UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

CENTRO DE TECNOLOGIA - CT

CIRCUITOS DIGITAIS

MÁQUINA DE TROCOS

ELE2715 - Grupo 01 - Implementação - Problema 05

Alysson Ferreira da Silva

João Matheus Bernardo Resende

Isaac de Lyra Junior

Vinicius Souza Fonseca

Natal, 4 de Abril de 2021

Alysson Ferreira da Silva
João Matheus Bernardo Resende

Isaac de Lyra Junior

Vinicius Souza Fonseca

MÁQUINA DE TROCOS

Projeto da disciplina de Circuitos Digitais do Departamento de Engenharia Elétrica da Universidade do Rio Grande do Norte para relatório das atividades.

Docente: Samaherni Morais Dias

Natal, 4 de Abril de 2021

RESUMO

O seguinte relatório tem como objetivo desenvolver uma solução para o projeto de um circuito digital que simula uma máquina de troco que libera, ou não, em moedas, um valor determinado colocado em sua entrada, de acordo com a disponibilidade no cofre. No decorrer do texto serão mencionadas todas as condições de funcionamento de acordo com o que é solicitado, além de uma proposta de solução. A ideia é projetar uma resolução capaz de ser reproduzida, futuramente, em *softwares* de simulação de circuitos digitais, além de linguagens de descrição de *hardware*, como o VHDL. Serão abordados, assim, para o desenvolvimento, conceitos de máquinas de estados finitos, circuitos sequenciais e combinacionais, e tendo como principal foco a realização do projeto em RTL.

Palavras-chave: Máquinas de Estados Finitos; RTL; Circuitos Sequenciais; Blocos Operacionais; Blocos de Controle.

SUMARIO

1 INTRODUÇÃO 5
2 DESENVOLVIMENTO 5
PROJETO RTL 7
Máquina de estados de alto nível 7
BOTÃO SÍNCRONO 13
3 RESULTADOS 14
3.1 BLOCO OPERACIONAL 14
3.1.1 Processa Troco 14
3.1.2 Acumulador 29
3.1.3 Cofre 31
3.1.4 LED 31
3.2 BLOCO DE CONTROLE 32
3.2.1 Máquina de Estado 32
3.2.2 Botão Síncrono 35
3.3 LAYOUT 36
3.4 TÓPICO DE MUDANÇAS 37
4 CONCLUSÃO 37
5 REFERÊNCIAS 40
ANEXOS 41
ANEXO B - Tabela de condições 45
ANEXO D - CÓDIGO EM VHDL 47
ANEXO E - IMPLEMENTAÇÃO DOS CIRCUITOS 63

1 INTRODUÇÃO

Em sistemas digitais, o aumento da complexidade dos circuitos gera a necessidade de arranjos lógicos que possam abranger um circuito com múltiplas funções. Um desses arranjos são os blocos de controle, estes são úteis em implementações de sistemas que necessitam de entradas e saídas para controlar um determinado comportamento, como por exemplo, a mudança de estados (VAHID, 2008).

Nesse sentido, o bloco de controle atua em junção com outro bloco construtivo que possua as entradas e saídas de dados do sistema, em particular, esse bloco deve conter registradores para armazenar e unidades funcionais para operar esses dados. Esse arranjo é conhecido como componente do nível de transferência entre registradores, do inglês *Register-Transfer-Level* (RTL). Um circuito composto por tais componentes é nomeado bloco operacional (VAHID, 2008).

A combinação de um bloco operacional com um bloco de controle gera um processador. Há diversos métodos de se projetar um processador, o mais comum é o projeto em nível de transferência entre registradores, ou projeto RTL. Nele são especificados os registradores do circuito, as possíveis transferências e operações que serão feitas com os dados de entrada; saída e dos registradores, além de definir o controle que coordena quando e como transferir e operar dados (VAHID, 2008).

Nesse contexto, para descrever o comportamento de um sistema no método RTL, há as máquinas de estados de alto e baixo nível. As máquinas de alto nível buscam descrever o comportamento do sistema de forma literal, assim, diferente das máquinas de baixo nível, as condições e ações para transição não se limitam ao uso de variáveis booleanas e sim de uma descrição do que aquela variável representa (VAHID, 2008).

Diante do que foi exposto, o presente relatório irá discutir colocar em prática todos esses conceitos na realização de um projeto de uma máquina de trocos utilizando RTL.

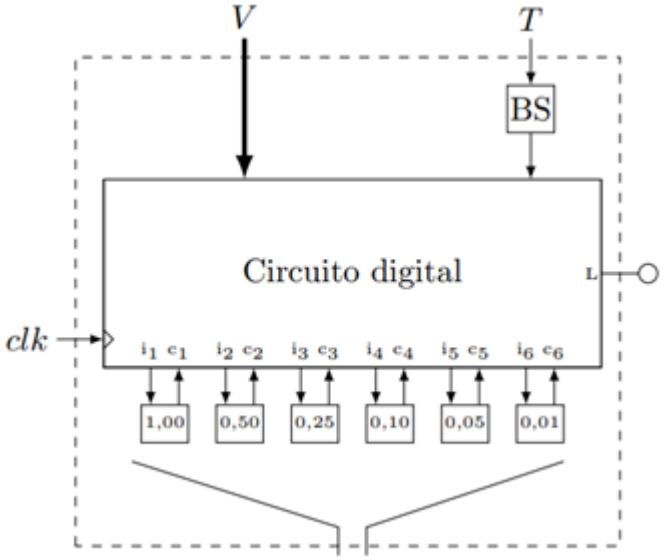
2 DESENVOLVIMENTO

A problemática consiste em projetar uma Máquina de Troco (Figura 1), a qual tem a capacidade de dar um valor, em moedas, determinado pelo usuário. A liberação das moedas ocorre através de um sistema cofre que libera uma moeda sempre que em sua entrada ix (onde x = 1, 2, …, 6) existir um nível lógico alto e ocorrer um pulso de clock. A máquina possui como entrada o valor do troco (V), em binário; e uma entrada T, referente ao pulso gerado a partir da saída do circuito de um botão sincronizado (BS). Terá também uma saída

L, um LED no qual, quando está piscando, indica que a máquina está processando a informação para liberar o troco e qualquer outra solicitação de troco será ignorada.

A máquina também é dotada da capacidade de verificar se algum dos cofres de moedas está vazio (cx = 0, onde x = 1, 2, …, 6) e faz o recálculo para fazer a liberação de moedas apenas por onde terá valores. Durante o processo de verificação de troco, como já foi dito, um LED ficará piscando. A máquina também tem a capacidade de informar quando não há troco para o valor fornecido, mantendo a saída LED em nível lógico alto até que um novo valor de troco seja fornecido.

Figura 1 - Diagrama de blocos da máquina de troco.



Fonte: Dados do problema.

A liberação das moedas do cofre (ix, x=1,2 ...6) e a indicação de cofre vazio (cx, x=1,2 ...6) ocorrem da forma apresentada no Quadro 1. Além disso, a entrada do circuito será realizada ao se definir um valor binário V, entre 0 e 10 reais, e fazer T = 1. A máquina só processa um troco por vez.

Quadro 1 - Liberação das moedas e indicação de cofre vazio.

Identificação	Valor	Identificação	Valor
i1 = 1	R$ 1,00	c1 = 1	Há moedas de R$ 1,00
i2 = 1	R$ 0,50	c2 = 1	Há moedas de R$ 0,50
i3 = 1	R$ 0,25	c3 = 1	Há moedas de R$ 0,25
i4 = 1	R$ 0,10	c4 = 1	Há moedas de R$ 0,10

i5 = 1	R$ 0,05	c5 = 1	Há moedas de R$ 0,05
i6 = 1	R$ 0,01	i6 = 1	Há moedas de R$ 0,01

Fonte: Elaboração própria.

PROJETO RTL

Para projetar a máquina de trocos, foi seguida a metodologia descrita por VAHID (2008), a qual divide o método de projeto RLT em 4 passos: 1º Descrever o comportamento do circuito (máquina de estados de alto nível); 2º criar bloco operacional; 3º conectar bloco operacional a um bloco de controle e 4º obter a máquina de estados finitos (FMS).

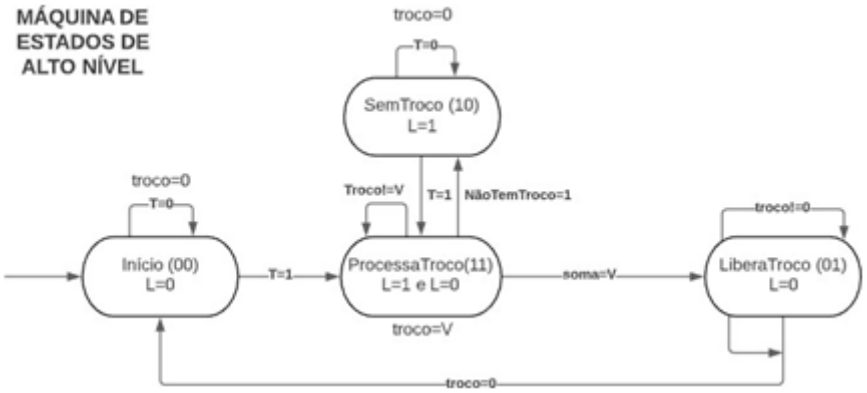
Máquina de estados de alto nível

Primeiramente, a máquina de trocos foi pensada para que na sua entrada seja colocado o valor do troco em centavos, logo, ela irá operar numa faixa de 0 a 1000 centavos. Em binário esse número pode ser representado em até 12 bits. Para saída foi pensada em uma ordem de precedência, assim foi dada prioridade às moedas de valores mais altos.

Para descrever o comportamento desse sistema foi montada a máquina de estados de alto nível mostrada na Figura 2. Esta conta com 4 estados: Início; ProcessaTroco; LiberaTroco e SemTroco. O estado Início significa que nenhum valor foi inserido, ou seja, o botão não foi pressionado, além disso, o LED deve estar apagado. Para sair desse estado o botão deve ser pressionado (T=1) e o próximo estado será ProcessaTroco.

No estado ProcessaTroco o LED fica piscando e uma operação para verificar se a máquina possui ou não troco deve ser realizada. Caso não haja moedas suficientes para o troco (NãoTemTroco=1), o próximo estado será SemTroco (LED aceso); caso haja moedas (soma=V), o próximo estado será LiberaTroco. Em LiberaTroco deverão ser “setadas” as saídas de acordo com a ordem de precedência e o LED tem que ficar apagado.

Figura 2 - Máquina de estados de alto nível da máquina de trocos.



Fonte: Elaboração própria.

Bloco operacional

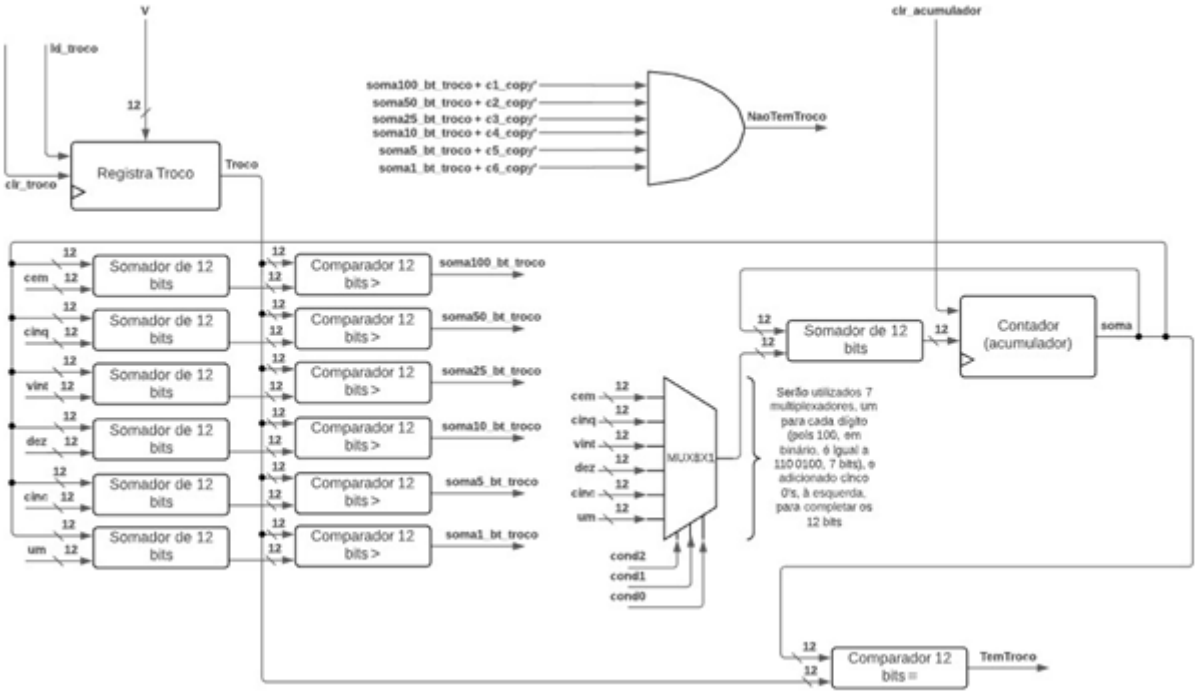
Para que a máquina funcione conforme instruído, foi criado um bloco operacional que contém todas as operações que ocorrem nos estados ProcessaTroco e LiberaTroco.

Em ProcessaTroco, recomenda-se que o valor V seja guardado em um registrador, visando evitar que o usuário altere V durante o processamento. Assim, a verificação será feita com o valor guardado. Para verificar se há ou não troco na máquina, recomenda-se o uso de somadores, subtratores e comparadores mediante condições pré-definidas.

As condições foram criadas para identificar se, com as moedas disponíveis, é possível gerar o troco solicitado. Diante disso, primeiramente, foi definida uma quantidade de moedas para cada cofre, no caso 7 (111, em binário). Então, para verificar se ainda há moedas, recomenda-se o uso de uma porta *OR*, assim, enquanto qualquer um dos 3 bits estiverem em nível lógico alto, há moedas no cofre (ci_copy=1, onde i corresponde ao valor descrito no Quadro 1); caso contrário, não há moedas naquele cofre de moedas.

Além disso, as condições também são resultado da análise da soma das moedas que o cofre possui, visto que, é preciso verificar se a máquina possui moedas suficientes para o troco ser dado. Então, usa-se um contador, em que sua saída irá alimentar 6 somadores (para cada valor de moeda), conforme mostrado na Figura 3, os quais estarão conectados à comparadores. Diante disso, caso a soma realizada seja menor que o valor armazenado (e há moedas) ele continuará somando na mesma condição, quando o valor exceder o troco (ou a moeda acabar), a próxima condição será verificada.

Figura 3 - Diagrama de blocos das operações do estado ProcessaTroco.



Fonte: Elaboração própria.

Enquanto essa verificação é feita, o número de moedas armazenadas irá diminuindo à medida que o acumulador for utilizando os valores necessários para o troco. Assim, se, por exemplo, o troco for de R$2,50 (partindo do pressuposto que todos os cofres estão cheios) serão utilizadas duas moedas de R$1,00 e uma moeda de R$0,50, assim o cofre de R$1,00 ficará com 101 bits e o de R$0,50 com 110 bits. O número de bits subtraído do cofre será guardado em liberai (para i=1, 2, …, 6).

O resultado do contador será comparado com o valor inserido. Caso os valores sejam iguais, TemTroco=1 e o próximo estado será LiberaTroco. Mas, para verificar se não há troco, uma condição diferente precisa ser verificada, pois TemTroco=0 já na primeira verificação, por exemplo, fazendo uma falsa afirmação que não tem troco, enquanto ele ainda nem comparou. Sendo assim,, a condição para não ter troco é caso todas as condições tenham sido verificadas e não haja mais moedas de 1 centavo. O Anexo B mostra todas as condições.

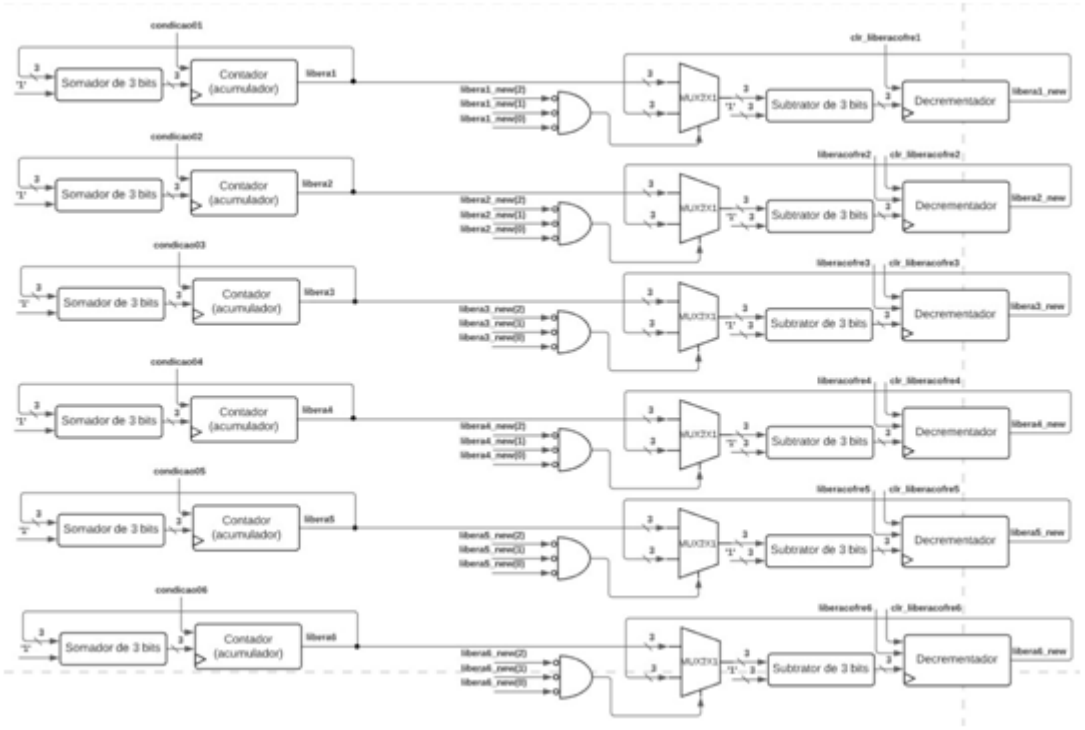
Já no estado LiberaTroco, as saídas serão setadas utilizando a variável liberai, mencionada anteriormente, que informa o número de moedas de cada cofre que deve ser liberado. Isso será feito de forma gradativa, do maior para o menor, diminuindo um bit de

cada liberai até todos chegarem em 0. Para identificar se todas as moedas necessárias para dar o troco já estão zeradas, há a variável cofrei_liberado que verifica se liberai é igual a 0.

Assim, a liberação ocorre conforme descrito: primeiro libera as moedas de R$1, em seguida, caso já tenha liberado as moedas do cofre de 1 real (cofre1_liberado), então permite liberar as moedas de 50 centavos (liberacofre2); após liberadas (cofre2_liberado), permite-se liberar as moedas de 25 centavos (liberacofre3); e assim por diante até que todas as moedas necessárias forem liberadas. A variável que define que todo o troco já foi dado é TrocoLiberado, a qual terá nível lógico alto quando todas as moedas dos cofres forem liberadas.

A sinalização da saída será as variáveis ix em nível lógico alto, enquanto estiverem sendo liberadas as moedas do seu respectivo cofre. Se for liberada mais de uma moeda do mesmo valor, o nível lógico deverá ficar alto por quantos pulsos de *clock* forem necessários para todo o valor ser dado. No exemplo do troco de R$2,50, a saída i1 deverá ficar em 1 por dois pulsos de clock e, em seguida, a saída i2 por apenas um.

Figura 4 - Diagrama de blocos das operações do estado LiberaTroco.



