Licznik symboli na platformie ZYBO

Bartosz Bryk, Katarzyna Ladra

Opis projektu

Projekt zakładał stworzenie dedykowanego procesora do zliczania symboli w kodzie ASCII. Taki procesor mógłby być częścią większego modułu wykonującego kodowanie Huffmana.

W pierwszej części wykonano model behawioralny algorytmu w języku MATLAB. Następnie, na jego podstawie opracowano kod w Vivado, składający się z dwóch modułów. Pierwszy odpowiadał za porównanie dwóch symboli i zliczanie wystąpień. Drugi moduł był nadrzędy w stosunku do tego pierwszego i odpowiadał za:

- Zapis symboli i wyników porównań do pamięci (256 bajtowe rejestry),
- odczyt z pamięci w celu wypisania wyników,
- inicjalizację i usuwanie danych z rejestrów,
- generację podrzędnych modułów.

Zachowanie algorytmu zobrazowano w maszynie stanów na rysunku 1. Dokonano symulacji i wstępnych testów. Kod testbench znajduje się w listingu kodu na końcu dokumentu.

W kolejnym kroku wygenerowano moduł jako IP Core w formacie peryferium AXI4. Kod umieszczono w stosownym miejscu, a blok posłużył do zbudowania schematu blokowego z procesorem ogólnego przeznaczenia. W pierwszej implementacji korzystano z procesora Microblaze, jednak jak się okazało na późniejszych etapach projektowania - uruchomienie go na płytce Zybo było dość problematyczne. Zmieniono więc jednostkę procesora na ZYNQ7. Ostateczny schemat można zobaczyć na rysunku 2. W jego skład wchodzą również dwa bloki AXI GPIO - jeden do wyświetlania za pomocą LED poprawności działania programu, a drugi do ustawiania sygnałów sterujących za pomocą przełączników.

Dokonano syntezy i implementacji kodu HDL na platformę Zybo. Raporty zużywanych zasobów można znaleźć wraz z kodem w repozytorium projektu.

Wygenerowany bitstream wraz z plikami hardware wyeksportowano do programu Xilinx SDK, gdzie napisano prostą aplikację do testowania w języku C. Udało się pomyślnie wgrać program na platformę ZYBO i uruchomić go. Wypisywane wyniki można obserwować za pomocą konsoli - są one poprawne, jednak nie udało się usunąć wszystkich błędów (np. program czasem wpada w nieskończoną pętlę).

Działanie modułu

Moduł SymbolsCounter sterowany jest za pomocą czterech sygnałów:

- Reset,
- start,
- mode,
- oraz end_flag.

Sygnał reset (aktywny stanem wysokim) powoduje wyczyszczenie rejestrów pamięci oraz inicjalizację pozostałych sygnałów. Stan wysoki sygnału start powoduje przejście maszyny stanów do trybu przetwarzania.

Sygnał mode odpowiada za wybieranie trybu wprowadzania symboli. W przypadku stanu niskiego, wszystkie wprowadzone symbole będą zapisywane do kolejnych rejestrów pamięci alphabet_memory. W przypadku stanu wysokiego, dokonane zostają porównania wprowadzanych symboli z pamięcią alphabet_memory, wynik jest zapisywany w pamięci result_memory.

