Departamento de Engenharia Eletrotécnica e de Computadores

CSD

2023/2024 – 1º Semestre

Relatório Trabalho 1 - Relógio de Parede

Identificação

Turno: P1

Data: 18 / 11 /2023

Aluno: Guilherme Concha	Número: 60143
Aluno: Pedro Peres	Número: 60495
Aluno: António Alves	Número: 58339

Índice

Solução Implementada

Este trabalho tem como objetivo a implementação de um relógio de parede através da ferramenta Xilinx ISE e com um FPGA.

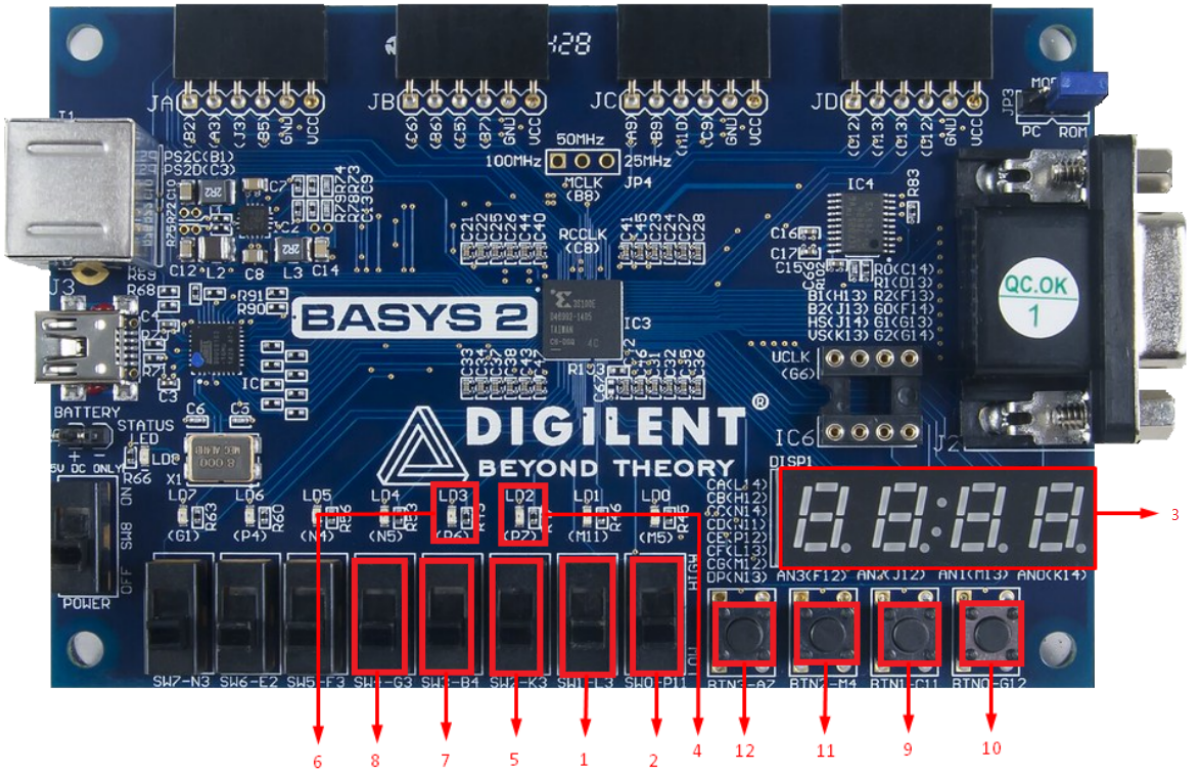
A solução implementada para este problema foi um ***Relógio*** com a funcionalidade de podermos acertar horas, minutos e segundos. Existe um **interruptor de edição**[1] que faz parar o relógio para o fazermos. Para essa edição existe um **botão**[12] para selecionarmos Horas, Minutos ou Segundos e outro **botão**[11] para incrementar o que selecionamos através do botão mencionado anteriormente. Existe também um **interruptor**[2] para mostrar no **Display**[3] do FPGA HH:MM e MM:SS.

Tem uma funcionalidade de ***Alarme*** que acende um **LED**[4] quando está a “tocar”. Possui um **interruptor**[5] de edição do alarme como o do relógio.

Tem também uma funcionalidade de ***Temporizador*** que possui um **LED**[6] quando está **ativo**. Possui um **interruptor**[7] que quando está ativo podemos meter o tempo que queremos que seja contado, e quando for desativado o tempo começa a contar até chegar a **zero**.

Por fim tem uma funcionalidade de ***Cronómetro***, sendo o mais simples possível com um **interruptor**[8] que quando está ativo o tempo encontra-se parado e quando está desativado o tempo está a contar. Possui um **botão de reset**[8] para este voltar a zero.

Para a Seleção de modo temos um **botão**[9], e para reset global temos um **botão**[10] também.

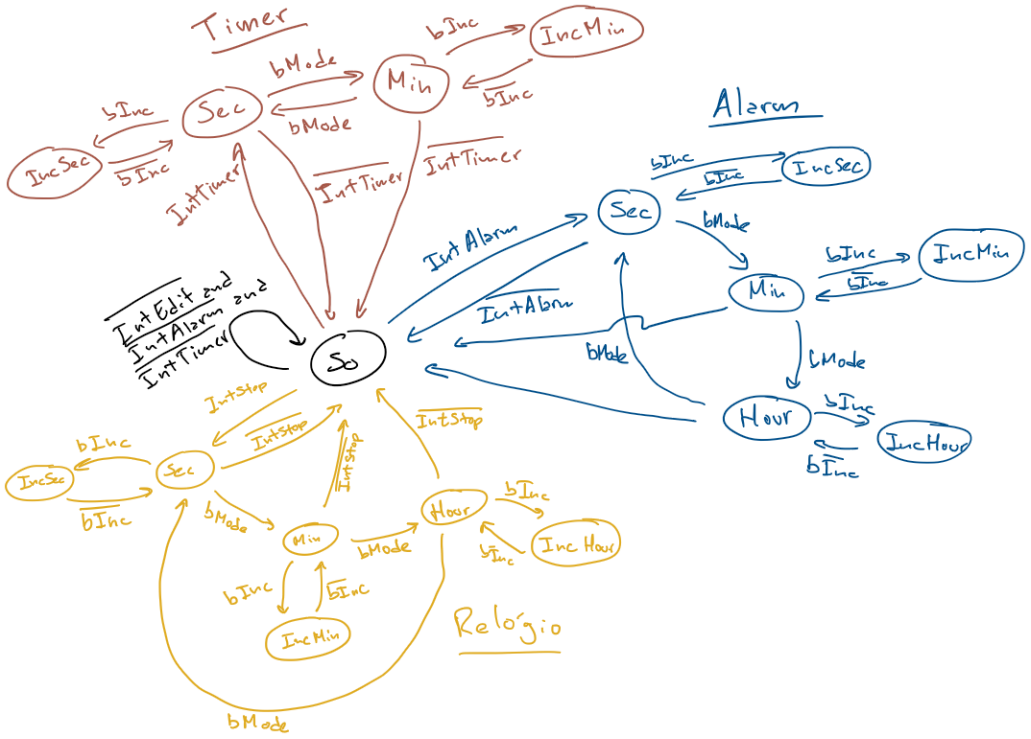


Análise da solução

Para a realização da nossa solução necessitamos de desenvolver duas máquinas de estados, para além da programação dos componentes Relógio, Alarme, Temporizador e Cronómetro.