Fonte: Elaboração própria

As variáveis condicao0i (com i=1, 2, …, 6) são de acordo com as condições vistas no Anexo B, por exemplo: *condicao01=cond(2)=0 && cond(1)=0 && cond(0)=1; condicao02: cond(2)=0 && cond(1)=1 && cond(0)=0;* e assim por diante. E as variáveis clr_liberacofrei

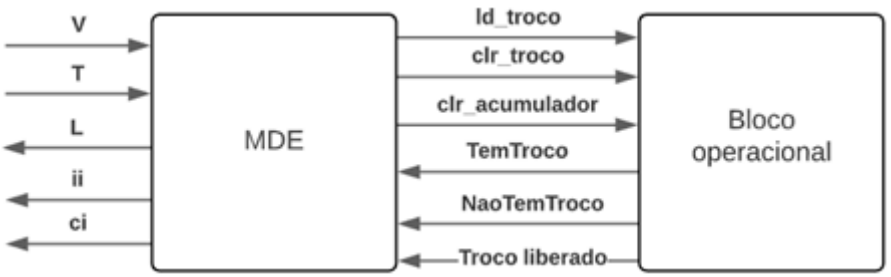
(com i=1, 2, …, 6) são utilizadas para o decrementador fixar em ‘0’ quando não for necessário decrementar, sendo assim, utilizando a variável cofrei liberado.

Bloco operacional e bloco de controle

O bloco operacional descrito no tópico anterior será conectado a um bloco de controle, o qual possui como entradas V e T; e como saídas L, libera moeda ou não (ix, x=1, 2, …, 6) e os cofres (cx, x=1, 2, …, 6). Já as entradas do bloco operacional são ld_troco, clr_troco e clr_acumulador; e as saídas são TemTroco; NãoTemTroco e TrocoLiberado.

No que tange as entradas do bloco operacional, ld_troco será o responsável por permitir o registro do valor inserido, clr_troco irá limpar esse registrador quando não houver mais necessidade do uso do valor inserido e por último clr_acumulador vai limpar o contador que está acumulando os valores em ProcessaTroco. A Figura 5 ilustra como estão conectados.

Figura 5 - Bloco de controle e bloco operacional da máquina de trocos.

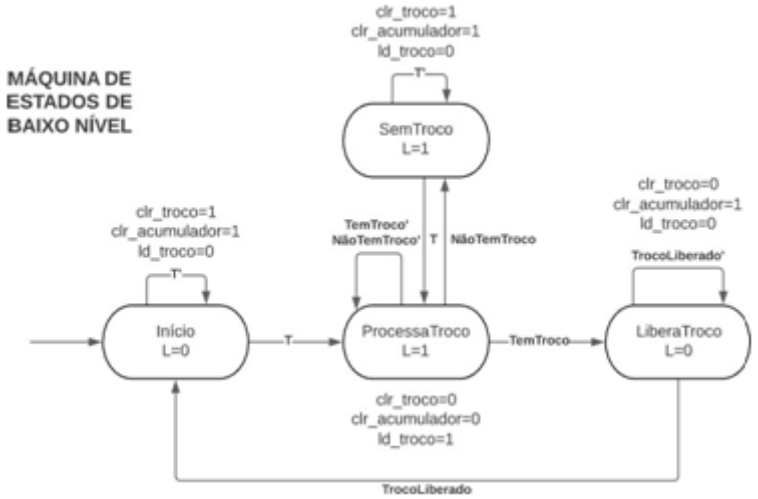


Fonte: Elaboração própria.

Máquina de estados finitos

Após definidos os blocos de controle e operacional, a máquina de estados finita foi montada. Esta é descrita pela máquina de estados de baixo nível mostrada na Figura 6.

Figura 6 - Máquina de estados de baixo nível da máquina de trocos.



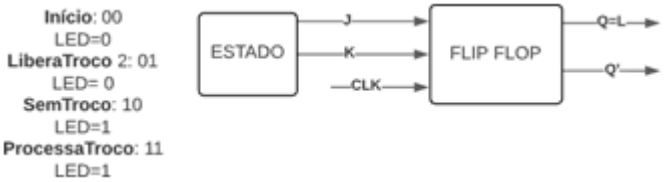
Fonte: Elaboração própria.

A partir dela foi montada a tabela estados mostrada no Anexo C, contendo as variáveis de estado, estados atual e futuro e as saídas.

Note que a saída LED é somente 0 ou 1 na tabela de estados, no entanto, conforme mencionado na problemática, o LED possui 3 comportamentos diferentes de acordo com o estado. Nos estados Início e LiberaTroco o LED deve está apagado, em SemTroco o LED deve está aceso, já em ProcessaSenha o LED deve está piscando. Para tornar isso possível, foi criada uma lógica pautada nos estados e na saída LED de cada um.

Assim, o estado atual *AND* L são colocados na entrada no Flip-flop JK. Devido a configuração do flip-flop o LED será capaz de piscar quando estiver no estado ProcessaTroco. Isso acontece pois, ao inserir 1 nas entradas J e K, o flip-flop fica instável mudando sua saída a cada pulso. Nos demais estados, o LED irá se comportar conforme exigido. A Figura 7 ilustra o que foi descrito.

Figura 7 - Lógica do acendimento do LED.

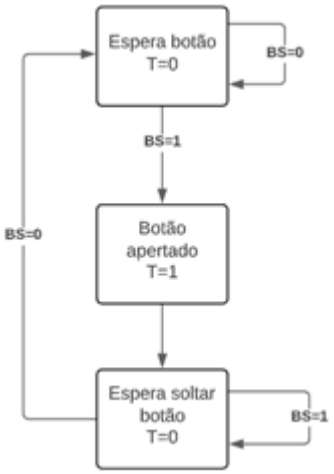


Fonte: Elaboração própria.

BOTÃO SÍNCRONO

Para a criação do botão síncrono foi feita a máquina de estados mostrada na Figura 8. Esta máquina irá garantir que, quando pressionado o botão, T terá nível lógico alto durante apenas 1 *clock*, solucionando o problema de que, por exemplo, o usuário aperte o botão e fique pressionado por um certo tempo.

Figura 8 - Máquina de estados do botão síncrono.



Fonte: Elaboração própria.

3 RESULTADOS

Softwares utilizados para a realização da implementação do projeto:

1. Loginsim-ITA, sua versão: 2.16.1.2
2. ModelSim - Intel FPGA starter, edição 2021

   A relação dos componentes usados na implementação do projeto máquina de troco, pode ser observado na tabela 1.

Tabela 1 - Componentes comerciais

PORTAS LÓGICAS	CI COMERCIAL
AND	7408
OR	7432 / 7411
NOT	7404
FLIP-FLOP D	7479
MULTIPLEXADORES (8X1)/(2x1)	74451 / 74257
CONTADOR	74592
SUBTRATOR	74385
SOMADOR	74283
COMPARADOR	74682

Fonte: Autores

3.1 BLOCO OPERACIONAL

Bloco responsável onde ocorre todo o processo do funcionamento da máquina de troco.

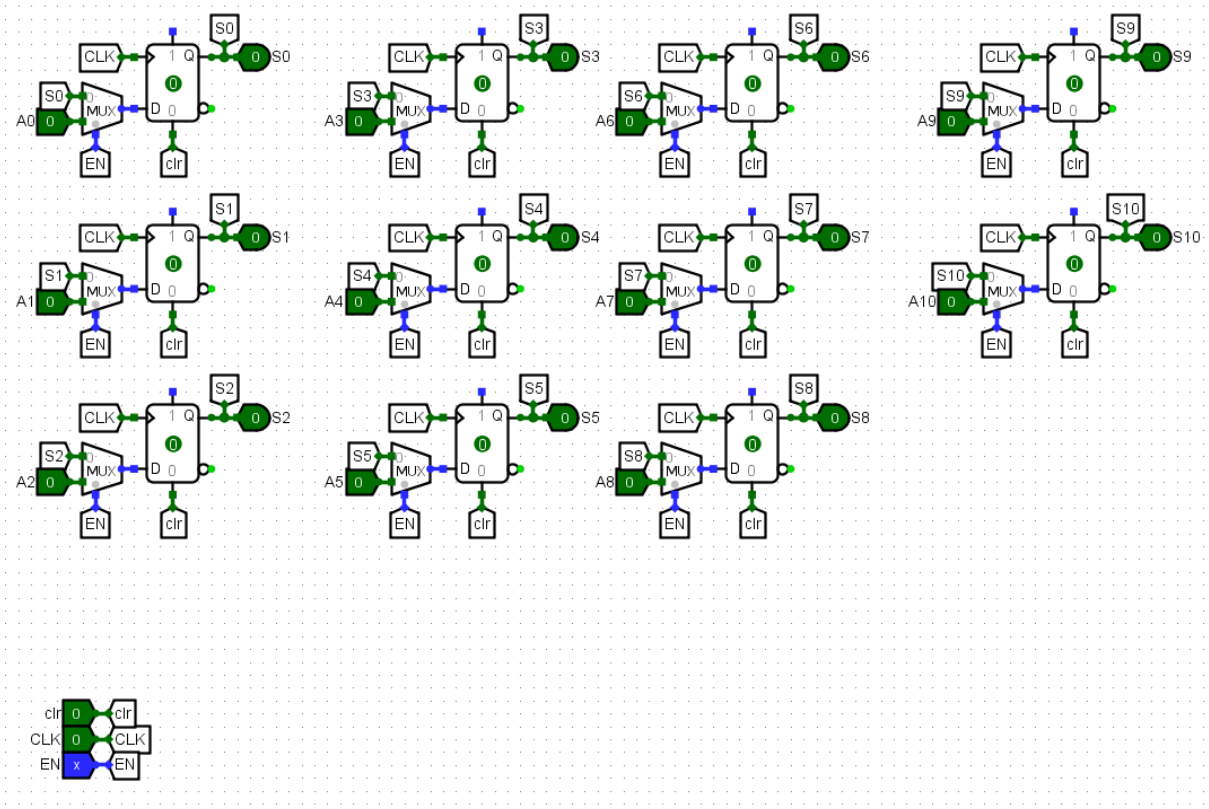
3.1.1 Processa Troco

A implementação do bloco “Processa Troco" partiu da criação de um registrador de 11 bits, através de 11 flip-flops síncronos do tipo D. Dito isso, a entrada D destes flip flops será composta por um MUX 2x1, onde este mux irá selecionar para a entrada D a saída

S(0:10) do registrador ou o número em bits referente ao dinheiro solicitado pelo usuário A(0:10). Nesse sentido, a chave seletora deste MUX (EN) será a variável ld_troco.

Por causa disso, este registrador irá registrar a sua própria saída sempre que a variável ld_troco possuir valor lógico baixo. Entretanto, ao assumir valor lógico alto, será registrado o valor solicitado pelo usuário dando início ao processo da máquina que efetuará as operações necessárias para retornar o troco ao usuário. O registrador de 11 bits supracitado pode ser observado na figura 9.

Figura 9 - Registrador de 11 bits



Fonte: Autores

Com o registrador montado no Logisim foi possível realizar a implementação do código no programa ModelSim, no intuito de efetuar a simulação do componente através do código em VHDL. Nesse sentido, para realizar os testes no registrador de 11 bits, implementamos uma entidade chamada REG11. Nesta entidade, atribuímos um vetor “I” de 11 bits, duas variáveis de 1 bit referente ao CLK (clock) e CLR (clear), assim como, uma saída “O” de 11 bits.

Para estruturar lógica deste circuito foi implementada na arquitetura um componente flip-flop do tipo D, onde nomeamos como FFD. Nessa continuidade, utilizamos o recurso PORT MAP de 11 FFD, com o objetivo de registrar as entradas para cada flip-flop presente

no registrador. A implementação do registrador de 11 bits em VHDL está presente na figura 10.

Figura 10 - Código VHDL do registrador

```
--======================--
--  REGISTRADOR 11 BITS --
--======================--

ENTITY REG11 IS
    PORT( I: IN BIT_VECTOR(10 DOWNTO 0);
        CLK, CLR: IN BIT;
        O: OUT BIT_VECTOR(10 DOWNTO 0));
END REG11;

ARCHITECTURE CKT OF REG11 IS

COMPONENT ffd IS
    port ( clk ,D ,P , C : IN BIT ;
    q : OUT BIT );
END COMPONENT;

BEGIN

    CLEAR <= NOT CLR;

    FFD1: ffd PORT MAP (CLK, I(0), '1', CLEAR, O(0));
    FFD2: ffd PORT MAP (CLK, I(1), '1', CLEAR, O(1));
    FFD3: ffd PORT MAP (CLK, I(2), '1', CLEAR, O(2));
    FFD4: ffd PORT MAP (CLK, I(3), '1', CLEAR, O(3));
    FFD5: ffd PORT MAP (CLK, I(4), '1', CLEAR, O(4));
    FFD6: ffd PORT MAP (CLK, I(5), '1', CLEAR, O(5));
    FFD7: ffd PORT MAP (CLK, I(6), '1', CLEAR, O(6));
    FFD8: ffd PORT MAP (CLK, I(7), '1', CLEAR, O(7));
    FFD9: ffd PORT MAP (CLK, I(8), '1', CLEAR, O(8));
    FFD10: ffd PORT MAP (CLK, I(9), '1', CLEAR, O(9));
    FFD11: ffd PORT MAP (CLK, I(10), '1', CLEAR, O(10));

END CKT;
```

Fonte: Autores

Em sequência, implementou-se a entidade do MUX 2x1, sendo registrada como MUX21. Nesta entidade, foram implementadas 3 variáveis de entrada de 1 bit que se referem a A, B e S. Tão quanto sua saída “O” com valor de 1 bit. Na sua estrutura é denominado uma equação booleana utilizando as entradas citadas, determinando o seu comportamento de saída específico. No qual pode ser analisada na figura 11.

Figura 11 - Código VHDL do Multiplexador 2x1

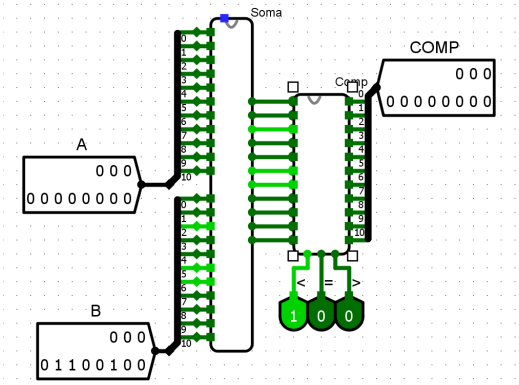
```
--====================--
-- MULTIPLEXADOR 2x1 --
--====================--

entity MUX21 is
    port(A, B, S: in bit;
        O: out bit);
 end MUX21;

 architecture CKT of MUX21 is

 begin

     O <= (B and S) or (A and (not S));

 end CKT;
```

Fonte: Autores.

Ao realizar a implementação do registro, é possível dar início a montagem do circuito. Para isso, é necessário que a saída do registrador seja encaminhada para 6 circuitos implementados com a função de realizar uma soma com uma moeda X (cujo os valores podem assumir o valor R$ 1, R$ 0,50, R$ 0,25, R$ 0,10, R$ 0,05 e R$ 0,01), e verificar se o valor atual na máquina com a soma da moeda será maior que o valor inserido pelo usuário.

Este circuito somador-comparador apresenta entradas denominadas A e B, onde A faz menção ao valor atual guardado na máquina, e B é um valor constante que representa uma moeda em centavos. Essas entradas são encaminhadas para um somador de 11 bits, que realizará a soma entre os valores e encaminhará o resultado para um comparador de 11 bits. O comparador irá verificar se o resultado da soma entre o valor atual do sistema e a moeda é maior que o valor inserido pelo usuário. Caso o valor seja maior, será utilizado o resultado da soma de outro somador-comparador, que possuirá em B o valor de outra moeda guardada, para realimentar o sistema. Esta realimentação deverá ocorrer até que o valor atual da máquina seja igual ao valor inserido pelo usuário. A implementação do somador-comparador pode ser vista na figura 12.

Figura 12 - Somador-Comparador



Fonte: Autores.

Com a implementação do componente feita no Logisim, é possível realizar a implementação em código no Modelsim, sendo baseado na linguagem VHDL. Dessa forma, para demonstrar o funcionamento do componente, construímos uma entidade denominada COMP_BIT sendo que na entidade implementada, possuem entrada denominadas por A, B, maior, igual, menor sendo cada um valor unitário de bit, o mesmo pode ser referido a saída de nome AgtB, AeqB e AltB.

A arquitetura do componente foi denominada de CKT. Para compor a lógica desta arquitetura foram adicionadas equações booleanas que determinam se A é maior, menor ou igual a B. Na sequência foram atribuídos os resultados as respectivas saídas AltB, AeqB e AgtB. O comparador em VHDL pode ser observado na figura 13.

Figura 13 - Código VHDL de um comparador 1 bit

```
--==================================--
-- COMPARADOR DE MAGNITUDE (1 bits) --
--==================================--

entity COMP_BIT is
    port(A, B, maior,igual, menor: in bit;
                AgtB, AeqB, AltB: out bit);
end COMP_BIT;

ARCHITECTURE CKT OF COMP_BIT IS

BEGIN

    AgtB <= (((not B) and A) and igual) or maior;
    AeqB <= ((A xnor B) and igual);
    AltB <= (((not A) and B) and igual) or menor;

END CKT;
```

Fonte: Autores.

Ao ser implementado o código de um comparador de 1 bit, é refeito a sua estrutura para um valor de 11 bits, onde é denominado *COMP11*. Nele são utilizados 11 *port maps* do comparador de 1 bit, além disso, é atribuído aos vetores de entrada A e B um valor de 11 bits. Nessa entidade, a saída permanece igual a do comparador de 1 bit mencionado anteriormente.

Ademais, ao ser implementado esse componente, é feito um sinal de vetor 11 bits sendo nomeado por GT, EQ e LT. Nessa continuidade, utilizamos a função PORT MAP sendo chamada por COMP_BIT, com objetivo de fazer a comparação de todos os bits deste vetor.

Figura 14 - Código VHDL de um comparador 11 bit

```
--================================--
-- COMPARADOR DE MAGNITUDE 11 BITS --
--================================--

entity COMP11 is
    port(A,B: in bit_vector(10 downto 0);
             AgtB, AeqB, AltB: out bit);
end COMP11;

ARCHITECTURE CKT OF COMP11 IS

component COMP_BIT is
     port(A, B, maior,igual, menor: in bit;
              AgtB, AeqB, AltB: out bit);
end component;

signal GT, EQ, LT: bit_vector(10 downto 0);

begin

      COMP1: COMP_BIT port map(A(10), B(10),'0', '1','0', GT(10), EQ(10), LT(10));
      COMP2: COMP_BIT port map(A(9), B(9), GT(10), EQ(10), LT(10), GT(9), EQ(9), LT(9));
      COMP3: COMP_BIT port map(A(8), B(8), GT(9), EQ(9), LT(9), GT(8), EQ(8), LT(8));
      COMP4: COMP_BIT port map(A(7), B(7), GT(8), EQ(8), LT(8), GT(7), EQ(7), LT(7));
      COMP5: COMP_BIT port map(A(6), B(6), GT(7), EQ(7), LT(7), GT(6), EQ(6), LT(6));
      COMP6: COMP_BIT port map(A(5), B(5), GT(6), EQ(6), LT(6), GT(5), EQ(5), LT(5));
      COMP7: COMP_BIT port map(A(4), B(4), GT(5), EQ(5), LT(5), GT(4), EQ(4), LT(4));
      COMP8: COMP_BIT port map(A(3), B(3), GT(4), EQ(4), LT(4), GT(3), EQ(3), LT(3));
      COMP9: COMP_BIT port map(A(2), B(2), GT(3), EQ(3), LT(3), GT(2), EQ(2), LT(2));
      COMP10: COMP_BIT port map(A(1), B(1), GT(2), EQ(2), LT(2), GT(1), EQ(1), LT(1));
      COMP11: COMP_BIT port map(A(0), B(0), GT(1), EQ(1), LT(1), GT(0), EQ(0), LT(0));

      AgtB <= GT(0);
      AeqB <= EQ(0);
      AltB <= LT(0);

end CKT;
```

Fonte: Autores.

Na implementação em VHDL do somador completo é apresentada a função que realiza a soma do valor atual da máquina com o valor constante da moeda específica em centavos. Para isso, foi designada a construção da entidade denominada COMP_ADD que possui 4 entradas unitárias de bit denominadas A, B, CI, e sua saída unitária em bit S e CO.

Na sua arquitetura desenvolvida CKT, é demonstrada uma equação booleana para cada saída, no qual foi especificada antes, sendo cada uma possuindo sua resposta única. Como pode ser observado na figura 15.