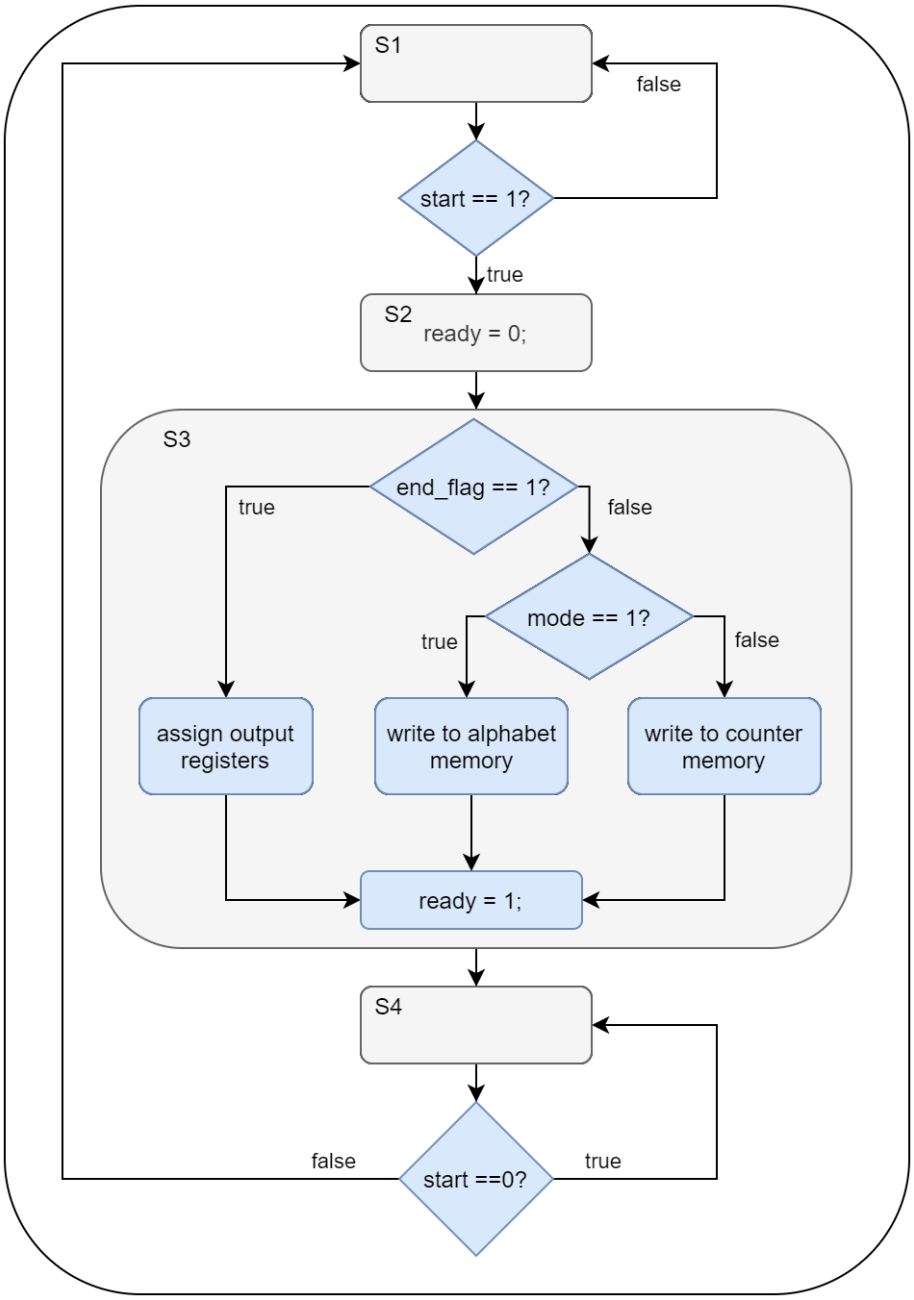
Sygnał end_flag informuje moduł o zakończeniu przetwarzania. W takim przypadku przy każdym kolejnym sygnale start zostają na wyjście podane pary: symbol oraz ilość wystąpień. Kiedy flaga jest aktywna stanem wysokim, sygnał mode nie ma żadnego wpływu na działanie modułu.

Kiedy przetwarzanie zostanie zakończone, sygnał ready_out zostanie postawiony w stan wysoki. Razem z sygnałem start można kontrolować działanie modułu przy pomocy np. testbencha albo aplikacji uruchomieniowej.