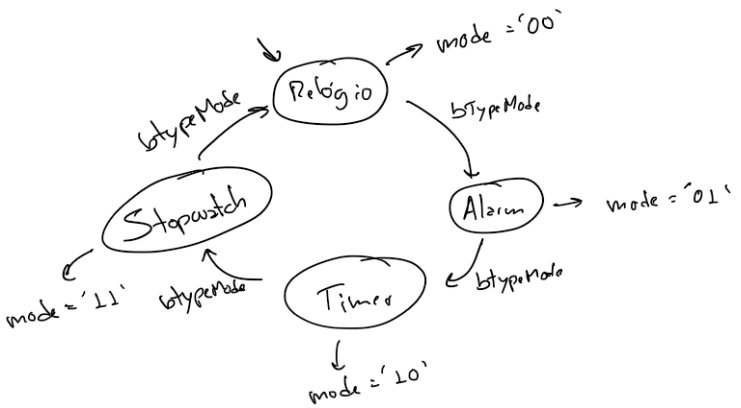
1. Máquina de Estados de Edição

Esta máquina de estados permite-nos enviar para cada um dos componentes um signal para ser incrementado o valor de Segundos / Minutos / Horas.



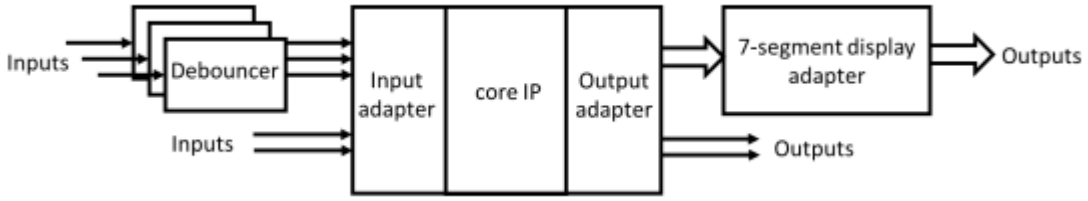
2. Máquina de Estados de Modo

Esta máquina de estados permite ao Display saber o que tem que mostrar ao Utilizador. Através do buttonTypeMode é alternado o estado Relógio, Alarme, Temporizador e Cronómetro, sendo que em cada estado uma variável tem um valor diferente correspondente a cada um dos modos.



Implementação

O projeto é composto por um core IP, um Debouncer e um Display.



No **core IP**, ou “principal” (nome atribuído no projeto) é responsável por tudo o que aparece no Display, sendo preciso o processamento dividido por vários componentes:

- Relógio*;
- Alarm*;
- Timer*;
- StopWatch*;
- Conversor do Display*;
- Máquina de Estados do Modo*;
- Máquina de Estados da Edição*.

O **Debouncer*** tem o objetivo de retirar o “lag” dos botões do FPGA, ou seja, quando clicamos, apenas ser contado o click durante um clock.

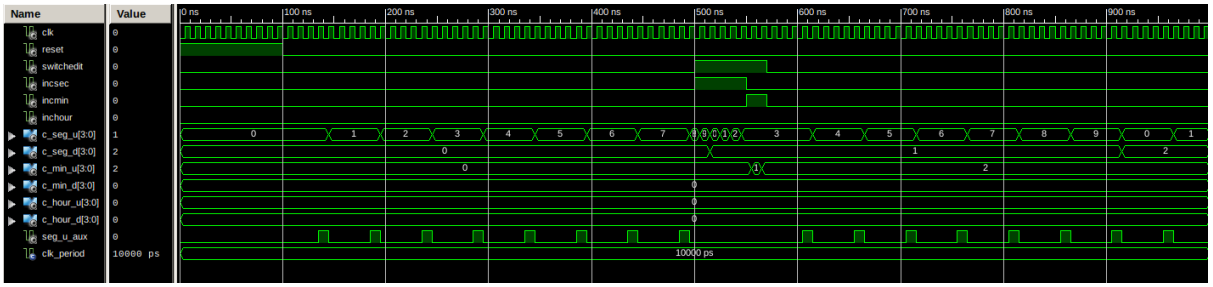
No **Display** aparece o que o User quer, sendo o relógio, alarme, temporizador ou cronómetro.

*Os códigos de cada um dos componentes encontra-se em anexo.

```
entity principal is
    Port ( reset : in std_logic;
           clk : in std_logic;
           algarismo : out std_logic_vector(7 downto 0);
           switch1 : in std_logic;
           switchEDIT : in std_logic;
           switchTIMER : in std_logic;
           switchALARM : in std_logic;
           switchStopWatch : IN std_logic;
           buttonMODE : in std_logic;
           buttonTYPEMODE : in std_logic;
           buttonINC : in std_logic;
           LED_Alarm : out std_logic;
           LED_Timer : out std_logic;
           an3 : out std_logic;
           an2 : out std_logic;
           an1 : out std_logic;
           an0 : out std_logic);
end principal;
```

Resultados

1. Teste do Relógio

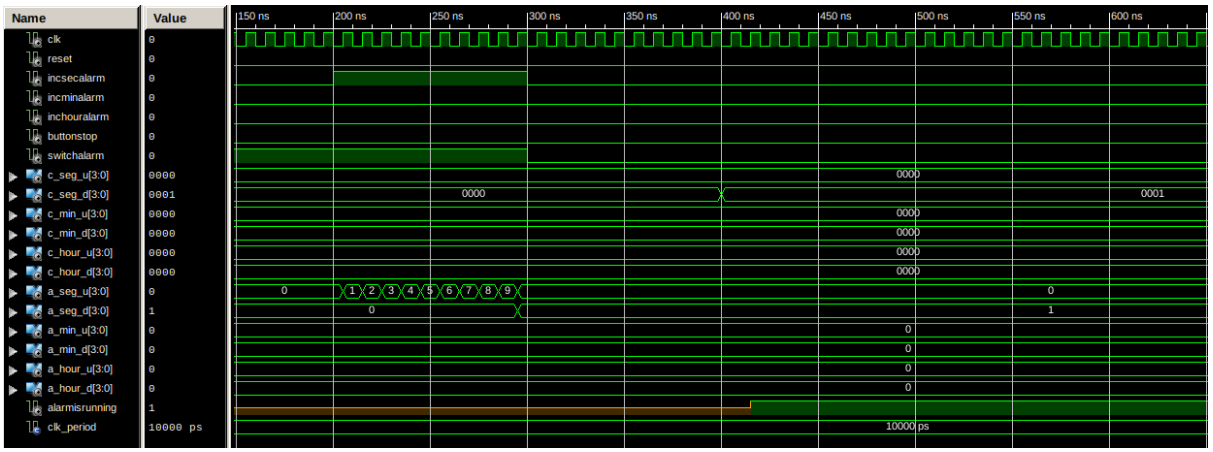


Através desta simulação podemos ver o comportamento do relógio. Temos a variável seg_u_aux que nos diz se o número de clocks que passou representa 1 segundo para o contador dos segundos incrementar.

Em 500 ns na simulação, liguei o interruptor de edição, o relógio parou como era suposto, e incrementei 5 vezes o número de segundos e 2 vezes o número de minutos.

