    = I                           ;
PIN 10   = B                         ;
PIN 11   = P                          ;
PIN 13   = K                          ;


/* *************** OUTPUT PINS *********************/
PIN 14  =  Y0;
PIN 15  =  AL                             ;
PIN 16  =  ClearCode                            ;
PIN 17  =  L                            ;
PIN 18  =  Error                           ;
PIN 19  =  X1                           ;
PIN 20  =  Y2                           ;
PIN 21  = InitTime;
PIN 22  = X0                ;
PIN 23  = SelCode;

PINNODE [25..28] = [D0..3];
PINNODE [29..31] = [Z0..2];
PINNODE 32 = X2;
```

*CUPL 4, fase 2*

```
/*Controlo*/
AL = K & (!V # (!I&!P));

/*MUX para CA e CB*/
S = SelCode $ I;

Y0 = CA0&!S # CB0&S;
Y1 = CA1&!S # CB1&S;
Y2 = CA2&!S # CB2&S;

/*Cont Up botao B*/
Start = B;
[Z2..0].ar = ClearCode;
[Z2..0].sp = 'b'0 ;
[Z2..0].ck = B;

Z0.T = 'b'1;
Z1.T = Z0;
Z2.T = Z0&Z1;


/*Comparador*/
Error = (Z0 $ Y0) # (Z1 $ Y1) # (Z2 $ Y2);
```

```
/*Timer*/
[D3..0].ar = InitTime ;
[D3..0].sp = 'b'0 ;
[D3..0].ck = UMHRTZ ;

D0.T = 'b'1;
D1.T = D0;
D2.T = D0&D1;
D3.T = D0&D1&D2;

TCode = !D3 & D2 & !D1 & D0;
TOpen = D3 & !D2 & D1 & !D0;

/*CTRL*/
[X2..0].ar = 'b'0;
[X2..0].sp = 'b'0;
[X2..0].ck = MCLK;
```

*CUPL 5, fase 2*

```
SEQUENCET [X2..X0]{
    PRESENT 0
        out InitTime;
        if Start next 1;
        default next 0;
    PRESENT 1
        out L;
        if TCode & Error next 3;
        if TCode & !Error next 2;
        default next 1;
    PRESENT 2
        out ClearCode;
        default next 4;
    PRESENT 3
        out ClearCode;
        if TOpen next 0;
        default next 3;
    PRESENT 4
        out SelCode;
        if TOpen & Error next 3;
        if TOpen & !Error next 5;
        default next 4;
    PRESENT 5
        out InitTime;
        default next 6;
    PRESENT 6
        out V,L;
        if TOpen next 7;
        default next 6;
    PRESENT 7
        out V, L;
        if !K next 0;
        default next 7;

