**Modelowanie liczników w języku Verilog i ich implementacja w strukturze FPGA**

## Licznik binarny

Licznik binarny jest najprostszym i najpojemniejszym licznikiem. Kod 4 bitowego synchronicznego licznika binarnego pokazano poniżej:

```
module bin_cnt(clk, en, cnt);
        input clk;
        input en;    // enable
        output [3:0] cnt;

        reg [3:0] cnt = 0;

        always @ (posedge clk)
                cnt <= en ? (cnt + 1) : cnt;
endmodule
```

```
cnt = 0000
cnt = 0001
cnt = 0010
cnt = 0011
cnt = 0100
cnt = 0101
cnt = 0110
cnt = 0111
cnt = 1000
```

## Licznik pierścieniowy

Licznik pierścieniowy to układ sekwencyjny pracujący w kodzie “1 z n”. W trakcie działania tylko jeden z bitów może mieć wartość „1” dla układu z “krążącą jedynką” (lub „0” dla „krążącego zera”). Kod 4 bitowego licznika pierścieniowego z krążącą jedynką pokazano poniżej:

```
module ring(clk, cnt);
        input clk;
        output [3:0] cnt;

        reg [3:0] cnt = 4'b0001;

        always @ (posedge clk)
                cnt <= {cnt[2:0], cnt[3]};

endmodule
```

```
cnt = 0001
cnt = 0010
cnt = 0100
cnt = 1000
cnt = 0001
cnt = 0010
cnt = 0100
cnt = 1000
cnt = 0001
```

Działanie tego licznik można przetestować korzystając z poniższego modułu testowego:

```
module ring_tb();
        reg clk;
        wire [3:0] LED;
        ring ring_u(clk, LED);

        reg [3:0] i;
        initial begin
                $dumpfile("./ring.vcd");        // zapisz wynik symulacji
                $dumpvars(0);                   // wartości wszystkich zmiennych (rejestry)

                for(i=0; i < 10; i=i+1) begin
                        #1 clk = 0; #1 clk = 1;
                        $display("%d: %b", i, LED);
                end
        end
endmodule
```

**Symulacja pracy licznika**

W niniejszej instrukcji proponuje się przeprowadzenie symulacji z wykorzystaniem narzędzi opensource. W tym celu:

a) pobieramy pakiet symulator + przeglądarka

http://bleyer.org/icarus/iverilog-10.0-x86_setup.exe

b) Instalujemy w katalogu domyślnym tj. *C:\iverilog*, wraz z przeglądarką GTKWave

c) Uruchamiamy konsolę poleceniem *cmd*, za pomocą polecenia *cd* przechodzimy do katalogu w którym znajduje się plik z kodem testowanego licznika tj. *ring.v* i kompilujemy go komendą

*c:\iverilog\bin\iverilog* ring.v

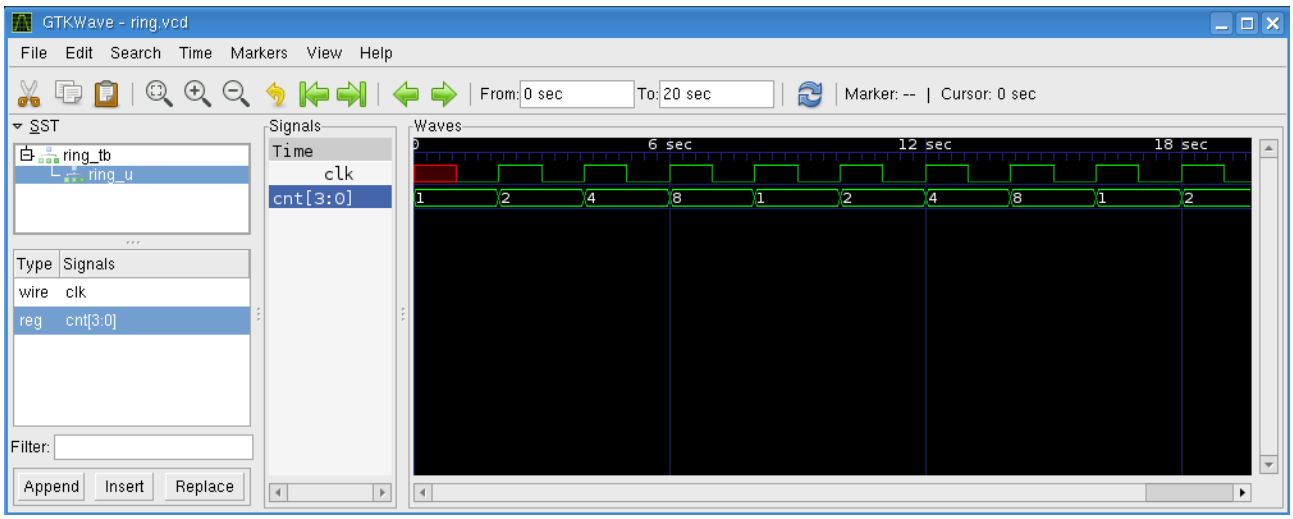
Wynik kompilacja umieszczany zostaje w pliku *a.out*. Symulację uruchamiamy komendą:

