به نام خدا

گزارش کار آزمایشگاه معماری کامپیوتر

پروژه نهایی

عنوان آزمایش:

NEC Protocol IR Receiver and LCD Presentation

نام استاد:

استاد علی جوادی

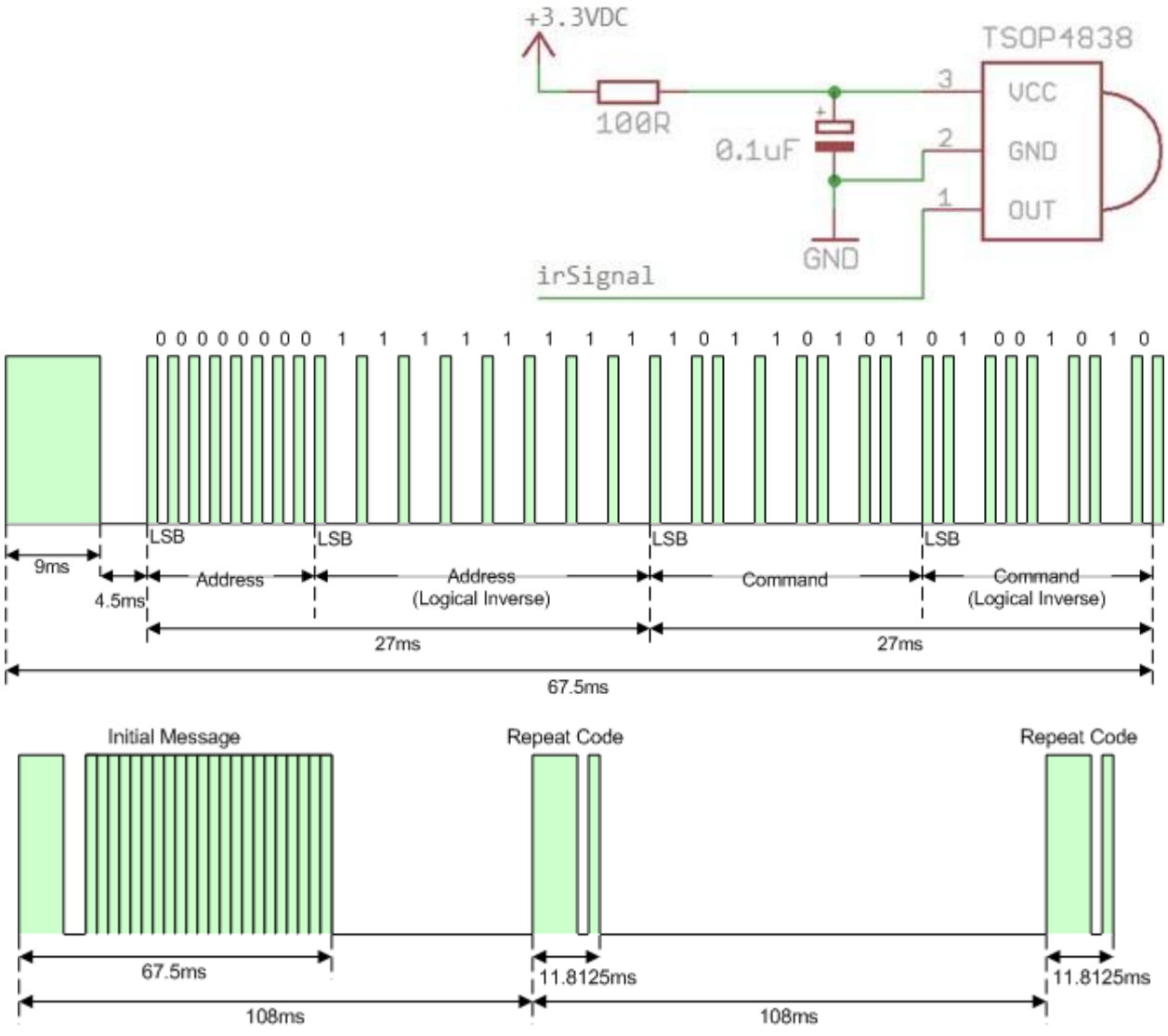
اعضای گروه:

غزل عربعلی – بهاره کاوسی نژاد

آزمایش: NEC Protocol IR Receiver and LCD Presentation

هدف آزمایش: دریافت داده از پروتکل NEC و نمایش روی LCD

تئوری آزمایش:



روش و چگونگی انجام آزمایش:

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


entity nec_receiver is -- asm_with_datapath
  port(

    --Main system clock signal; used for reading the NEC signal's timing.
    clk_40MHz : in std_ulogic;

    --Active high reset signal
    reset : in std_ulogic := '0';

    -- The NEC infrared input.
    nec_in : in std_ulogic;

    --The most recently received command.
    last_received_command : out std_ulogic_vector(7 downto 0) :=
"00000000";
    --High iff the given key is currently down, to the best of the
    --receiver's knowledge.
    key_down : out std_ulogic

  );
end nec_receiver;
```

ورودی و خروجی ها را مشخص می کنیم.