Figura 15 - Código VHDL de Somador Completo

```
--==================--
-- SOMADOR COMPLETO --
--==================--

entity COMP_ADD is
    port(A, B, CI: in bit;
             S, CO: out bit);
end COMP_ADD;

architecture CKT of COMP_ADD is

begin

    S <= A xor B xor CI;
    CO <= (B and CI) or (A and CI) or (A and B);

end CKT;
```

Fonte: Autores.

Ao implementar o somador completo, é preciso lembrar que em VHDL se do meio somador. Para isso, foi designada a construção da entidade denominada HALF_ADD que possui 4 entradas unitárias de bit denominadas A, B, CI, e sua saída unitária em bit S e CO.

Na sua arquitetura desenvolvida CKT, é demonstrada uma equação booleana para cada saída, no qual foi especificada antes, sendo cada uma possuindo sua resposta única. Como pode ser observado na figura 16.

Figura 16 - Código VHDL de Somador Completo

```
--===============--
-- MEIO SOMADOR --
--===============--

entity HALF_ADD is
    port(A, B: in bit;
             S, CO: out bit);
end HALF_ADD;

architecture CKT of HALF_ADD is

begin

    S <= A xor B;
    CO <= A and B;

end CKT;
```

Fonte: Autores.

Tendo em mente a função somador completo demonstrada anteriormente, utilizou-se dentro do componente um somador de 11 bits. Sendo assim, o somador completo e meio somador servirá de apoio na montagem da arquitetura do componente.

Sendo dessa forma, denominamos entidade ADD11, o qual apresenta entrada vetor de 11 Bits A e B, também saída em vetor 11 bits de nome “O” e “CO”como valor unitário de bit.

Dentro da arquitetura CKT é realizado uma chamada do componente HALF_ADD de entrada única de bit apresentado por A e B, e sua saída em 1 bit S e CO. Para que é possível realizar a soma completa do componente, é preciso também criar uma nova chamada de componente, no qual, desta vez é o somador completo, denominado COMP_ADD, vale ressaltar, que foi utilizado os mesmos sinais de entrada e saída demonstrada na figura 15. Proximo a ser implementado é um sinal vetor de 10 bits declarado VAI_UM.

Arquitetura quase finalizada, é considerado a implementação usando código do PORT MAP sendo cetada a saída do vetor de 11 bits “O” pode ser observado na figura 17

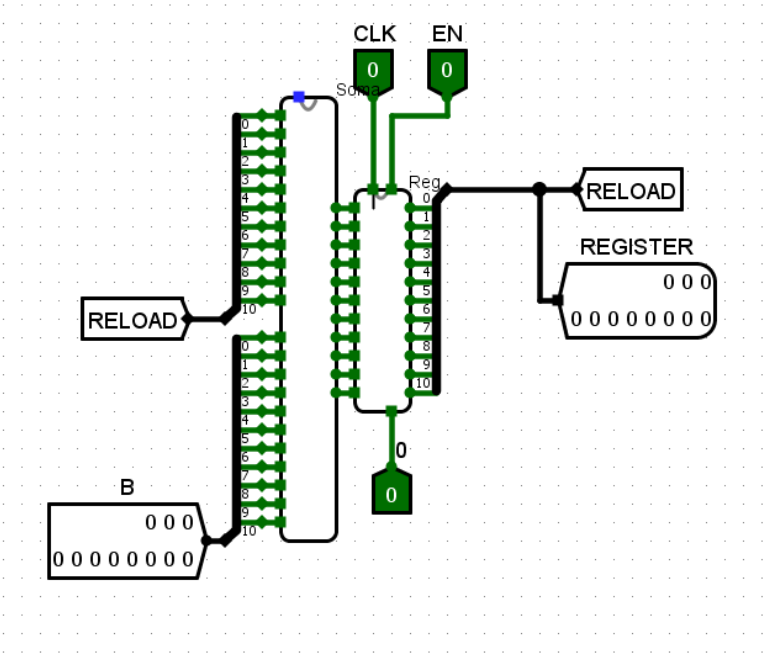
Figura 17 - Código VHDL de Somador Completo

```
--==================--
-- SOMADOR 11 BITS --
--==================--

entity ADD11 is
    port(A, B: in bit_vector(10 downto 0);
            O: out bit_vector(10 downto 0);
            CO: out bit);
end ADD11;

architecture CKT of ADD11 is

component HALF_ADD is
    port(A, B: in bit;
         S, CO: out bit);
end component;

component COMP_ADD
    port(A, B, CI: in bit;
            S, CO: out bit);
end component;

signal VAI_UM: bit_vector(9 downto 0);

begin

    S0: HALF_ADD port map(A(0), B(0), O(0), VAI_UM(0));
    S1: COMP_ADD port map(A(1), B(1), VAI_UM(0), O(1), VAI_UM(1));
    S2: COMP_ADD port map(A(2), B(2), VAI_UM(1), O(2), VAI_UM(2));
    S3: COMP_ADD port map(A(3), B(3), VAI_UM(2), O(3), VAI_UM(3));
    S4: COMP_ADD port map(A(4), B(4), VAI_UM(3), O(4), VAI_UM(4));
    S5: COMP_ADD port map(A(5), B(5), VAI_UM(4), O(5), VAI_UM(5));
    S6: COMP_ADD port map(A(6), B(6), VAI_UM(5), O(6), VAI_UM(6));
    S7: COMP_ADD port map(A(7), B(7), VAI_UM(6), O(7), VAI_UM(7));
    S8: COMP_ADD port map(A(8), B(8), VAI_UM(7), O(8), VAI_UM(8));
    S9: COMP_ADD port map(A(9), B(9), VAI_UM(8), O(9), VAI_UM(9));
    S10: COMP_ADD port map(A(10), B(10), VAI_UM(9), O(10), CO);

end CKT;
```

Fonte: Autores.

Este circuito Somador-Registrador apresenta entradas denominadas RELOAD e B, onde B faz menção ao valor em bit a saída de um MUX 8x1 sendo a seleção desse mux valor de 3 bits, e RELOAD é um valor atual setada na saída de um registrador presente no componente. Essas entradas são encaminhadas para um somador de 11 bits, que realizará a soma entre os valores e encaminhará o resultado para um registrador de 11 bits. O Registrador irá registrar o resultado da soma entre o valor atual do componente e a saída constante selecionada no MUX onde será cetada para a saída nomeada REGISTER. A implementação do somador-comparador pode ser vista na figura 16.

Figura 18 - Esquemático do Somador Registrador



Fonte: Autores

Ao ser finalizado a implementação do somador registrador, é possível obter a simulação em código do VHDL usando o programa ModelSim, no qual, a entidade construida é denominada SOMAREG, apresentando vetor de entrada 11 bits nomeadas B, tão qual, entras de valor bit unitário CLK, CLR, EN. Ao setar as entradas, devemos colocar o valor de saída, no qual, foi definido um vetor de 11 bits nomeado como AC.

Na sua arquitetura construida, apresentou nome de CKT onde teve chama de componentes ADD11 de entrada vetor 11 bits A e B. com saída vetor O de 11 bits, e colocando o CO como saída de 1 bit. Fora o ADD11 foi chamado o componente REG11, no qual possui vetor “I” de entrada 11 bit, no qual, o valor de entrada sendo de 1 bit é nomeado CLK, CLR e EN, sendo apresentado com saída O de vetor 11 bits.

Na arquitetura não pode faltar o sinal de bit, sendo nomeada por CO e também sinal vetor de 11 bits sendo denominada RELOAD e RES. No final da arquitetura é utilizado o comando PORT MAP para a saída ADD11 e REG11, também como saída AC sendo usada o sinal vetor RELOAD de 11 bits. Como pode ser verificado na figura 19.

Figura 19 - Código VHDL de Somador Registrador

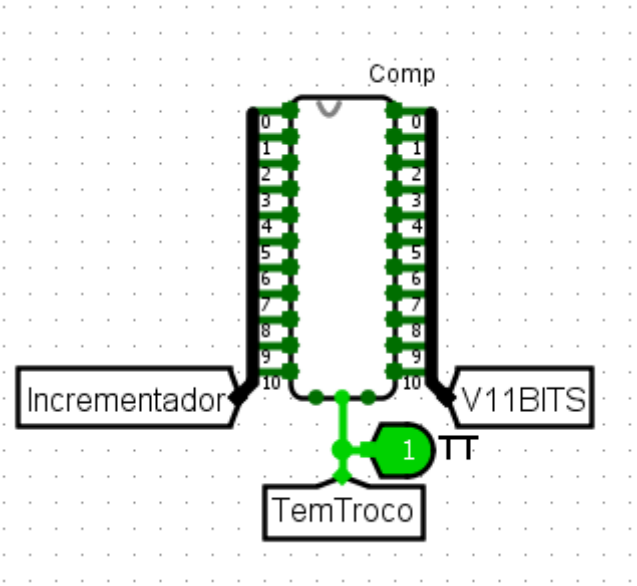
```
--=======================--
-- SOMADOR + REGISTRADOR --
--=======================--

ENTITY SOMAREG IS
    PORT(B: IN BIT_VECTOR(10 DOWNTO 0);
        CLK, CLR, EN: IN BIT;
        AC: OUT BIT_VECTOR(10 DOWNTO 0));
END SOMAREG;

ARCHITECTURE CKT OF SOMAREG IS

COMPONENT ADD11 is
    port(A, B: in bit_vector(10 downto 0);
            O: out bit_vector(10 downto 0);
            CO: out bit);
end COMPONENT;

COMPONENT REG11 IS
    PORT( I: IN BIT_VECTOR(10 DOWNTO 0);
        CLK, CLR, EN: IN BIT;
        O: OUT BIT_VECTOR(10 DOWNTO 0));
END COMPONENT;

SIGNAL RELOAD, RES: BIT_VECTOR(10 DOWNTO 0);
SIGNAL CO: BIT;

BEGIN

ADICAO: ADD11 PORT MAP (RELOAD, B, RES, CO);
REGISTRA: REG11 PORT MAP (RES, CLK, CLR, EN, RELOAD);
AC <= RELOAD(10 DOWNTO 0);

END CKT;
```

Fonte: Autores

No final do processo, na saída haverá mais um comparador, que tem a função de liberar a moeda, se e somente se, o valor do registrador fornecido pelo usuário for igual ao do processamento, no qual a máquina terá como saída denominada "*TemTroco*", caso essa ação seja contraditório ao esperado a máquina terá resposta para a não realização da função "*NãoTemTroco*".

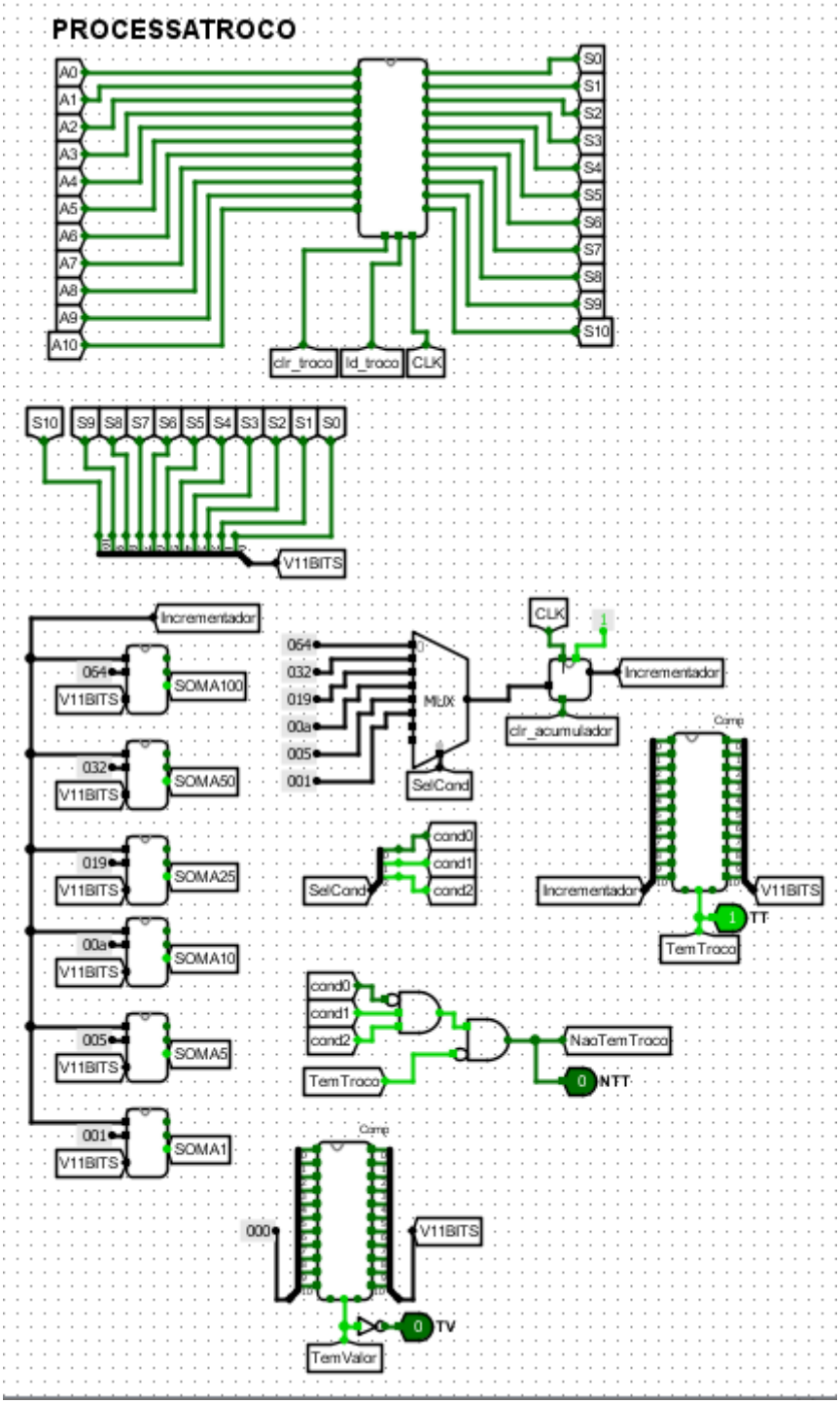
Ao ser implementado, vale relembrar, que o comparador de 11 bits é utilizado a mesma estrutura citada anteriormente, no qual pode ser consultado na figura 12. Este componente comparador apresenta entradas denominadas INCREMENTADOR e V11BITS, onde INCREMENTADOR faz menção ao valor atual registrado do componente somador registrador, e V11BITS é o valor inserido pelo usuário na máquina. A implementação do comparador pode ser vista na figura 20.

Figura 20- Código VHDL de Comparador



Fonte: Autores

Figura 21- esquemático completo do processa troco



Fonte: Autores

Após essas explicações, é necessário demonstrar o funcionamento da seleção do MUX 8x1 onde é possível fazer a definição do valor de entrada para a realização da contagem no processo do troco. Para que o número da moeda correspondente seja determinado, inicialmente o multiplexador deve ter sua seleção setada corretamente. Para isso, é necessário ter em mente, em que dentro do processa troco apresenta uma análise do valor do usuário o qual foi setada na máquina juntamente com as moedas disponíveis dentro da máquina, onde o valor setado pelo usuário passa no registro de 11 bits como mencionado anteriormente, na saída do registrador será servido de entrada para 6 somadores, cada um correspondente ao valor das moedas e cada saída dos somadores apresentará um comparador.

Vale ressaltar que a condição do valor da moeda será válida sendo menor que o valor do troco "*V11BITS*", onde essa saída terá valor lógico alto. Para realizar a seleção, foi pensado na equação booleana que foi nos demonstrado na tabela de condição, no qual, pode ser encontrado no **ANEXO B**, onde traduzimos cada condição com sua respectiva linha de lógica, baseado nas condições demonstradas, precisávamos obter as informações fornecidas. Portanto bastava juntar as equações correspondentes com uma porta OR onde cada CONDIÇÃO é demonstrada no seu valor analógico alto e conseguimos encontrar a equação lógica de cada condição *COND0*, *COND1* e *COND2*. Como pode ser visto na tabela 2

Tabela 2 - Equação de condiçõeS

Condições	Equação Booleana
COND 0	CONDICAO1 OR CONDICAO3 OR CONDICAO5
COND 1	CONDICAO2 OR CONDICAO3 OR CONDICAO6
COND 2	CONDICAO4 OR CONDICAO5 OR CONDICAO6

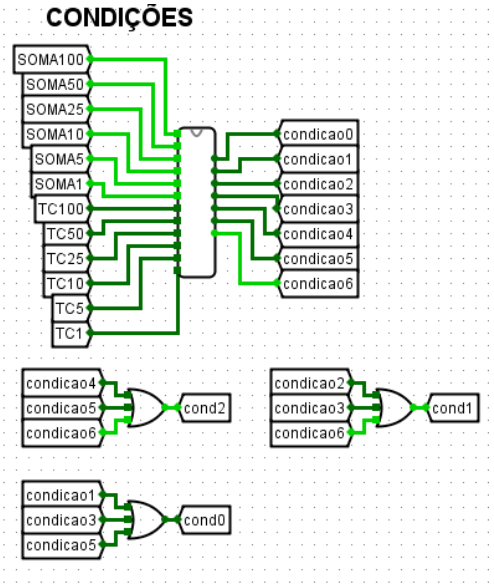
Fonte: Autores

Determinado a equação das condições, o próximo passo foi realizado a implementação do circuito lógico, onde é esperado o resultado da seleção correta de moedas a serem processadas até que o valor correspondente do usuário seja satisfatório.

Na implementação dos componentes condições, as entradas "SOMAX" (X pode ser considerado o valor da moeda em centavos) são cetadas pela saída dos soma-comparadores, onde o nível analogico alto de cada somador é a entrada da implementação do esquemático, juntamente com o valor disponível do cofre da máquina "TCX" (X equivalente ao valor da moeda em centavos).

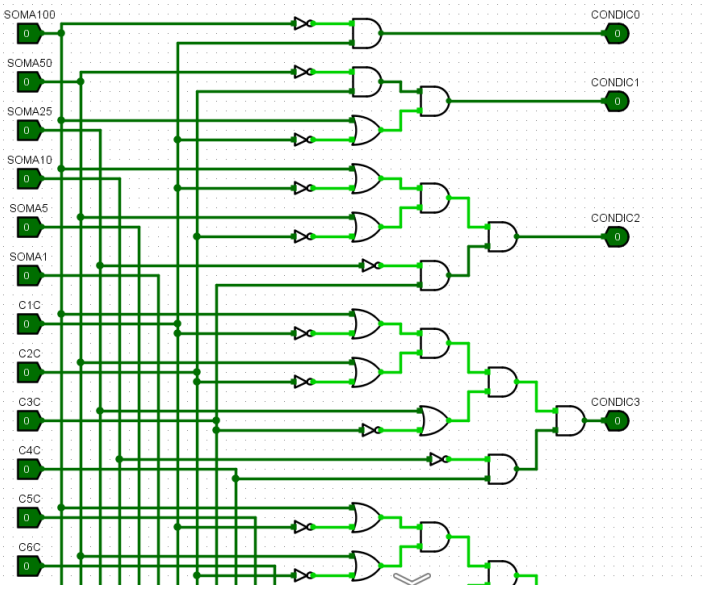
Após ser computado deverá ter saída de nível alto para cada CONDICAOX efetuada pela equação booleana determinada. Com isso, as saídas são juntadas pelas portas OR e denomidas CONDX, onde cada uma é respondida pela equação booleana adquirida na tabela 2.

Figura 22- Implementação das condições



Fonte: Autores

Figura 23- esquemático das condições



Fonte: Autores

Por fim, a lógica para que o valor demonstre, será a entrada de nível alto onde ao ser selecionada no MUX, o mesmo irá ativar as entradas corretas para o funcionamento da contagem do troco. Dessa forma iniciando a contagem do 0 até o valor do troco registrado na máquina.

Na implementação do código VHDL das condições é usando a identidade nomeada CONDICOES, no qual, será apresentado entrada de 1 bit para os valores SOMAX (X equivale aos valores: 100, 50, 25, 20, 5, 1), TCX (X: 100, 50, 25, 10, 5, 1). No qual, é nomeado a saída unitária de bit, denominada CONDICAO0, CONDICAO1, CONDICAO2, CONDICAO3, CONDICAO4, CONDICAO5, CONDICAO6, COND0, COND1, COND2.