*c:\iverilog\bin\vvp* a.out

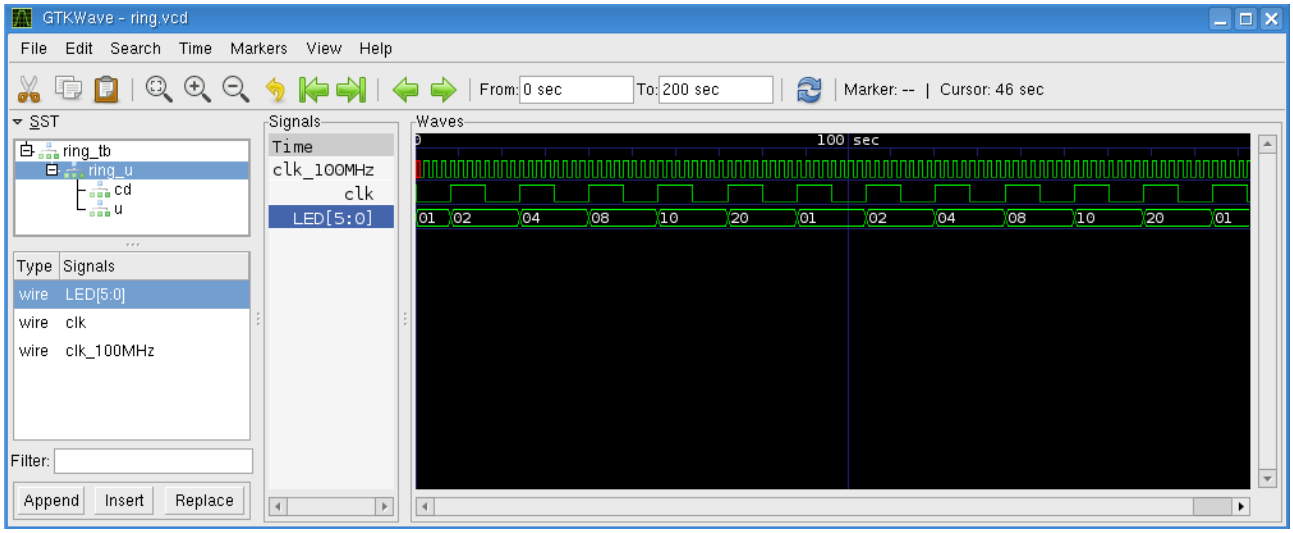
d) Przegladarkę uruchamiamy komendą

*c:\iverilog\gtkwave\bin\gtkwave* -f ring.vcd

Następnie wybieramy sygnały które chcemy obejrzeć (Rys poniżej):




Rys. Przebiegi symulacji pracy 4-bitowego licznika pierścieniowego.




Rys. Przebiegi symulacji pracy 6-bitowego licznika pierścieniowego z układem dodatkowego dzielnika częstotliwości przez 8.

**Testowanie licznik pierścieniowego na FPGA**

Na płytce Basys3 dostępny jest zegar 100 MHz. Częstotliwość tego zegara jest za wysoka do sterowania wyświetlaczem LED, dlatego musi być ona podzielona, tak by uzyskać sygnał o

częstotliwości rzędu 1 kHz (lub mniej, w zależności od zastosowania). W tym celu należy skorzystać z przygotowanego modułu *clk_divider*, którego kod wraz z licznikiem umieszczony jest w pliku *ring_fpga.v*. Zanim przeprowadzisz implementację na FPGA wykonaj symulację i obejrzyj przebiegi czasowe w układzie. Jak widać (rysunek wyżej) zegar taktujący licznik *ring*, tj. sygnał *clk* w module *ring_top*, ma częstotliwość 8 razy mniejszą jak sygnał *clk_100MHz* pochodzący z generatora zewnętrznego; dlaczego ?.

Zmień współczynnik podziału tak by licznik pracował z taką częstotliwością by można było zaobserwować efekt „krążącej jedynki" tj ok 10 Hz. Zwróć uwagę, że rejestr *reg [20:0] cnt* w dzielniku *clk_divider* ma 21 bitów, co pozwala na zliczanie do wartości 2^21-1, czyli 2097151 max. Czy to wystarczy do wygenerowania 10 Hz z sygnału 100 MHz ?

**Uwaga:** Vivado jest mało „inteligentny" i wymaga wskazania modułu TOP. W przypadku symulacji modułem TOP jest *ring_tb*, który nie ma żadnych portów. Do implementacji modułem TOP jest *ring_top*(clk_100MHz, LED), który ma dwa porty. Przypominam o konieczności dołączenia i modyfikacji pliku XDC, tak by uwzględnić te porty.

