Univerzitet u Sarajevu
Elektrotehnički fakultet u Sarajevu
Odsjek za računarstvo i informatiku

Projektni zadatak

Tema: ***Ploča A sistema adaptivnog upravljanja saobraćajem***

Predmet: ***Projektovanje i sinteza digitalnih sistema***

Akademska godina: 2024/2025

Student: ***Nedim Kalajdžija, Sara Kardaš,***
Amila Kukić, Harun Goralija

Predmetni profesor:

Red. prof. dr. Amila Akagić, dipl. el. ing.

Predmetni asistenti:

Muhamed Zukić

Sadržaj

Dokumentacija FPGA Ploče A

1. Uvod: svrha i motivacija projekta

Pozadina i motivacija
U modernim gradovima adaptivno upravljanje saobraćajem omogućuje optimizaciju protoka vozila, smanjenje gužvi i povećanje sigurnosti. Ključni dio takvih sustava je pouzdano mjerenje brzine vozila u realnom vremenu na FPGA platformi. Ploča A se fokusira na očitavanje i pohranu mjerenja, dok Ploča B vrši predikciju i daljnju obradu.

Specifičan zadatak Ploče A
Ploča A očitava dva ultrazvučna senzora postavljena na poznatoj udaljenosti (D cm). Echo impulsi mjere se brojačem na taktu FPGA (npr. 50 MHz). Vrijeme između detekcije na senzoru 1 i senzoru 2 omogućuje izračun brzine: $v = d/(t2-t1)$

Ploča A također:

- Generira periodične trigger pulseve za senzore.
- Mjeri širinu echo signala i izračunava udaljenost dok objekt prolazi pored senzora.
- Mjeri interval između senzora za brzinu.
- Upozorava buzzerom ako je izračunata brzina > prag.
- Pohranjuje izmjerene brzine u RAM modul.
- Prikazuje posljednju brzinu na četiri seven-segment displeja.
- Komunicira s Pločom B putem UART (još nije implementirano) radi slanja pohranjenih podataka.

Ciljevi dokumentacije

- Detaljno opisati module prema stvarnoj VHDL implementaciji.
- Obezbijediti informacije za održavanje i eventualne nadogradnje.
- Pomoć u verifikaciji i integraciji s Pločom B.

2. Specifikacija Ploče A

2.1 Funkcionalni zahtjevi

- **MODE kontrola**: preko ulaza `sw0`; '1' = INACTIVE, '0' = SEMI_ACTIVE (aktivno mjerenje i alarm, ali bez slanja podataka).
- **Generiranje triggera**: širina `TRIGGER_WIDTH` (500 ciklusa), period `TRIGGER_PERIOD` (7_000_000 ciklusa) na taktu FPGA.
- **Mjerenje echo signala**: brojač echo1_width i echo2_width mjeri trajanje impulsa.

- **Izračun udaljenosti**: `distance = echo_width / 2900` (približno, pri 50 MHz taktu i brzini zvuka).
- **State Machine**: stanja IDLE, WAITING_FOR_FIRST, TIMING, CALCULATE_SPEED, DISPLAY_RESULT.
- **Izračun brzine**:
    - Ako `timer > 50_000` (1 ms),
      [speed = rac{DISTANCE_BETWEEN_SENSORS imes 50,000,000}{timer}]
    - Inače postaviti `speed = 999` (maksimalna prikazana).
- **Alarm**: buzzer aktivan 3 sekunde u 5-sekundnom prikazu ako `speed > SPEED_LIMIT`.
- **Prikaz**: četiri seven-segment modula (`hex1`, `hex2`, `hex3`, `hex4`), pri čemu `hex4` inicijalno prikazuje blank ("1111111").
- **RAM pohrana**: modul ram_module s 256 lokacija (8-bit podaci).
- **Komunikacija**: UART interfejs planiran, još nije integriran u kod.
- **LED indikatori**: `led1`, `led2` pokazuju detekciju objekta na senzoru 1 i 2.

2.2 Konstante iz koda

- `DISTANCE_BETWEEN_SENSORS`: integer := 10 (cm).
- `TRIGGER_WIDTH`: integer := 500 (ciklusa).
- `TRIGGER_PERIOD`: integer := 7_000_000 (ciklusa).
- `DETECTION_THRESHOLD`: integer := 30 (cm).
- `SPEED_LIMIT`: integer := 50 (cm/s).
- Takt FPGA: implicitno 50 MHz (na osnovu perioda i kalkulacija).