Após ter desligado o interruptor de edição ele continuou a contar a partir do tempo que eu inseri.

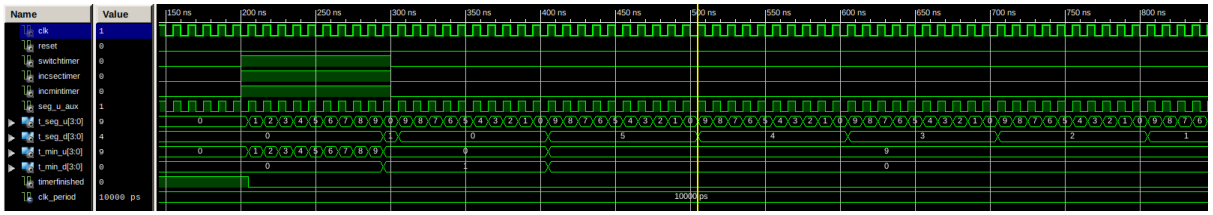
2. Teste do Alarme



Através desta simulação podemos ver o comportamento do alarme. Começando por ativar o interruptor de edição (switchAlarm), incremento os segundos 10 vezes.

Em 400 ns na simulação, coloquei o tempo do relógio igual ao tempo do alarme e podemos ver que a variável “alarmIsRunning” ficou a 1.

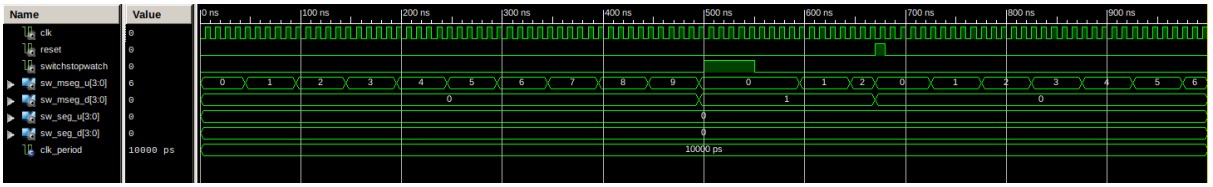
3. Teste do Temporizador



Através desta simulação podemos ver o comportamento do temporizador. Começo por ativar o interruptor de edição (switchTimer), para incrementar os Minutos e os Segundos 10 vezes, obtendo 10:10.

Em 300 ns na simulação desativo o interruptor e o tempo começa a decrementar como podemos ver.

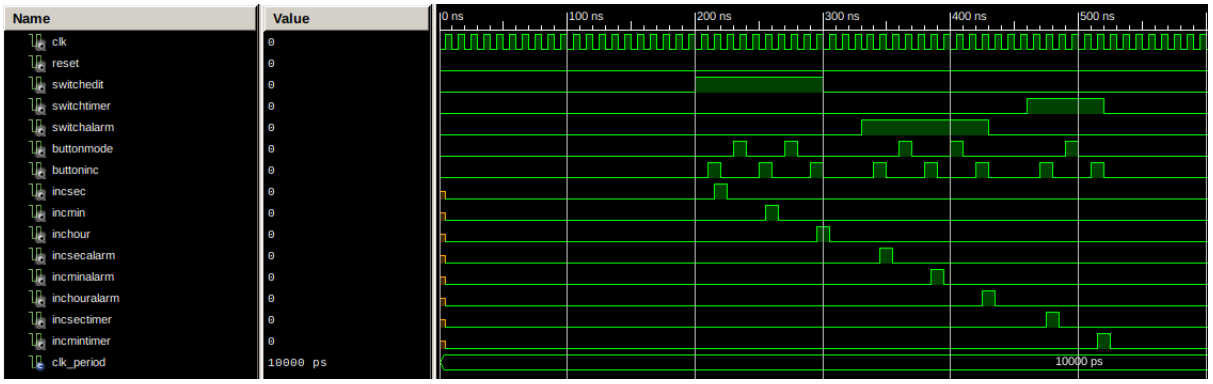
4. Teste do Cronómetro



Através desta simulação podemos ver o comportamento do cronómetro. O tempo começa logo a contar, e em 500 ns quando ativo o interruptor o mesmo para de incrementar.

Desligando o interruptor o tempo volta a ser incrementado e em 670 ns clico no reset e o tempo volta a 0, começando a contar de 0.

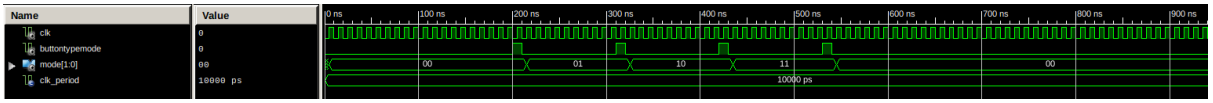
5. Teste da Máquina de Estados de Edição



Através desta simulação podemos ver o comportamento da Máquina de Estados de Edição. O botão de incremento faz uma variável ficar a 1 consoante o modo em que se encontra, e o botão de modo faz alternar entre modos, seg/min/hour.

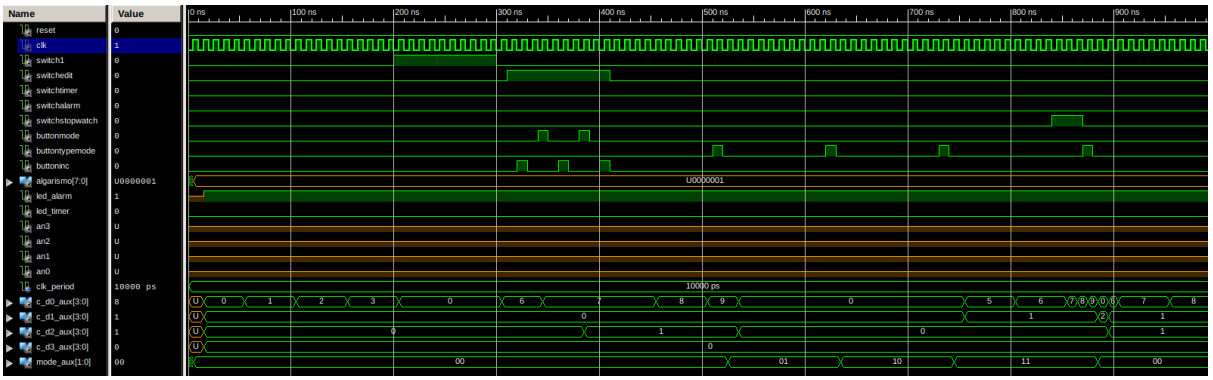
Entre 200 ns e 300 ns testámos a parte de acerto do relógio. Entre 330 ns e 430 ns testámos a parte de acerto do alarme. E por fim entre 460 ns e 520 ns testámos a parte de acerto do temporizador.

6. Teste da Máquina de Estados de Modo



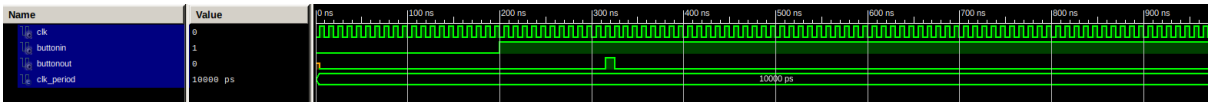
Através desta simulação podemos ver o comportamento da Máquina de Estados de Edição. O trabalho desta máquina de estados é cada vez que um botão é clicado, a variável de saída muda de valor, como podemos confirmar na simulação.

