Logika Cyfrowa

Jakub Gałaszewski

May 6, 2024

1 Poniższy kod implementuje rejestr przesuwny z liniowym sprzężeniem zwrotnym (linear-feedback shift register, LFSR). Jak wygląda jego sekwencja odliczania (zaczynając od 001)?

```
module lfsr(
    output logic [2:0] q,
    input [2:0] d,
    input load, clk
);
    always_ff @(posedge clk)
        if (load) q <= d;
        else q <= {q[0], q[0] ^ q[2], q[1]};
endmodule
```

$$q <= \{q[0], q[0] \oplus q[2], q[1]\};$$

$q[0]$	$q[1]$	$q[2]$	$q[0]'$	$q[1]'$	$q[2]'$
0	0	0	0	0	0
0	0	1	1	1	0
1	1	0	0	1	1
0	1	1	1	1	1
1	1	1	1	0	1
1	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	0	1

2 Poniższy kod implementuje inny rejestr przesuwny z liniowym sprzężeniem zwrotnym. Jak wygląda jego sekwencja odliczania?

```
module lfsr(
    output logic [3:0] q,
    input clk,
    input nrst
);
    always_ff @(posedge clk or negedge nrst)
        if (!nrst) q <= 4'b0 ;
        else q <= {q[2:0], q[3] ~^ q[1]};
endmodule
```

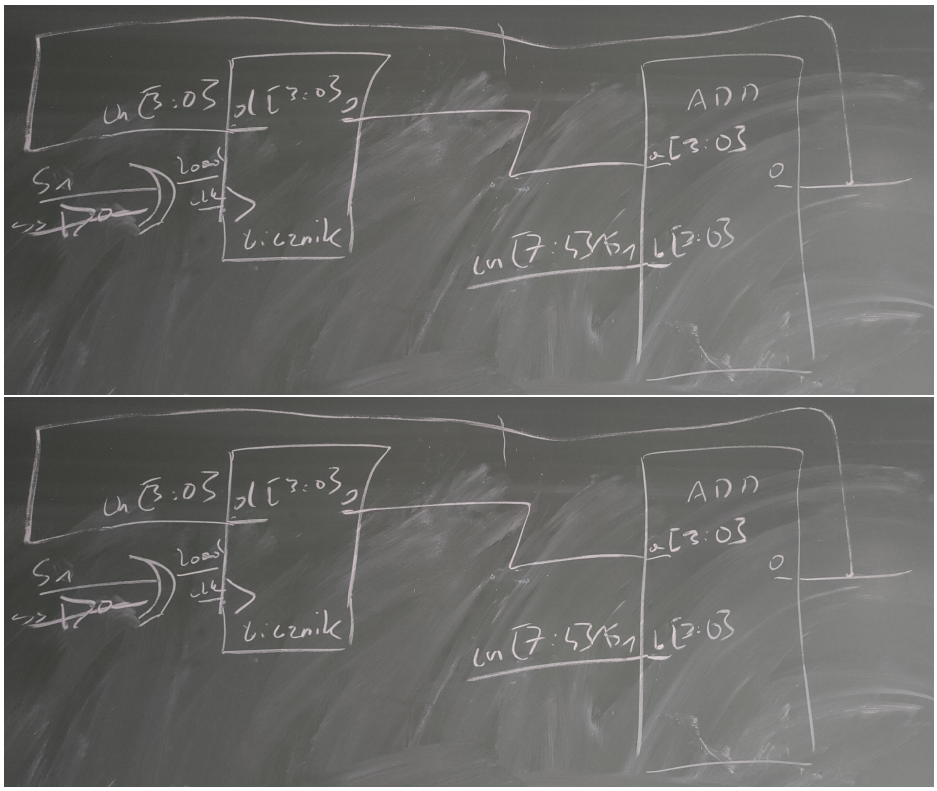
$$q <= \{q[2:0], q[3] \neg \oplus q[1]\}$$

$q[0]$	$q[1]$	$q[2]$	$q[3]$	$q[0]'$	$q[1]'$	$q[2]'$	$q[3]'$
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	1
0	0	1	1	0	1	1	0
0	1	1	0	1	1	0	0
1	1	0	0	1	0	0	0
1	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0
0	1	0	0	1	0	0	1
1	0	0	1	0	0	1	0
0	1	0	1	1	0	1	1
1	0	1	1	0	1	1	1
0	1	1	1	1	1	1	0
1	1	1	0	1	1	0	1
1	1	0	1	1	0	1	0
1	0	1	0	0	1	0	1
1	1	1	1	1	1	1	1

3 Rozważ poniższy blok always$_{ff}$:

```
always_ff @(posedge clk)
    if (s1) r1 <= r1 + r2;
    else if (s2) r1 <= r1 + 1;
    else r1 <= r1
```

Narysuj możliwy sposób zsyntezowania układu opisanego powyższym kodem, wykorzystujący 4-bitowy licznik z ładowaniem równoległym oraz sumator 4-bitowy. Nie używaj multiplekserów.


ADD
load
clk
licznik
ADD
load
clk
licznik

4 Jaka będzie wartość e po jednym cyklu zegara, jeśli wartość ra i rb przed tym cyklem była równa 1, dla poniższych bloków? Dlaczego?

1.
```
always_latch if (clk) begin
    rb = ra - 1;
    if (rb == 0) e = 1;
    else e = 0;
end
```

2.
```
always_ff @(posedge clk) begin
    rb <= ra - 1;
    if (rb == 0) e <= 1;
    else e <= 0;
end
```

Oturz dla pierwszego $e = 1$, natomiast dla drugiego $e = 0$, jest tak ponieważ operator = przypisuje od razu, <= natomiast odracza aby operacje były współbieżne.

5 (2 pkt) Zaimplementuj przy użyciu układu sekwencyjnego algorytm Euklidesa (w wariancie z odejmowaniem) znajdujący największy wspólny dzielnik (NWD) dwóch liczb 16-bitowych bez znaku. Podążaj przy tym za metodą przedstawioną na wykładzie na przykładzie obliczania silni. W szczególności, proszę podać:

- potrzebne sygnały wejściowe i wyjściowe oraz ich rozmiar,
- potrzebne rejestry oraz ich rozmiar, należy postarać się nie wprowadzać zbędnego stanu,
- zależność stanu w następnej chwili czasu od stanu w poprzedniej chwili czasu oraz wejść,
- schemat układu, bez rozpisywania poszczególnych bramek,
- implementację w SystemVerilogu.

```
module GCD(input  clk,
           input  [15:0] a,
```

```
            input [15:0] b,
            input ini,
            output [15:0]e,
            output fin
                  );
  assign fin = (!newb || !newa);
  logic [15:0]newa;
  logic [15:0]newb;
  assign e = fin ? (newb==0 ? newa : newb) : 0;
  always_ff @(posedge clk)
    if(ini) begin
      newa <= a;
      newb <= b;
    end else if(newa > newb) begin
          newa <= newa-newb;
          newb <= newb;
    end else begin
      newa <= newa;
      newb <= newb-newa;
    end
endmodule
```