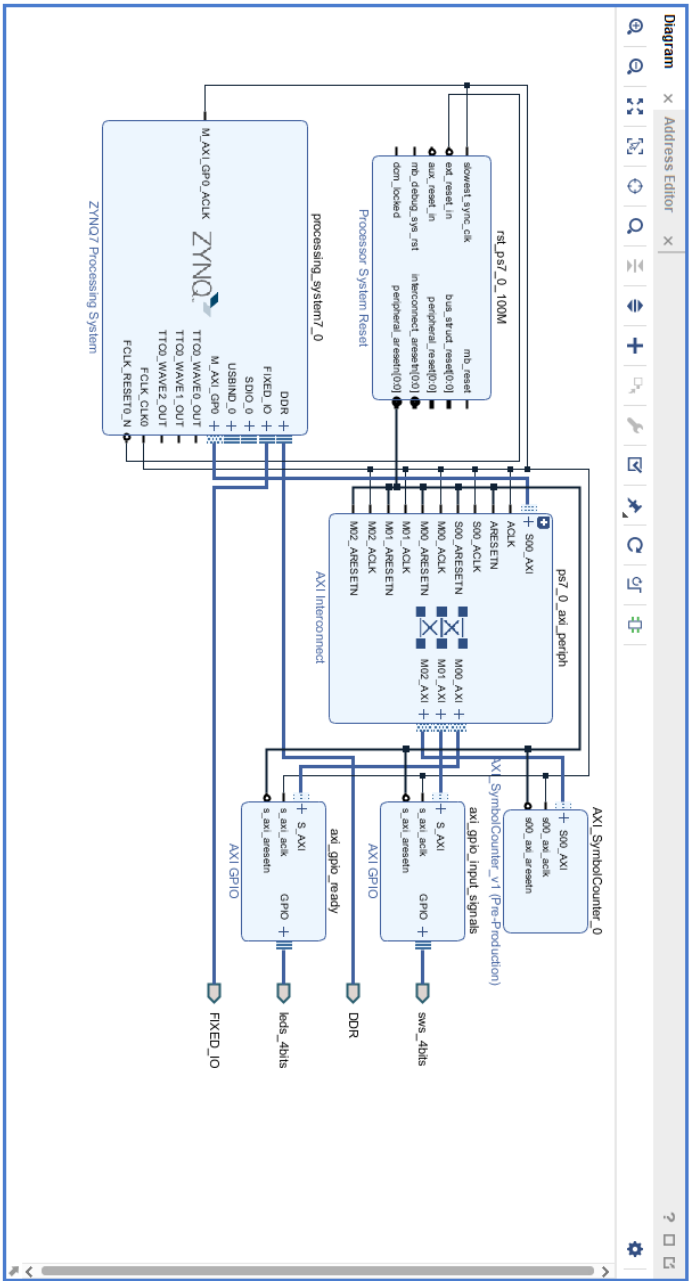
Wprowadzanymi symbolami mogą być dowolne ciągi bitów o długości od 1 do 16 bitów. Długość można zadeklarować parameterem S_WIDTH (w przypadku syntezowanego kodu musi to być wcześniej określona wartość, na etapie projektowania).

Symbole są przetwarzane w każdym z 256 wygenerowanych bloków modułu SingleCount.

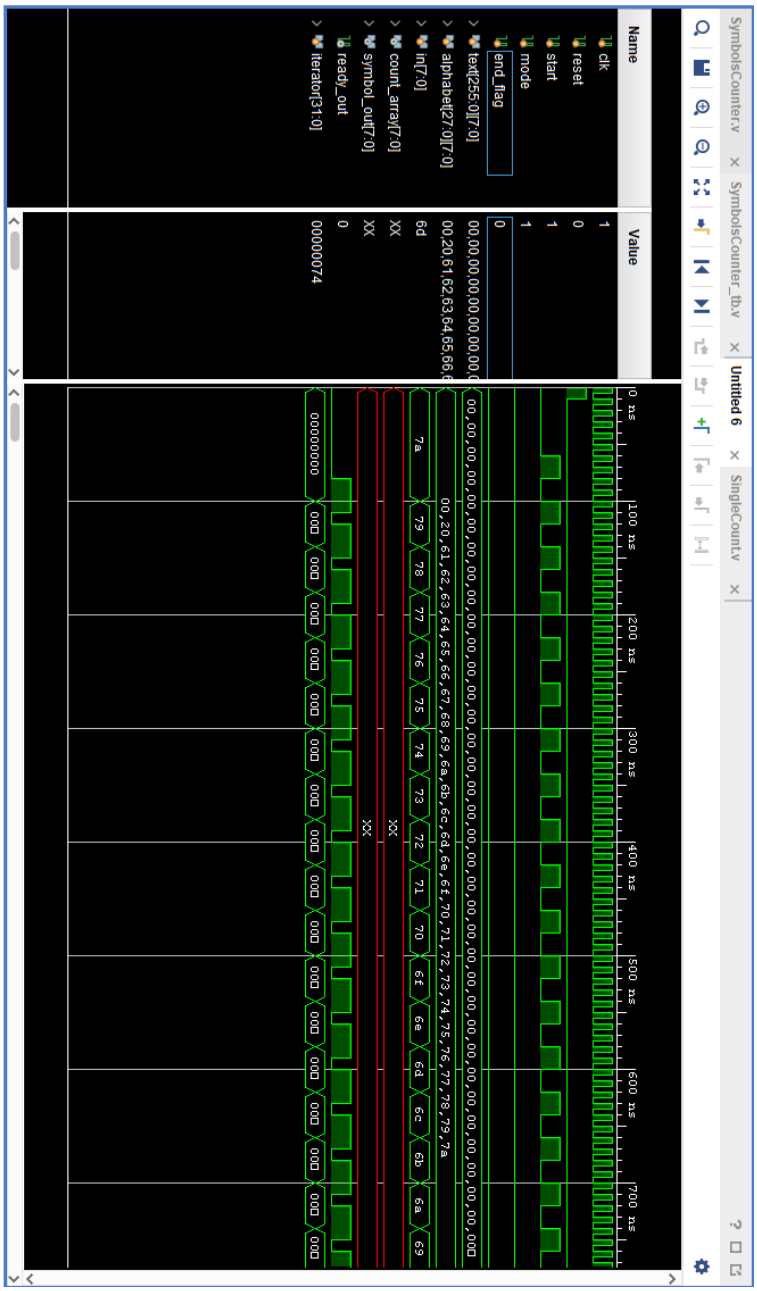
Na rysunku 3 można zaobserwować przykładowe przebiegi sygnałów, jakie uzyskano korzystając z zamieszczonego na końcu dokumentu kodu testbench.