**Licznik BCD**

Licznik binarny oferuje największą pojemność przy zadanej liczbie użytych przerzutników. Jeśli jednak chcemy wyświetlić liczbę zliczeń w postaci dziesiętnej na wyświetlaczu 7 segmentowym, to pojawia się problem; należy skonwertować liczbę binarną na postać dziesiętną. Problemu tego można uniknąć poprzez użycie licznika BCD (Binary-Coded Decimal). Kod BCD wykorzystuje 4 bitowy kod binarny do kodowania kolejnych cyfr dziesiętnych liczby. Przykładowo liczba dziesiętna 1234 ma reprezentacje:

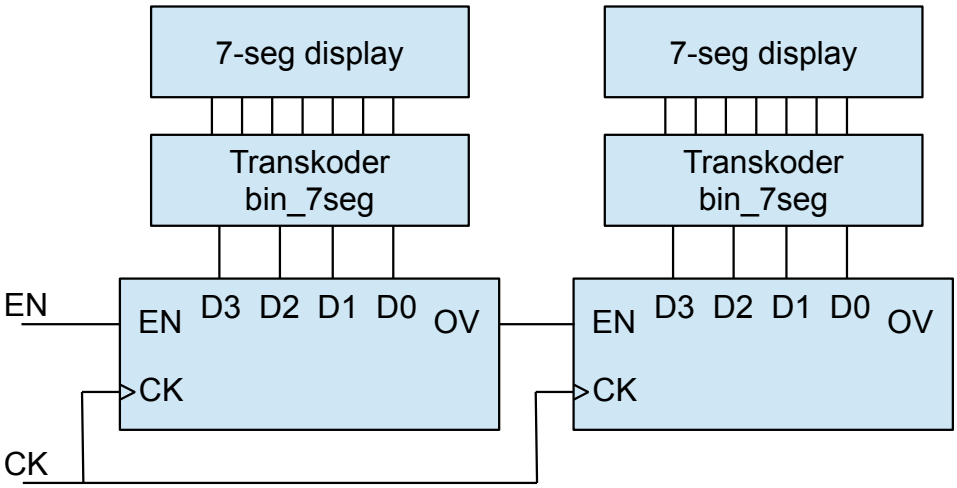
| | | | | |
|---|---|---|---|---|
| dziesiętnie: | 1 | 2 | 3 | 4 |
| BCD: | 0001 | 0010 | 0011 | 0100 |
| binarnie: | | 100 | 1101 | 0010 |

Jak widać, na ogół, zapis liczby w kodzie BCD wymaga więcej bitów niż zapis tej samej liczby w kodzie binarnym. Wynika to z faktu, że kolejne cztery bity zapisu BCD reprezentują liczbę w zakresie 0 do 9, podczas gdy w kodzie binarnym te same 4 bity reprezentują liczbę od 0 do 15. Dlatego kod BCD stosowany jest głównie w przypadku, gdy występuje konieczność wyświetlania wyniku zliczeń na wyświetlaczu 7 segmentowym.

Z powyższego przykładu wynika, że 4-pozycyjny licznik BCD można zbudować łącząc kaskadowo cztery liczniki binarne, których zakres zliczania ograniczony jest do 9. Przykład takiego licznika (do 9) w języku Verilog pokazany jest poniżej. Zwróć uwagę, że wartość rejestru *bcd* zwiększa się o 1 (modulo 10) podczas narastającego zbocza zegara tylko gdy *en* = 1. Przepełnienie licznika sygnalizowane jest flagą *ov*. Zauważ, że licznik składa się z dwóch części: kombinatorycznej i sekwencyjnej.

```
module bcd1(clk, en, bcd, ov);
	input clk;
	input en;
	output [3:0] bcd;
	output ov;

	reg [3:0] bcd_next, bcd = 0;
	reg ov;

	always @ (*)
		if(en==0)
			{ov, bcd_next} = {1'b0, bcd};
		else begin
```

```
                bcd_next = (bcd < 9) ? bcd + 1 : 0;
                ov = (bcd < 9) ? 0 : 1;
        end


    always @ (posedge clk)
        bcd <= bcd_next;

endmodule

module bcd1_tb();
    reg clk;

    wire [3:0] bcd;
    reg en;
    wire ov;
    bcd1 u(clk, en, bcd, ov);

    reg [7:0] i;
    initial begin
        $dumpfile("./cnt.vcd");
        $dumpvars(0);
        en = 1;
        for(i=0; i < 12; i=i+1) begin
            #1 clk = 0; #1 clk = 1;
            $display("i=%d: ov=%b, bcd=%d", i, ov, bcd);
        end
    end
endmodule
```




Rys. Licznik BCD dwu pozycyjny z wyświetlaczem (transkoder na kod wskaźnika 7 seg).

**Zadanie**: na podstawie powyższego przykładu zbuduj licznik BCD o dwóch pozycjach, tj. taki który zlicza w zakresie 0 do 99. Przeprowadź symulacje działania.

**Zadanie**: zaimplementuj na układzie FPGA licznik BCD o dwóch pozycjach (zakres zliczania 0 do 99). Do taktowania licznika wykorzystaj sygnał zegarowy o częstotliwości ok 1 Hz uzyskany z

podzielnika tak jak to omówiono w przypadku licznika pierścieniowego. Sygnał c_in dla pierwszego licznika powinien pochodzić z przycisku *sw*[0], czyli licznik zlicza, gdy przycisk jest wciśnięty. Licznik uzupełnij o moduł sterowania wyświetlaczem 7-segmentowmy.