7. Teste do Principal



Através desta simulação podemos ver o comportamento do ficheiro principal. Temos as variaveis c_d0_aux, c_d1_aux, c_d2_aux e c_d3_aux que correspondem aos dígitos do display. Mexemos com o relógio inicialmente, acertamos as hh:mm:ss, de seguida mudamos para o display do alarme, temporizador, cronómetro e de volta para o relógio.

8. Teste do Debouncer



Através desta simulação podemos ver o comportamento do Debouncer.

No real o botão tem uma oscilação e quando estabiliza é que é que envia o pulso a ON. Neste caso para testar, como não houve oscilações passados 10 clocks o pulso foi enviado.

Anexos

1. Relógio

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity relogio is
    Port ( clk : in std_logic;
           reset : in std_logic;
                                                    switchEDIT : in std_logic;
                                                    incSec : in std_logic;
                                                    incMin : in std_logic;
                                                    incHour : in std_logic;
           c_seg_u : out std_logic_vector(3 downto 0);
           c_seg_d : out std_logic_vector(3 downto 0);
           c_min_u : out std_logic_vector(3 downto 0);
           c_min_d : out std_logic_vector(3 downto 0);
           c_hour_u : out std_logic_vector(3 downto 0);
           c_hour_d : out std_logic_vector(3 downto 0);
                                                    seg_u_aux : out std_logic
                                                    );
end relogio;

architecture Behavioral of relogio is

signal cont_subseg : integer range 0 to 50000000;

signal cont_seg_u, cont_min_u, cont_hour_u : integer range 0 to 9;
signal cont_seg_d, cont_min_d, cont_hour_d : integer range 0 to 9;
signal seg_u, seg_d, min_u, min_d, hour_u, hour_d : std_logic;

begin

process (clk, reset)
begin
                    if reset = '1' then
    cont_subseg <= 0;
                    elsif clk'event and clk = '1' and switchEdit = '0' then
                                        if cont_subseg = 49999999 then--49999999 --4
      cont_subseg <= 0;
                                        else cont_subseg <= cont_subseg + 1;
    end if;
  end if;
end process;

seg_u <= '1' when cont_subseg = 49999999 else '0';--49999999 --4
seg_u_aux <= seg_u;

process (clk, reset)
begin
                    if reset = '1' then
    cont_seg_u <= 0;

  elsif clk'event and clk = '1' then
    if seg_u = '1' or incSec = '1' then
      if cont_seg_u = 9 then
        cont_seg_u <= 0;
      else cont_seg_u <= cont_seg_u + 1;
      end if;
    end if;
  end if;
end process;

seg_d <= '1' when cont_seg_u = 9 and (seg_u = '1' or incSec = '1') else '0';
c_seg_u <= std_logic_vector(to_unsigned(cont_seg_u,c_seg_u'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_seg_d <= 0;
  elsif clk'event and clk = '1' then
    if seg_d = '1' then
      if cont_seg_d = 5 then
        cont_seg_d <= 0;
      else cont_seg_d <= cont_seg_d + 1;
      end if;
    end if;
  end if;
end process;

min_u <= '1' when cont_seg_d = 5 and seg_d = '1' and incSec = '0' else '0';
c_seg_d <= std_logic_vector(to_unsigned(cont_seg_d,c_seg_d'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_min_u <= 0;

  elsif clk'event and clk = '1' then
    if min_u = '1' or incMin = '1' then
      if cont_min_u = 9 then
        cont_min_u <= 0;
      else cont_min_u <= cont_min_u + 1;
      end if;
    end if;
  end if;
```

```
end process;

min_d <= '1' when cont_min_u = 9 and (min_u = '1' or incMin = '1') else '0';
c_min_u <= std_logic_vector(to_unsigned(cont_min_u,c_min_u'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_min_d <= 0;
  elsif clk'event and clk = '1' then
    if min_d = '1' then
      if cont_min_d = 5 then
        cont_min_d <= 0;
      else cont_min_d <= cont_min_d + 1;
      end if;
    end if;
  end if;
end process;

hour_u <= '1' when cont_min_d = 5 and min_d = '1' and incMin = '0' else '0';
c_min_d <= std_logic_vector(to_unsigned(cont_min_d,c_min_d'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_hour_u <= 0;

  elsif clk'event and clk = '1' then
    if hour_u = '1' or incHour = '1' then
                                        if cont_hour_d = 2 and cont_hour_u = 3 then
                                                                cont_hour_u <=0;
      elsif cont_hour_u = 9 then
        cont_hour_u <= 0;
      else cont_hour_u <= cont_hour_u + 1;
      end if;
    end if;
  end if;
end process;

hour_d <= '1' when (cont_hour_u = 9 or (cont_hour_u = 3 and cont_hour_d = 2)) and (hour_u = '1' or incHour = '1') else '0'; --or (cont_hour_u = 3 and cont_hour_d = 2))
c_hour_u <= std_logic_vector(to_unsigned(cont_hour_u,c_hour_u'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_hour_d <= 0;
  elsif clk'event and clk = '1' then
    if hour_d = '1' then
      if cont_hour_d = 2 and cont_hour_u = 3 then
        cont_hour_d <= 0;
      else cont_hour_d <= cont_hour_d + 1;
      end if;
    end if;
  end if;
end process;

c_hour_d <= std_logic_vector(to_unsigned(cont_hour_d,c_hour_d'length));