Após dessa realização é implementada a arquitetura de nome CKT. representando sinal de valor 1 bit para CONDIC0, CONDIC1, CONDIC2, CONDIC3, CONDIC4, CONDIC5, CONDIC6.

Ao ser feito as saídas é possível identificar que cada saída representada, tem sua resposta diferenciada pela equação booleana feita usando a tabela de condição, sendo encontrada no anexo b.a implementação do código em VHDL das condições pode ser observado na figura 24.

Figura 24- Código VHDL das condições

```
--------------
-- CONDICOES --
--------------

ENTITY CONDICOES IS
    port(SOMA100, SOMA50, SOMA25, SOMA10, SOMA5, SOMA1, TC100, TC50, TC25, TC10, TC5, TC1: IN BIT;
         CONDICAO0, CONDICAO1, CONDICAO2, CONDICAO3, CONDICAO4, CONDICAO5, CONDICAO6, COND0, COND1, COND2: OUT BIT);
END CONDICOES;

ARCHITECTURE CKT OF CONDICOES IS

SIGNAL CONDIC0,CONDIC1,CONDIC2,CONDIC3,CONDIC4,CONDIC5,CONDIC6: BIT;

BEGIN

CONDIC0 <= NOT SOMA100 AND TC100;
CONDIC1 <= (SOMA100 OR NOT TC100) AND (NOT SOMA50 AND TC50);
CONDIC2 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (NOT SOMA25 AND TC25);
CONDIC3 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (SOMA25 OR NOT TC25) AND (NOT SOMA10 AND TC10);
CONDIC4 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (SOMA25 OR NOT TC25) AND (SOMA10 OR NOT TC10) AND (NOT SOMA5 AND TC5);
CONDIC5 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (SOMA25 OR NOT TC25) AND (SOMA10 OR NOT TC10) AND (SOMA5 OR NOT TC5) AND (NOT SOMA1 AND TC1);
CONDIC6 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (SOMA25 OR NOT TC25) AND (SOMA10 OR NOT TC10) AND (SOMA5 OR NOT TC5) AND (SOMA1 OR NOT TC1);

CONDICAO0 <= CONDIC0 ;
CONDICAO1 <= CONDIC1 ;
CONDICAO2 <= CONDIC2 ;
CONDICAO3 <= CONDIC3 ;
CONDICAO4 <= CONDIC4 ;
CONDICAO5 <= CONDIC5 ;
CONDICAO6 <= CONDIC6 ;

COND0 <= CONDIC1 OR CONDIC3 OR CONDIC5;
COND1 <= CONDIC2 OR CONDIC3 OR CONDIC6;
COND2 <= CONDIC4 OR CONDIC5 OR CONDIC6;

END CKT;
```

Fonte: Autores.

Ao ser demonstrado o VHDL de condição, para que possa haver atividade simultaneamente na selação do mux é realizado esse codigo em VHDL de um MUX 8x1 11 bits.

Onde é demonstrado na entidade MUX8X1_11INPUTS entrada de vetor 11 bits denominados, I0, I1, I2, I3, I4, I5, I6, I7. no qual, é apresentado outra entrada de vetor 3 bits nomeado SEL.

Sua saída implementada é de vetor 11 bits denominada “O”, sendo a arquitetura contruida nomeada CKT, apresenta chamada de componente MUX81de entrada valor unitario em bit I0, I1, I2, I3, I4, I5, I6, I7, S0, S1, S2, representando saída O de 1 bit . Vale ressaltar, que nessa arquitetura é utilizado codigo de PORT MAP de 11 bits, sendo nomeadas saídas de MUX1, MUX2, MUX3, MUX4, MUX5, MUX6, MUX7, MUX8, MUX9, MUX10 e MUX11. Como pode ser visualizado na figura 25

Figura 25- código VHDL do multiplexador 8x1 11bits

```
--============================--
-- MULTIPLEXADOR 8X1 11 BITS --
--============================--

ENTITY MUX8X1_11INPUTS IS
    PORT( I0, I1, I2, I3, I4, I5, I6, I7: IN BIT_VECTOR(10 DOWNTO 0);
        SEL: IN BIT_VECTOR(2 DOWNTO 0 );
        O: OUT BIT_VECTOR(10 DOWNTO 0));
END MUX8X1_11INPUTS;

ARCHITECTURE CKT OF MUX8X1_11INPUTS IS

COMPONENT MUX81 is
    port(I0, I1, I2, I3, I4, I5, I6, I7, S0, S1, S2: in bit;
       O: out bit);
end COMPONENT;

BEGIN

    MUX1: MUX81 PORT MAP(I0(0),I1(0), I2(0), I3(0), I4(0), I5(0), I6(0), I7(0), SEL(0), SEL(1), SEL(2), O(0));
    MUX2: MUX81 PORT MAP(I0(1),I1(1), I2(1), I3(1), I4(1), I5(1), I6(1), I7(1), SEL(0), SEL(1), SEL(2), O(1));
    MUX3: MUX81 PORT MAP(I0(2),I1(2), I2(2), I3(2), I4(2), I5(2), I6(2), I7(2), SEL(0), SEL(1), SEL(2), O(2));
    MUX4: MUX81 PORT MAP(I0(3),I1(3), I2(3), I3(3), I4(3), I5(3), I6(3), I7(3), SEL(0), SEL(1), SEL(2), O(3));
    MUX5: MUX81 PORT MAP(I0(4),I1(4), I2(4), I3(4), I4(4), I5(4), I6(4), I7(4), SEL(0), SEL(1), SEL(2), O(4));
    MUX6: MUX81 PORT MAP(I0(5),I1(5), I2(5), I3(5), I4(5), I5(5), I6(5), I7(5), SEL(0), SEL(1), SEL(2), O(5));
    MUX7: MUX81 PORT MAP(I0(6),I1(6), I2(6), I3(6), I4(6), I5(6), I6(6), I7(6), SEL(0), SEL(1), SEL(2), O(6));
    MUX8: MUX81 PORT MAP(I0(7),I1(7), I2(7), I3(7), I4(7), I5(7), I6(7), I7(7), SEL(0), SEL(1), SEL(2), O(7));
    MUX9: MUX81 PORT MAP(I0(8),I1(8), I2(8), I3(8), I4(8), I5(8), I6(8), I7(8), SEL(0), SEL(1), SEL(2), O(8));
    MUX10:MUX81 PORT MAP(I0(9),I1(9), I2(9), I3(9), I4(9), I5(9), I6(9), I7(9), SEL(0), SEL(1), SEL(2), O(9));
    MUX11:MUX81 PORT MAP(I0(10), I1(10), I2(10), I3(10), I4(10), I5(10), I6(10), I7(10), SEL(0), SEL(1), SEL(2), O(10));

END CKT;
```

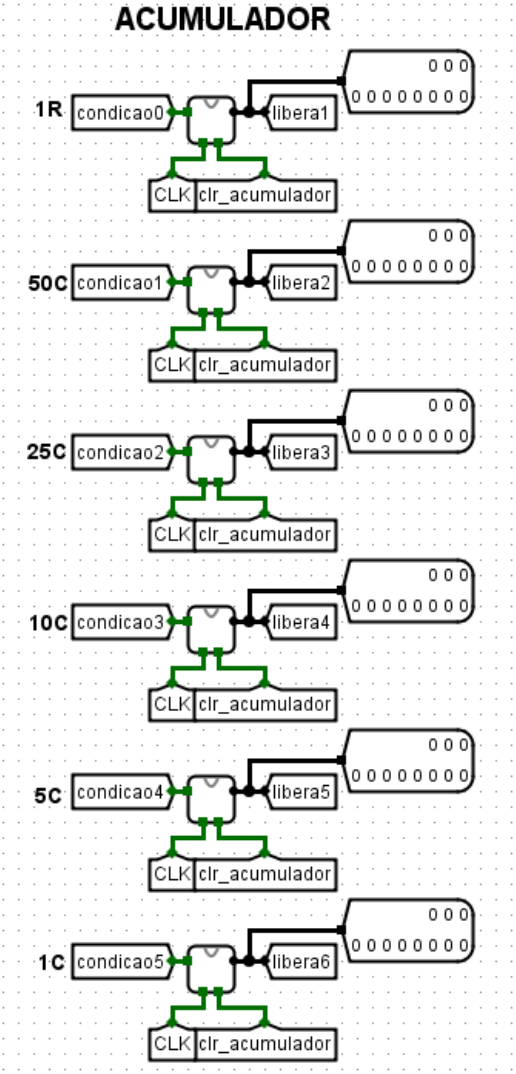
Fonte: Autores

3.1.2 Acumulador

Tendo o conhecimento do funcionamento do processa troco, um ponto a lembrar que existe outro circuito que estará trabalhando de maneira simultânea com o circuito mencionado na seção 3.1, o qual é registrado a quantidade de moedas retiradas do cofre dependendo da

entrada condição que foi implementada, como pode ser observado na figura 17, um exemplo do funcionamento: se o valor de entrada V = 500 (5 reais), no circuito implementado, será realizado a operação de contador de valor inicial 0 e a medida que a entrada condição for valor analógico alto, será computado o acréscimo de 1 bit, quivalente a 1 moeda, portanto, o cofre terá como resposta a liberação de 5 moedas de 1 Real, ou seja "*libera1=100"*. Vale ressaltar, que cada registro do contador é realizado por pulso de clock.

Figura 26-- Esquemático do acumulador

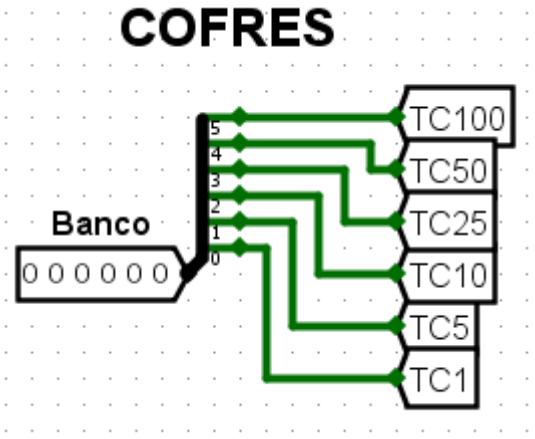


Fonte: Autores

3.1.3 Cofre

A máquina de troco tem como objetivo alem dos processos, armazenar a quantidade de moedas presente dentro da máquina, ela é responsável por avisar ao usuário se as moedas armazenadas é suficiente para a liberação do troco correto "*TemTroco*", caso contrário terá como resposta a impossibilidade de liberação do troco "*NãoTemTroco*". Primeiramente projetamos para a realização de moeda infinita, com o objetivo de estudo do funcionamento. Como o relatório sugere o implemento do cofre finito, não foi possível realizar devido a problemas nos dados fornecidos e a implementação do circuito, encontramos problemas no registro de saída onde não ficava coerente com o esperado de resposta, então foi discutido a ideia e chegamos a conclusão de usar o dinheiro infinito no qual será setado no bloco operacional, para que seja feita a realização do processamento.

Figura 27cofre



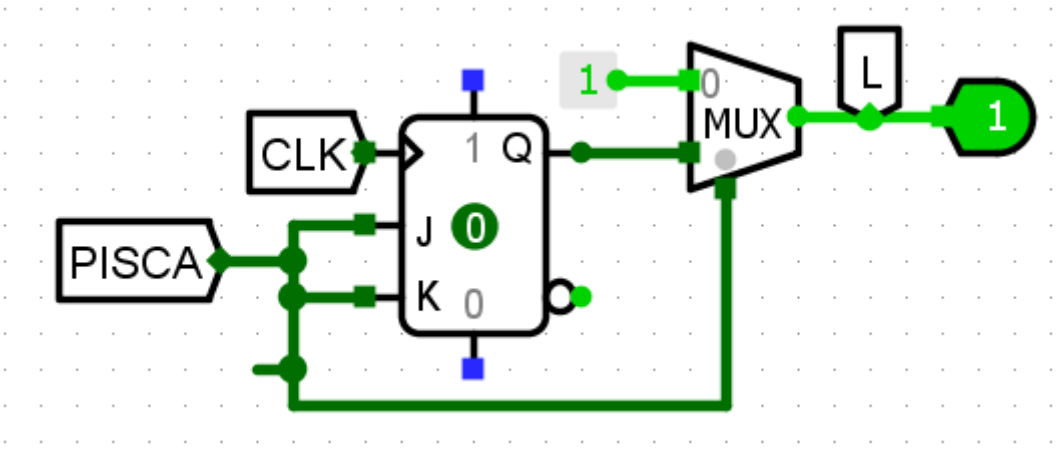
Fonte: Autores

3.1.4 LED

Implementação do circuito baseado no led de informação da máquina, quando o led está aceso irá demonstrar o funcionamento do circuito foi ativado, a segunda função ela deverá ficar piscando enquanto estiver no estado processo de troco, onde após receber o valor do dinheiro pelo usuário, a máquina estará computando o seu valor em moedas, e no final ele irá ficar aceso quando houver liberação das moedas, seja ela satisfatória ou não. Utilizamos a saída de um MUX 2x1 onde colocamos uma das entradas em valor analógico alto e a segunda entrada, sendo conectada em uma saída de um flip-flop tipo JK. A ideia é justamente a entrada "*PISCA*" ficará no analógico alto alimentando a entrada J e K do flip-flop, simultaneamente na seleção do mux, onde será possível fazer com que o led alterne o seu

valor analógico em high e low, sendo possível realizar a função de acender e apagar o LED durante o processo da máquina.

Figura 28- Implementação do LED



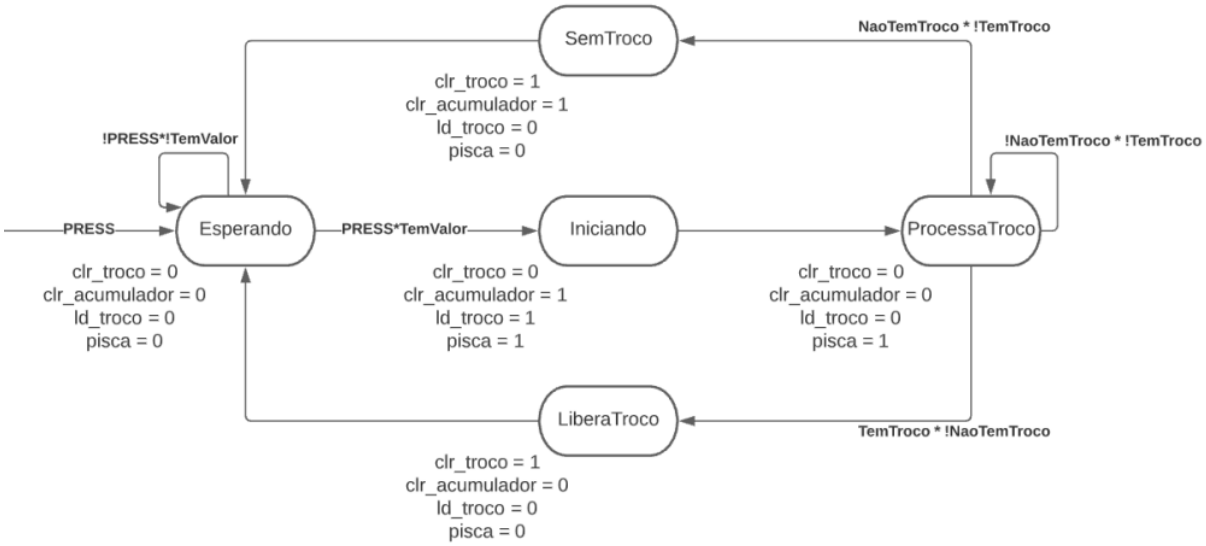
Fonte: Autores

3.2 BLOCO DE CONTROLE

3.2.1 Máquina de Estado

Ao ser implementado a máquina de estado, precisamos realizar o diagrama da máquina de troco sendo representada como deve se comportar de cada ação e seu respectivo estado seguinte, como pode ser visto na figura 29.

Figura 29- Tabela de transição de estados



Fonte: Autores

Para ser possível a implementação do bloco de controle da máquina, primeiramente precisamos refazer a tabela de estado fornecido no relatório, como pode ser visto no anexo C.

Pois como apresentava ausências de dados nas entradas, optamos por redefinir a tabela para melhor ser trabalhada na implementação, como pode ser conferido na figura 30.

Figura 30- Tabela de transição de estados

ESTADOS	ENTRADAS							ESTADOS	SAIDAS						
	Q2	Q1	Q0	PRESS	TemValor	TemTroco	NaoTemTroco		D2	D1	D0	CLR_TROCO	CLR_ACUMULADOR	LD_TROCO	PISCA
ESPERANDO	0	0	0	0	0	X	X	ESPERANDO	0	0	0	0	0	0	0
	0	0	0	0	1	X	X	ESPERANDO	0	0	0	0	0	0	0
	0	0	0	1	0	X	X	ESPERANDO	0	0	0	0	0	0	0
	0	0	0	1	1	X	X	INICIANDO	0	0	1	0	0	0	0
INICIANDO	0	0	1	X	X	X	X	PROCESSATROCO	0	1	0	0	1	1	1
PROCESSATROCO	0	1	0	X	X	0	0	PROCESSATROCO	0	1	0	0	0	0	1
	0	1	0	X	X	0	1	SEMTROCO	0	1	1	0	0	0	1
	0	1	0	X	X	1	0	TEMTROCO	1	0	0	0	0	0	1
SEMTROCO	0	1	1	X	X	X	X	ESPERANDO	0	0	0	1	1	0	0
LIBERATROCO	1	0	0	X	X	X	X	ESPERANDO	0	0	0	1	0	0	0

Fonte: Autores

Tabela feita, poderemos realizar o próximo passo da implementação, no qual é a realização do mapa de Karnaugh. Onde poderemos retirar os dados oferecidos na tabela e convertê-los em equações de fácil entendimento para a construção do circuito que dará os processos corretos para a máquina de troco, pode ser analisada na figura 31.