```
architecture asm_with_datapath of nec_receiver is

  --
  -- NEC protocol timings.
  --

  -- The NEC header is a 9ms pulse; these constants specify the ideal and realistic timings of that
header.
  constant HEADER_PULSE_TIME_ALLOWANCE : integer := 1000;
  constant HEADER_PULSE_TIME_BY_SPEC   : integer := 360000; -- 288000; -- 9ms @ 32MHz
  constant HEADER_MINUMUM_PULSE_LENGTH : integer := HEADER_PULSE_TIME_BY_SPEC -
HEADER_PULSE_TIME_ALLOWANCE;
  constant HEADER_MAXIMUM_PULSE_LENGTH : integer := HEADER_PULSE_TIME_BY_SPEC +
HEADER_PULSE_TIME_ALLOWANCE;

  -- After the header, there's a window about 2.25ms long which will always be low. The value /after/
this
  -- window can be used to detect whether the packet is a repeat command.
  constant HEADER_REPEAT_WINDOW_START  : integer := 92000; -- 73600;

  -- Stores the minimum pulse distance which should be considered a one. Taken directly from the NEC
spec.
  constant DATA_MINIMUM_TIME_FOR_ONE   : integer := 22500; -- 18000;

  -- Stores the maximum amount of time we'll wait for a "repeat packet". If we don't receieve a repeat
  -- packet in this time, we'll decide that the key must have been released.
  constant KEY_PRESS_TIMEOUT : integer := 4320000 + 1000; -- 3456000 + 1000;

  --
  --Core controller FSM logic.
  --
  type receiver_state is (WAIT_FOR_HEADER, READ_HEADER, WAIT_FOR_REPEAT_WINDOW, WAIT_FOR_IDLE,
WAIT_FOR_COMMAND, WAIT_FOR_END_OF_DATA_PULSE, COUNT_DATA_PULSE_SPACING, PROCESS_PACKET);

  --Current and next state logic for the FSM.
  signal current_state : receiver_state := WAIT_FOR_HEADER;
  signal next_state, next_state_with_reset : receiver_state;

  -- Control output signals.
  signal repeat_code_received, command_packet_started : std_ulogic;
  signal data_bit_received, packet_received : std_ulogic;

  --
  -- Datapath signals.
  --

  --Time counter, which counts the amount of time that has passed.
  signal time_counter_clear : std_ulogic;
  signal time_counter_value : unsigned(21 downto 0) := (others => '0');

  --Comparator output, which determine if the current time counter
  --value would be interpreted as a '0' or a '1' when interpreted as
  --a pulse spacing.
  signal current_bit_value : std_ulogic;

  --Pulse counter, which counts the amount of pulses passed.
  signal prior_input_value, is_rising_edge_of_input : std_ulogic;
  signal pulse_counter_clear          : std_ulogic;
  signal pulse_counter_value          : unsigned(4 downto 0);

  -- "Shift register" which stores the 16 most recent received bits.
  signal received_bits      : std_ulogic_vector(15 downto 0);
  signal data_is_valid : std_ulogic;