end Behavioral;
```

2. Alarm

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Alarm is
    Port (              clk : in  std_logic;
                                                        reset : in  std_logic;
                                                        incSecAlarm : in std_logic;
                                                        incMinAlarm : in std_logic;
                                                        incHourAlarm : in std_logic;
                                                        buttonSTOP : in std_logic;
                                                        switchALARM : in std_logic;
                                                        c_seg_u : in std_logic_vector(3 downto 0);
                                                        c_seg_d : in std_logic_vector(3 downto 0);
                                                        c_min_u : in std_logic_vector(3 downto 0);
                                                        c_min_d : in std_logic_vector(3 downto 0);
                                                        c_hour_u : in std_logic_vector(3 downto 0);
                                                        c_hour_d : in std_logic_vector(3 downto 0);
                                                        a_seg_u : out std_logic_vector(3 downto 0);
                                                        a_seg_d : out std_logic_vector(3 downto 0);
                                                        a_min_u : out std_logic_vector(3 downto 0);
                                                        a_min_d : out std_logic_vector(3 downto 0);
                                                        a_hour_u : out std_logic_vector(3 downto 0);
                                                        a_hour_d : out std_logic_vector(3 downto 0);
                                                        alarmIsRunning : out std_logic
                                                );
end Alarm;

architecture Behavioral of Alarm is
```

```
signal cont_seg_u, cont_min_u, cont_hour_u : integer range 0 to 9;
signal cont_seg_d, cont_min_d, cont_hour_d : integer range 0 to 9;
signal c_seg_u_aux, c_seg_d_aux, c_min_u_aux, c_min_d_aux, c_hour_u_aux, c_hour_d_aux : std_logic_vector(3 downto 0);
signal seg_d, min_d, hour_d: std_logic;

begin

---------- ALARM IS RUNNING ----------
process(clk)
		begin
		if clk'event and clk = '1' then
				if (switchALARM = '0') then
						if (std_logic_vector(to_unsigned(cont_seg_u,a_seg_u'length)) = c_seg_u_aux and

std_logic_vector(to_unsigned(cont_seg_d,a_seg_d'length)) = c_seg_d_aux and

std_logic_vector(to_unsigned(cont_min_u,a_min_u'length)) = c_min_u_aux and

std_logic_vector(to_unsigned(cont_min_d,a_min_d'length)) = c_min_d_aux and

std_logic_vector(to_unsigned(cont_hour_u,a_hour_u'length)) = c_hour_u_aux and

std_logic_vector(to_unsigned(cont_hour_d,a_hour_d'length)) = c_hour_d_aux) then
								alarmIsRunning <= '1';
						elsif ((buttonSTOP = '1') or (std_logic_vector(to_unsigned(cont_seg_u,a_seg_u'length)) = c_seg_u_aux and

std_logic_vector(to_unsigned(cont_seg_d,a_seg_d'length)) = c_seg_d_aux and

std_logic_vector(to_unsigned(cont_min_u,a_min_u'length)) = c_min_u_aux and

std_logic_vector(to_unsigned(cont_min_d,a_min_d'length)) = c_min_d_aux and

std_logic_vector(to_unsigned(cont_hour_u+1,a_hour_u'length)) = c_hour_u_aux and

std_logic_vector(to_unsigned(cont_hour_d,a_hour_d'length)) = c_hour_d_aux)) then
								alarmIsRunning <= '0';
						end if;
				end if;
		end if;
end process;

---------- ALARM EDIT ----------
process (clk, reset, incSecAlarm)
		begin
				if reset = '1' then
				 cont_seg_u <= 0;

		 elsif clk'event and clk = '1' then
				 if incSecAlarm = '1' then
						if cont_seg_u = 9 then
						  cont_seg_u <= 0;
						else cont_seg_u <= cont_seg_u + 1;
						end if;
				 end if;
		 end if;

end process;

seg_d <= '1' when cont_seg_u = 9 and incSecAlarm = '1' else '0';
a_seg_u <= std_logic_vector(to_unsigned(cont_seg_u,a_seg_u'length));

process (clk, reset)
		begin
		 if reset = '1' then
				cont_seg_d <= 0;

		 elsif clk'event and clk = '1' then
				if seg_d = '1' then
						if cont_seg_d = 5 then
						  cont_seg_d <= 0;
						else cont_seg_d <= cont_seg_d + 1;
						end if;
				end if;
		 end if;
end process;

a_seg_d <= std_logic_vector(to_unsigned(cont_seg_d,a_seg_d'length));


process (clk, reset, incMinAlarm)
		begin
		 if reset = '1' then
				cont_min_u <= 0;

		 elsif clk'event and clk = '1' then
				if incMinAlarm = '1' then
						if cont_min_u = 9 then
						  cont_min_u <= 0;
						else cont_min_u <= cont_min_u + 1;
						end if;
				end if;
		 end if;
end process;

min_d <= '1' when cont_min_u = 9 and incMinAlarm = '1' else '0';
a_min_u <= std_logic_vector(to_unsigned(cont_min_u,a_min_u'length));


process (clk, reset)
		begin
		 if reset = '1' then
				cont_min_d <= 0;

		 elsif clk'event and clk = '1' then
```

```
                        if min_d = '1' then
                                if cont_min_d = 5 then
                                  cont_min_d <= 0;
                                else cont_min_d <= cont_min_d + 1;
                                end if;
                        end if;
                end if;
end process;

a_min_d <= std_logic_vector(to_unsigned(cont_min_d,a_min_d'length));


process (clk, reset, incHourAlarm)
                begin
                  if reset = '1' then
                        cont_hour_u <= 0;

                  elsif clk'event and clk = '1' then
                        if incHourAlarm = '1' then
                                if cont_hour_d = 2 and cont_hour_u = 3 then
                                        cont_hour_u <=0;
                                elsif cont_hour_u = 9 then
                                  cont_hour_u <= 0;
                                else cont_hour_u <= cont_hour_u + 1;
                                end if;
                        end if;
                end if;
end process;

hour_d <= '1' when (cont_hour_u = 9 or (cont_hour_u = 3 and cont_hour_d = 2)) and incHourAlarm = '1' else '0'; --or (cont_hour_u = 3 and cont_hour_d = 2))
a_hour_u <= std_logic_vector(to_unsigned(cont_hour_u,a_hour_u'length));


process (clk, reset)
                begin
                  if reset = '1' then
                        cont_hour_d <= 0;

                  elsif clk'event and clk = '1' then
                        if hour_d = '1' then
                                if cont_hour_d = 2 and cont_hour_u = 3 then
                                  cont_hour_d <= 0;
                                else cont_hour_d <= cont_hour_d + 1;
                                end if;
                        end if;
                end if;
end process;

a_hour_d <= std_logic_vector(to_unsigned(cont_hour_d,a_hour_d'length));


process(clk)
                begin
                if(clk'event and clk = '1') then
                        c_hour_d_aux <= c_hour_d;
                        c_hour_u_aux <= c_hour_u;
                        c_min_d_aux <= c_min_d;
                        c_min_u_aux <= c_min_u;
                        c_seg_d_aux <= c_seg_d;
                        c_seg_u_aux <= c_seg_u;
                end if;
end process;

end Behavioral;
```

3. Timer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Timer is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
                                                switchTimer : in std_logic;
                                                incSecTimer : in std_logic;
                                                incMinTimer : in std_logic;
           t_seg_u : out std_logic_vector(3 downto 0);
           t_seg_d : out std_logic_vector(3 downto 0);
           t_min_u : out std_logic_vector(3 downto 0);
           t_min_d : out std_logic_vector(3 downto 0);
                                                seg_u_aux : in std_logic;
                                                timerFinished : out std_logic
                                                );
end Timer;

architecture Behavioral of Timer is

signal cont_seg_u, cont_min_u: integer range 0 to 9;
signal cont_seg_d, cont_min_d: integer range 0 to 9;
signal iseg_d, imin_d, dseg_d, dmin_u, dmin_d, timerFinished_aux : std_logic;
```

```
begin

-------------------- RUNNING TIMER ---------------------

process(clk, cont_seg_u, cont_min_u, cont_seg_d, cont_min_d)
                begin
                                if(cont_seg_u = 0 and cont_seg_d = 0 and cont_min_u = 0 and cont_min_d = 0) then
                                                timerFinished_aux <= '1';
                                else
                                                timerFinished_aux <= '0';
                                end if;
end process;

timerFinished <= timerFinished_aux;

---------------------- EDIT TIMER ----------------------

process (clk, reset)
                begin
                                if reset = '1' then
                                 cont_seg_u <= 0;

                                elsif clk'event and clk = '1' then
                                -- ACERTO DOS SEGUNDOS UNIDADES --
                                 if incSecTimer = '1' then
                                                if cont_seg_u = 9 then
                                                  cont_seg_u <= 0;
                                                else cont_seg_u <= cont_seg_u + 1;
                                                end if;