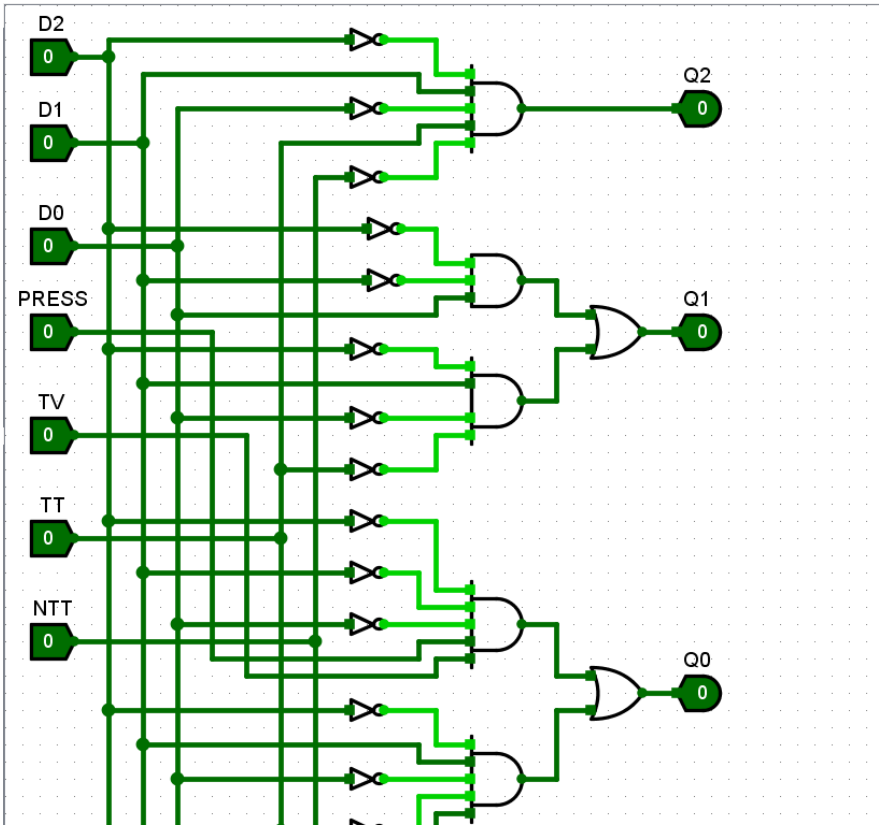
Figura 31- Tabela Combinacional

SAÍDAS	LÓGICA COMBINACIONAL
Q2	D2'*D1*D0'*TT*NTT'
Q1	(D2'*D1'*D0) + (D2'*D1*D0'*TT'*NTT') + (D2'*D1*D0'*TT'*NTT)
Q0	(D2'*D1*D0'*TT'*NTT) + (D2'*D1'*D0'*PRESS*TV)
CLR_TROCO	(D2'*D1*D0) + (D2*D1'*D0')
CLR_ACUMULADOR	(D2'*D1'*D0) + (D2'*D1* D0)
LD_TROCO	(D2'*D1'*D0)
PISCA	(D2'*D1'*D0) + (D2'*D1*D0'*TT'*NTT') + (D2'*D1*D0'*TT'*NTT) + (D2'*D1*D0'*TT*NTT')

Fonte: Autores

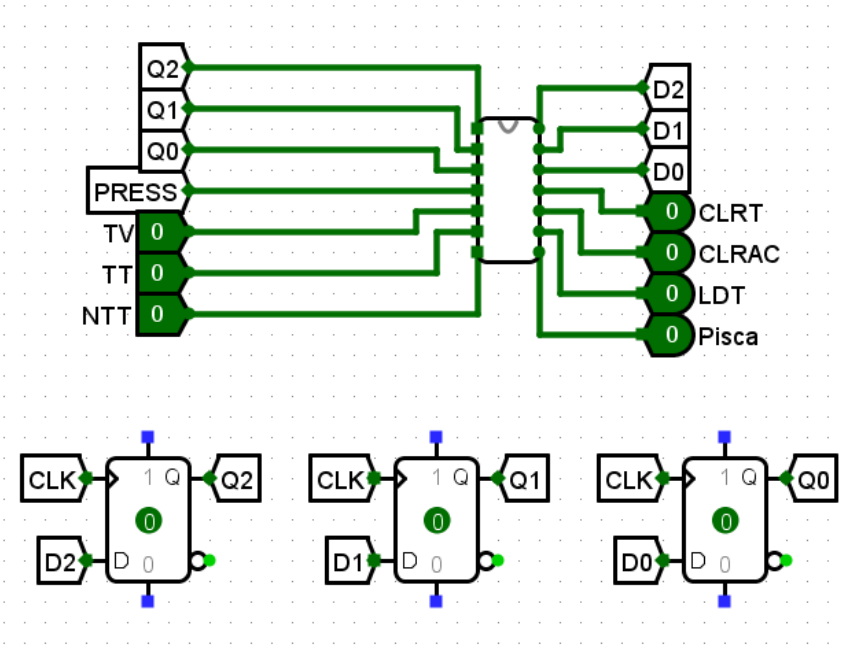
Equação cumprida, temos dados suficientes para realizar a implementação do circuito da máquina de estado onde usamos o programa *Logisim-ITA* para a realização do circuito.

Figura 32: Implementação do circuito da máquina



Fonte: Autores

Figura 33: Máquina de Estado

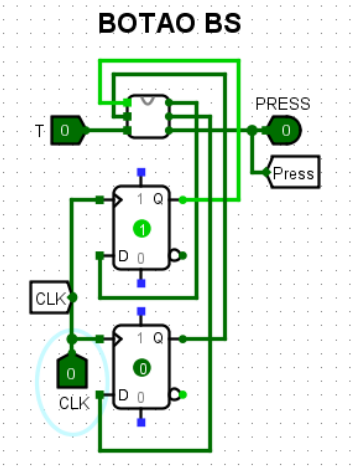


Fonte: Autores

3.2.2 Botão Síncrono

A realização da criação do botão síncrono (BS) como máquina de estado mostrada na Figura 32. Esta máquina garante que, quando pressionado o botão, T terá nível lógico alto durante apenas 1 *clock*, essa ideia intensifica em um problema de o usuário apertar o botão e deixar pressionado por um certo tempo.

Figura 34: Botão Síncrono

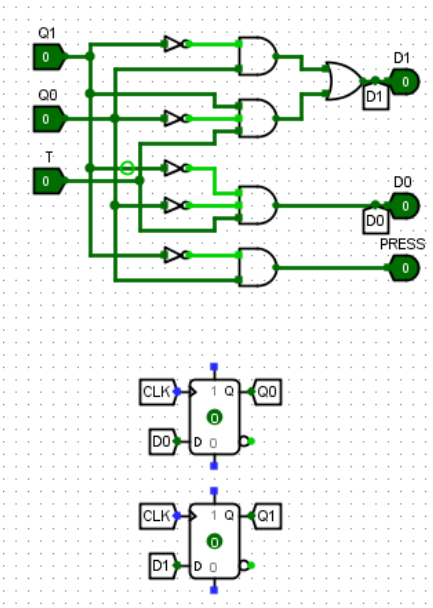


Fonte: Autores

Onde o funcionamento desse circuito utilizando flip flop do tipo D, para ser realizado a contagem do tempo baseado no pulso de clock. Assim, satisfazendo a sugestão fornecida no relatório recebido, a título de informação, implementamos um circuito que é responsável pela função do botão, pode ser observada na figura 33. Como saída do botão BS para a máquina é

representado pelo "*PRESS*", onde ele terá como entrada a máquina de estado com função da ativação da máquina para o inicio do processo.

Figura 35: Implementação do push botton

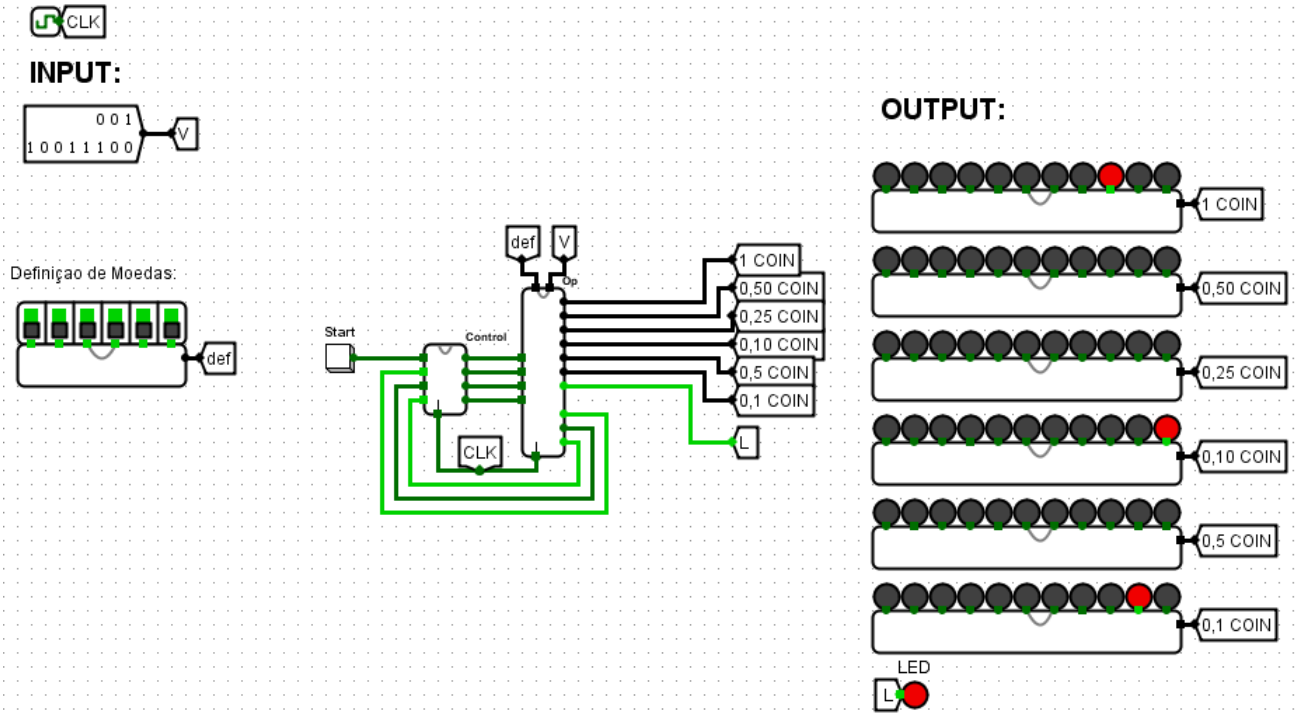


Fonte: Autores

3.3 LAYOUT

Ao ser demonstrado todo o esquemático da máquina de troco, optamos em realizar um layout para que possa ser demonstrado o funcionamento de entrada e saída final do circuito completo, como pode ser analisada na figura 36.

Figura 36- layout da máquina de troco



Fonte: Autores

Cada etapa do layout tem sua respectiva função:

INPUT	será o valor de entrada que o usuário deseja receber em moedas na máquina.
Definição de Moedas	Define qual moeda deseja receber, a esquerda mais significativo e mais a direita menos significativo
Bloco de Controle	Onde o usuário pode colocar o valor do dinheiro na máquina e possa pressionar o botão de ativação
Bloco Operacional	Bloco responsável pelo processamento completo da máquina
OUTPUT	Valor da quantidade de moedas usadas em bit, onde o led mais significativo é a esquerda e o menos significativo é a direita
L	Led indicador da máquina, se o mesmo está sendo ativada, processando troco

3.4 TÓPICO DE MUDANÇAS

No decorrer da implementação, as mudanças foram realizadas da versão original foi a alteração da tabela de estado fornecida no relatório, devido a falta de dados suficientes na implementação da máquina de estado principalmente a ausência de entradas na realização da máquina.

Outro ponto a ser mencionado é o valor de bits armazenados na máquina, originalmente necessitava o uso de 12 bits no total, o que foi alterado para uso de 11 bits sendo que o 11º bit é considerado como o bit de sinal, no qual era mudado o valor de entrada apresentando modo negativo.

4 CONCLUSÃO

A complexidade dos sistemas digitais exige combinações de circuitos sequenciais, combinacionais e uma padronização/sistematização no processo. Uma dessas combinações, nomeada processador, faz uso de blocos de controle e blocos operacionais. Uma aplicação do dia-a-dia desse conceito é uma máquina de trocos, a qual foi projetada no presente relatório.

A máquina projetada precisa dar um valor em moedas, determinado pelo usuário. A liberação das moedas ocorre através de um sistema cofre que libera uma moeda sempre que em sua entrada ix (onde x = 1, 2, …, 6) existir um nível lógico alto e ocorrer um pulso de clock. A máquina possui como entrada o valor do troco em binário (V), uma entrada T, referente ao pulso gerado a partir da saída do circuito de um botão sincronizado (BS). E uma saída L, na qual quando está piscando indica que a máquina está processando a informação para liberar o troco e qualquer outra solicitação de troco será ignorada.

O circuito também tem a capacidade de informar quando não há troco para o valor fornecido, mantendo a saída LED em nível lógico alto até que um novo valor de troco seja fornecido.

A abordagem de implementação do projeto proposto foi centrada na utilização de somadores, registradores e comparadores. Nesse sentido, foi criado um registrador de 11 bits utilizando 11 flip flops do tipo D. Este registrador foi utilizado na implementação do registro de valor do cofre, no acumulador de troco, utilizado na verificação de disponibilidade da máquina para liberação de dinheiro, e no conjunto de 6 registradores de 11 bits para fazer a distinção de quantas moedas de cada tipo devem ser entregues para o cliente.

Ademais, um somador de 11 bits foi implementado e também compõe parte da lógica de processamento de troco junto do registrador supracitado. A soma, junto ao registrador, foi utilizada no intuito de informar a máquina, a cada pulso de clock, o quanto de dinheiro já havia sido guardado em moedas. Além disso, compôs a lógica do acumulador para contar quantas moedas deveriam ser despejadas da máquina.

Os comparadores utilizados no projeto atuam consoantes aos registradores e somadores. É neste componente onde são realizadas as verificações que fazem menção às condições de comparação do troco atual na máquina com o dinheiro solicitado, se a soma de uma moeda com o dinheiro acumulado é maior ou não que o dinheiro solicitado e se o valor monetário inserido na máquina é diferente de 0.

Somado aos 3 tipos de componentes mencionados anteriormente foi implementada uma lógica combinacional. Esta foi adicionada ao bloco operacional no intuito de discernir qual resultado de soma deveria ser colocado no acumulador de dinheiro. Isto visto que a soma do valor atual com algumas moedas poderia ultrapassar o valor solicitado pelo usuário.

Por fim, o bloco de controle utilizado na implementação do projeto descrito neste relatório é composto por: uma máquina de estados, um registrador de estados e um botão síncrono. A máquina de estados é capacitada para informar ao bloco de controle quando deverá ser realizada as operações de cálculo de troco e liberação de troco. O botão síncrono fornece o valor booleano para a máquina de estados informar ao bloco operacional quando iniciar a operação, e o registrador de estados guarda a cada clock em que estado a máquina se encontra e qual deverá ser o próximo passo mediante as condições estabelecidas.

Devido a presença de um bloco de controle e um operacional, haja vista o disposto na literatura, o projeto foi centrado na abordagem de desenvolvimento de processadores RTL. Além disso, mediante o apresentado na implementação do projeto, vemos que o projeto proposto foi exequível e funcional após a alteração de alguns componentes como a máquina de estados. Dito isso, constata-se além da plausibilidade de execução do projeto, a prática de estudo e experiência de aplicação acerca de circuitos digitais no que tange somadores, registradores, circuitos sequenciais, combinacionais e máquinas de estado finitos.

5 REFERÊNCIAS

VAHID, Frank. **Sistemas digitais: projeto, otimização e HDLS**. Rio Grande do Sul: Artmed Bookman, 2008. 558 p.

ANEXOS

ANEXO A - RELATO SEMANAL

Líder: Isaac de Lyra Junior

1 EQUIPE

Tabela 1 - Identificação da Equipe

Função do grupo:	Discente
Redator	Vinicius Souza Fonseca
Debatedor	Alysson Ferreira da Silva
Videomaker	João Matheus Bernardo Resende
Auxiliar	-

Fonte: Produzido pelos autores.

A.2 DEFINA O PROBLEMA

O problema consistia em implementar uma máquina de troco capaz de receber um valor em binário na entrada e fornecer como saída a quantidade de moedas de cada tipo que devem sair da máquina para formar o troco inserido. A liberação das moedas é realizada por um sistema cofre que libera uma moeda sempre que em sua entrada i_x(onde x = 1,2,...,6) existir um nível lógico alto e ocorrer um pulso de clock. Além disso, a máquina também possui uma saída L que quando está piscando indica que a máquina está processando a informação para liberar o troco e qualquer outra solicitação de troco será ignorada. A máquina de troco possui também a capacidade de verificar se os cofres de moedas estão vazios ou não e recalcula o troco para liberar moedas apenas dos cofres que não estão vazios.

A.3 REGISTRO DO BRAINSTORMING

O primeiro brainstorming ocorreu na terça-feira, onde foi estudado o projeto recebido a fim de identificar falhas que poderiam comprometer a implementação do mesmo. Ao final do primeiro brainstorming foi decidido que seria melhor iniciar a implementação através do bloco responsável pelo estado “ProcessaTroco”, pois nele estaria a parte principal do processamento da máquina. Na segunda reunião foi continuado a implementação das operações do estado “ProcessaTroco”, que é responsável pelo fornecimento das entradas internas TemValor, TemTroco, NaoTemTroco do bloco de controle, foi concluído todas as operações desse estado ainda nesta reunião e iniciado a discussão sobre o estado “LiberaTroco”, ao tentarmos implementar as operações deste estado percebemos diversos *bugs* no diagrama de blocos dado no relatório de projeto, estes *bugs* tiveram como principal causador a utilização do conceito de moedas finitas que, para implementar, utilizaria um registrador em conjunto com um subtrator,

tal registrador receberia a princípio a quantidade de moedas que deveria ter em cada cofre de moedas e a cada pulso dado este valor deveria decrementar até chegar a zero. A implementação deste conceito foi fonte de diversas discussões internas e decidimos abandonar a ideia de moedas finitas.

As moedas finitas foram trocadas por sinalizadores de cofres, tais sinalizadores são bits que vão definir se tem ou não moedas de um tipo específico, foram denominados de TC_x(onde x=100, 50, 25, 10, 5, 1), quando TC=1 indica que tem moeda no cofre, do contrário, não tem. Através desta mudança no projeto foi possível concluir a implementação do esquemático do circuito no decorrer da semana com tranquilidade, ficando em déficit apenas a implementação em VHDL do circuito implementado.

A.4 PONTOS-CHAVES

Os pontos chaves para a realização da implementação foi entender como a conexão entre bloco de controle e bloco operacional de uma máquina de estados ocorre, o uso de uma tabela de condições, que analisa as condições necessárias para contagem de uma determinada moeda e por fim o entendimento de como utilizar registradores, comparadores e somadores na prática.

A.5 QUESTÕES DE PESQUISA

Os principais tópicos de pesquisa para realizar a implementação estão listados abaixo:

1. Máquina de Estados;
2. Conexão entre bloco de controle e bloco operacional de uma máquina de estados;
3. Registradores;
4. Comparadores;
5. Somadores;
6. Multiplexadores.

A.6 PLANEJAMENTO DA PESQUISA

As pesquisas realizadas durante a semana tiveram como fonte o projeto recebido pelo grupo 2 anterior, o livro texto “Sistemas digitais: projeto, otimização e HDLS” de F. VAHID, como também foi utilizado video-aulas. Primeiramente foi estudado o projeto recebido e para implementação foi estudado os conceitos presentes na seção A.5.

ANEXO B - Tabela de condições

CONDIÇÕES	Cond 2	Cond 1	Cond 0	Expressão
0	0	0	0	SOMA100'*TC100
1	0	0	1	(SOMA100 + TC100') * (SOMA50' + TC50)
2	0	1	0	(SOMA100 + TC100') * (SOMA50 + TC50') * (SOMA25' * TC25)
3	0	1	1	(SOMA100 + TC100') * (SOMA50 + TC50') * (SOMA25 + TC25') * (SOMA10' * TC10)
4	1	0	0	(SOMA100 + TC100') * (SOMA50 + TC50') * (SOMA25 + TC25') * (SOMA10 + TC10') * (SOMA5' * TC5)
5	1	0	1	(SOMA100 + TC100') * (SOMA50 + TC50') * (SOMA25 + TC25') * (SOMA10 + TC10') * (SOMA5 + TC5') * (SOMA1' * TC1)
6	1	1	0	(SOMA100 + TC100') * (SOMA50 + TC50) * (SOMA25 + TC25') * (SOMA10 + TC10') * (SOMA5 + TC5') * (SOMA1 + TC1')

Fonte: Elaboração do grupo 01 <ADAPTADA>.

