Osnove računarske tehnike - LPRS
Vežba 4
Napredne sekvencijalne mreže
Huffman kodovanje

Miloš Subotić

4. decembar 2015

1 Uvod

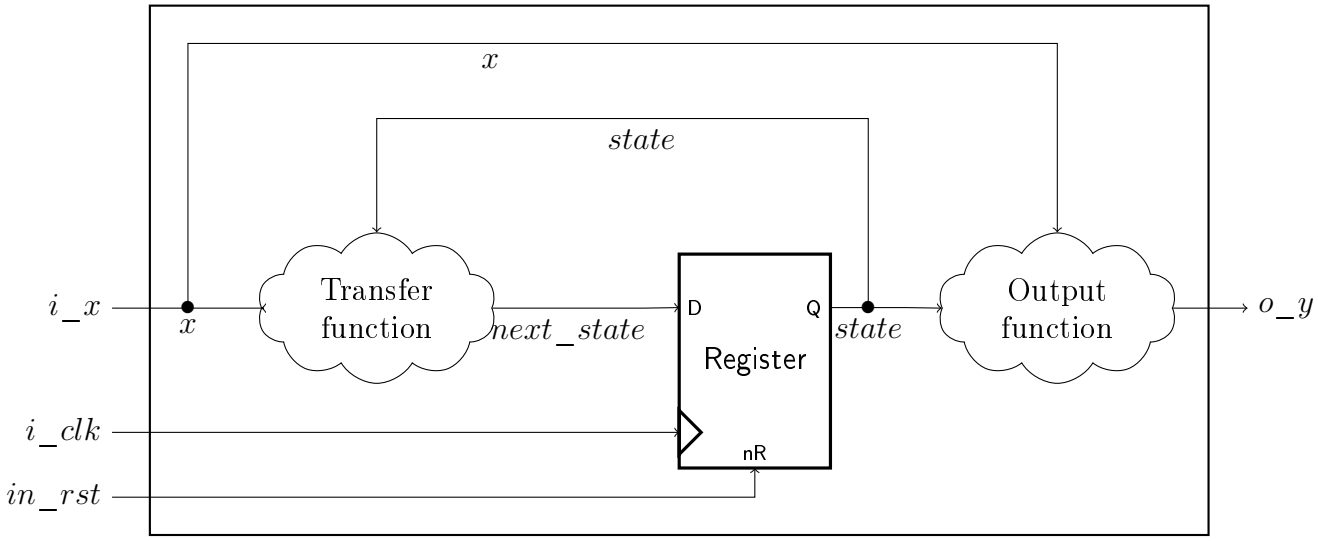
Projektovanje kompleksnijih sekvencijalnih mreža će biti prikazano na primeru realizacije Huffman kodovanja za statističku kompresiju podataka.

2 Automati

Automat sa konačnim brojem stanja je generalizovana sekvencijalna mreža, pogodnog oblika za projektovanje. Automat se sastoji od registra koji čuva trenutno stanje automata *state*, funkcije prelaza koja je kombinaciona mreža za određivanje narednog stanja *next_state* na osnovu ulaza *i_x* odnosno *x* i trenutnog stanja *state*, i izlazne funkcije koja je kombinaciona mreža za određivanje izlaza *o_y* na osnovu ulaza *i_x* odnosno *x* i trenutnog stanja *state*. Blok šema automata je prikazana na Slici 1.

Neka je na primer potrebno izraditi osnovni automat za rešenje zadatka koji će vršiti prihvat podataka sa AXI Stream magistrale, davati 4-bitne simbole na izlazu, upravljati kontrolne signale, upravljati brojačem faze. Na Slici 2 je prikazan diagram stanja automata.

Početno stanje je **`NEW_CHAR`**. U tom stanju izlaz **`s_axis_tready`** je postavljen na 1 što govori spoljnom rukovaocu (engl. Bus Master) da je primalac spreman za prihvat podataka (engl. Bus Slave). U tom stanju se očekuje da je signal **`s_axis_tvalid`** posavljen na 1 što označava da je novi bajt postavljen na **`s_axis_tdata`**. Kada su postavljeni **`s_axis_tvalid`** i **`s_axis_tready`** na 1 to je znak za rukovaoca i primaoca da je podatak preuzet. Kada je automat u



Slika 1: Automat

stanju **NEW_CHAR** na **o_sym** se prosleđuju donja 4 biti ulaznog bajta. Kada je automat u stanju **NEW_CHAR** i **s_axis_tvalid** posavljen na 1 potrebno je sačuvati gornja 4 bita ulaznog bajta, koji će biti prosleđeni na **o_sym** u idućem taktu.