3. Arhitektura Ploče A

3.1 Blok dijagram

- **Top-level**: entitet `speed_measure`.
- **Moduli**:
    - Trigger & Echo mjerenje (integrirano unutar top-level koda).
    - State Machine unutar `speed_measure`.
    - RAM modul: `ram_module`.
    - Seven-segment dekoder: `sevenDisplay`.
    - LED i buzzer logika.
    - UART interfejs (predviđeno, još nije implementirano).

3.2 Signalni tok

- `clk` sinkronizira sve registre i brojače.
- Generiranje triggera: periodični impulsi na `trig1`, `trig2`.
- Echo mjerenje: echo1_width i echo2_width brojači resetiraju se kad signal padne, distance se izračuna.

- State Machine prati distance signale i upravlja timer.
- Nakon izračuna brzine, piše se u RAM i prikazuje na seven-segment.
- LED i buzzer logika bazirana na signalima iz state machine.
- UART se može dodati za čitanje iz RAM-a i slanje van.

4. Detaljni opis modula

4.1 Modul RAM: `ram_module`

```
entity ram_module is
    Port (
        clk             : in  std_logic;
        we              : in  std_logic;                          -- write enable
        data_in         : in  std_logic_vector(7 downto 0);       -- podaci za upis
        data_out        : out std_logic_vector(7 downto 0);       -- podaci za čitanje
        first_write_addr : out std_logic_vector(7 downto 0);      -- prva adresa na koju je upisano
        last_write_addr  : out std_logic_vector(7 downto 0)       -- poslednja adresa na koju je up
    );
end ram_module;

architecture Behavioral of ram_module is
    type ram_type is array (0 to 255) of std_logic_vector(7 downto 0);
    signal ram : ram_type := (others => (others => '0'));
    signal first_written : boolean := false;
    signal first_addr    : std_logic_vector(7 downto 0) := (others => '0');
    signal last_addr     : std_logic_vector(7 downto 0) := (others => '0');
    signal write_ptr     : std_logic_vector(7 downto 0) := (others => '0'); -- internal address counter
begin

    process(clk)
    begin
        if rising_edge(clk) then
            if we = '1' then
                -- Upis na adresu write_ptr
                ram(to_integer(unsigned(write_ptr))) <= data_in;
                -- Čuvanje prve i posljednje adrese upisa
                last_addr <= write_ptr;
                if not first_written then
                    first_addr <= write_ptr;
                    first_written <= true;
                end if;
                -- Povećanje write_ptr ili wrap-around
```

```
            if write_ptr = "11111111" then
                write_ptr <= (others => '0');
            else
                write_ptr <= std_logic_vector(unsigned(write_ptr) + 1);
            end if;
        end if;
        -- Uvijek čitanje s posljednje upisane adrese
        data_out <= ram(to_integer(unsigned(last_addr)));
    end if;
  end process;

  -- Izlazi entiteta
  first_write_addr <= first_addr;
  last_write_addr  <= last_addr;
end Behavioral;
```

- **Opis**:
  - Memorija dubine 256, širina 8 bita.
  - `we` kontrolira upis; svaki upis pomiče `write_ptr`, čuva `first_addr` i `last_addr`.
  - `data_out` uvijek vraća sadržaj na `last_addr`.
  - Wrap-around kada se dostigne adresa 255.
  - `first_write_addr` signalizira adresu prvog upisa nakon resetiranja FPGA.

4.2 Modul seven-segment: `sevenDisplay`

```
entity sevenDisplay is
  port (
    BCD: in std_logic_vector(3 downto 0);
    HEX: out std_logic_vector(6 downto 0)
  );
end sevenDisplay;

architecture segment of sevenDisplay is
begin
  HEX <= "1000000" when (BCD = "0000") else -- 0
       "1111001" when (BCD = "0001") else -- 1
       "0100100" when (BCD = "0010") else -- 2
       "0110000" when (BCD = "0011") else -- 3
       "0011001" when (BCD = "0100") else -- 4
       "0010010" when (BCD = "0101") else -- 5
       "0000010" when (BCD = "0110") else -- 6
       "1111000" when (BCD = "0111") else -- 7
```

```
        "0000000" when (BCD = "1000") else -- 8
        "0010000" when (BCD = "1001") else -- 9
        "1111111"; -- blank ili minus
end segment;
```

- **Opis**:
  - Kombinacijski mapping BCD → kod segmenata.
  - Zadnja opcija `"1111111"` za nenormalne vrijednosti (prikazuje blank ili minus).
  - Aktivni nivo za segment upaljen: '0' (ovisno o hardveru).