ANEXO C - Tabela de Estados

ESTADOS	ESTADO ATUAL		ENTRADA				ESTADO FUTURO		SAÍDA
	Q1	Q0	T	TEMTROCO	ÑTEM TROCO	TROCOL IBERADO	E1	E0	L
INÍCIO	0	0	0	X	X	X	0	0	0
	0	0	1	X	X	X	1	1	0
PROCESSA TROCO	1	1	X	0	0	X	1	1	1
	1	1	X	1	0	X	0	1	1
	1	1	X	0	1	X	1	0	1
SEM TROCO	1	0	0	X	X	X	1	0	1
	1	0	1	X	X	X	1	1	1
LIBERA TROCO	0	1	X	X	X	0	0	1	0
	0	1	X	X	X	1	0	0	0

ANEXO D - CÓDIGO EM VHDL

```
--==============--
-- MEIO SOMADOR --
--==============--

entity HALF_ADD is
    port(A, B: in bit;
         S, CO: out bit);
end HALF_ADD;

architecture CKT of HALF_ADD is

begin

    S <= A xor B;
    CO <= A and B;

end CKT;


--==================--
-- SOMADOR COMPLETO --
--==================--

entity COMP_ADD is
    port(A, B, CI: in bit;
         S, CO: out bit);
end COMP_ADD;

architecture CKT of COMP_ADD is

begin

    S <= A xor B xor CI;
    CO <= (B and CI) or (A and CI) or (A and B);

end CKT;


--=================--
-- SOMADOR 11 BITS --
--=================--

entity ADD11 is
    port(A, B: in bit_vector(10 downto 0);
         O: out bit_vector(10 downto 0);
```

```
          CO: out bit);
end ADD11;

architecture CKT of ADD11 is

component HALF_ADD is
   port(A, B: in bit;
       S, CO: out bit);
end component;

component COMP_ADD
   port(A, B, CI: in bit;
          S, CO: out bit);
end component;

signal VAI_UM: bit_vector(9 downto 0);

begin

   S0: HALF_ADD port map(A(0), B(0), O(0), VAI_UM(0));
   S1: COMP_ADD port map(A(1), B(1), VAI_UM(0), O(1), VAI_UM(1));
   S2: COMP_ADD port map(A(2), B(2), VAI_UM(1), O(2), VAI_UM(2));
   S3: COMP_ADD port map(A(3), B(3), VAI_UM(2), O(3), VAI_UM(3));
   S4: COMP_ADD port map(A(4), B(4), VAI_UM(3), O(4), VAI_UM(4));
   S5: COMP_ADD port map(A(5), B(5), VAI_UM(4), O(5), VAI_UM(5));
   S6: COMP_ADD port map(A(6), B(6), VAI_UM(5), O(6), VAI_UM(6));
   S7: COMP_ADD port map(A(7), B(7), VAI_UM(6), O(7), VAI_UM(7));
   S8: COMP_ADD port map(A(8), B(8), VAI_UM(7), O(8), VAI_UM(8));
   S9: COMP_ADD port map(A(9), B(9), VAI_UM(8), O(9), VAI_UM(9));
   S10: COMP_ADD port map(A(10), B(10), VAI_UM(9), O(10), CO);

end CKT;


--====================--
-- MULTIPLEXADOR 2x1 --
--====================--

entity MUX21 is
   port(A, B, S: in bit;
      O: out bit);
 end MUX21;

 architecture CKT of MUX21 is

 begin

    O <= (B and S) or (A and (not S));
```

```
 end CKT;


--====================--
-- MULTIPLEXADOR 8x1 --
--====================--

entity MUX81 is
    port(I0, I1, I2, I3, I4, I5, I6, I7, S0, S1, S2: in bit;
       O: out bit);
 end MUX81;

 architecture CKT of MUX81 is

 begin

     O <= (I0 AND (NOT S2 AND NOT S1 AND NOT S0))  OR (I1 AND (NOT S2 AND NOT S1
AND S0))
        OR (I2 AND (NOT S2 AND S1 AND NOT S0)) OR  (I3 AND (NOT S2 AND S1 AND S0))
        OR (I4 AND (S2 AND NOT S1 AND NOT S0)) OR  (I5 AND (S2 AND NOT S1 AND S0))
        OR (I6 AND (S2 AND S1 AND NOT S0)) OR (I7  AND (S2 AND S1 AND S0));

end CKT;

--============================--
-- MULTIPLEXADOR 8X1 11 BITS --
--============================--

ENTITY MUX8X1_11INPUTS IS
    PORT( I0, I1, I2, I3, I4, I5, I6, I7: IN BIT_VECTOR(10 DOWNTO 0);
        SEL: IN BIT_VECTOR(2 DOWNTO 0 );
        O: OUT BIT_VECTOR(10 DOWNTO 0));
END MUX8X1_11INPUTS;

ARCHITECTURE CKT OF MUX8X1_11INPUTS IS

COMPONENT MUX81 is
    port(I0, I1, I2, I3, I4, I5, I6, I7, S0, S1, S2: in bit;
       O: out bit);
end COMPONENT;

BEGIN

    MUX1: MUX81 PORT MAP(I0(0),I1(0), I2(0), I3(0), I4(0), I5(0), I6(0), I7(0), SEL(0),
SEL(1), SEL(2), O(0));
    MUX2: MUX81 PORT MAP(I0(1),I1(1), I2(1), I3(1), I4(1), I5(1), I6(1), I7(1), SEL(0),
SEL(1), SEL(2), O(1));
```

```
   MUX3: MUX81 PORT MAP(I0(2),I1(2), I2(2), I3(2), I4(2), I5(2), I6(2), I7(2), SEL(0),
SEL(1), SEL(2), O(2));
   MUX4: MUX81 PORT MAP(I0(3),I1(3), I2(3), I3(3), I4(3), I5(3), I6(3), I7(3), SEL(0),
SEL(1), SEL(2), O(3));
   MUX5: MUX81 PORT MAP(I0(4),I1(4), I2(4), I3(4), I4(4), I5(4), I6(4), I7(4), SEL(0),
SEL(1), SEL(2), O(4));
   MUX6: MUX81 PORT MAP(I0(5),I1(5), I2(5), I3(5), I4(5), I5(5), I6(5), I7(5), SEL(0),
SEL(1), SEL(2), O(5));
   MUX7: MUX81 PORT MAP(I0(6),I1(6), I2(6), I3(6), I4(6), I5(6), I6(6), I7(6), SEL(0),
SEL(1), SEL(2), O(6));
   MUX8: MUX81 PORT MAP(I0(7),I1(7), I2(7), I3(7), I4(7), I5(7), I6(7), I7(7), SEL(0),
SEL(1), SEL(2), O(7));
   MUX9: MUX81 PORT MAP(I0(8),I1(8), I2(8), I3(8), I4(8), I5(8), I6(8), I7(8), SEL(0),
SEL(1), SEL(2), O(8));
   MUX10:MUX81 PORT MAP(I0(9),I1(9), I2(9), I3(9), I4(9), I5(9), I6(9), I7(9), SEL(0),
SEL(1), SEL(2), O(9));
   MUX11:MUX81 PORT MAP(I0(10), I1(10), I2(10), I3(10), I4(10), I5(10), I6(10), I7(10),
SEL(0), SEL(1), SEL(2), O(10));

END CKT;


--=============--
-- FlipFlop JK --
--=============--

ENTITY ffjk IS
port (clk ,J,K,P,C: IN BIT;
q: OUT BIT );
END ffjk ;

ARCHITECTURE ckt OF ffjk IS

SIGNAL qS: BIT;

BEGIN

PROCESS (clk ,P,C)
BEGIN
IF P = '0' THEN qS <= '1';
ELSIF C = '0' THEN qS <= '0';
ELSIF clk ='1' AND clk ' EVENT THEN
IF J = '1' AND K = '1' THEN qS <= NOT qS;
ELSIF J = '1' AND K = '0' THEN qS <= '1';
ELSIF J = '0' AND K = '1' THEN qS <= '0';
END IF;
END IF;
END PROCESS ;
```

```
q <= qS;

END ckt;

--=============--
-- FlipFlop D --
--=============--

ENTITY ffd IS
        port ( clk ,D ,P , C : IN BIT ;
                q : OUT BIT );
END ffd ;

ARCHITECTURE ckt OF ffd IS
        SIGNAL qS : BIT;
BEGIN
   PROCESS ( clk ,P ,C )
            BEGIN
            IF P = '0' THEN qS <= '1';
            ELSIF C = '0' THEN qS <= '0';
            ELSIF clk = '1' AND clk ' EVENT THEN
            qS <= D ;
            END IF;
   END PROCESS ;
q <= qS ;
END ckt ;


--================================--
-- COMPARADOR DE MAGNITUDE 1 BIT --
--================================--

entity COMP_BIT is
   port(A, B, maior,igual, menor: in bit;
          AgtB, AeqB, AltB: out bit);
end COMP_BIT;

ARCHITECTURE CKT OF COMP_BIT IS

BEGIN

   AgtB <= (((not B) and A) and igual) or maior;
   AeqB <= ((A xnor B) and igual);
   AltB <= (((not A) and B) and igual) or menor;

END CKT;
```

```
--==================================--
-- COMPARADOR DE MAGNITUDE 11 BITS --
--==================================--

entity COMP11 is
   port(A,B: in bit_vector(10 downto 0);
           AgtB, AeqB, AltB: out bit);
end COMP11;

ARCHITECTURE CKT OF COMP11 IS

component COMP_BIT is
    port(A, B, maior,igual, menor: in bit;
           AgtB, AeqB, AltB: out bit);
end component;

signal GT, EQ, LT: bit_vector(10 downto 0);

begin

     COMP1: COMP_BIT port map(A(10), B(10),'0', '1','0', GT(10), EQ(10), LT(10));
     COMP2: COMP_BIT port map(A(9), B(9), GT(10), EQ(10), LT(10), GT(9), EQ(9), LT(9));
     COMP3: COMP_BIT port map(A(8), B(8), GT(9), EQ(9), LT(9), GT(8), EQ(8), LT(8));
     COMP4: COMP_BIT port map(A(7), B(7), GT(8), EQ(8), LT(8), GT(7), EQ(7), LT(7));
     COMP5: COMP_BIT port map(A(6), B(6), GT(7), EQ(7), LT(7), GT(6), EQ(6), LT(6));
     COMP6: COMP_BIT port map(A(5), B(5), GT(6), EQ(6), LT(6), GT(5), EQ(5), LT(5));
     COMP7: COMP_BIT port map(A(4), B(4), GT(5), EQ(5), LT(5), GT(4), EQ(4), LT(4));
     COMP8: COMP_BIT port map(A(3), B(3), GT(4), EQ(4), LT(4), GT(3), EQ(3), LT(3));
     COMP9: COMP_BIT port map(A(2), B(2), GT(3), EQ(3), LT(3), GT(2), EQ(2), LT(2));
     COMP10: COMP_BIT port map(A(1), B(1), GT(2), EQ(2), LT(2), GT(1), EQ(1), LT(1));
     COMP11: COMP_BIT port map(A(0), B(0), GT(1), EQ(1), LT(1), GT(0), EQ(0), LT(0));

     AgtB <= GT(0);
     AeqB <= EQ(0);
     AltB <= LT(0);

end CKT;


--======================--
--  REGISTRADOR 11 BITS --
--======================--

ENTITY REG11 IS
   PORT( I: IN BIT_VECTOR(10 DOWNTO 0);
      CLK, CLR, EN: IN BIT;
      O: OUT BIT_VECTOR(10 DOWNTO 0));
END REG11;
```

```
ARCHITECTURE CKT OF REG11 IS

COMPONENT ffd IS
   port ( clk ,D ,P , C : IN BIT ;
   q : OUT BIT );
END COMPONENT;

COMPONENT MUX21 is
   port(A, B, S: in bit;
      O: out bit);
 end COMPONENT;

SIGNAL CLEAR:BIT;
SIGNAL Q,D: BIT_VECTOR(10 DOWNTO 0);

BEGIN

   CLEAR <= NOT CLR;

   MUX1:  MUX21 PORT MAP(Q(0), I(0), EN, D(0));
   MUX2:  MUX21 PORT MAP(Q(1), I(1), EN, D(1));
   MUX3:  MUX21 PORT MAP(Q(2), I(2), EN, D(2));
   MUX4:  MUX21 PORT MAP(Q(3), I(3), EN, D(3));
   MUX5:  MUX21 PORT MAP(Q(4), I(4), EN, D(4));
   MUX6:  MUX21 PORT MAP(Q(5), I(5), EN, D(5));
   MUX7:  MUX21 PORT MAP(Q(6), I(6), EN, D(6));
   MUX8:  MUX21 PORT MAP(Q(7), I(7), EN, D(7));
   MUX9:  MUX21 PORT MAP(Q(8), I(8), EN, D(8));
   MUX10: MUX21 PORT MAP(Q(9), I(9), EN, D(9));
   MUX11: MUX21 PORT MAP(Q(10), I(10), EN, D(10));


   FFD1: ffd PORT MAP (CLK, D(0), '1', CLEAR, Q(0));
   FFD2: ffd PORT MAP (CLK, D(1), '1', CLEAR, Q(1));
   FFD3: ffd PORT MAP (CLK, D(2), '1', CLEAR, Q(2));
   FFD4: ffd PORT MAP (CLK, D(3), '1', CLEAR, Q(3));
   FFD5: ffd PORT MAP (CLK, D(4), '1', CLEAR, Q(4));
   FFD6: ffd PORT MAP (CLK, D(5), '1', CLEAR, Q(5));
   FFD7: ffd PORT MAP (CLK, D(6), '1', CLEAR, Q(6));
   FFD8: ffd PORT MAP (CLK, D(7), '1', CLEAR, Q(7));
   FFD9: ffd PORT MAP (CLK, D(8), '1', CLEAR, Q(8));
   FFD10: ffd PORT MAP (CLK, D(9), '1', CLEAR, Q(9));
   FFD11: ffd PORT MAP (CLK, D(10), '1', CLEAR, Q(10));


   O<= Q;
```

```
END CKT;


--======================--
-- SOMADOR + COMPARADOR --
--======================--

ENTITY SOMACOMP IS
   PORT(A, B, C: IN BIT_VECTOR(10 DOWNTO 0);
      MAIOR, IGUAL, MENOR: OUT BIT);
END SOMACOMP;

ARCHITECTURE CKT OF SOMACOMP IS

COMPONENT ADD11 is
   port(A, B: in bit_vector(10 downto 0);
         O: out bit_vector(10 downto 0);
         CO: out bit);
end COMPONENT;

COMPONENT COMP11 IS
   PORT(A,B: in bit_vector(10 downto 0);
      AgtB, AeqB, AltB: out bit);
END COMPONENT;

SIGNAL RES: BIT_VECTOR(10 DOWNTO 0);
SIGNAL CO: BIT;

BEGIN

ADICAO: ADD11 PORT MAP (A,B, RES,CO);
COMPARACAO: COMP11 PORT MAP(RES, C, MAIOR, IGUAL, MENOR);

END CKT;


--======================--
-- SOMADOR + REGISTRADOR --
--======================--

ENTITY SOMAREG IS
   PORT(B: IN BIT_VECTOR(10 DOWNTO 0);
      CLK, CLR, EN: IN BIT;
      AC: OUT BIT_VECTOR(10 DOWNTO 0));
END SOMAREG;

ARCHITECTURE CKT OF SOMAREG IS
```

```
COMPONENT ADD11 is
   port(A, B: in bit_vector(10 downto 0);
        O: out bit_vector(10 downto 0);
        CO: out bit);
end COMPONENT;

COMPONENT REG11 IS
   PORT( I: IN BIT_VECTOR(10 DOWNTO 0);
      CLK, CLR, EN: IN BIT;
      O: OUT BIT_VECTOR(10 DOWNTO 0));
END COMPONENT;

SIGNAL RELOAD, RES: BIT_VECTOR(10 DOWNTO 0);
SIGNAL CO: BIT;

BEGIN

ADICAO: ADD11 PORT MAP (RELOAD, B, RES, CO);
REGISTRA: REG11 PORT MAP (RES, CLK, CLR, EN, RELOAD);
AC <= RELOAD(10 DOWNTO 0);

END CKT;


--============--
-- CONDICOES --
--============--

ENTITY CONDICOES IS
   port(SOMA100, SOMA50, SOMA25, SOMA10, SOMA5, SOMA1, TC100, TC50, TC25,
TC10, TC5, TC1: IN BIT;
      CONDICAO0, CONDICAO1, CONDICAO2, CONDICAO3, CONDICAO4, CONDICAO5,
CONDICAO6, COND0, COND1, COND2: OUT BIT);
END CONDICOES;

ARCHITECTURE CKT OF CONDICOES IS

SIGNAL CONDIC0,CONDIC1,CONDIC2,CONDIC3,CONDIC4,CONDIC5,CONDIC6: BIT;

BEGIN

CONDIC0 <= NOT SOMA100 AND TC100;
CONDIC1 <= (SOMA100 OR NOT TC100) AND (NOT SOMA50 AND TC50);
CONDIC2 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (NOT
SOMA25 AND TC25);
CONDIC3 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (SOMA25
OR NOT TC25) AND (NOT SOMA10 AND TC10);
```

```
CONDIC4 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT TC50) AND (SOMA25
OR NOT TC25) AND (SOMA10 OR NOT TC10) AND (NOT SOMA5  AND TC5);
CONDIC5 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT  TC50) AND (SOMA25
OR NOT TC25) AND (SOMA10 OR NOT TC10) AND (SOMA5 OR  NOT TC5) AND (NOT
SOMA1 AND TC1);
CONDIC6 <= (SOMA100 OR NOT TC100) AND (SOMA50 OR NOT  TC50) AND (SOMA25
OR NOT TC25) AND (SOMA10 OR NOT TC10) AND (SOMA5 OR  NOT TC5) AND (SOMA1
OR NOT TC1);

CONDICAO0 <= CONDIC0 ;
CONDICAO1 <= CONDIC1 ;
CONDICAO2 <= CONDIC2 ;
CONDICAO3 <= CONDIC3 ;
CONDICAO4 <= CONDIC4 ;
CONDICAO5 <= CONDIC5 ;
CONDICAO6 <= CONDIC6 ;

COND0 <= CONDIC1 OR CONDIC3 OR CONDIC5;
COND1 <= CONDIC2 OR CONDIC3 OR CONDIC6;
COND2 <= CONDIC4 OR CONDIC5 OR CONDIC6;

END CKT;


--====================--
-- BLOCO OPERACIONAL --
--====================--

ENTITY OP_BLOCK IS
   PORT(VALOR: IN BIT_VECTOR(10 DOWNTO 0);
      BANCO: IN BIT_VECTOR(5 DOWNTO 0);
      CLK, CLRT, CLRAC, LDT, PISCA: IN BIT;
      LIBERA100,LIBERA50,LIBERA25,LIBERA10,LIBERA5,LIBERA1: OUT
BIT_VECTOR(10 DOWNTO 0);
      TV, TT, NTT, L: OUT BIT);
END OP_BLOCK;

ARCHITECTURE CKT OF OP_BLOCK IS

COMPONENT MUX21 is
   port(A, B, S: in bit;
     O: out bit);
end COMPONENT;

COMPONENT MUX8X1_11INPUTS IS
   PORT( I0, I1, I2, I3, I4, I5, I6, I7: IN BIT_VECTOR(10 DOWNTO 0);
      SEL: IN BIT_VECTOR(2 DOWNTO 0 );
      O: OUT BIT_VECTOR(10 DOWNTO 0));
```

```
END COMPONENT;

COMPONENT ffjk IS
port (clk ,J,K,P,C: IN BIT;
q: OUT BIT );
END COMPONENT ;

COMPONENT REG11 IS
   PORT( I: IN BIT_VECTOR(10 DOWNTO 0);
      CLK, CLR, EN: IN BIT;
      O: OUT BIT_VECTOR(10 DOWNTO 0));
END COMPONENT;

COMPONENT COMP11 is
   port(A,B: in bit_vector(10 downto 0);
         AgtB, AeqB, AltB: out bit);
end COMPONENT;

COMPONENT SOMACOMP IS
   PORT(A, B, C: IN BIT_VECTOR(10 DOWNTO 0);
      MAIOR, IGUAL, MENOR: OUT BIT);
END COMPONENT;

COMPONENT SOMAREG IS
   PORT(B: IN BIT_VECTOR(10 DOWNTO 0);
      CLK, CLR, EN: IN BIT;
      AC: OUT BIT_VECTOR(10 DOWNTO 0));
END COMPONENT;

COMPONENT CONDICOES IS
   port(SOMA100, SOMA50, SOMA25, SOMA10, SOMA5, SOMA1, TC100, TC50, TC25,
TC10, TC5, TC1: IN BIT;
      CONDICAO0, CONDICAO1, CONDICAO2, CONDICAO3, CONDICAO4, CONDICAO5,
CONDICAO6, COND0, COND1, COND2: OUT BIT);
END COMPONENT;


SIGNAL V11BITS, INCREMENTADOR, MUXS, C100, C50, C25, C10, C5, C1, ZERO, ONE:
BIT_VECTOR(10 DOWNTO 0);
SIGNAL SELCOND: BIT_VECTOR (2 DOWNTO 0);
SIGNAL SOMA100, SOMA50, SOMA25, SOMA10, SOMA5, SOMA1, TC100, TC50, TC25,
TC10, TC5, TC1: BIT;
SIGNAL CONDICAO0, CONDICAO1, CONDICAO2, CONDICAO3, CONDICAO4,
CONDICAO5, CONDICAO6, COND0, COND1, COND2: BIT;
SIGNAL MAIOR, IGUAL, MENOR: BIT_VECTOR(5 DOWNTO 0);
SIGNAL MAIOR2, MENOR2: BIT_VECTOR (1 DOWNTO 0);
SIGNAL STT,QLED:BIT;
```

```
BEGIN

C100 <= "00001100100";
C50 <=  "00000110010";
C25 <=  "00000011001";
C10 <=  "00000001010";
C5 <=   "00000000101";
C1 <=   "00000000001";
ZERO <= "00000000000";
ONE <=  "00000000001";

SELCOND(2) <= COND2;
SELCOND(1) <= COND1;
SELCOND(0) <= COND0;

TC100 <= BANCO(5);
TC50  <= BANCO(4);
TC25  <= BANCO(3);
TC10  <= BANCO(2);
TC5   <= BANCO(1);
TC1   <= BANCO(0);

REGVAL: REG11 PORT MAP(VALOR, CLK, CLRT, LDT, V11BITS);
MUX81: MUX8X1_11INPUTS PORT MAP(C100, C50, C25, C10, C5, C1, ZERO, ZERO,
SELCOND, MUXS);
INCREMENT: SOMAREG PORT MAP(MUXS, CLK, CLRAC, '1', INCREMENTADOR);
ADDCOMP1: SOMACOMP PORT MAP(INCREMENTADOR, C100,V11BITS, SOMA100,
IGUAL(5), MENOR(5));
ADDCOMP2: SOMACOMP PORT MAP(INCREMENTADOR, C50, V11BITS,  SOMA50,
IGUAL(4), MENOR(4));
ADDCOMP3: SOMACOMP PORT MAP(INCREMENTADOR, C25, V11BITS,  SOMA25,
IGUAL(3), MENOR(3));
ADDCOMP4: SOMACOMP PORT MAP(INCREMENTADOR, C10, V11BITS,  SOMA10,
IGUAL(2), MENOR(2));
ADDCOMP5: SOMACOMP PORT MAP(INCREMENTADOR, C5, V11BITS,   SOMA5 ,
IGUAL(1), MENOR(1));
ADDCOMP6: SOMACOMP PORT MAP(INCREMENTADOR, C1, V11BITS,   SOMA1 ,
IGUAL(0), MENOR(0));
COND: CONDICOES PORT MAP(SOMA100, SOMA50, SOMA25, SOMA10, SOMA5,
SOMA1, TC100, TC50, TC25, TC10, TC5, TC1, CONDICAO0,  CONDICAO1, CONDICAO2,
CONDICAO3, CONDICAO4, CONDICAO5, CONDICAO6, COND0, COND1, COND2);
COMPARACAO1: COMP11 PORT MAP(ZERO, V11BITS, MAIOR2(1), TV, MENOR2(1) );
COMPARACAO2: COMP11 PORT MAP(INCREMENTADOR, V11BITS, MAIOR2(0), STT,
MENOR2(0));

NTT <= (NOT COND0 AND COND1 AND COND2) AND (NOT STT);
TT <= STT;
```

```
ACUMULADOR1: SOMAREG PORT MAP (ONE, CLK, CLRAC, CONDICAO0, LIBERA100);
ACUMULADOR2: SOMAREG PORT MAP (ONE, CLK, CLRAC, CONDICAO1, LIBERA50);
ACUMULADOR3: SOMAREG PORT MAP (ONE, CLK, CLRAC, CONDICAO2, LIBERA25);
ACUMULADOR4: SOMAREG PORT MAP (ONE, CLK, CLRAC, CONDICAO3, LIBERA10);
ACUMULADOR5: SOMAREG PORT MAP (ONE, CLK, CLRAC, CONDICAO4, LIBERA5);
ACUMULADOR6: SOMAREG PORT MAP (ONE, CLK, CLRAC, CONDICAO5, LIBERA1);

FFLED: ffjk PORT MAP(CLK, PISCA, PISCA, '1', '1', QLED);
MUXLED: MUX21 PORT MAP('1', QLED, PISCA, L);

END CKT;


--================================--
-- LÃ“GICA COMBINACIONAL BOTÃƒO BS --
--================================--

ENTITY LOGIC_BS IS
   PORT(Q1, Q0, T: IN BIT;
      D1, D0, PRESS: OUT BIT);
END LOGIC_BS;

ARCHITECTURE CKT OF LOGIC_BS IS

BEGIN

D1 <= (NOT Q1 AND Q0) OR (Q1 AND NOT Q0 AND T);
D0 <= NOT Q1 AND NOT Q0 AND T;
PRESS <= NOT Q1 AND Q0;

END CKT;


--==========--
-- BOTÃƒO BS --
--==========--

ENTITY BS IS
   PORT(CLK, T: IN BIT;
      PRESS: OUT BIT);
END BS;

ARCHITECTURE CKT OF BS IS

COMPONENT ffd is
   PORT ( clk ,D ,P , C : IN BIT ;
                  q : OUT BIT );
END COMPONENT;
```