begin
```

ماژول `irReceiver` دارای 2 سیگنال ورودی و یک بردار خروجی است. از یک ساعت 40 مگاهرتز برای اعمال منطق داخلی استفاده می‌شود. سیگنال حسگر از طریق ورودی `irSignal` به این ماژول می‌رسد و پس از رمزگشایی، یک بردار خروجی با مقدار دریافتی به صورت باینری تولید می‌شود. در پاراگراف‌های زیر، عملکرد داخلی ماژول توضیح داده خواهد شد.

ماژول `irReceiver` شامل یک واحد کنترل و یک واحد فرآیند است. برای جمع‌آوری داده‌ها و رمزگشایی آنها، یک ماشین حالت طراحی شده است که در شکل زیر نمایش داده شده است.

Header در این پروتکل 9 میلی ثانیه است بعد از آن یک پنجره 2.25 میلی ثانیه ای داریم و از طریق این پنجره متوجه می شویم که repeat code است یا خیر. سپس مینیمم فاصله ای که لازم است در نظر گرفته شود تا 1 به دست بیاید.

سپس مقدار ماکسیمم زمانی که برای repeat packet لازم داریم را در key_press_timeout ذخیره می کنیم.

وارد قسمت منطق FSM می شویم.

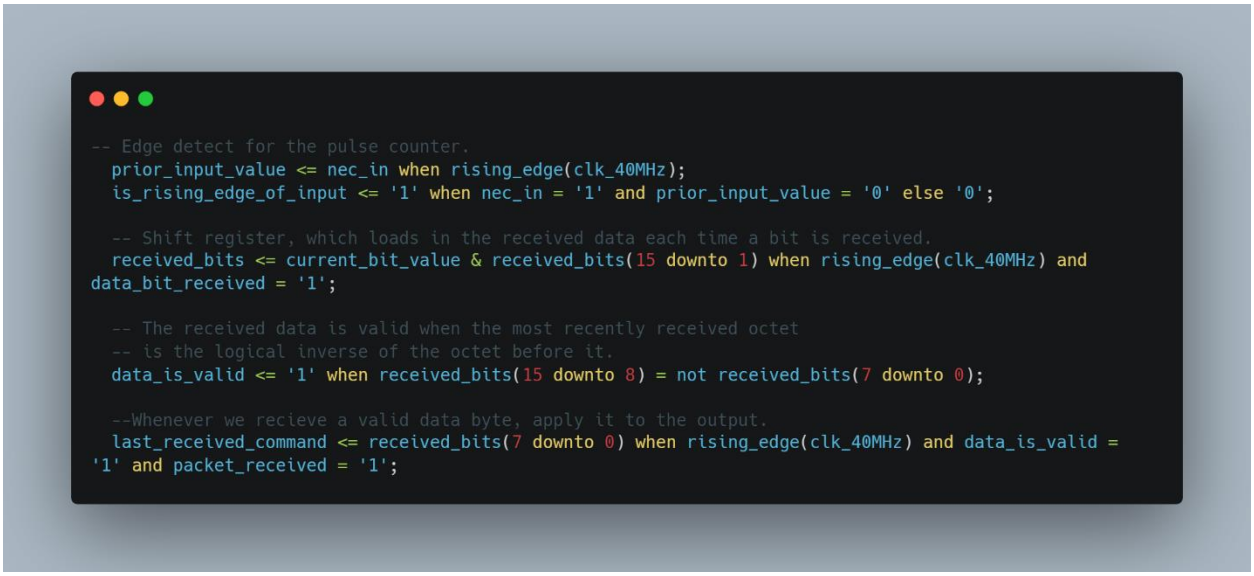
تایپ receiver_state را تعریف می کنیم. تعدادی سیگنال برای ذخیره کردن stateها لازم داریم.

به قسمت تعریف Datapath می رسیم. یک سیگنال برای track کردن زمان سپری شده و بیت کنونی در نظر می گیریم. یک سیگنال دیگر برای شمارنده pulse لازم داریم. در یک شیفت رجیستر 16 بیت اخیر دریافت شده را ذخیره می کنیم.

یک کلاک 40 MHz تعریف می کنیم. مقدار بیت کنونی را در current_bit_value ذخیره می کنیم.

یک PULSE_COUNTER بر اساس کلاک 40 مگاهرتز تعریف می کنیم و مقدار pulse_counter_clear را در آن مقداردهی می کنیم.

```
--
-- Datapath.
--

-- Core counter, which counts the amount of cycles that have passed since
-- its last reset. (Note that this could alternatively be accomplished with a
-- process, but this method is less ugly.)
TIME_COUNTER:
process(clk_40MHz)
begin
  if rising_edge(clk_40MHz) then

    --if our clear signal is high, clear the counter.
    if time_counter_clear = '1' then
      time_counter_value <= (others => '0');

    --otherwise, count normally.
    else
      time_counter_value <= time_counter_value + 1;
    end if;
  end if;
end process;


-- Comparator, which determines whether the current time-counter value
-- would be considered a '0' or a '1'.
current_bit_value <= '1' when time_counter_value > DATA_MINIMUM_TIME_FOR_ONE else '0';


-- Pulse counter, which counts the amount of pulses that have passed since
-- its last reset.
PULSE_COUNTER:
process(clk_40MHz)
begin
  if rising_edge(clk_40MHz) then

    --If our clear signal is high, clear the pulse counter.
    if pulse_counter_clear = '1' then
      pulse_counter_value <= (others => '0');

    --Otherwise, count rising edges.
    elsif is_rising_edge_of_input = '1' then
      pulse_counter_value <= pulse_counter_value + 1;