                                -- TIMER A DECREMENTAR --
                                 elsif seg_u_aux = '1' and timerFinished_aux = '0' and switchTimer = '0' then
                                                if cont_seg_u = 0 then
                                                  cont_seg_u <= 9;
                                                else cont_seg_u <= cont_seg_u - 1;
                                                end if;
                                 end if;
                 end if;

end process;

iseg_d <= '1' when cont_seg_u = 9 and incSecTimer = '1' else '0';
dseg_d <= '1' when (cont_seg_u = 0 and timerFinished_aux = '0' and switchTimer = '0' and (cont_seg_d > 0 or cont_min_u > 0 or cont_min_d > 0)) else '0';
t_seg_u <= std_logic_vector(to_unsigned(cont_seg_u,t_seg_u'length));

process (clk, reset)
                begin
                 if reset = '1' then
                                cont_seg_d <= 0;
                 elsif clk'event and clk = '1' then
                                if iseg_d = '1' then
                                                if cont_seg_d = 5 then
                                                  cont_seg_d <= 0;
                                                else cont_seg_d <= cont_seg_d + 1;
                                                end if;
                                elsif seg_u_aux = '1' and dseg_d = '1' then
                                                if cont_seg_d = 0 then
                                                  cont_seg_d <= 5;
                                                else cont_seg_d <= cont_seg_d - 1;
                                                end if;
                                end if;
                 end if;
end process;

dmin_u <= '1' when ((cont_seg_d = 0 and cont_seg_u = 0) and timerFinished_aux = '0' and switchTimer = '0' and (cont_min_u > 0 or cont_min_d > 0)) else '0';
t_seg_d <= std_logic_vector(to_unsigned(cont_seg_d,t_seg_d'length));

process (clk, reset)
                begin
                 if reset = '1' then
                                cont_min_u <= 0;
                 elsif clk'event and clk = '1' then
                                if incMinTimer = '1' then
                                                if cont_min_u = 9 then
                                                  cont_min_u <= 0;
                                                else cont_min_u <= cont_min_u + 1;
                                                end if;
                                elsif seg_u_aux = '1' and dmin_u = '1' then
                                                if cont_min_u = 0 then
                                                  cont_min_u <= 9;
                                                else cont_min_u <= cont_min_u - 1;
                                                end if;
                                end if;
                 end if;
end process;

imin_d <= '1' when cont_min_u = 9 and incMinTimer = '1' else '0';
dmin_d <= '1' when ((cont_min_u = 0 and cont_seg_d = 0 and cont_seg_u = 0) and timerFinished_aux = '0' and switchTimer = '0' and cont_min_d > 0) else '0';
t_min_u <= std_logic_vector(to_unsigned(cont_min_u,t_min_u'length));

process (clk, reset)
                begin
                 if reset = '1' then
                                cont_min_d <= 0;
                 elsif clk'event and clk = '1' then
                                if imin_d = '1' then
                                                if cont_min_d = 5 then
                                                  cont_min_d <= 0;
                                                else cont_min_d <= cont_min_d + 1;
                                                end if;
                                elsif seg_u_aux = '1' and dmin_d = '1' then
                                                if cont_min_d = 0 then
                                                  cont_min_d <= 5;
                                                else cont_min_d <= cont_min_d - 1;
                                                end if;
```

```
                                end if;
                        end if;
end process;

t_min_d <= std_logic_vector(to_unsigned(cont_min_d,t_min_d'length));

end Behavioral;
```

4. StopWatch

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity StopWatch is
    Port ( clk : in  STD_LOGIC;
           reset : in  STD_LOGIC;
                                                    switchStopWatch : in std_logic;
           sw_mseg_u : out std_logic_vector(3 downto 0);
           sw_mseg_d : out std_logic_vector(3 downto 0);
           sw_seg_u : out std_logic_vector(3 downto 0);
           sw_seg_d : out std_logic_vector(3 downto 0));
end StopWatch;

architecture Behavioral of StopWatch is

signal cont_msubseg : integer range 0 to 500000;

signal cont_mseg_u, cont_seg_u : integer range 0 to 9;
signal cont_mseg_d, cont_seg_d : integer range 0 to 9;
signal mseg_u, mseg_d, seg_u, seg_d : std_logic;

begin

process (clk, reset)
begin
                        if reset = '1' then
    cont_msubseg <= 0;
                        elsif clk'event and clk = '1' and switchStopWatch = '0' then
                                if cont_msubseg = 4 then--499999 --4
      cont_msubseg <= 0;
                                else cont_msubseg <= cont_msubseg + 1;
    end if;
   end if;
end process;

mseg_u <= '1' when cont_msubseg = 4 else '0';--499999 --4

process (clk, reset)
begin
                        if reset = '1' then
    cont_mseg_u <= 0;
  elsif clk'event and clk = '1' then
    if mseg_u = '1' then
      if cont_mseg_u = 9 then
        cont_mseg_u <= 0;
      else cont_mseg_u <= cont_mseg_u + 1;
      end if;
    end if;
   end if;
end process;

mseg_d <= '1' when cont_mseg_u = 9 and mseg_u = '1' else '0';
sw_mseg_u <= std_logic_vector(to_unsigned(cont_mseg_u,sw_mseg_u'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_mseg_d <= 0;
  elsif clk'event and clk = '1' then
    if mseg_d = '1' then
      if cont_mseg_d = 9 then
        cont_mseg_d <= 0;
      else cont_mseg_d <= cont_mseg_d + 1;
      end if;
    end if;
  end if;
end process;

seg_u <= '1' when cont_mseg_d = 9 and mseg_d = '1' else '0';
sw_mseg_d <= std_logic_vector(to_unsigned(cont_mseg_d,sw_mseg_d'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_seg_u <= 0;

  elsif clk'event and clk = '1' then
    if seg_u = '1' then
      if cont_seg_u = 9 then
        cont_seg_u <= 0;
      else cont_seg_u <= cont_seg_u + 1;
```

```
    end if;
   end if;
  end if;
end process;

seg_d <= '1' when cont_seg_u = 9 and seg_u = '1' else '0';
sw_seg_u <= std_logic_vector(to_unsigned(cont_seg_u,sw_seg_u'length));

process (clk, reset)
begin
  if reset = '1' then
    cont_seg_d <= 0;
  elsif clk'event and clk = '1' then
    if seg_d = '1' then
      if cont_seg_d = 5 then
        cont_seg_d <= 0;
      else cont_seg_d <= cont_seg_d + 1;
      end if;
    end if;
  end if;
end process;

sw_seg_d <= std_logic_vector(to_unsigned(cont_seg_d,sw_seg_d'length));

end Behavioral;
```