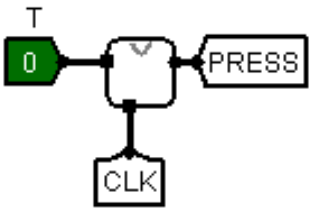
```
COMPONENT LOGIC_BS IS
   PORT(Q1, Q0, T: IN BIT;
      D1, D0, PRESS: OUT BIT);
END COMPONENT;

SIGNAL Q1,Q0,D1,D0:BIT;

BEGIN

LOGIC: LOGIC_BS PORT MAP (Q1, Q0, T, D1, D0, PRESS);
FFD1: ffd PORT MAP(CLK, D1, '1', '1', Q1);
FFD2: ffd PORT MAP(CLK, D0, '1', '1', Q0);

END CKT;


--=======================================--
-- LOGICA COMBINACIONAL BLOCO DE CONTROLE --
--=======================================--

ENTITY LOGIC_COMB IS
   PORT(Q2,Q1,Q0, PRESS, TV, TT, NTT: IN BIT;
      D2, D1, D0, CLRT, CLRAC, LDT, PIS: OUT BIT);
END LOGIC_COMB;

ARCHITECTURE CKT OF LOGIC_COMB IS

BEGIN

D2 <= NOT Q2 AND Q1 AND NOT Q0 AND TT AND NOT NTT;
D1 <= (NOT Q2 AND NOT Q1 AND Q0) OR (NOT Q2 AND Q1  AND NOT Q0 AND NOT TT);
D0 <= (NOT Q2 AND NOT Q1 AND NOT Q0 AND PRESS AND  TV) OR (NOT Q2 AND Q1
AND NOT Q0 AND NOT TT AND NTT);
CLRT <= (NOT Q2 AND Q1 AND Q0) OR (Q2 AND NOT Q1 AND  NOT Q0);
CLRAC <= (NOT Q2 AND Q0);
LDT <= (NOT Q2 AND NOT Q1 AND Q0);
PIS <= (NOT Q2 AND NOT Q1 AND Q0) OR (NOT Q2 AND Q1  AND NOT Q0 AND NOT TT)
OR (NOT Q2 AND Q1 AND NOT Q0 AND NOT NTT);

END CKT;


--====================--
-- BLOCO DE CONTROLE --
--====================--

ENTITY CONTROL_BLOCK IS
```

```
   PORT(CLK, T, TV, TT, NTT: IN BIT;
          CLRT, CLRAC, LDT, PISCA: OUT BIT);
END CONTROL_BLOCK;

ARCHITECTURE CKT OF CONTROL_BLOCK IS

COMPONENT ffd IS
   port ( clk ,D ,P , C : IN BIT ;
   q : OUT BIT );
END COMPONENT;

COMPONENT LOGIC_COMB IS
   PORT(Q2,Q1,Q0, PRESS, TV, TT, NTT: IN BIT;
      D2, D1, D0, CLRT, CLRAC, LDT, PIS: OUT BIT);
END COMPONENT;

COMPONENT BS IS
   PORT(CLK, T: IN BIT;
      PRESS: OUT BIT);
END COMPONENT;


SIGNAL Q2,Q1,Q0,D2,D1,D0,PRESS: BIT;

BEGIN

BOTTON: BS PORT MAP(CLK, T, PRESS);
LOGIC: LOGIC_COMB PORT MAP (Q2, Q1, Q0, PRESS, TV, TT, NTT, D2, D1, D0, CLRT,
CLRAC, LDT, PISCA);
FFD1: ffd PORT MAP(CLK, D2, '1', '1', Q2);
FFD2: ffd PORT MAP(CLK, D1, '1', '1', Q1);
FFD3: ffd PORT MAP(CLK, D0, '1', '1', Q0);

END CKT;


--==================--
-- MÃ  QUINA DE TROCO --
--==================--

ENTITY maqtroco IS
         PORT(VALOR:IN BIT_VECTOR(10 DOWNTO 0);
         BANCO: IN BIT_VECTOR(5 DOWNTO 0);
   CLK, T: IN BIT;
   C100, C50, C25, C10, C5, C1: OUT BIT_VECTOR(10 DOWNTO 0);
   L: OUT BIT);
END maqtroco;
```

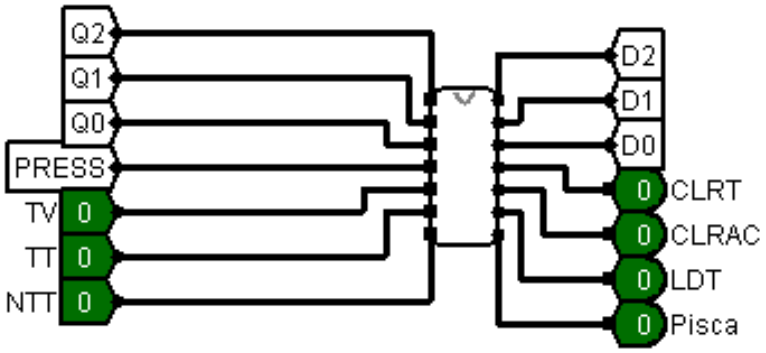
```
ARCHITECTURE CKT OF maqtroco IS

COMPONENT CONTROL_BLOCK IS
   PORT(CLK, T, TV, TT, NTT: IN BIT;
          CLRT, CLRAC, LDT, PISCA: OUT BIT);
END COMPONENT;

COMPONENT OP_BLOCK IS
   PORT(VALOR: IN BIT_VECTOR(10 DOWNTO 0);
      BANCO: IN BIT_VECTOR(5 DOWNTO 0);
      CLK, CLRT, CLRAC, LDT, PISCA: IN BIT;
      LIBERA100,LIBERA50,LIBERA25,LIBERA10,LIBERA5,LIBERA1: OUT
BIT_VECTOR(10 DOWNTO 0);
      TV, TT, NTT, L: OUT BIT);
END COMPONENT;

SIGNAL CLRT, CLRAC, LDT, PISCA, TV, TT, NTT: BIT;

BEGIN

CBLOCK: CONTROL_BLOCK PORT MAP(CLK, T, TV, TT, NTT, CLRT, CLRAC, LDT,
PISCA);
OBLOCK: OP_BLOCK PORT MAP(VALOR, BANCO, CLK, CLRT, CLRAC, LDT, PISCA,
C100, C50, C25, C10, C5, C1, TV, TT, NTT, L);

END CKT;
```

