Współczesne systemy wbudowane

Sprawozdanie z projektu

Paweł Bielecki
01.01.2024

Algorytm

Dos sterowania sygnałem LED został wykorzystany sygnał o regulowanym wypełnieniu. Zazwyczaj takie sterowniki LED są sterowane przy pomocy sygnału PWM. W trakcie jednego cyklu PWM, sygnał jest na początku włączony, a po pewnym momencie przez resztę cyklu wyłączony. Jest to nieefektywne, gdyż zegar służy jedynie do oddzielenia momentu zmiany sygnału mimo iż musi liczyć cały czas. W tym celu opracowano algorytm, który w każdym cyklu zegarowym określa stan sygnału z możliwie jak najczęstszymi zmianami wartości. Zwiększenie częstotliwości zmian ma na zminimalizowanie efektu migotania led. Porównanie sygnałów PWM i opracowanego dla wypełnienia z rozdzielczością czterech bitów, przedstawiono w Tabeli 1.

Tabela 1: Porównanie generowania sygnałów

Wartość	PWM	Poprawiony generator
0	000000000000000	000000000000000
1	100000000000000	000000010000000
2	110000000000000	000100000001000
3	111000000000000	000100010001000
4	111100000000000	010001000100010
5	111110000000000	010001010100010
6	111111000000000	010101000101010
7	111111100000000	010101010101010
8	111111110000000	101010101010101
9	111111111000000	101010111010101
10	111111111100000	101110101011101
11	111111111110000	101110111011101
12	111111111111000	111011101110111
13	111111111111100	111011111110111
14	111111111111110	111111101111111
15	111111111111111	111111111111111

Algorytm do generowania używa prawdopodobieństwa przełączenia się zera na jedynkę w binarnym zapisie pozycyjnym licznika. Prawdopodobieństwa przedstawiono w Tabeli 2. Bit najmłodszy zmienia się co drugą inkrementację, a każdy kolejny dwa razy rzadziej. Dodatkowo prawdopodobieństwa te są całkowicie rozdzielne. Bit *n*-ty zmieni się na jedynkę, w momencie gdy wszystkie młodsze bity zmienią się na zera (przykład 0111 → 1000).

Tabela 2: Prawdopodobieństwa zmian bitów z zera na jedynkę

Bit	3	2	1	0
Prawdopodobieństwo	0,0625	0,125	0,25	0,5

Rozdzielność prawdopodobieństw pozwala na wyznaczenie sygnału wyjściowego poprzez sumę logiczną zmian z zera na jedynkę na pożądanych bitach. Np. uzyskanie wypełnienia 0,75 odbywa się przez generowanie sygnału na dwóch najmłodszych bitach. Do sterownia wymagane jest utworzenie sterownika będącego maską bitową. Odbywa się to poprzez odwrócenie kolejności bitów licznika ułamka o mianowniku 2^r, gdzie *r* jest rozdzielczością bitową. Przykładowo: użyta w poprzednim przykładzie wartość 0,75 dla rozdzielczości 4-bitowej to ułamek 12/16, czyli w zapisie binarnym 1100. Po odwróceniu kolejności bitów uzyska się 0011, czyli sprawdzanie dwóch najmłodszych bitów.

Wyliczenie wartości wyjściowych może odbywać się poprzez prostą operację logiczną:
~ (*(~counter) & (counter+1) & driver)*

gdzie:

counter – zmienna licznika,
driver – maska / sterownik.

Wytłumaczenie:
To co było zerem przed inkrementacją i jest jedynką po inkrementacji i jednocześnie jest bitem, który należy brać pod uwagę. Jeśli na jakiejkolwiek pozycji będzie jedynka (suma logiczna wektora bitów), to na wyjściu powinno się otrzymać jedynkę. W przeciwnym wypadku zero.

Algorytm porównujący generowanie przy pomocy generowania sygnałów przy pomocy PWM, oraz poprawionego, zaimplementowanego algorytmu pokazuje Tabela 3. Generowany sygnał dla o rozdzielczości *r* bitów może przyjąć 2^r wartości i ma przy tym długość 2^r-1 bitów.

