Electrical and Computer Engineering

Asynchronous Communication

By Prawat Nagvajara

Synopsis

This note covers an example of asynchronous communication (handshake protocol) between client and server processes. The example depicts a pedestrian lights controller that uses a timer (egg timer) services to time different light intervals. The controller and the timer are running on different clock signals. The control policy is to allow the traffic to flow at least certain amount of time, for example 5 minutes, before allowing pedestrian to cross.

Code

```
----------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- Handshake async communication
--
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity handshake is
Port(ck,reset,pedX_button: in std_logic;
     P,T : out std_logic_vector(2 downto 0));
end handshake;

architecture Behavioral of handshake is
-- T = max_count * F_clk
-- F_div = 1/T
-- F_clk = 100MHz
constant max_count_clk_div: natural:=1;-- 33333;--0.33333ms
signal max_count_timer : natural;-- 3000=1s, 15000=5s
------------------
-- client's flags
signal request_service, done_received: std_logic;
------------------
-- server's flags
signal request_received, service_done : std_logic;
signal start_job, job_completed       : std_logic;
------------------
-- Ped light controller flags and capture pedx button
signal start,ready, time_exp, pedX, reset_pedx : std_logic;
------------------
-- clock div
signal ck_div: std_logic;
begin
time_exp <= service_done;
-------------------------
-- pedX button capture into Flip-flop
-------------------------
process(ck)
begin
if ck='1' and ck'event then
 if reset_pedx = '1' then pedX <= '0';
 elsif pedX_button = '1' then pedX <= '1';
 end if;
end if;
end process;
-------------------------
-- Ped light controller State Machine uses egg timer
-------------------------
process(ck)
type my_state is (
-- Ped_light Traffic_light wait on timer exp
GR, wait_on_GR_exp,
YR, wait_on_YR_exp,
RG, wait_on_RG_exp, poll_PedX,
RY, wait_on_RY_exp);
variable n_s : my_state;
begin
if ck = '1' and ck'event then
  if reset = '1' then
  n_s:=GR; start<='0'; max_count_timer<=2;--6000;
  reset_pedx <= '1';
  else
  case n_s is -- "RYG" Red Yellow Green
 when GR => P <= "001"; T <= "100";
 if ready = '1' then n_s := wait_on_GR_exp;
   start <= '1'; max_count_timer <= 2;--6000;
   reset_pedx <= '0';
 else start  <= '0';
 end if;
 when wait_on_GR_exp =>
 P <= "001"; T <= "100"; start <= '0'; reset_pedx <= '0';
 if time_exp = '1' then n_s := YR; end if;

 when YR => P <= "010"; T <= "100";
 if ready = '1' then  n_s := wait_on_YR_exp;
   start <= '1'; max_count_timer <= 1;--3000;
   reset_pedx <= '0';
 else start  <= '0';
 end if;
 when wait_on_YR_exp =>
 P <= "010"; T <= "100"; start <= '0'; reset_pedx <= '0';
 if time_exp = '1' then n_s := RG; end if;
```

```
  when RG => P <= "100"; T <= "001";
  if ready = '1' then  n_s := wait_on_RG_exp;
    start <= '1'; max_count_timer <= 3;--15000;
    reset_pedx <= '0';
  else start  <= '0';
  end if;
  when wait_on_RG_exp =>
  P <= "100"; T <= "001"; start <= '0'; reset_pedx <= '0';
  if time_exp = '1' then n_s := poll_PedX; end if;
  when poll_PedX =>
  P <= "100"; T <= "001"; start <= '0';
  if pedX = '1' then n_s := RY; reset_pedx <= '1';
  else reset_pedx <= '0'; end if;

  when RY => P <= "100"; T <= "010";
  if ready = '1' then  n_s := wait_on_RY_exp;
    start <= '1'; max_count_timer <= 1;--3000;
    reset_pedx <= '0';
  else start  <= '0';
  end if;
  when wait_on_RY_exp =>
  P <= "100"; T <= "010"; start <= '0'; reset_pedx <= '0';
  if time_exp = '1' then n_s := GR; end if;

  end case;
 end if;
 end if;
 end process;
-------------------------
-- client
-------------------------
process(ck)
type my_state is (
-------------------------------------------------
--wait on server flag then action on transition
-------------------------------------------------
 wait_on_start,            -- then request service
 wait_on_request_received,--   then lower request
 wait_on_request_received_low,-- no action
 wait_on_service_done,   -- then flag done received
 wait_on_service_done_low--then lower done_received
 );
variable n_s : my_state;
```

```
begin
if ck = '1' and ck'event then
  if reset = '1' then
  n_s := wait_on_start;
  request_service <= '0';
  done_received <= '0';
  ready <= '1';
  else
  case n_s is
  when wait_on_start =>
  if start = '1' then
  n_s := wait_on_request_received;
  request_service <= '1';
  done_received <= '0';
  ready <= '0';
  else
  request_service <= '0';