Kada je **s_axis_tvalid** posavljen na 1, što je prikazano na luku na Slici 2, prelazi se u stanje **UPPER_NIBBLE**. Sada se **s_axis_tready** postavlja na 0 kako bi zaustavili prihvat podataka sa AXI Stream magistrale, jer još nismo prosledili gornji simbol. U tom stanju na **o_sym** izlaz se prosleđuje gornji simbol. Iz to tog stanja se uobičajeno prelazi ponovo u **NEW_CHAR**, ali ako je brojač faze **stage** 15 potrebno je preći u stanje **LAST_STAGE**.

LAST_STAGE stanje služi da se potroši jedan takt viška dok je brojač faze **stage** jednak 16. Kada je brojač faze **stage** 16 onda se podaci iz jednog bloka obrade prosleđuju u drugi blok obrade. Iz ovog stanja se prelazi direktno u **NEW_CHAR**.

Za realizaciju prvo je potrebno definisati tip sa enumerisanim stanjima automata, kao na Listingu 1.

Naredni deo je definisanje signala potrebnih za automat kao na Listingu 2. Ovi signali će biti tipa **t_states**. Za početak su definisani samo trenutno stanje **state** i naredno stanje **next_state**, ali moguće je po potrebi definisati i još neke signale.

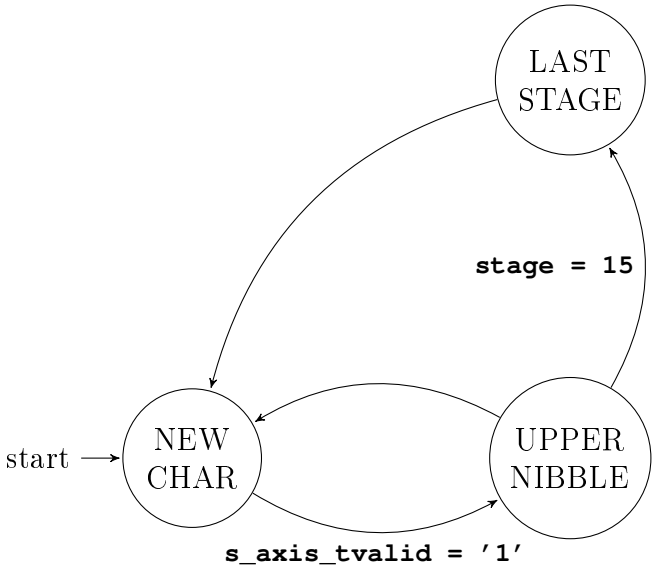
Dalje je potrebno definisati registar za čuvanje stanja automata. VHDL sintaksa za definisanje registra je data na Listingu 3. Linija **state <= NEW_CHAR;** postavlja početno stanje kada je reset aktivan (**in_rst = '0'**). Linija **state <= next_state;** označava da je ulaz registra **next_state** saču-

```
...
architecture ...

    type t_states is (
        NEW_CHAR,
        UPPER_NIBBLE,
        LAST_STAGE
    );

    ...

begin
    ...
```

Listing 1: Tip sa enumerisanim stanjima automata

```
...
architecture ...
    ...

    signal state      : t_states;
    signal next_state : t_states;

    ...
begin
    ...
```

Listing 2: Tip sa enumerisanim stanjima automata



Slika 2: Osnovni automat za rešenje zadatka

van na rastuću ivicu takta (`rising_edge(i_clk)`). Izlaz registra je `state`.

Funkcija prelaza i izlaza se realizuju kao klasične kombinacione mreže. Praktičano je praviti multiplekser `with state select` i po trenutnom stanju određivati koji prosleđivati konkretnu vrednost ili izračunat signal.

`o_sym` je multipleksiran na donja 4 bita ulaznog podatka u `NEW_CHAR`, dok je u stanju `UPPER_NIBBLE` postavljen na gornja 4 bita sačuvana u registar u `NEW_CHAR` stanju.

`o_pipe_en` signal bi trebao da je uvek posavljen na 1 sem u slučaju kad je trenutno stanje `NEW_CHAR` i nema validnog ulaza tj. `s_axis_tvalid` je 0.