4.3 Top-level entitet: `speed_measure`

```
entity speed_measure is
   port(
      clk: in std_logic;
      echo1: in std_logic;
      echo2: in std_logic;
      sw0: in std_logic;
      trig1: out std_logic;
      trig2: out std_logic;
      led1: out std_logic;
      led2: out std_logic;
      hex3, hex2, hex1, hex4: out std_logic_vector(6 downto 0);
      buzzer: out std_logic
   );
end speed_measure;
```

- **Opis portova**:
  - `clk`: taktski signal (50 MHz).
  - `echo1`, `echo2`: echo impulsi ultrazvučnih senzora.
  - `sw0`: kontrola stanja (INACTIVE/SEMI_ACTIVE).
  - `trig1`, `trig2`: izlazni trigger impulsi.
  - `led1`, `led2`: signalizacija detekcije objekta.
  - `hex1`, `hex2`, `hex3`, `hex4`: seven-segment prikazi.
  - `buzzer`: aktivacija zvuka pri prekoračenju.

Arhitektura i signali

- **Konstante**: definirane kao u kodu (`DISTANCE_BETWEEN_SENSORS`, `TRIGGER_WIDTH`, itd.).
- **Tipovi i signali**:
  - State machine: `state_type` (IDLE, WAITING_FOR_FIRST, TIMING, CALCULATE_SPEED, DISPLAY_RESULT).

- Brojači: `period1`, `period2`, `echo1_width`, `echo2_width`, `timer`, `display_timer`, `buzzer_timer`.
- Detekcija: `object1_detected`, `object2_detected`.
- Brzina: `speed`, `speed_calculated`, `speed_exceeded`, `buzzer_active`.
- RAM signali: `ram_we`, `ram_data_in`, `ram_data_out`, `ram_first_addr`, `ram_last_addr`.
- Kontrola: `traffic_monitoring_enabled`, `alarm_enabled`, `board_state`.

Trigger & Echo mjerenje

- **Periodični brojači** `period1` i `period2`: resetiraju se kad dostignu `TRIGGER_PERIOD` i enable je aktivan.
- **Trigger output**: `'1'` dok `period < TRIGGER_WIDTH`.
- **Echo width mjerenje**: kad `echo='1'`, brojač inkrementira; kad `echo='0'` i width>0, distance izračuna `echo_width/2900`.
- Reset echo_width nakon mjerenja ili kad monitoring isključen.

State Machine logika

- **IDLE**:
  - Sve signale resetira.
  - Ako `traffic_monitoring_enabled='1'`, prelazi u WAITING_FOR_FIRST.
- **WAITING_FOR_FIRST**:
  - Čeka `distance1 < DETECTION_THRESHOLD` i `object1_detected='0'`.
  - Ako detektirano, postavlja `object1_detected_next='1'`, reset timer, `timing_active='1'`, prelazi u TIMING.
- **TIMING**:
  - Ako `timing_active='1'`, `timer_next <= timer+1`.
  - Ako `timer > 100_000_000`, timeout, vraća se u IDLE.
  - Ako `distance2 < DETECTION_THRESHOLD` i `object2_detected='0'`, postavlja `object2_detected_next='1'`, `timing_active_next='0'`, `measured_time<=timer`, prelazi u CALCULATE_SPEED.
- **CALCULATE_SPEED**:
  - Ako `timer > 50_000`, računa `speed_next <= (DISTANCE_BETWEEN_SENSORS * 50_000_000) / timer`.
    - Ako > `SPEED_LIMIT`: `speed_exceeded_next='1'`, `buzzer_active_next='1'`.
    - Inače: `speed_exceeded_next='0'`, `buzzer_active_next='0'`.
  - Inače (`timer <= 50_000`): `speed_next<=999`, `speed_exceeded_next='1'`, `buzzer_active_next='1'`.
  - `speed_calculated_next='1'`, reset `display_timer_next`.
  - Piše u RAM: `ram_we<='1'`, `ram_data_in<=std_logic_vector(to_unsigned(speed_next,8))`.
  - Prelazi u DISPLAY_RESULT.