Rysunek 1: Algorytm wykorzystanej maszyny stanów



Rysunek 2: Wygenerowany block design w Vivado



Rysunek 3: Przykładowe sygnały z symulacji

module SymbolsCounter.v

```
module SymbolsCounter
#(
    parameter S_WIDTH = 8 //symbol width (for both alphabet and text)
)(
    input clock, //clock input
    input reset, //module reset - activated with high state
    input start, //initial signal for processing
    input mode, //input mode 0 - input alphabet, 1 - input text
    input end_flag, //enables output - activated with high state
    input [S_WIDTH-1:0]symbol_in, //input symbol (goes to alphabet or processor,
    //depending on mode state)
    output reg [7:0]count_array, //how many given symbol was present in text
    output reg [S_WIDTH-1:0]symbol_out, //alphabet output symbol
    output ready_out //'processing ended' flag
);

reg [S_WIDTH-1:0]alphabet_memory[255:0]; //memory for storing alphabet symbols
reg [7:0]alphabet_counter = 0; //counter for alphabet symbols
reg [15:0]ready_cnt = 1; //decrementing counter
reg ready = 0;

reg [7:0]result_memory[255:0]; //memory for storing the number of symbol occurrences
reg [7:0]result_counter = 1;
reg [7:0]output_counter = 0;

parameter S1 = 4'h01, S2 = 4'h02, S3 = 4'h03, S4 = 4'h04;
reg [2:0] state;

wire [S_WIDTH-1:0]memory_write; //auxiliary signal
wire [7:0]result_write[255:0]; //auxiliary signal

assign memory_write = symbol_in;

integer i = 0;
initial
    begin
    alphabet_counter <= 0;
    state <= S1;
    ready <= 0;
    for(i = 0; i < 256; i = i + 1)
    begin
        alphabet_memory[i] <= 0;
        result_memory[i] <= 0;
    end
end
```

```
always @(posedge clock)
begin
    if(reset == 1'b1) //reset section
    begin
        result_counter <= 0;
        alphabet_counter <= 0;
        output_counter <= 0;
        ready <= 0;
        state <= S1;
        for(i = 0; i < 256; i = i + 1)
        begin
            result_memory[i] <= 0;
            alphabet_memory[i] <= 0;
        end
    end
    else //insert symbols section state-machine
    begin
        case(state)
        S1:
            begin
                if(start == 1'b1) state <= S2; else state <= S1;
            end
        S2:
            begin
                ready <= 0;
                state <= S3;
            end
        S3:
            begin
                if (end_flag != 1'b1)
                begin
                    case(mode)
                        0:
                        begin //alphabet mode
                            alphabet_memory[alphabet_counter] <= memory_write;
                            alphabet_counter <= alphabet_counter + 1;
                            result_counter <= alphabet_counter + 1;
                        end
                        1:
                        begin //text mode
                            for(i = 0; i < result_counter; i = i + 1)
                            begin
                                result_memory[i] <= result_write[i];
                            end
                        end
                    endcase
                end
                else // end_flag == 1'b1, output mode
                begin
                    if(result_counter > 0)
```

```
                    begin
                        count_array <= result_memory[output_counter];
                        symbol_out <= alphabet_memory[output_counter];
                        result_counter <= result_counter - 1;
                        output_counter <= output_counter + 1;
                    end
                    else
                    begin
                        result_counter <= output_counter;
                        output_counter <= 0;
                    end
                end
                ready <= 1;
                state <= S4;
            end
        S4:
            begin
                if(start == 1'b0) state <= S1; else state <= S4;
            end

        endcase
    end
end

assign ready_out = ready; // && (end_flag == 1) ? 1'b1 : 1'b0;

genvar j;
//generating processing blocks for each symbol in alphabet
generate for (j=0; j<256; j=j+1)
    begin: counter_loop
    SingleCount #(.S_WIDTH(S_WIDTH)) counter_step_0 (
        .mode(mode),
        .alphabet_symbol(alphabet_memory[j]),
        .symbol_in(symbol_in),
        .symbol_cnt_in(result_memory[j]),
        .symbol_cnt_out(result_write[j])
    );
    end
endgenerate

endmodule
```