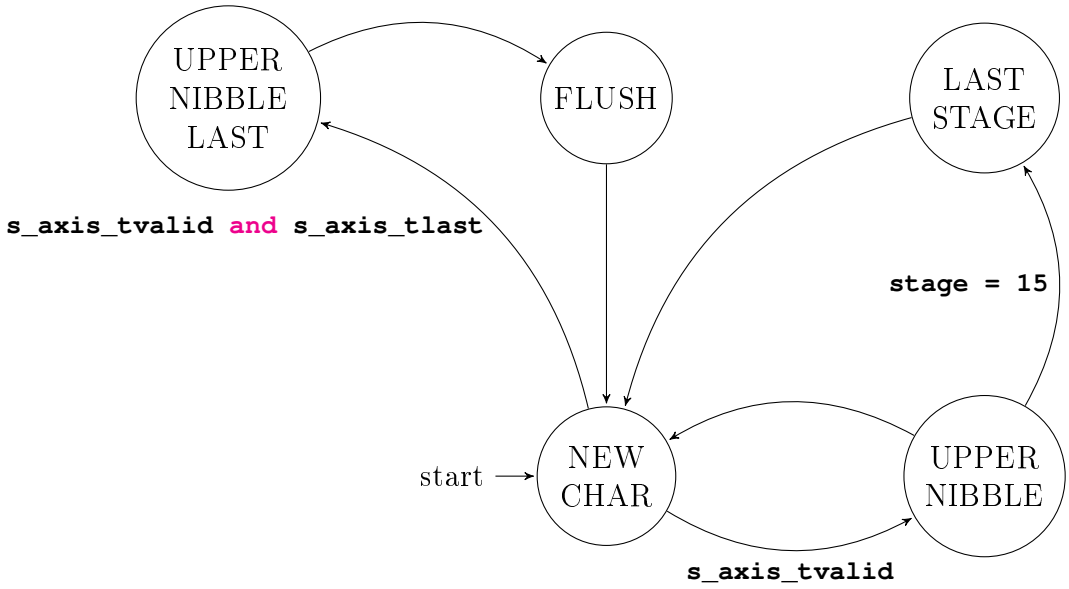
```
...
begin

    process(i_clk, in_rst)
    begin
        if in_rst = '0' then
            state <= NEW_CHAR;
        elsif rising_edge(i_clk) then
            state <= next_state;
        end if;
    end process;
    ...
```

Listing 3: Registar stanja automata

Još je potrebno realizovati brojač faze da broji od 0 do 16. Njegov signal dozvole se isto ponaša kao i **o_pipe_en**.

3 Zadatak

Na Slici 3 je prikazan automat koji je potrebno implementirati u prvom bloku (**text2sym_conv_and_stage_cnt**). Razlika u odnosu na automat sa slike Slici 2 je da postoje 2 nova stanja. U stanje **UPPER_NIBBLE_LAST** se prelazi kada su **s_axis_tvalid** i **s_axis_tlast** postavljeni na 1. **s_axis_tlast** označava da je na AXI Stream magistrali zadnji bajt zadnjeg bloka. U **UPPER_NIBBLE_LAST** stanju se prosleđuje zadnji simbol na izlaz i prelazi se u **FLUSH** stanje, u kom bi trebalo da se čeka sve dok se celokupan sistem ne isprazni. Dok je trenutno stanje **FLUSH** potrebno je postaviti izlaz **o_pipe_flush** na 1. Iz stanja **FLUSH** bi trebalo izaći nakon nekoliko ciklusa od 17 taktova. Zasad to nije bitno realizovati jer još nema dovoljno blokova obrade koji zahtevaju pražnjenje.



Slika 3: Automat za rešenje zadatka

Pored ovo ulaznog bloka potrebno je realizovati **histogram**. **histogram** treba da broji koliko se koji simbol pojavio dok je **o_pipe_en**. Kada je **i_stage** 16 proslediti u naredni blok za sortiranje. Koristiti tipove iz ***global.vhd*** kao što su **t_sym** i **t_freq_array**.

U bloku za sortiranje koristiti paralelizovani bubble sort algoritam za sortiranje. Potrebno je naizmenično porediti i zamenjivati ako je potrebno

elemente na sledeći način: ako je parna faza (gledati najniži bit **`i_stage`** signala) onda porediti parove 0 i 1, 2 i 3, 4 i 5 itd. dok ako je neparna faza porediti parove 1 i 2, 3 i 4, 5 i 6 itd. Pogodno je koristiti strukturu **`t_sym_and_freq`** za lakšu realizaciju.

Koristiti C++ model ***huffman_coding.cpp***, ali pre svega ***huffman_coding.log*** kao zlatnu referencu za realizaciju algoritma. C++ model i ***huffman_encoder_tb.vhd*** imaju istu pobudu na ulazu, tako da je moguće proveriti rezultate određenih blokova.

Obavezna je izrada blok diagrama rešenja i po potrebi šemtaski diagram nekih blokova. Nije potreban šematski diagram opštepoznatih blokova kao što je brojač. Nije potrebno crtati kompletnu šemu za svih 16 elemenata, dovoljno je prikazati samo par njih a one između zamisliti da postoje i razdvojiti tačkicama.