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



```
-- Edge detect for the pulse counter.
  prior_input_value <= nec_in when rising_edge(clk_40MHz);
  is_rising_edge_of_input <= '1' when nec_in = '1' and prior_input_value = '0' else '0';

  -- Shift register, which loads in the received data each time a bit is received.
  received_bits <= current_bit_value & received_bits(15 downto 1) when rising_edge(clk_40MHz) and
data_bit_received = '1';

  -- The received data is valid when the most recently received octet
  -- is the logical inverse of the octet before it.
  data_is_valid <= '1' when received_bits(15 downto 8) = not received_bits(7 downto 0);

  --Whenever we recieve a valid data byte, apply it to the output.
  last_received_command <= received_bits(7 downto 0) when rising_edge(clk_40MHz) and data_is_valid =
'1' and packet_received = '1';
```

Valid بودن دیتای دریافتی را چک می کنیم. Valid بودن این دیتا بر این اساس است که 8 بیت آخر not هشت بیت قبل از آن باشد. هنگامی که یک valid data دریافت کردیم، آن را در last_received_command ذخیره می کنیم.

```
--KEY_PRESS_TRACKER:
  process(clk_40MHz)
  begin

    if rising_edge(clk_40MHz) then

      --Once we receive a new packet, mark the given key as pressed.
      if packet_received = '1' then
        key_down <= '1';

      --If we've received a new command packet, we must be starting
      --a new button press. Indicate a key release.
      elsif command_packet_started = '1' then
        key_down <= '0';

      -- If we're able to reach the keypress timeout without a new
      -- key press occurring, then indicate a key release.
      elsif time_counter_value > KEY_PRESS_TIMEOUT  then
        key_down <= '0';
      end if;


    end if;
  end process;
```

در یک process دیگر که حساس به کلاک 40 مگاهرتز است بررسی می کنیم که آیا دکمه ای فشرده شده است یا خیر. اگر دکمه ای فشرده شده بود، packet_received یک می شود. اگر که یک packet دستور یا command دریافت کنیم، مقدار key_down را 0 قرار می دهیم.

در استیت WAIT_FOR_REPEAT_WINDOW پک تکرار را تشخیص می دهیم.

در استیت WAIT_FOR_IDLE مطمئن می شویم که core timer صفر می ماند.

یک WAIT_FOR_COMMAND نیز داریم که منتظر 16 بیتی میماند.

در حالت WAIT_FOR_END_OF_DATA_PULSE برای بررسی داده ورودی منتظر ورودی می مانیم تا صفر شود.

در استیت COUNT_DATA_PULSE_SPACING می شماریم که چقدر طول می کشد که سیگنال دیتا high شود.

در استیت آخر که PROCESS_PACKET نام دارد، پس از کامل شدن دریافت packet پیام را extract می کنیم.