5. Conversor do Display

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity conv_displays is
    Port ( clk : in std_logic;
           reset : in std_logic;
           seg_u : in std_logic_vector(3 downto 0);
           seg_d : in std_logic_vector(3 downto 0);
           min_u : in std_logic_vector(3 downto 0);
           min_d : in std_logic_vector(3 downto 0);
           an3 : out std_logic;
           an2 : out std_logic;
           an1 : out std_logic;
           an0 : out std_logic;
           algarismo : out std_logic_vector(7 downto 0));
end conv_displays;

architecture Behavioral of conv_displays is
signal alg : integer range 0 to 9;
signal cont_clock : integer range 0 to 499999;
signal num_alg : std_logic_vector(1 downto 0);

begin

co: process (clk, reset)
begin
 if reset = '1' then
    cont_clock <= 0;
  elsif clk'event and clk = '1' then
    if cont_clock = 499999 then
      cont_clock <= 0;
      num_alg <= "00";
    else
      if cont_clock = 124999 then
        num_alg <= "01";
      elsif cont_clock = 249999 then
        num_alg <= "10";
      elsif cont_clock = 374999 then
        num_alg <= "11";
      end if;
      cont_clock <= cont_clock + 1;
    end if;
  end if;
end process;

an : process(clk, reset)
begin
  if reset = '1' then
    an3 <= '0';
    an2 <= '0';
    an1 <= '0';
    an0 <= '0';
    alg <= 0;
    algarismo(7) <= '1';
  elsif clk'event and clk = '1' then
    if num_alg = "00" then
      an3 <= '0';
      an2 <= '1';
      an1 <= '1';
      an0 <= '1';
      alg <= to_integer(unsigned(seg_u));
      algarismo(7) <= '1';
    elsif num_alg = "01" then
      an3 <= '1';
      an2 <= '0';
      an1 <= '1';
      an0 <= '1';
      alg <= to_integer(unsigned(seg_d));
      algarismo(7) <= '1';
    elsif num_alg = "10" then
      an3 <= '1';
      an2 <= '1';
```

```
    an1 <= '0';
                                    an0 <= '1';
    alg <= to_integer(unsigned(min_u));
    algarismo(7) <= '0';
   elsif num_alg = "11" then
    an3 <= '1';
    an2 <= '1';
    an1 <= '1';
    an0 <= '0';
    alg <= to_integer(unsigned(min_d));
    algarismo(7) <= '1';
   end if;
  end if;
end process;

al : process(clk, reset)
begin
  if reset = '1' then
                algarismo(6 downto 0) <= "0000001";
  elsif clk'event and clk = '1' then
    case alg is
                when 0 => algarismo(6 downto 0) <= "0000001";
                when 1 => algarismo(6 downto 0) <= "1001111";
                when 2 => algarismo(6 downto 0) <= "0010010";
                when 3 => algarismo(6 downto 0) <= "0000110";
                when 4 => algarismo(6 downto 0) <= "1001100";
                when 5 => algarismo(6 downto 0) <= "0100100";
                when 6 => algarismo(6 downto 0) <= "0100000";
                when 7 => algarismo(6 downto 0) <= "0001111";
                when 8 => algarismo(6 downto 0) <= "0000000";
                when 9 => algarismo(6 downto 0) <= "0000100";
                when others => algarismo(6 downto 0) <= "0111000";
    end case;
   end if;
end process;

end Behavioral;
```

6. Máquina de Estados do Modo

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Mode_State_Machine is
    Port (
                                        clk : in  STD_LOGIC ;
                                        buttonTYPEMODE : in STD_LOGIC ;
                                        mode : out STD_LOGIC_VECTOR(1 DOWNTO 0)
                                        );
                end Mode_State_Machine;

                architecture Behavioral of Mode_State_Machine is
                --Use descriptive names for the states, like st1_reset, st2_search
                type state_type is (watch, alarm, timer, stopwatch); --
                signal state, next_state : state_type;

                --Declare internal signals for all outputs of the state-machine
                signal mode_i : std_logic_vector(1 downto 0);

                --signal mode_aux : integer range 0 to 2;

begin

SYNC_PROC: process (clk)
   begin
      if (clk'event and clk = '1') then
          state <= next_state;
                                                        --mode_i <= std_logic_vector(to_unsigned(mode_aux,mode_i'length));
          mode <= mode_i;
        -- assign other outputs to internal signals
      end if;
   end process;

   --MOORE State-Machine - Outputs based on state only
   OUTPUT_DECODE: process (state)
   begin
      --insert statements to decode internal output signals
      --below is simple example
      if state = watch then
      else
      end if;
   end process;

   NEXT_STATE_DECODE: process (state, buttonTYPEMODE)
   begin
      --declare default state for next_state to avoid latches
      next_state <= state;  --default is to stay in current state
      --insert statements to decode next_state
      --below is a simple example
      case (state) is
         when watch =>
                                                        mode_i <= "00";
          if buttonTYPEMODE = '1' then
```

```
            next_state <= alarm;
                                                                    else
                                                                            next_state <= watch;
          end if;
        when alarm =>
                                                                    mode_i <= "01";
          if buttonTYPEMODE = '1' then
            next_state <= timer;
                                                                    else
                                                                            next_state <= alarm;
          end if;
        when timer =>
                                                                    mode_i <= "10";
                                                                    if buttonTYPEMODE = '1' then
            next_state <= stopwatch;
                                                                    else
                                                                            next_state <= timer;
          end if;
                                                        when stopwatch =>
                                                                    mode_i <= "11";
                                                                    if buttonTYPEMODE = '1' then
            next_state <= watch;
                                                                    else
                                                                            next_state <= stopwatch;
          end if;
        when others =>
          next_state <= watch;
      end case;
    end process;


end Behavioral;
```

7. Máquina de Estados da Edição

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity State_machine is
--Insert the following in the architecture before the begin keyword
                Port (
                                clk : in std_logic;
                                reset : in std_logic;
                                switchEDIT : in std_logic;
                                switchTIMER : in std_logic;
                                switchALARM : in std_logic;
                                buttonMODE : in std_logic;
                                buttonINC : in std_logic;
                                incSec : out std_logic;
                                incMin : out std_logic;
                                incHour : out std_logic;
                                incSecAlarm : out std_logic;
                                incMinAlarm : out std_logic;
                                incHourAlarm : out std_logic;
                                incSecTimer : out std_logic;
                                incMinTimer : out std_logic
                );
                end State_machine;

                architecture Behavioral of State_machine is

  --Use descriptive names for the states, like st1_reset, st2_search
  type state_type is (          st0_idle, st1_seconds, st2_minutes, st3_hours, st4_incsec, st5_incmin, st6_inchour,
                                                                                                  secAlarm, minAlarm, hourAlarm, SincSecAlarm, SincMinAlarm,
SincHourAlarm,
                                                                                                  secTimer, minTimer, SincSecTimer, SincMinTimer);

  signal state, next_state : state_type;

  --Declare internal signals for all outputs of the state-machine
  signal incSec_i : std_logic;
  signal incMin_i : std_logic;
  signal incHour_i : std_logic;

                signal incSecAlarm_i : std_logic;
  signal incMinAlarm_i : std_logic;
  signal incHourAlarm_i : std_logic;

                signal incSecTimer_i : std_logic;
  signal incMinTimer_i : std_logic;

                --other outputs
begin

SYNC_PROC: process (clk)
  begin
    if (clk'event and clk = '1') then
      if (reset = '1') then
        state <= st0_idle;
        incSec <= '0';
        incMin <= '0';
        incHour <= '0';
                                                                    incSecAlarm <= '0';
        incMinAlarm <= '0';
```

```
      incHourAlarm <= '0';
                                                    incSecTimer <= '0';
      incMinTimer <= '0';
    else
      state <= next_state;
      incSec <= incSec_i;
      incMin <= incMin_i;
      incHour <= incHour_i;
                                                    incSecAlarm <= incSecAlarm_i;
      incMinAlarm <= incMinAlarm_i;
      incHourAlarm <= incHourAlarm_i;
                                                    incSecTimer <= incSecTimer_i;
      incMinTimer <= incMinTimer_i;
    -- assign other outputs to internal signals
    end if;
  end if;
end process;