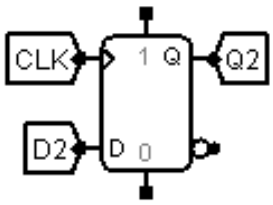
ANEXO E - IMPLEMENTAÇÃO DOS CIRCUITOS

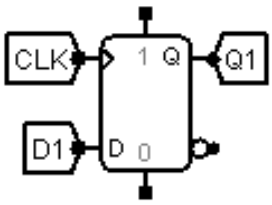
BOTAO BS

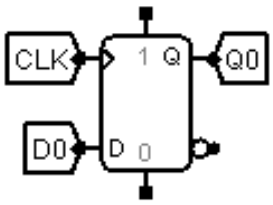


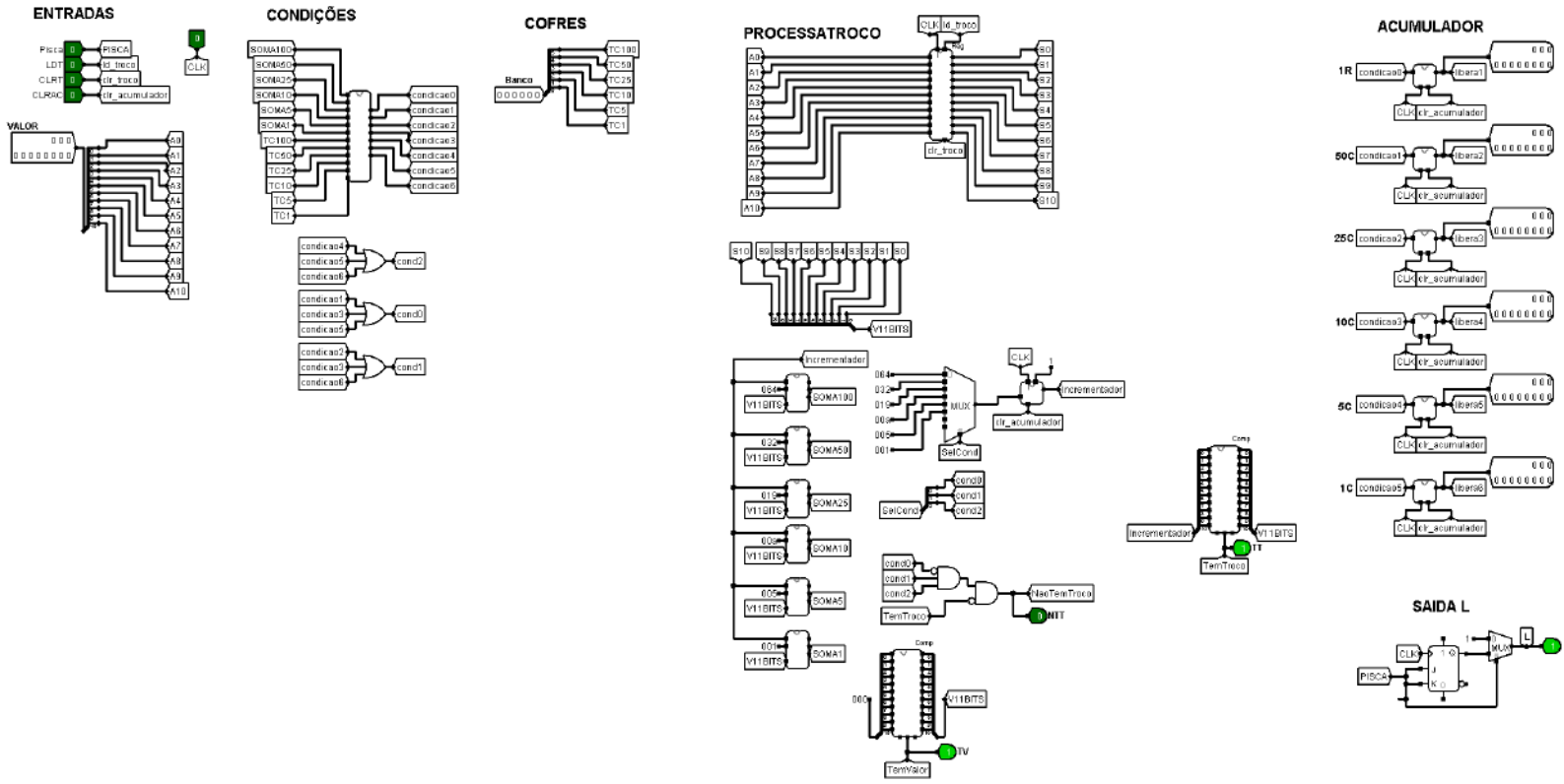
MAQUINA DE ESTADOS

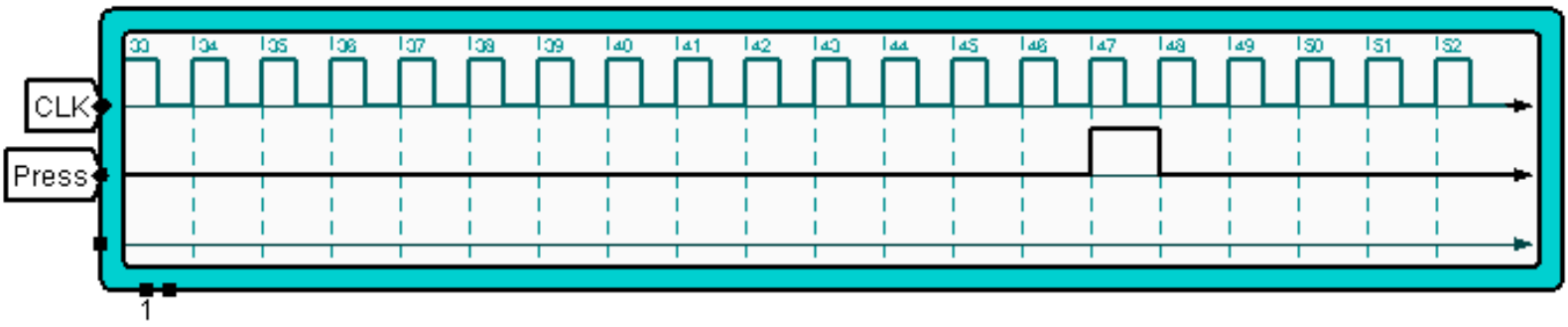




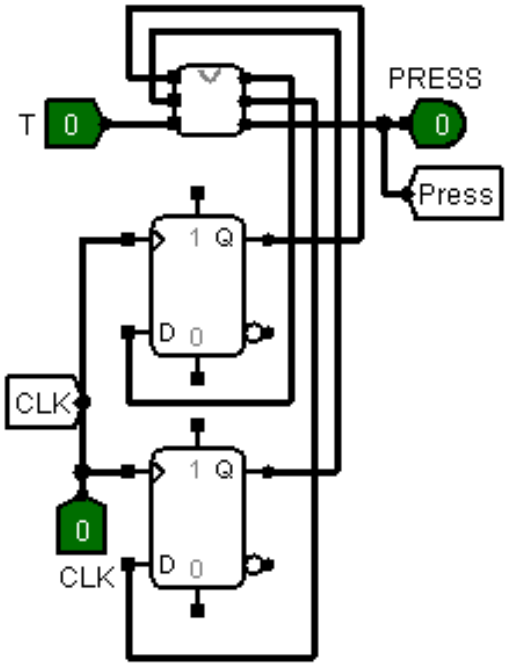


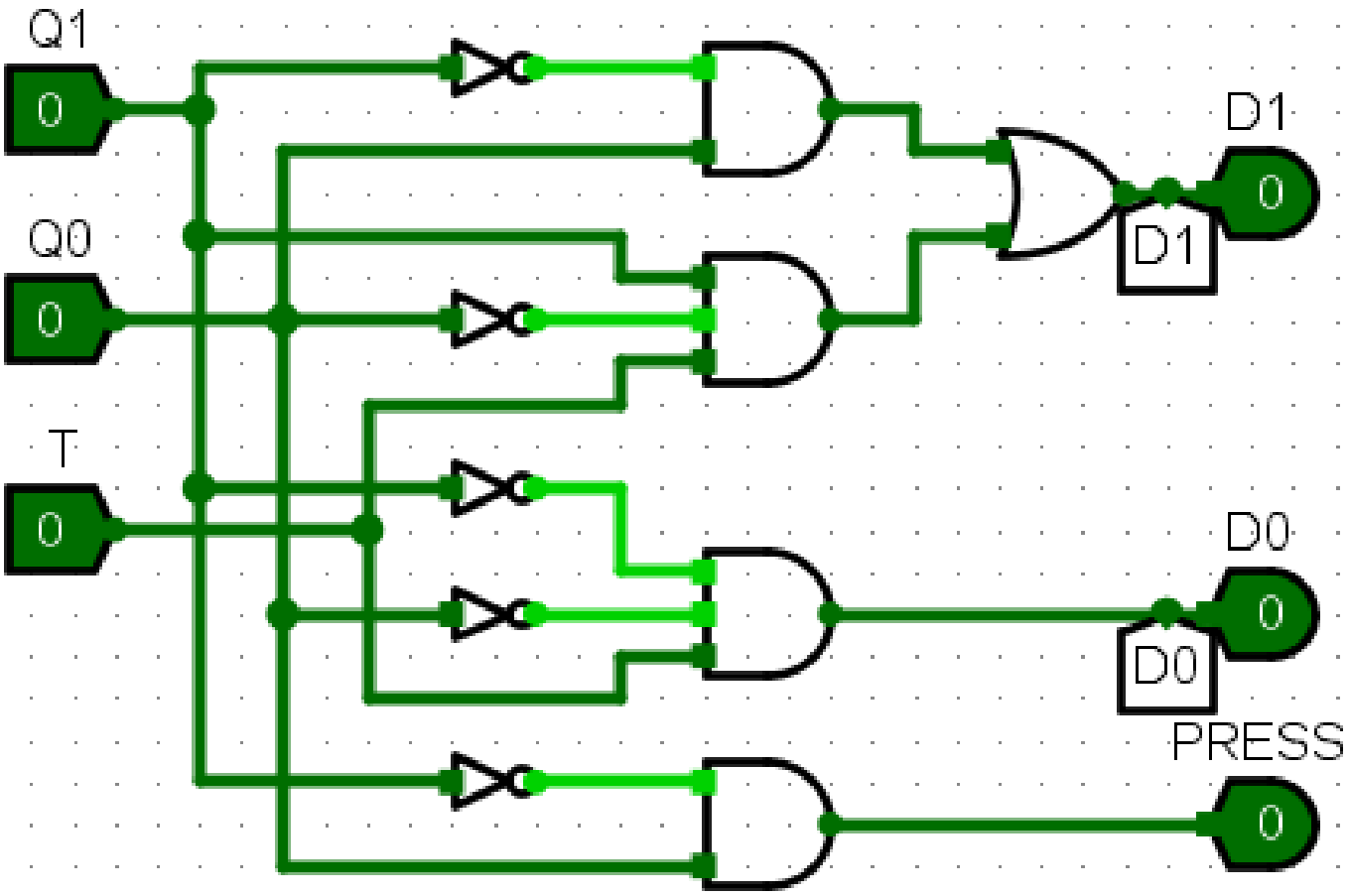


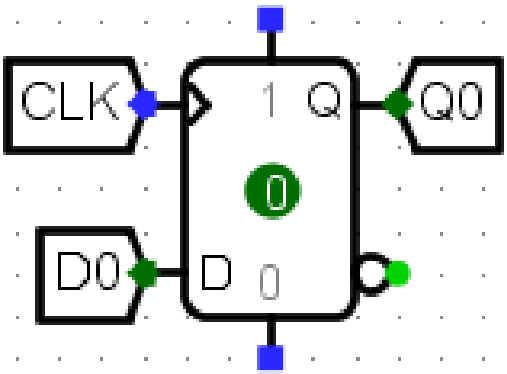

ENTRADAS
Pisca
LDT
CLRT
CLRAC
PISCA
ld_troco
clr_troco
clr_acumulador
CLK
VALOR
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
CONDIÇÕES
SOMA100
SOMA50
SOMA25
SOMA10
SOMA5
SOMA1
TC100
TC50
TC25
TC10
TC5
TC1
condicao0
condicao1
condicao2
condicao3
condicao4
condicao5
condicao6
cond2
cond0
cond1
COFRES
Banco
PROCESSATROCO
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
S10
V11BITS
Incrementador
MUX
SelCond
TemTroco
NaoTemTroco
INTT
Comp
TV
TemValor
ACUMULADOR
1R
50C
25C
10C
5C
1C
libera1
libera2
libera3
libera4
libera5
libera6
clr_acumulador
SAIDA L
L
MUX

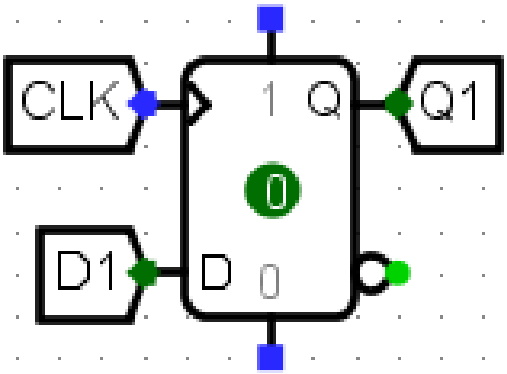


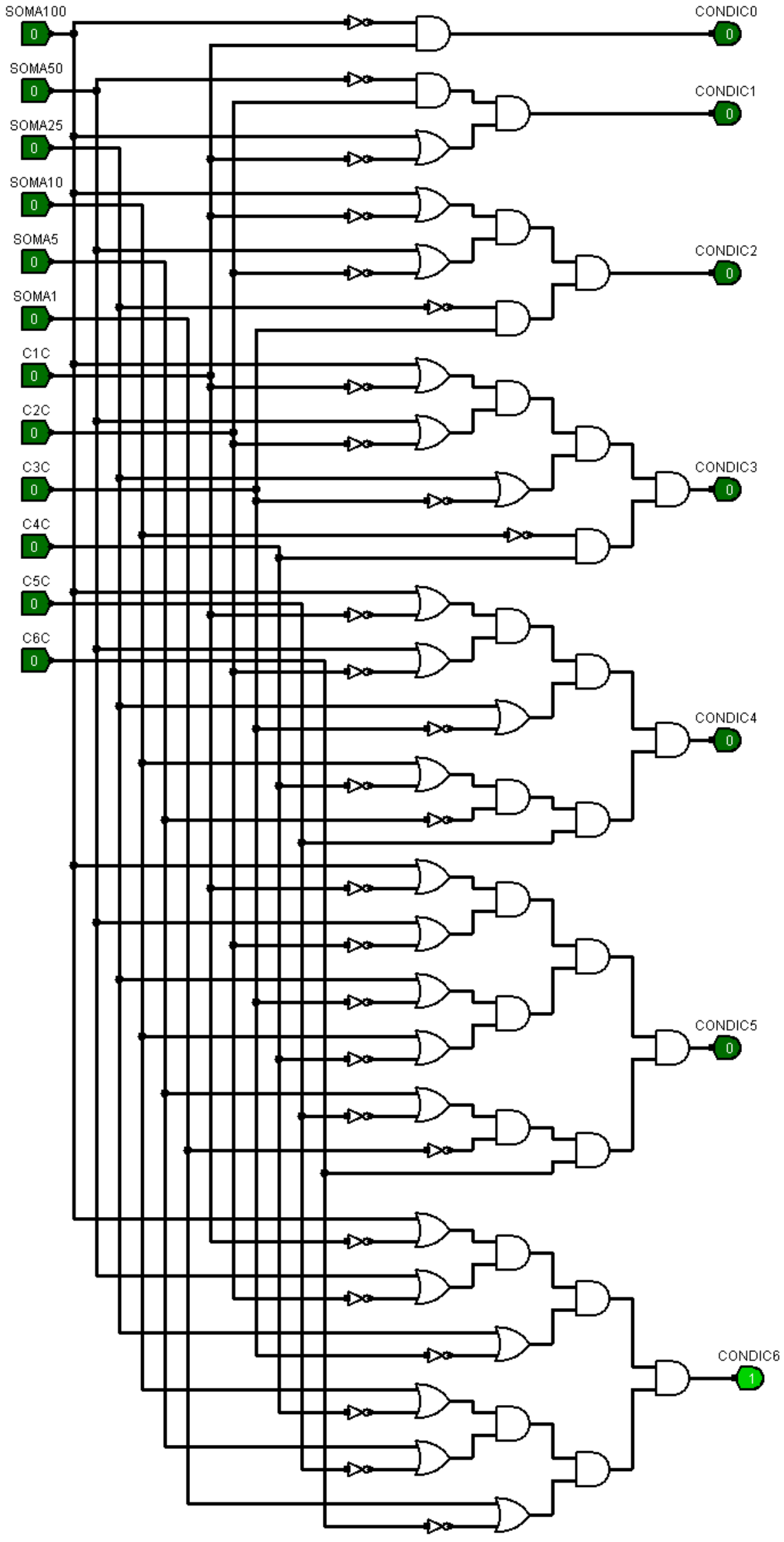
BOTAO BS



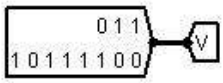

Q1
0
Q0
0
T
0
D1
0
D1
D0
0
D0
PRESS
0


CLK
1
Q
Q0
0
D0
D
0

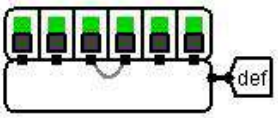

CLK
1
Q
Q1
0
D1
D
0

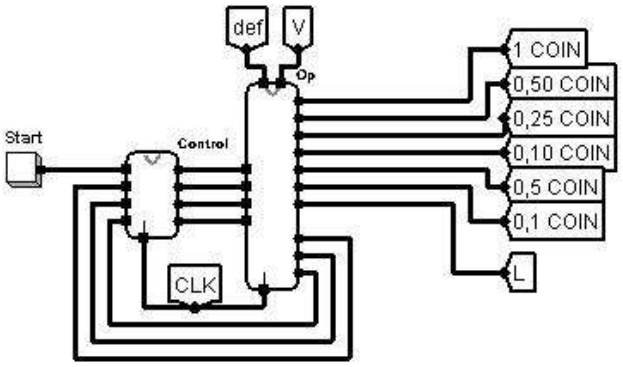

SOMA100
0
SOMA50
0
SOMA25
0
SOMA10
0
SOMA5
0
SOMA1
0
C1C
0
C2C
0
C3C
0
C4C
0
C5C
0
C6C
0
CONDIC0
0
CONDIC1
0
CONDIC2
0
CONDIC3
0
CONDIC4
0
CONDIC5
0
CONDIC6
1

INPUT:

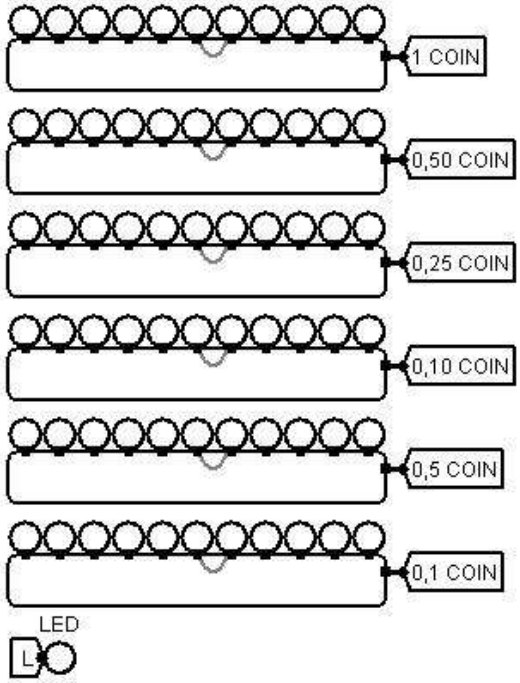


Definiçao de Moedas:

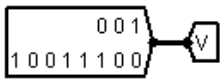




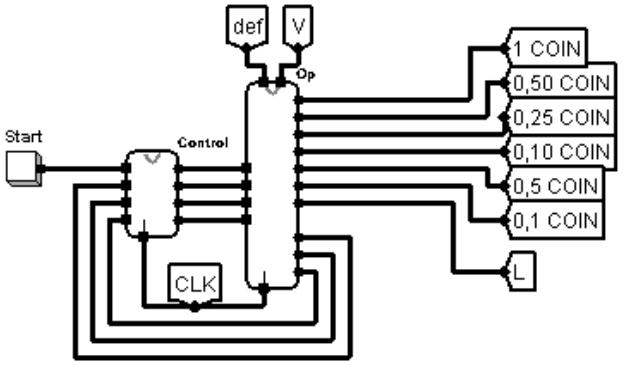
OUTPUT:



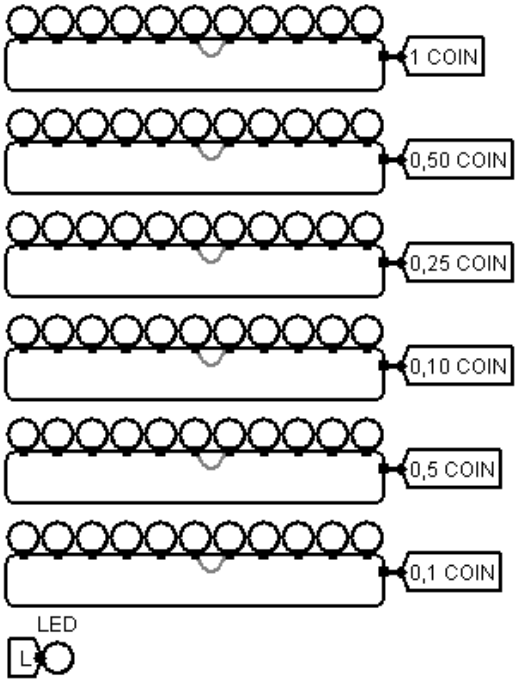
INPUT:

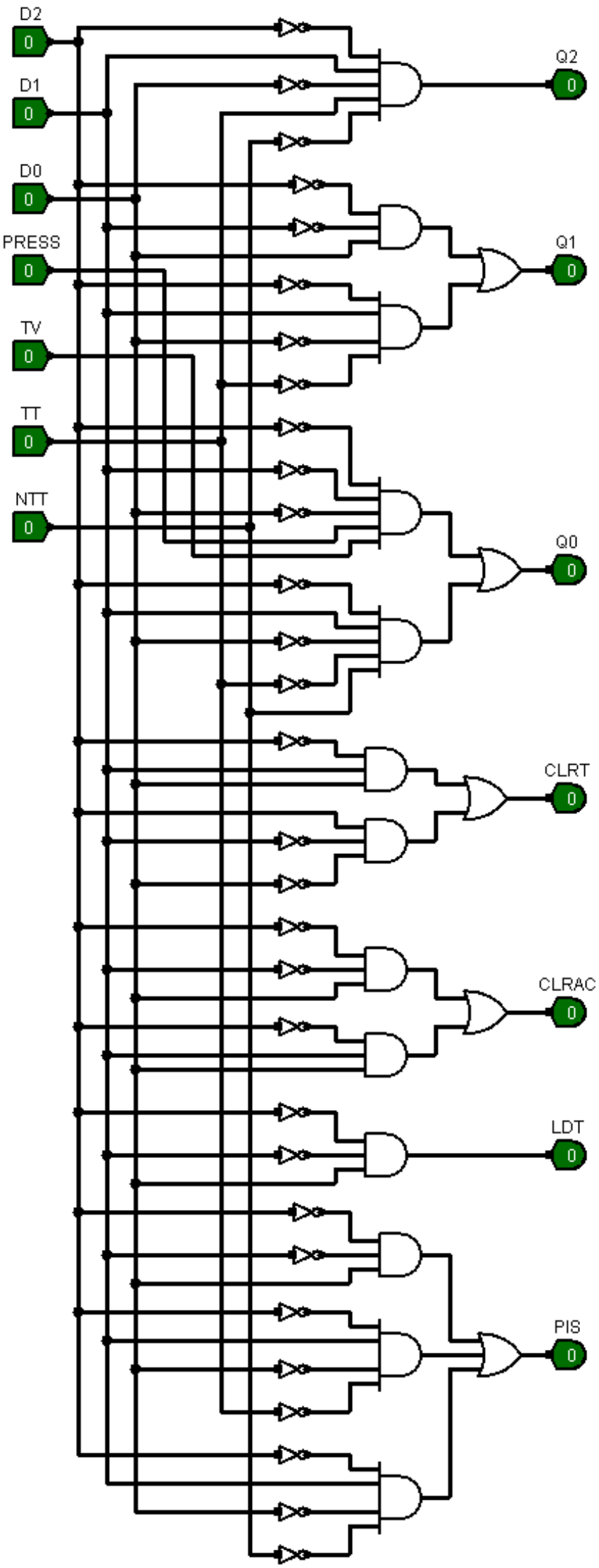


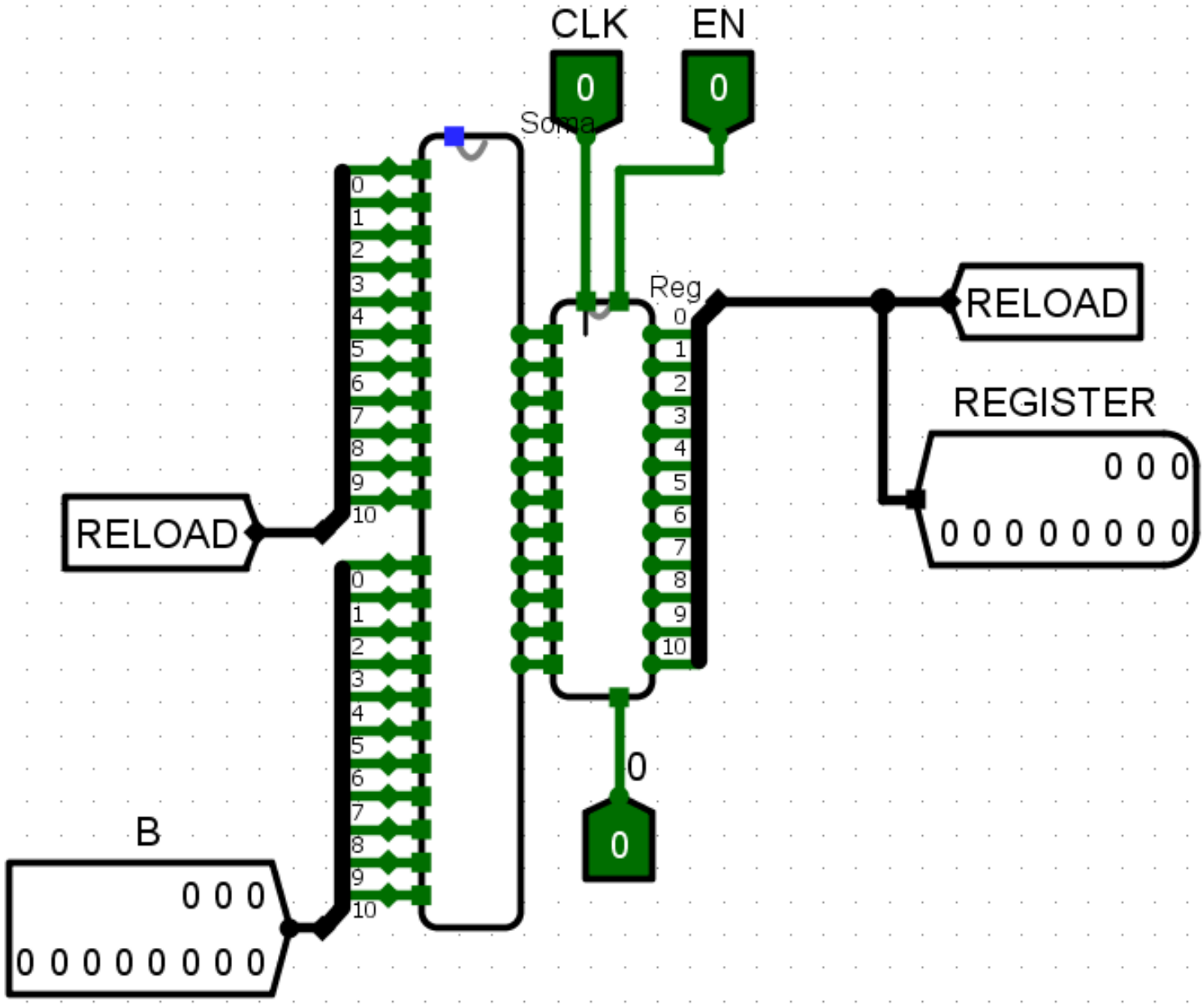
Definiçao de Moedas:



OUTPUT:




D2
0
D1
0
D0
0
PRESS
0
TV
0
TT
0
NTT
0
Q2
0
Q1
0
Q0
0
CLRT
0
CLRAC
0
LDT
0
PIS
0


CLK
EN
0
0
Soma
Reg
RELOAD
REGISTER
0 0 0
0 0 0 0 0 0 0 0
RELOAD
B
0 0 0
0 0 0 0 0 0 0 0
0
0
0 1 2 3 4 5 6 7 8 9 10