}
```

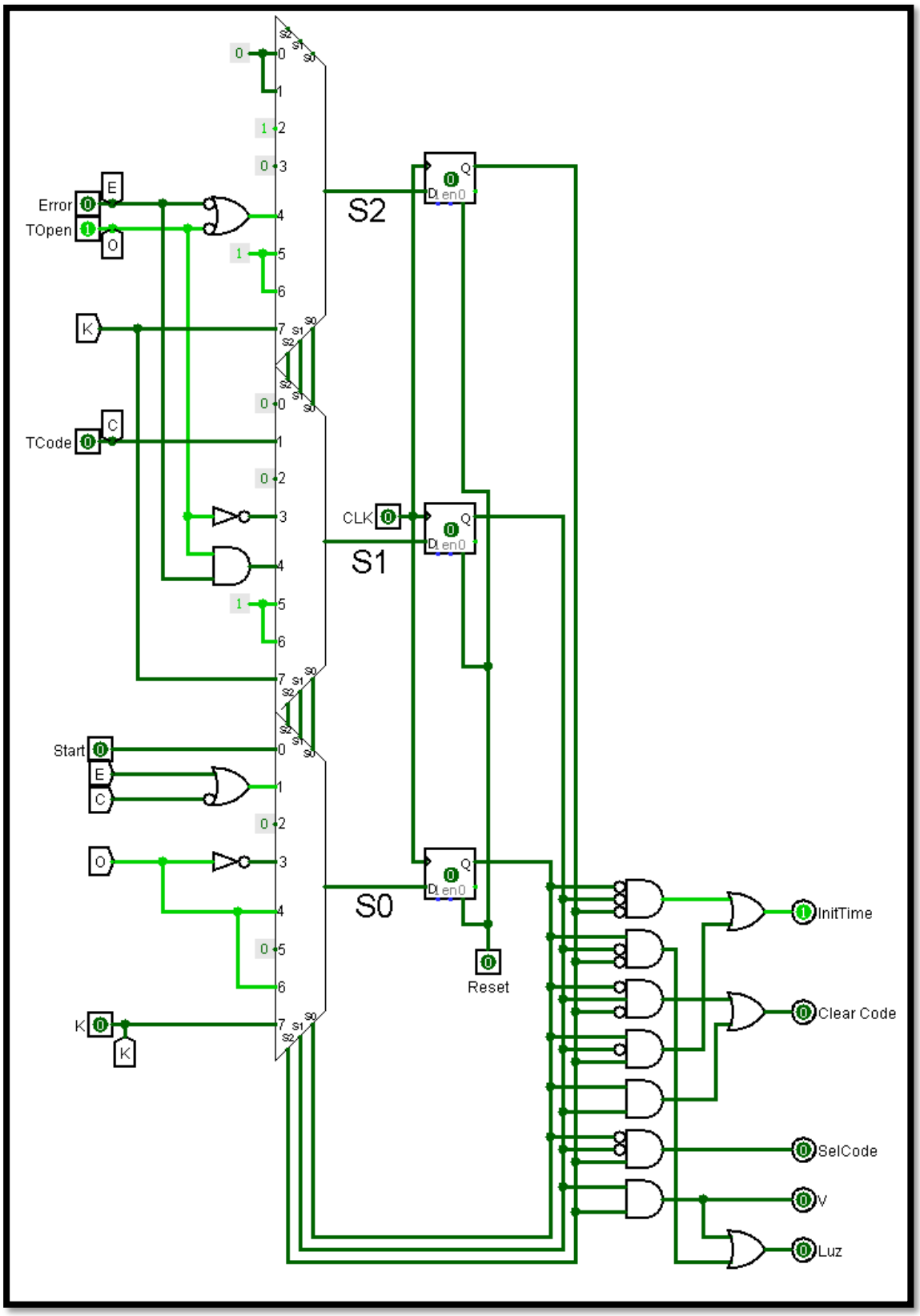
*CUPL 6, fase 2*

```
                 _____________
                |  COFREFASE2  |
  MCLK x---|1            24|---x Vcc
UMHRTZ x---|2            23|---x SelCode
   CA2 x---|3            22|---x X0
   CA1 x---|4            21|---x InitTime