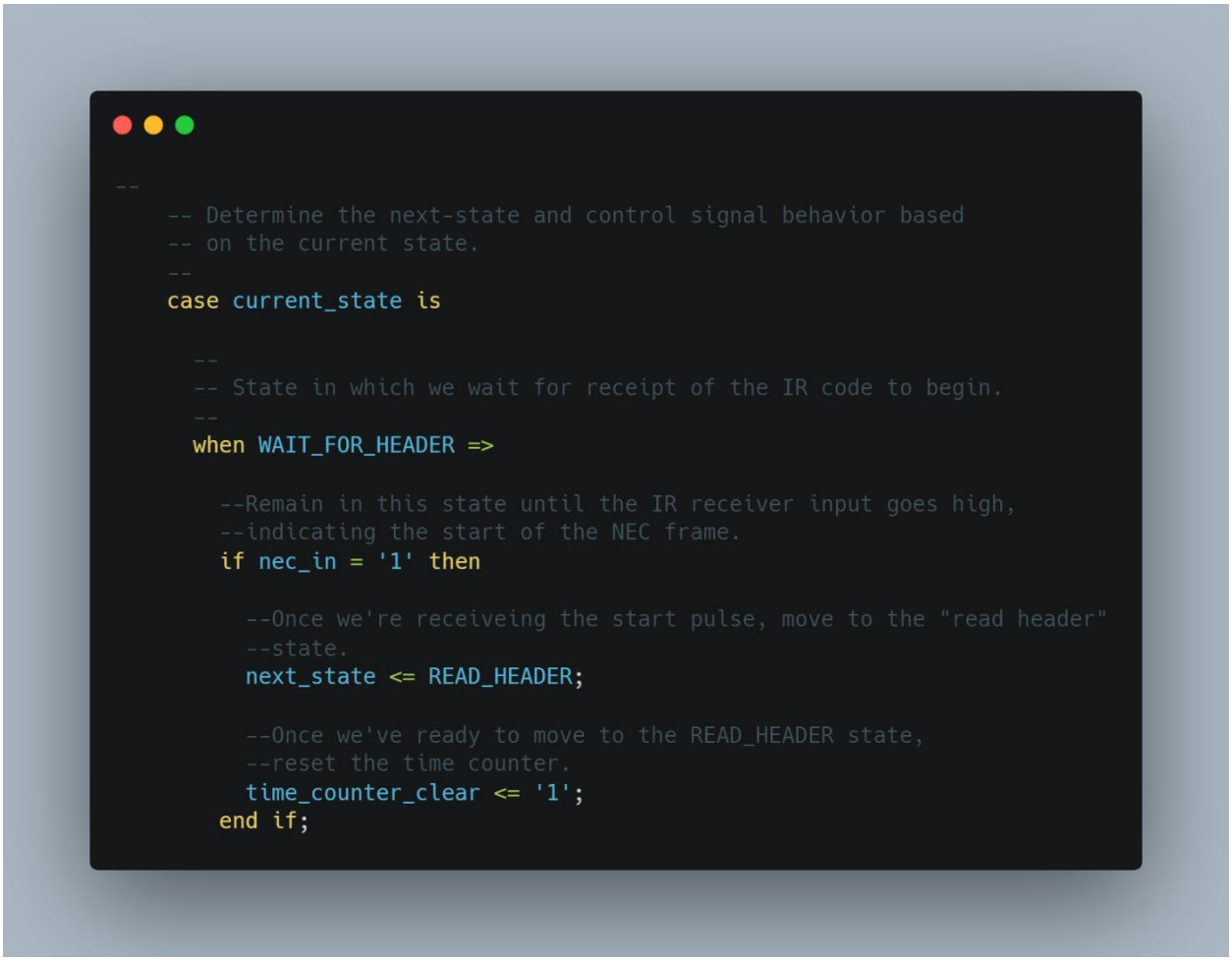
```
--
  -- Controller.
  --

  --Move to the next state at each clock edge.
  current_state <= next_state_with_reset when rising_edge(clk_40MHz);

  --FSM reset logic.
  next_state_with_reset <= WAIT_FOR_HEADER when reset = '1' else next_state;


  --
  -- Next-state and control logic.
  -- Note: This could be optimized further by preventing the counters from counting
  --       when not in use. This would incur the use of slightly more logic, but save
  --       dynamic power.
  --
  --
  process(nec_in, current_state, time_counter_value, pulse_counter_value)
  begin

    --Assume that the control signals are zero unless explicitly
    --asserted.
    packet_received <= '0';
    data_bit_received <= '0';
    time_counter_clear <= '0';
    pulse_counter_clear <= '0';
    repeat_code_received <= '0';
    command_packet_started <= '0';

    --Assume that we stay in the current state, unless the FSM
    --specifies otherwise. (Don't remove this! The behavior may
    --seem to be the same, but you'll cause the synthesis tools
    --to infer latches!)
    next_state <= current_state;
```

وارد بخش پیاده سازی controller می شویم. State ها را در current_state و next_state_with_reset ذخیره سازی می کنیم.