Tabela 3: Porównanie kodów źródłowych generatorów

PWM	Poprawiony generator

module pwm_driver #(parameter R=10) (
input logic clk,
input logic[R-1:0] value,
output logic out,
input logic reset
);
 	
logic[R-1:0] counter;
wire[R-1:0] next_counter;
assign next_counter = counter + 1;
 	
always @(posedge clk)
begin
counter <= (reset | &next_counter) ? 0 : next_counter;
out <= counter < value;
end
 	
endmodule
	module led_driver #(parameter R=10) (
	input logic clk,
	input logic[R-1:0] value,
	output logic out,
	input logic reset
	);
	
	logic[R-1:0] counter;
	wire[R-1:0] driver;
	wire[R-1:0] next_counter;
	assign next_counter = counter + 1;
	
	genvar i;
	generate
	for (i = 0; i < R; i=i+1) begin : gen_loop
	assign driver[i] = value[R-1-i];
	end
	endgenerate
	
	always @(posedge clk)
	begin
	counter <= (reset || &next_counter) ? 0 : next_counter;
	out <= |((~counter) & next_counter & driver);
	end
	
	endmodule

Figury 1 i 2 przedstawiają porównanie generowanych sygnałów dla obu generatorów.

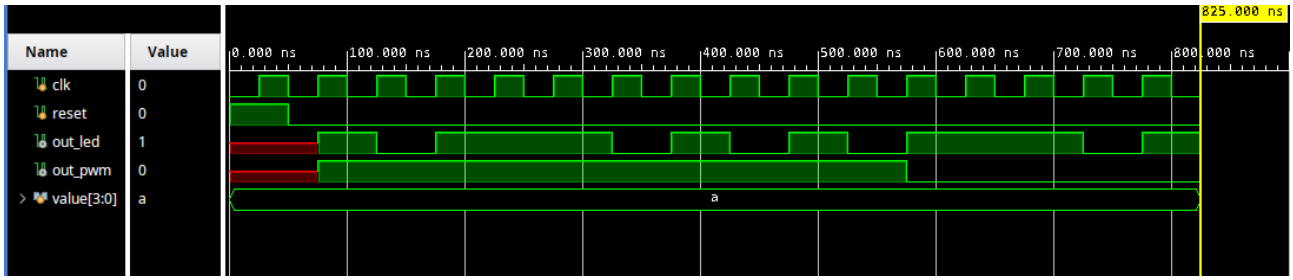

Figura 1: Wykres sygnałów dla wypełnienia 10 przy rozdzielczości 4 bitów

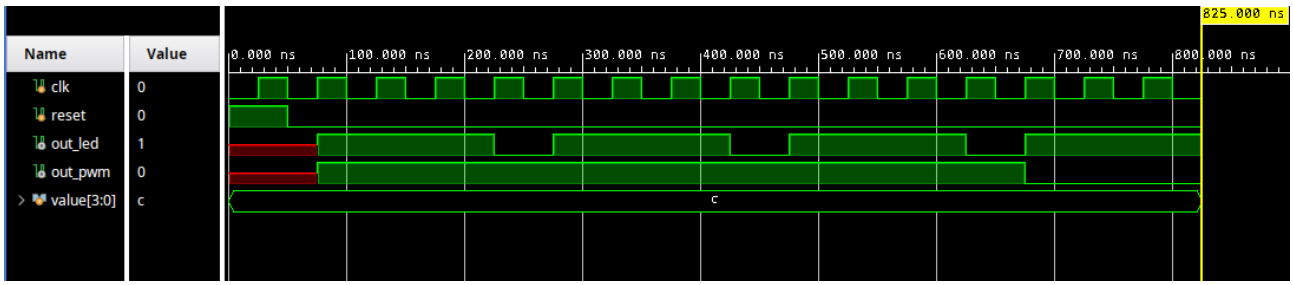

Figura 2: Wykres sygnałów dla wypełnienia 12 przy rozdzielczości 4 bitów

Implementacja

Algorytm zdecydowano się zaimplementować na układzie FPGA na płytce *Mimas A7 Mini*. Z peryferiów przydanych w tym projekcie płytka jest wyposażona w cztery przyciski, osiem monochromatycznych diod led i jedną diodę RGB. W celu prezentacji możliwości generatora zaprogramowano układ o następującej funkcjonalności:

- przycisk 0 – reset układu;
- przycisk 1 – zmniejszanie wartości kanału koloru;
- przycisk 2 – zwiększanie wartości kanału koloru;
- przycisk 3 – przełączanie zmienianego kanału koloru, kolejno:
  brak, czerwony, zielony, niebieski;
- diody monochromatyczne – wyświetlanie ośmiu najstarszych bitów wartości;
- dioda RGB – wartość ustawionego koloru.

Film z działania został dołączony do sprawozdania.

Kod z repozytorium z projektem Vivado: https://github.com/pawel2000pl/VerilogLedDriver