- **DISPLAY_RESULT**:
  - `ram_we<='0'`.
  - Ako `display_timer < 250_000_000`, incremenitra `display_timer_next`; kontroliše `buzzer_timer_next` dok `<150_000_000`.
  - Inače prelazi u IDLE.
- **Default**: vraća u IDLE.

RAM integracija

```
ram_inst: ram_module port map (
   clk            => clk,
   we             => ram_we,
   data_in         => ram_data_in,
   data_out        => ram_data_out,
   first_write_addr => ram_first_addr,
   last_write_addr  => ram_last_addr
);
```

- **Podržava**: pohranu sekvencijalnih mjerenja s wrap-around, vraća posljednju vrijednost na `data_out`.
- **first_write_addr** i **last_write_addr** dostupni, mogu se koristiti za slanje više mjerenja preko UART-a.

Seven-segment integracija

```
hex4 <= "1111111"; -- blank/minus
seg1_disp: sevenDisplay port map(
   BCD => ram_data_out(3 downto 0),
   HEX => hex1
);
seg2_disp: sevenDisplay port map(
   BCD => std_logic_vector(to_unsigned((to_integer(unsigned(ram_data_out)) / 10) mod 10,
4)),
   HEX => hex2
);
seg3_disp: sevenDisplay port map(
   BCD => std_logic_vector(to_unsigned((to_integer(unsigned(ram_data_out)) / 100) mod 10,
4)),
   HEX => hex3
);
-- hex4 je blank ili minus ako se želi signalizirati specijalno
```

- **Opis**:
  - Čita 8-bit `ram_data_out`, pretvara u decimalne cifre.

- `hex1` = jedinice, `hex2` = desetice, `hex3` = stotine, `hex4` ostaje blank.

LED i buzzer

- LED:
  led1 <= object1_detected when traffic_monitoring_enabled = '1' else '0';
  led2 <= object2_detected when traffic_monitoring_enabled = '1' else '0';
- Buzzer:
  buzzer <= '0' when (current_state = DISPLAY_RESULT and speed_exceeded = '1' and buzzer_active = '1' and alarm_enabled = '1') else '1';

- **Napomena**: aktivno nisko ('0' pali buzzer), ovisno o hardveru.

4.4 Kontrola stanja ploče

- `process(sw0)` upravlja `board_state`:
  - `'1'` => INACTIVE, `'0'` => SEMI_ACTIVE.
- `traffic_monitoring_enabled` i `alarm_enabled` deriviraju se iz `board_state`.
- Moguće proširenje: dodatni switch-i za ACTIVE s komunikacijom.

4.5 UART interfejs

- UART interfejs je implementiran kroz dodatni entitet `uart_sender` koji upravlja serijskim slanjem podataka iz RAM memorije preko linije `uart_tx_out`. Slanje se inicira ulaznim signalom `send_trigger`, a status toka prenosa se prikazuje signalom `uart_busy`. Modul koristi interni FSM za sekvencijalno čitanje iz RAM-a i slanje svakog bajta preko `UART_TX` komponente.

  ### Entitet uart_sender

  ```
  entity uart_sender is
  Port (
          clk             : in  std_logic;
          send_trigger    : in  std_logic;  -- High za start slanja
          ram_data_out    : in  std_logic_vector(7 downto 0);
          ram_last_addr   : in  std_logic_vector(7 downto 0);
          uart_tx_out         : out std_logic;
          ram_addr        : out std_logic_vector(7 downto 0);
          uart_busy       : out std_logic   -- '1' dok traje slanje
          );
  end uart_sender;
  ```

  FSM stanja:

- **IDLE** – čeka aktivaciju `send_trigger`;
- **READ** – čita RAM sadržaj na adresi `read_address`;
- **SEND** – šalje bajt preko UART (aktivira `tx_dv`);
- **WAIT_DONE** – čeka signal `tx_done`;
- **DONE** – vraća se u IDLE nakon što se pošalje bajt na `ram_last_addr`.

Dodatni modul: `UART_TX`

Ova komponenta (`UART_TX`) implementira nizak nivo UART protokola: generiše start bit, 8-bit podatak i stop bit pri 115200 baud (uz `g_CLKS_PER_BIT = 434` za takt 50 MHz).