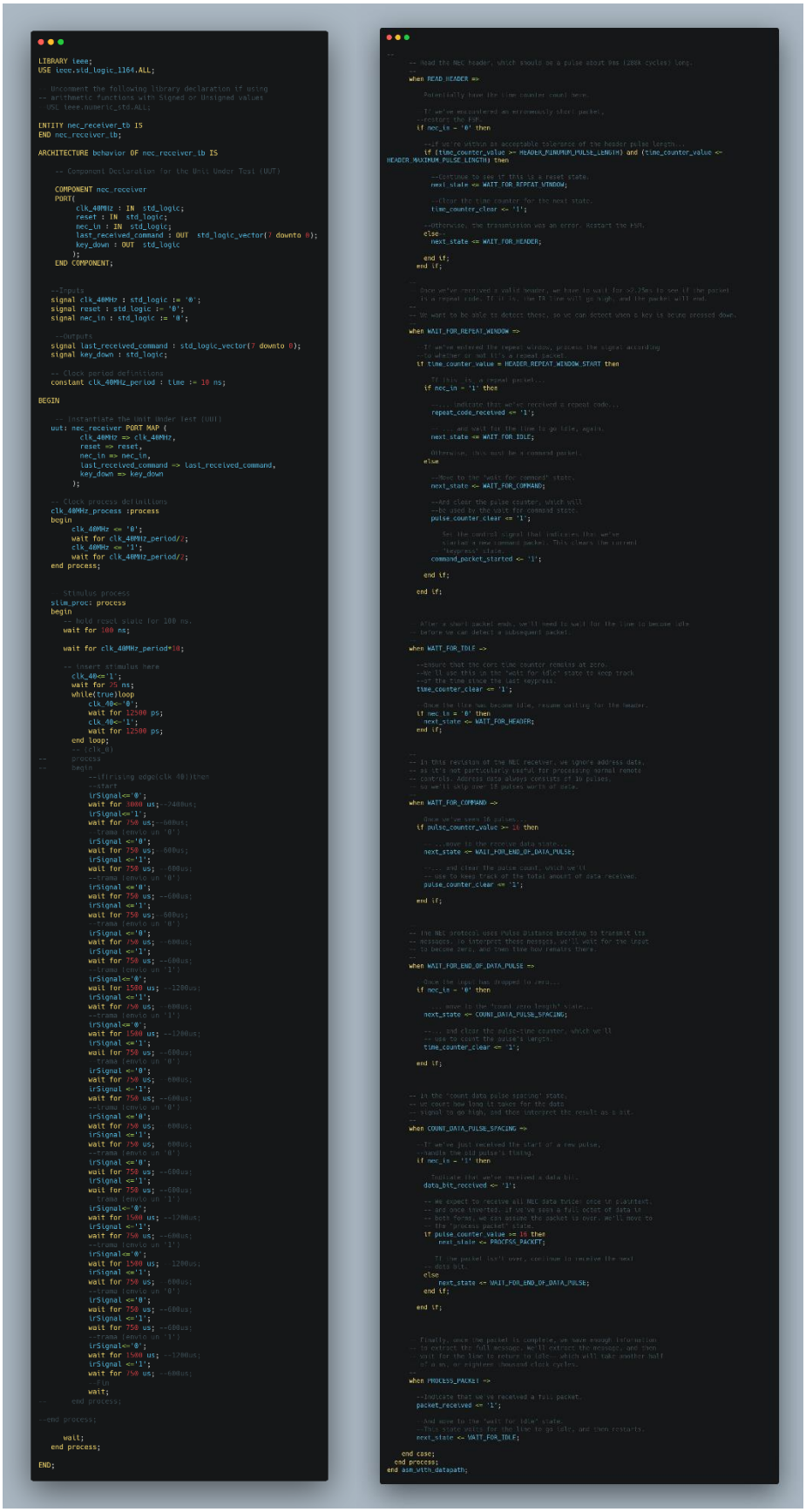
در یک process که به nec_in و current_state و time_counter_value و pulse_counter_value حساس است، منطق control logic استیت بعدی را پیاده سازی می کنیم. در این process مقادیر سیگنال ها را 0 مقدار دهی اولیه می کنیم.

با استفاده از caseها state بعدی را پیدا می کنیم. اگر WAIT_FOR_HEADER باشد، استیتی است که دریافت داده از IR آغاز می شود. در این استیت تا زمانی که IR receiver یک یا high شود می مانیم. به محض دریافت پالس شروع، به استیت read header می رویم.

در این state اگر به یک packet بسیار کوتاه رسیدیم restart می کنیم.