   CA0 x---|5            20|---x Y2
   CB2 x---|6            19|---x X1
   CB1 x---|7            18|---x Error
   CB0 x---|8            17|---x L
     I x---|9            16|---x ClearCode
     B x---|10           15|---x AL
     P x---|11           14|---x Y0
   GND x---|12           13|---x K
           |_____________|
```

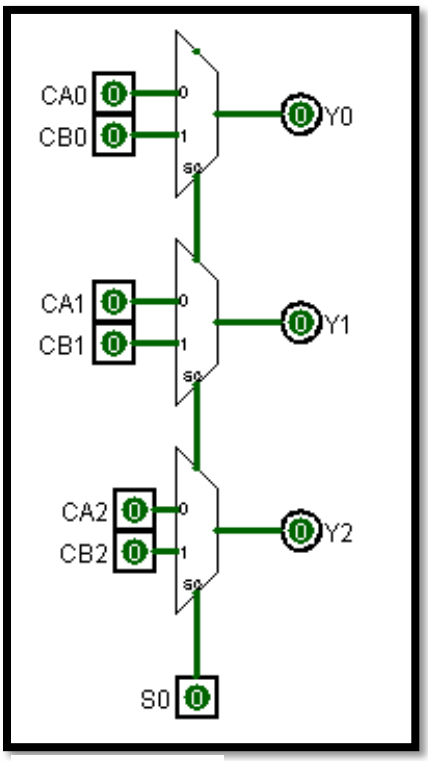
CUPL 7, fase 2

# Logisim:

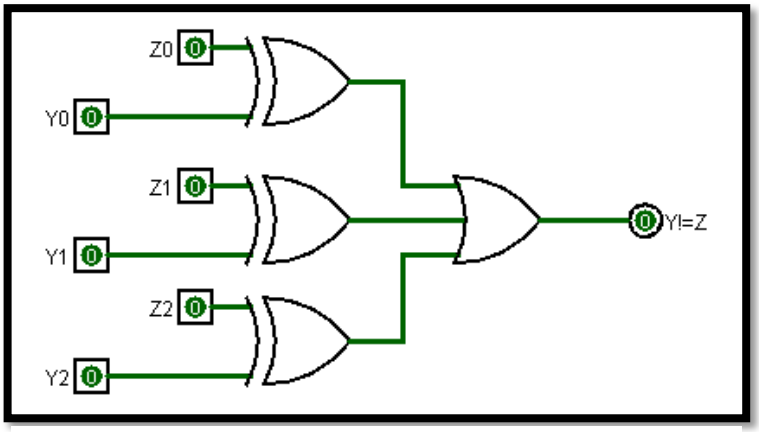
Foi usado 2 MUX's 4X1 e 1 2X1 para criar o 8X1.



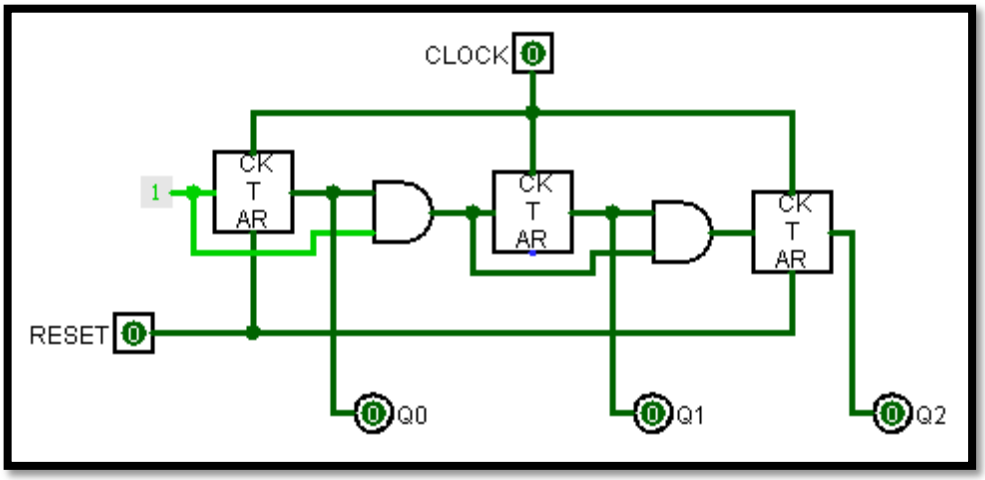

*Logisim 1,*
*CTRL*



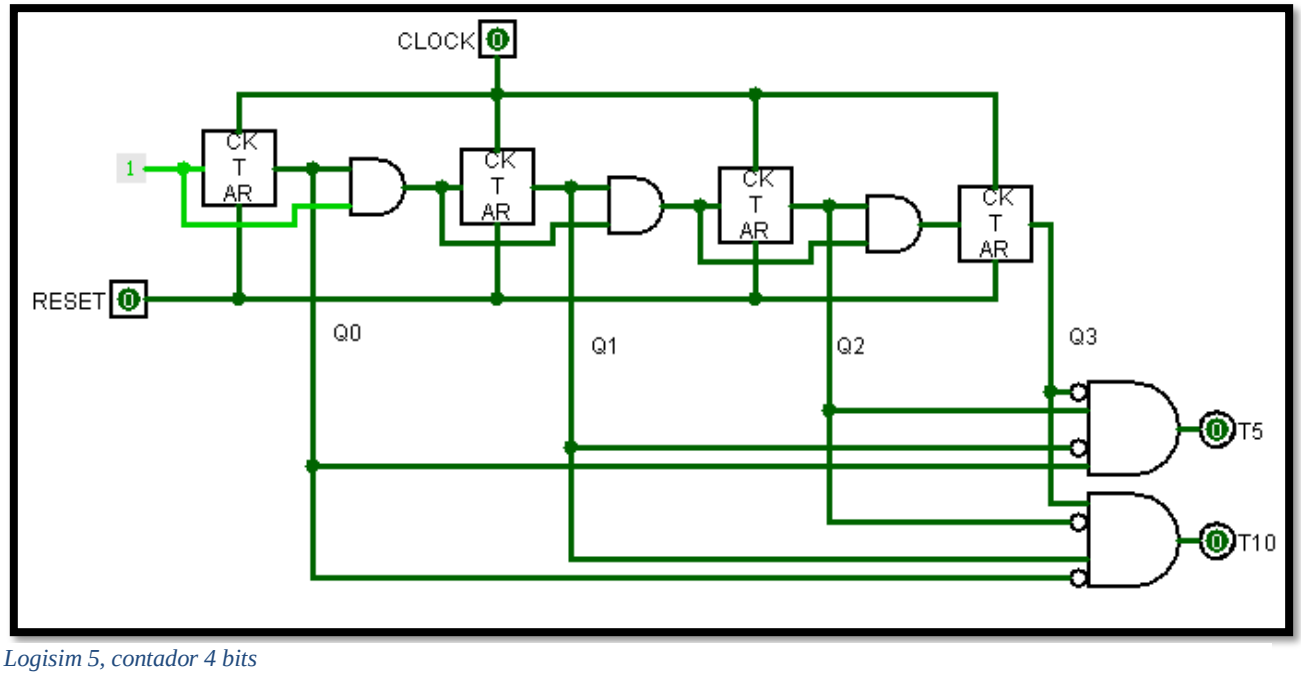

*Logisim 2, MUX 2X3*