module SingleCount.v

```
module SingleCount
#(
    parameter S_WIDTH = 8 //width of single symbol
)(
    input mode, //high state turns module ON
```

```
    input [S_WIDTH-1:0]alphabet_symbol, //compared alphabet symbol
    input [S_WIDTH-1:0]symbol_in, //compared text symbol
    input [7:0]symbol_cnt_in, //counter input value
    output [7:0]symbol_cnt_out //counter output value
);

//porownanie wartosci symbolu wejsciowego i alfabetu i ewentualna inkrementacja liczn
assign symbol_cnt_out = mode && (alphabet_symbol == symbol_in) ?
    symbol_cnt_in + 1 : symbol_cnt_in;

endmodule
```

module SymbolsCounter_tb.v

```
module SymbolsCounter_tb();

reg clk;
reg reset;
reg start;
reg mode, end_flag;
reg [7:0]text[255:0]; //input text memory
reg [7:0]alphabet[27:0] = {8'h00, 8'h20, 8'h61, 8'h62,
    8'h63, 8'h64, 8'h65, 8'h66, 8'h67, 8'h68,
    8'h69, 8'h6A, 8'h6B, 8'h6C, 8'h6D, 8'h6E,
    8'h6F, 8'h70, 8'h71, 8'h72, 8'h73, 8'h74,
    8'h75, 8'h76, 8'h77, 8'h78, 8'h79, 8'h7A};
reg [7:0]in;

wire [7:0]count_array; //output vector for symbol occurences
wire [7:0]symbol_out;
wire ready_out;

integer iterator; //count occurences and iterate through symbols

// Reset stimulus
initial
begin
    reset = 1'b1;
    #50 reset = 1'b0;
end

// Clocks stimulus
initial
begin
    clk = 1'b0; //set clk to 0
    clk = 1'b1;
end
always
```

```
begin
    #5 clk = ~clk; //toggle clk every 5 time units
end

initial
begin
    iterator = 0;
    start = 0;
    end_flag = 0;
    mode = 0;
    in <= alphabet[iterator];
    text <= "";
    text <= "lorem␣ipsum␣dolor␣sit␣amet␣consectetur␣adipiscing␣elit
␣␣␣␣sed␣do␣eiusmod␣tempor␣incididunt␣ut␣labore␣et␣dolore␣magna␣aliqua";
end

always @ (posedge reset)
begin
    #10
    iterator = 0;
    start = 0;
    end_flag = 0;
    mode = 0;
    in <= alphabet[iterator];
    reset = 1'b0;
    #50 start = 1;
end

always @ (posedge ready_out)
begin
    if (reset == 1'b0)
    begin
        start = ~start; //
        #20
        start = ~start;
        if(end_flag == 1'b0)
        begin
            if(mode == 1'b0) //insert alphabet to processor memory
            begin
                if(alphabet[iterator] != 8'h00)
                begin
                    iterator = iterator + 1;
                    in <= alphabet[iterator];
                end
                else
                begin
                    mode <= 1;
                    iterator = 0;
                end
            end
```

```
            else //mode == 1, input text
            begin
                if(text[iterator] != 8'h00)
                begin
                    in <= text[iterator];
                    iterator = iterator + 1;
                end
                else
                begin
                    end_flag <= 1;
                    iterator = 0;
                end
            end //mode
        end
        else //end_flag == 1
        begin
            $display("SYMBOL: '%c' OCCURENCES: %d", symbol_out, count_array);
            if(symbol_out == 8'h00)
            begin
                end_flag = 1'b0;
                $display("SUM: %d", iterator);
                reset = 1'b1;
            end
            iterator = iterator + count_array;
        end  //end_flag
    end
    else // reset == 0
    begin
        iterator = 0;
    end //reset
end

//Instantiate tested module
SymbolsCounter SymbolsCounter_inst (
    .clock(clk),
    .reset,
    .start(start),
    .mode,
    .end_flag,
    .symbol_in(in),
    .count_array,
    .symbol_out,
    .ready_out);

endmodule
```