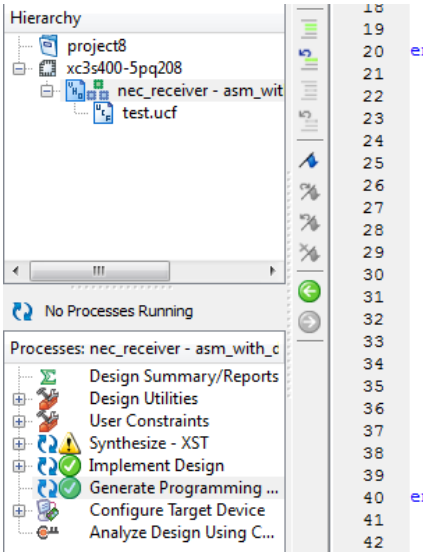


```
--
    -- Determine the next-state and control signal behavior based
    -- on the current state.
    --
    case current_state is

      --
      -- State in which we wait for receipt of the IR code to begin.
      --
      when WAIT_FOR_HEADER =>

        --Remain in this state until the IR receiver input goes high,
        --indicating the start of the NEC frame.
        if nec_in = '1' then

          --Once we're receiveing the start pulse, move to the "read header"
          --state.
          next_state <= READ_HEADER;

          --Once we've ready to move to the READ_HEADER state,
          --reset the time counter.
          time_counter_clear <= '1';
        end if;
```



```
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;

ENTITY nec_receiver_tb IS
END nec_receiver_tb;

ARCHITECTURE behavior OF nec_receiver_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT nec_receiver
    PORT(
         clk_40MHz : IN  std_logic;
         reset : IN  std_logic;
         nec_in : IN  std_logic;
         last_received_command : OUT  std_logic_vector(7 downto 0);
         key_down : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal clk_40MHz : std_logic := '0';
   signal reset : std_logic := '0';
   signal nec_in : std_logic := '0';

   --Outputs
   signal last_received_command : std_logic_vector(7 downto 0);
   signal key_down : std_logic;

   -- Clock period definitions
   constant clk_40MHz_period : time := 10 ns;

BEGIN

    -- Instantiate the Unit Under Test (UUT)
   uut: nec_receiver PORT MAP (
          clk_40MHz => clk_40MHz,
          reset => reset,
          nec_in => nec_in,
          last_received_command => last_received_command,
          key_down => key_down
        );

   -- Clock process definitions
   clk_40MHz_process :process
   begin
        clk_40MHz <= '0';
        wait for clk_40MHz_period/2;
        clk_40MHz <= '1';
        wait for clk_40MHz_period/2;
   end process;


   -- Stimulus process
   slim_proc: process
   begin
      -- hold reset state for 100 ns.
     wait for 100 ns;

     wait for clk_40MHz_period*10;

     -- insert stimulus here
       clk_40<='1';
       wait for 25 ns;
       while(true)loop
           clk_40<='0';
           wait for 12500 ps;
           clk_40<='1';
           wait for 12500 ps;
       end loop;
       -- clk_0)
--     process
--     begin
           --(flrising edge(clk 40))then
           --start
           irSignal<='0';
           wait for 3000 us;--7480us;
           irSignal<='1';
           wait for 750 us;--600us;
             --trama (envio un '0')
           irSignal <='0';
           wait for 750 us;  600us;
           irSignal <='1';
           wait for 750 us;  600us;
           --trama (envio un '0')
           irSignal <='0';
           wait for 750 us; --600us;
           irSignal <='1';
           wait for 750 us;  600us;
           --trama (envio un '0')
           irSignal <='0';
           wait for 750 us;  600us;
           irSignal <='1';
           wait for 750 us; --600us;
           --trama (envio un '1')
           irSignal='0';
           wait for 1580 us; --1200us;
           irSignal <='1';
           wait for 750 us;  600us;
           --trama (envio un '1')
           irSignal<='0';
           wait for 1500 us; --1200us;
           irSignal <='1';
           wait for 750 us; --600us;
             --trama (envio un '0')
           irSignal <='0';
           wait for 750 us;  600us;
           irSignal <='1';
           wait for 750 us; --600us;
           --trama (envio un '0')
           irSignal <='0';
           wait for 750 us;  600us;
           irSignal <='1';
           wait for 750 us;  600us;
           --trama (envio un '0')
           irSignal <='0';
           wait for 750 us; --600us;
           irSignal <='1';
           wait for 750 us; --600us;
           --trama (envio un '1')
           irSignal='0';
           wait for 1500 us; --1200us;
           irSignal <='1';
           wait for 750 us; --600us;
           --trama (envio un '1')
           irSignal<='0';
           wait for 1500 us;  1200us;
           irSignal <='1';
           wait for 750 us;  600us;
           --trama (envio un '0')
           irSignal <='0';
           wait for 750 us; --600us;
           irSignal <='1';
           wait for 750 us; --600us;
           --trama (envio un '1')
           irSignal<='0';
           wait for 1500 us; --1200us;
           irSignal <='1';
           wait for 750 us; --600us;
             fin
           wait;
--     end process;

--end process;

     wait;
   end process;

END;
```

```
--
    -- Read the NEC header, which should be a pulse about 9ms (288K cycles) long.
    --
    when READ_HEADER =>

      -- Potentially have the time counter count here.

      -- If we've encountered an erroneously short packet,
      -- restart the FSM.
      if nec_in = '0' then

        -- If we're within an acceptable tolerance of the header pulse length...
        if (time_counter_value >= HEADER_MINIMUM_PULSE_LENGTH) and (time_counter_value <=
HEADER_MAXIMUM_PULSE_LENGTH) then

          --Continue to see if this is a repeat state.
          next_state <= WAIT_FOR_REPEAT_WINDOW;

          --Clear the time counter for the next state.
          time_counter_clear <= '1';

        --Otherwise, the transmission was an error. Restart the FSM.