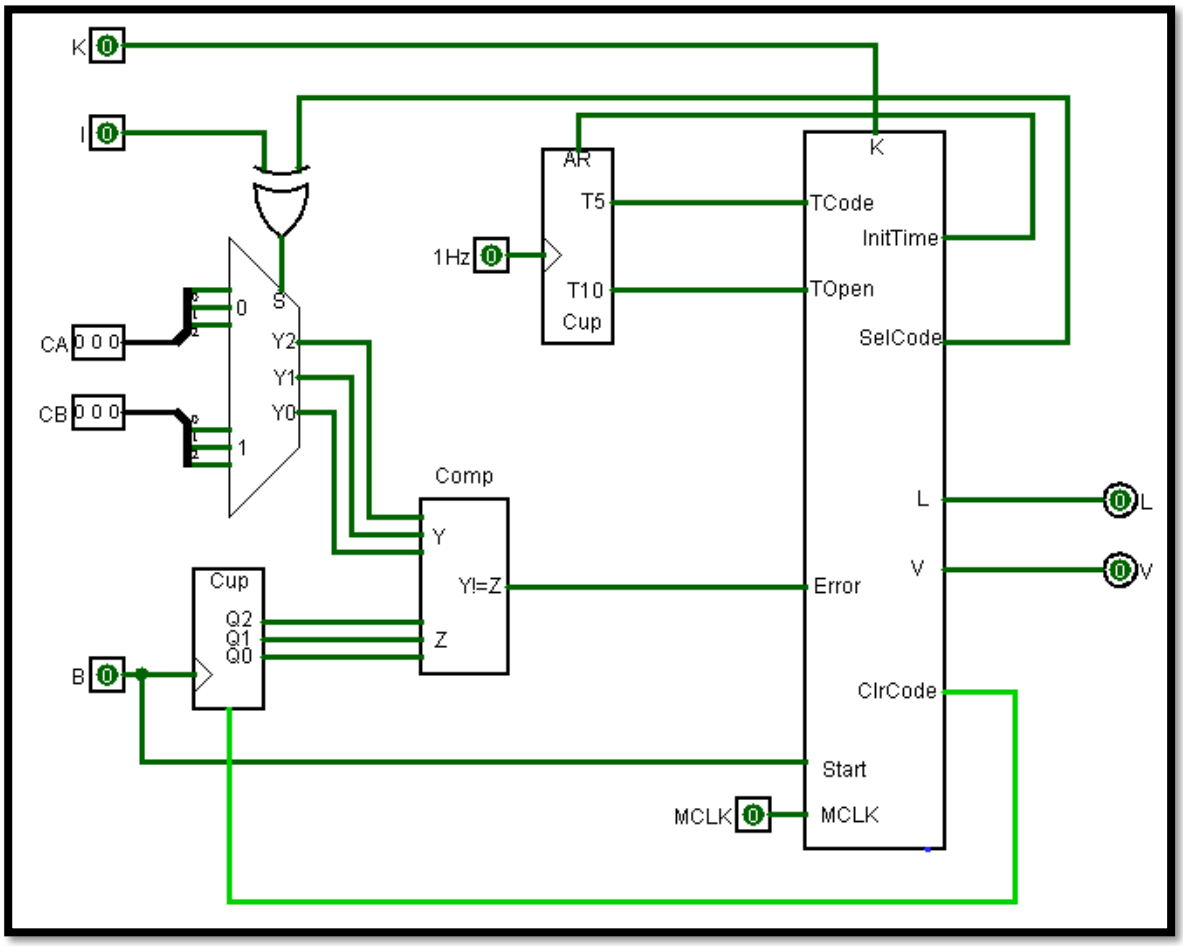

*Logisim 3, Comparador*



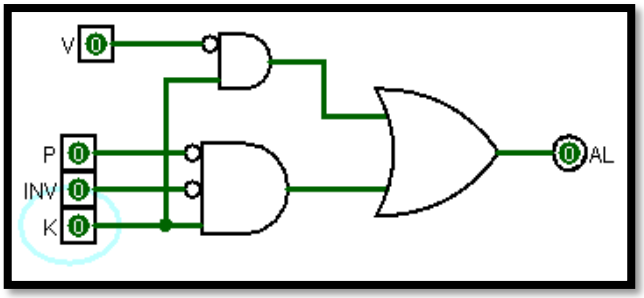

*Logisim 4, contador 3 bits*



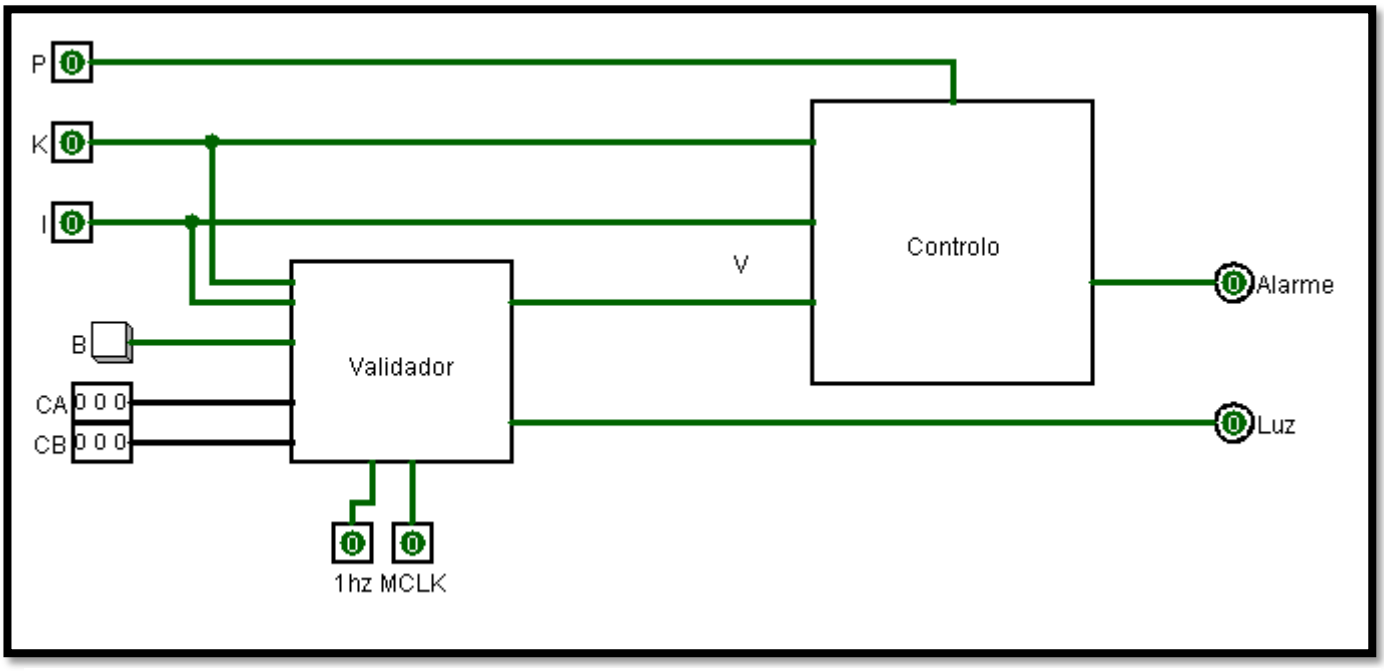

*Logisim 5, contador 4 bits*




*Logisim 6, validador*




*Logisim 7, controlo*




*Logisim 8, circuito todo*