  done_received <= '0';
  ready <= '1';
  end if;
 when wait_on_request_received =>
 if request_received = '1' then
 n_s := wait_on_request_received_low;
 -- stop requesting
 request_service <= '0';
 done_received <= '0';
 ready <= '0';
 else -- keep requesting
 request_service <= '1';
 done_received <= '0';
 ready <= '0';
 end if;
 when wait_on_request_received_low =>
 -- won't move if server doesn't low flag
 if request_received = '0' then
 n_s := wait_on_service_done;
 request_service <= '0';
 done_received <= '0';
 ready <= '0';
 else
 request_service <= '0';
 done_received <= '0';
 ready <= '0';
 end if;
```

```
  when wait_on_service_done =>
  if service_done = '1' then
  n_s := wait_on_service_done_low;
  request_service <= '0';
  done_received <= '1'; -- acknowledge received
  ready <= '0';
  else
  request_service <= '0';
  done_received <= '0';
  ready <= '0';
  end if;
  when wait_on_service_done_low =>
  -- won't move if server doesn't low done flag
  if service_done = '0' then
  n_s := wait_on_start;
  request_service <= '0';
  done_received <= '0'; -- lower done_received
  ready <= '0';
  else  -- keep sending done_received
  request_service <= '0';
  done_received <= '1';
  ready <= '0';
  end if;
  end case;
 end if;
end if;
end process;
--------------------------
--  server
--------------------------
process(ck_div)
type my_state is (
-----------------------------------------------
--wait on client flag then action on transition
-----------------------------------------------
 wait_on_request_service, --> acknowledge request
 wait_on_request_service_low,-->lower ack_request
 wait_on_job_completed,     --> send service_done
 wait_on_done_received,    --> lower service_done
 wait_on_done_received_low);        --> no action
 variable n_s : my_state;
```

```
begin
if ck_div = '1' and ck_div'event then
  if reset = '1' then
  n_s :=  wait_on_request_service;
  request_received <= '0';
  service_done <= '0';
  start_job <= '0';
  else
  case n_s is
  when wait_on_request_service =>
  if request_service = '1' then
  n_s := wait_on_request_service_low;
  request_received <= '1';
  service_done <= '0';
  start_job <= '0';
  else
  request_received <= '0';
  service_done <= '0';
  start_job <= '0';
  end if;
  when wait_on_request_service_low =>
  if request_service = '0' then
  n_s := wait_on_job_completed;
  request_received <= '0';
  service_done <= '0';
  start_job <= '1'; -- sync signaling
  else
  request_received <= '1';
  service_done <= '0';
  start_job <= '0';
  end if;
  when wait_on_job_completed =>
  if job_completed = '1' then -- sync signaling
  n_s := wait_on_done_received;
  request_received <= '0';
  service_done <= '1'; -- async signal to client
  start_job <= '0';
  else
  request_received <= '0';
  service_done <= '0';
  start_job <= '0';
  end if;
```

```
  when wait_on_done_received =>
  if done_received = '1' then
  n_s := wait_on_done_received_low;
  request_received <= '0';
  service_done <= '0';
  start_job <= '0';
  else
  request_received <= '0';
  service_done <= '1';
  start_job <= '0';
  end if;
  when wait_on_done_received_low =>
  if done_received = '0' then
  n_s := wait_on_request_service;
  request_received <= '0';
  service_done <= '0';
  start_job <= '0';
  else
  request_received <= '0';
  service_done <= '0';
  start_job <= '0';
  end if;
  end case;
 end if;
end if;
end process;
-----------------------
-- Server's job: Egg Timer
-- sync with server
-----------------------
process(ck_div)
variable count : natural;
type my_state is (counting, time_expired);
variable n_s : my_state;
begin
if ck_div = '1' and ck_div'event then
    if start_job = '1' then
      n_s := counting;
      count := max_count_timer;
      job_completed <= '0';
    else
    case n_s is
      when counting =>
        if count = 0 then
        -- flag job completed one cycle
            n_s := time_expired;
            job_completed <= '1';
        else
            count := count - 1;
        end if;
      when time_expired =>
        job_completed <= '0';
    end case;
    end if;
end if;
end process;

-----------------------
-- clock division
-----------------------
process(ck)
variable count : natural;
begin
if ck = '1' and ck'event then
    if reset = '1' then
        count := 0;
        ck_div <= '0';
    elsif reset = '0' then
        if count = max_count_clk_div then
            ck_div <= not ck_div;
            count := 0;
        else
            count := count + 1;
        end if;
    end if;
end if;
end process;

end Behavioral;
```