        else--
          next_state <= WAIT_FOR_HEADER;

        end if;
      end if;

      --
      -- Once we've received a valid header, we have to wait for >2.25ms to see if the packet
      -- is a repeat code. If it is, the TX line will go high, and the packet will end.
      --
      -- We want to be able to detect these, so we can detect when a key is being pressed down.
      --
    when WAIT_FOR_REPEAT_WINDOW =>

      -- If we've entered the repeat window, process the signal according
      --to whether or not it's a repeat packet.
      if time_counter_value = HEADER_REPEAT_WINDOW_START then

        -- If this _is_ a repeat packet...
        if nec_in = '1' then

          -- ... indicate that we've received a repeat code...
          repeat_code_received <= '1';

          -- ... and wait for the line to go idle, again.
          next_state <= WAIT_FOR_IDLE;

        -- Otherwise, this must be a command packet.
        else

          --Move to the 'wait for command' state.
          next_state <= WAIT_FOR_COMMAND;

          --And clear the pulse counter, which will
          --be used by the next few command states.
          pulse_counter_clear <= '1';

          -- Set the control signal that indicates that we've
          -- started a new command packet. This clears the current
          -- "keypress" state.
          command_packet_started <= '1';

        end if;

      end if;



      -- After a short packet ends, we'll need to wait for the line to become idle
    -- before we can detect a subsequent packet.
    --
    when WAIT_FOR_IDLE =>

      --ensure that the nec time counter remains at zero.
      --We'll use this in the "wait for idle" state to keep track
      --of the time since the last keypress.
      time_counter_clear <= '1';

      -- Once the line has become idle, resume waiting for the header.
      if nec_in = '0' then
        next_state <= WAIT_FOR_HEADER;
      end if;

      --
      -- In this revision of the NEC receiver, we ignore address data,
      -- as it's not particularly useful for processing normal remote
      -- controls. Address data always consists of 16 pulses,
      -- so we'll skip over 16 pulses worth of data.
      --
    when WAIT_FOR_COMMAND =>

        -- Once we've seen 16 pulses...
      if pulse_counter_value >= 16 then

        -- ... move to the receive data state...
        next_state <= WAIT_FOR_END_OF_DATA_PULSE;

        -- ... and clear the pulse count, which we'll
        -- use to keep track of the total amount of data received.
        pulse_counter_clear <= '1';

      end if;



    -- The NEC protocol uses Pulse Distance Encoding to transmit its
    -- messages. To interpret these messges, we'll wait for the input
    -- to become zero, and then time how remains there.
    --
    when WAIT_FOR_END_OF_DATA_PULSE =>

      -- Once the input has dropped to zero...
      if nec_in = '0' then

        -- ... move to the 'count zero length' state...
        next_state <= COUNT_DATA_PULSE_SPACING;

        -- ... and clear the pulse-time counter, which we'll
        -- use to count the pulse's length.
        time_counter_clear <= '1';

      end if;



    -- In the 'count data pulse spacing' state,
    -- we count how long it takes for the data
    -- signal to go high, and then interpret the result as a bit.
    --
    when COUNT_DATA_PULSE_SPACING =>

      --If we've just received the start of a new pulse,
      --handle the old pulse's timing.
      if nec_in = '1' then

        -- Indicate that we've received a data bit.
        data_bit_received <= '1';

        -- We expect to receive all NEC data pulses once in quadrant,
        -- and once inverted. If we've seen a full octet of data in
        -- both forms, we can assume the packet is over. We'll move to
        -- the 'process packet' state.
        if pulse_counter_value >= 16 then
            next_state <= PROCESS_PACKET;

        -- If the packet isn't over, continue to receive the next
        -- data bit.
        else
          next_state <= WAIT_FOR_END_OF_DATA_PULSE;
        end if;

      end if;



      -- Finally, once the packet is complete, we have enough information
      -- to extract the full message. We'll extract the message, and then
      -- wait for the line to return to idle... which will take another half
      -- of a 9-, or eighteen thousand clock cycles.
      --
    when PROCESS_PACKET =>

      --Indicate that we've received a full packet.
      packet_received <= '1';

      -- And move to the 'wait for idle' state.
      --This state waits for the line to go idle, and then restarts.
      next_state <= WAIT_FOR_IDLE;

    end case;
  end process;
end asm_with_datapath;
```




```
18
19
20  entity nec_receiver is -- asm_with_datapath
21    port(
22
23      --Main system clock signal; used for reading the NEC signal's timing.
24      clk_40MHz : in std_ulogic;
25
26      --Active high reset signal
27      reset : in std_ulogic := '0';
28
29      -- The NEC infrared input.
30      nec_in : in std_ulogic;
31
32      --The most recently received command.
33      last_received_command : out std_ulogic_vector(7 downto 0);
34
35      --High iff the given key is currently down, to the best of the
36      --receiver's knowledge.
37      key_down : out std_ulogic
38
39    );
40  end nec_receiver;
41
42
```