--MOORE State-Machine - Outputs based on state only
OUTPUT_DECODE: process (state)
begin
  --insert statements to decode internal output signals
  --below is simple example
  if state = st3_hours then
  else
  end if;
end process;

NEXT_STATE_DECODE: process (state, buttonMODE, switchEDIT, switchALARM, switchTIMER, buttonINC)
begin
  --declare default state for next_state to avoid latches
  next_state <= state;
  case (state) is
                                          when st0_idle =>
                                                    if switchEDIT = '1' then
                                                              next_state <= st1_seconds;
                                                    elsif switchALARM = '1' then
                                                              next_state <= secAlarm;
                                                    elsif switchTIMER = '1' then
                                                              next_state <= secTimer;
                                                    else
                                                              next_state <=st0_idle;
                                                    end if;

                                          -- WATCH STATE MACHINE PART --
    when st1_seconds =>
                                                    if switchEDIT = '0' then
                                                              next_state <= st0_idle;
                                                    elsif buttonINC = '1' then
                                                              next_state <= st4_incsec;
      elsif buttonMODE = '1' then
        next_state <= st2_minutes;
                                                    else
                                                              next_state <=st1_seconds;
      end if;

    when st2_minutes =>
                                                    if switchEDIT = '0' then
                                                              next_state <= st0_idle;
                                                    elsif buttonINC = '1' then
                                                              next_state <= st5_incmin;
      elsif buttonMODE = '1' then
        next_state <= st3_hours;
                                                              else
                                                              next_state <=st2_minutes;
      end if;

    when st3_hours =>
                                                    if switchEDIT = '0' then
                                                              next_state <= st0_idle;
                                                    elsif buttonINC = '1' then
                                                              next_state <= st6_inchour;
      elsif buttonMODE = '1' then
                                                              next_state <= st1_seconds;
                                                              else
                                                              next_state <=st3_hours;
                                                    end if;

                                          when st4_incsec =>
                                                    if buttonINC = '0' then
                                                              next_state <= st1_seconds;
                                                              else
                                                              next_state <=st4_incsec;
                                                    end if;

                                          when st5_incmin =>
                                                    if buttonINC = '0' then
                                                              next_state <= st2_minutes;
                                                              else
                                                              next_state <=st5_incmin;
                                                    end if;

                                          when st6_inchour =>
                                                    if buttonINC = '0' then
                                                              next_state <= st3_hours;
                                                              else
                                                              next_state <=st6_inchour;
                                                    end if;

                                          -- ALARM STATE MACHINE PART --
                                          when secAlarm =>
                                                    if switchALARM = '0' then
                                                              next_state <= st0_idle;
                                                    elsif buttonINC = '1' then
                                                              next_state <= SincSecAlarm;
```

```
        elsif buttonMODE = '1' then
                                                        next_state <= minAlarm;
                                                        else
                                                        next_state <=secAlarm;
                                                end if;

                                        when minAlarm =>
                                                if switchALARM = '0' then
                                                        next_state <= st0_idle;
                                                elsif buttonINC = '1' then
                                                        next_state <= SincMinAlarm;
        elsif buttonMODE = '1' then
                                                        next_state <= hourAlarm;
                                                        else
                                                        next_state <=minAlarm;
                                                end if;

                                        when hourAlarm =>
                                                if switchALARM = '0' then
                                                        next_state <= st0_idle;
                                                elsif buttonINC = '1' then
                                                        next_state <= SincHourAlarm;
        elsif buttonMODE = '1' then
                                                        next_state <= secAlarm;
                                                        else
                                                        next_state <=hourAlarm;
                                                end if;

                                        when SincSecAlarm =>
                                                if buttonINC = '0' then
                                                        next_state <= secAlarm;
                                                        else
                                                        next_state <=SincSecAlarm;
                                                end if;

                                        when SincMinAlarm =>
                                                if buttonINC = '0' then
                                                        next_state <= minAlarm;
                                                        else
                                                        next_state <=SincMinAlarm;
                                                end if;

                                        when SincHourAlarm =>
                                                if buttonINC = '0' then
                                                        next_state <= hourAlarm;
                                                        else
                                                        next_state <=SincHourAlarm;
                                                end if;

                                        -- TIMER STATE MACHINE PART --
      when secTimer =>
                                                if switchTIMER = '0' then
                                                        next_state <= st0_idle;
                                                elsif buttonINC = '1' then
                                                        next_state <= SincSecTimer;
        elsif buttonMODE = '1' then
          next_state <= minTimer;
                                                else
                                                        next_state <=secTimer;
        end if;

      when minTimer =>
                                                if switchTIMER = '0' then
                                                        next_state <= st0_idle;
                                                elsif buttonINC = '1' then
                                                        next_state <= SincMinTimer;
        elsif buttonMODE = '1' then
          next_state <= secTimer;
                                                else
                                                        next_state <= minTimer;
        end if;

                                        when SincSecTimer =>
                                                if buttonINC = '0' then
                                                        next_state <= secTimer;
                                                else
                                                        next_state <= SincSecTimer;
                                                end if;

                                        when SincMinTimer =>
                                                if buttonINC = '0' then
                                                        next_state <= minTimer;
                                                else
                                                        next_state <=SincMinTimer;
                                                end if;

        when others =>
          next_state <= st0_idle;
    end case;
  end process;

incSec_i <= '1' when (state = st1_seconds and buttonINC = '1') else '0';
incMin_i <= '1' when (state = st2_minutes and buttonINC = '1') else '0';
incHour_i <= '1' when (state = st3_hours and buttonINC = '1') else '0';

incSecAlarm_i <= '1' when (state = secAlarm and buttonINC = '1') else '0';
incMinAlarm_i <= '1' when (state = minAlarm and buttonINC = '1') else '0';
incHourAlarm_i <= '1' when (state = hourAlarm and buttonINC = '1') else '0';

incSecTimer_i <= '1' when (state = secTimer and buttonINC = '1') else '0';
incMinTimer_i <= '1' when (state = minTimer and buttonINC = '1') else '0';

end Behavioral;
```

8. Debouncer

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity Debouncer is
    Port (          clk : in  STD_LOGIC;
                                                        buttonIn : in STD_LOGIC;
                                                        buttonOut : out STD_LOGIC
                    );
end Debouncer;

architecture Behavioral of Debouncer is

                signal lastButtonState    : std_logic := '0';
                signal buttonState    : std_logic := '0';
                signal counter : integer range 0 to 10000 := 0;

begin
                process(clk)
                begin
                 if(clk'event and clk = '1') then
                                        if(buttonIn = '1') then      --assuming active-high
                                                if counter < 10000 then
                                                        counter <= counter + 1;
                                                else
                                                        buttonState <= '1';
                                                end if;
                                        else
                                                if counter > 0 then
                                                        counter <= counter - 1;
                                                else
                                                        buttonState <= '0';
                                                end if;
                            end if;
                 end if;
                end process;

                process(clk)
                begin
                            if(clk'event and clk = '1') then
                                        if buttonState = '1' and lastButtonState = '0' then
                                                buttonOut <= '1';
                                        else
                                                buttonOut <= '0';
                                        end if;
                                        lastButtonState <= buttonState;
                            end if;
                end process;

end Behavioral;
```