Interakcija sa RAM modulom

- `ram_data_out` prenosi vrijednost sa adrese `ram_addr`;
- `ram_last_addr` označava krajnju adresu slanja;
- Interno brojač `read_address` se koristi za adresiranje RAM-a tokom slanja;
- Nakon što je poslata vrijednost na `ram_last_addr`, FSM prelazi u stanje `DONE` i deaktivira `uart_busy`.

Integracija s RAM-om

RAM modul (`ram_module`) je proširen tako da podržava signal `read_addr` i dodatni izlaz `data_from_addr`. Ovo omogućava `uart_sender` modulu da čita sadržaj memorije na adresi `read_address`, nezavisno od `data_out` koji daje posljednju upisanu vrijednost.

Signalni dijagram

- `send_trigger = '1'` pokreće slanje;
- `uart_busy = '1'` dok traje slanje;
- `ram_addr` adresira sljedeći bajt za slanje;
- `uart_tx_out` prenosi serijski podatak (već priključen na vanjsku UART liniju);
- `tx_done` detektuje kraj prenosa jednog bajta.

5. Verifikacija i testiranje

U procesu implementacije projekta izvedeno je ručno testiranje svih modula te međusobne interakcije pojedinih, pri čemu je korišten veliki broj različitih ulaznih parametara i nasumično kombinovanje istih. Usljed nedostatka vremena pri implementaciji projekta, testbench simulacije su ostale ne-implementirane u ovom stadiju, te se u narednim podpoglavljima opisuje samo plan njihove implementacije, ali ne i stvarna izvedba.

5.1 Testbench za ram_module

- Simulirati:

    - Niz upisa podataka: provjeriti `first_write_addr` postavlja se na adresu prvog upisa.
    - Wrap-around: upis >256 puta vraća pointer na 0.
    - Čitanje `data_out` vraća posljednji upis.
- Assert primjeri:

```
assert first_write_addr = x"00" report "Prvi upis adresa" severity error;
assert last_write_addr = x"01" report "Drugi upis adresa" severity error;
```

5.2 Testbench za sevenDisplay

- Provjeriti mapping BCD ulaza 0–9 generira točan `HEX`.
- Ulazi izvan 0–9 => `HEX="1111111"`.

5.3 Testbench za speed_measure

- Scenariji:
    - Simulirani echo impulsi: definirati ulazne impulse za echo1 i echo2 s određenim latencama.
    - Provjeriti state transitions: IDLE→WAITING_FOR_FIRST→TIMING→CALCULATE_SPEED→DISPLAY_RESULT→IDLE.
    - Provjera `ram_data_out` s odgovarajućom zapisom brzine.
    - LED i buzzer signali prema očekivanju.
    - Timeout scenarij (echo2 ne stiže).
- Mogućnost simulacije slanja UART podataka: dummy handshake.

5.4 Testbench za uart_sender

- Scenariji testiranja:
    - `send_trigger='1'` i `ram_last_addr="00000010"` → treba poslati tri bajta;
    - Simulirati različite podatke u RAM memoriji na adresama 0, 1, 2;
    - Provjeriti da `uart_tx_out` daje očekivani niz bitova (start, data, stop);
    - Provjeriti da `uart_busy='1'` dok traje slanje, a zatim se vraća u `0`;
    - Provjeriti da `ram_addr` ide od 0 do `ram_last_addr`.

6. Pin assignment i hardware

- Navesti pinove FPGA za:
    - `echo1`, `echo2`, `trig1`, `trig2`
    - `led1`, `led2`
    - `hex1`–`hex4` (segmentni i enable pinovi)
    - `buzzer`
    - `sw0`

    - UART TX/RX pinovi
- Reference na constraints fajl.

7. Integracija s Pločom B

- Definirati format poruke: npr. svaki bajt predstavlja mjerenje brzine ili kontrolne okvire.
- Handshake: signal `send_request` od strane Ploče B.
- Čitanje RAM memorije od `first_write_addr` do `last_write_addr`.
- Iskoristiti implementirani UART sa obe strane za uspostavljanje komunikacije

8. Zaključak i buduće nadogradnje

- Dokument opisuje stvarnu implementaciju modula prema dostavljenom VHDL kodu.
- Buduće: parametrizacija pragova, testbench slučajevi za sve module, onakvi kakvi su planirani, ali radi nedostatka vremena nisu implementirani.