Generator PWM cu Interfață SPI

Documentație tehnică (versiune condensată)

Proiect Arhitectura Calculatoarelor

23 noiembrie 2025

1 Scopul proiectului

Proiectul implementează un periferic hardware de tip generator PWM (Pulse Width Modulation) controlat prin interfață SPI. Peripheralul permite configurarea perioadei, factorului de umplere, direcției de numărare și a modului de aliniere a semnalului PWM, toate prin registre accesibile prin SPI.

Obiectivele principale sunt:

- implementarea unei arhitecturi modulare, ușor de înțeles;
- folosirea exclusivă a elementelor de Verilog sintetizabile;
- compatibilitate cu diagrama de arhitectură și harta de registre date în enunț.

2 Arhitectura hardware

Perifericul este alcătuit din cinci module principale, instanțiate în `top.v`:

1. **spi_bridge** – convertește fluxul serial SPI în octeți paralele; generează semnalul `byte_sync` și livrează octeții către decodor.
2. **instr_dcd** – decodifică perechi de octeți (instrucțiune + date) în semnale de `read`, `write`, `addr` și `data_write` pentru blocul de registre.
3. **regs** – conține toate registrele de configurare: `PERIOD`, `COMPARE1`, `COMPARE2`, `PRESCALE`, semnalele de activare, direcția numărătorului și modul de funcționare al PWM.
4. **counter** – numără de la 0 la `period` (sau invers), folosind prescalerul, și livrează valoarea `count_val` către PWM.
5. **pwm_gen** – generează semnalul `pwm_out` pe baza `count_val`, `period` și `compare1`, în funcție de modul selectat în registrul `FUNCTIONS`.

2.1 Fluxul de date

Master-ul SPI trimite comenzi către `spi_bridge`, care acumulează 8 biți, generează `byte_sync` și livrează octetul în `data_in`. Modulul `instr_dcd` interpretează primul octet ca instrucțiune (bitul 7 = citire/scriere, biții 5..0 = adresă), iar al doilea octet ca date, activând `read` sau `write`. Blocul `regs` actualizează registrele sau returnează conținutul acestora. Numărătorul și generatorul PWM folosesc aceste registre pentru a genera semnalul de ieșire.

3 Interfața SPI și formatul comenzilor

3.1 Parametri SPI

Protocolul folosit este SPI clasic, cu următoarele ipoteze:

- CPOL = 0, CPHA = 0 (date eșantionate pe primul front crescător de `sclk`);
- transmisie MSB-first;
- fiecare transfer de registru se face în doi octeți.

3.2 Formatul instrucțiunii

Primul octet trimis reprezintă instrucțiunea:

- bitul 7: 1 = scriere, 0 = citire;
- biții 5..0: adresa registrului (0–63).

Al doilea octet reprezintă datele:

- la scriere: valoarea care se va încărca în registru;
- la citire: master-ul citește octetul furnizat pe linia MISO.

4 Harta registrelor esențiale

Sunt utilizate doar registrele cerute în enunț; valorile pe 16 biți sunt împărțite în LSB/MSB.

Addr	Nume	Biți	R/W	Descriere
0x00	PERIOD_LSB	[7:0]	RW	Octet inferior perioadă
0x01	PERIOD_MSB	[7:0]	RW	Octet superior perioadă
0x02	COUNTER_EN	[0]	RW	Activare numărător (`en`)
0x03	COMPARE1_LSB	[7:0]	RW	Prag PWM inferior
0x04	COMPARE1_MSB	[7:0]	RW	Prag PWM superior
0x0A	PRESCALE	[7:0]	RW	Prescaler pentru contor
0x0B	UPNOTDOWN	[0]	RW	1 = numărare în sus, 0 = în jos
0x0C	PWM_EN	[0]	RW	Activare ieșire PWM
0x0D	FUNCTIONS	[1:0]	RW	Modul PWM (0 = stânga, 1 = dreapta)

Registrul virtual `COUNTER_VAL` (0x08/0x09) este read-only și întoarce valoarea curentă a numărătorului, utilă la depanare.

5 Algoritmul de funcționare

5.1 Numărătorul cu prescaler

Contorul intern se comportă astfel:

- prescaler-ul `prescale` numără de la 0 până la valoare, apoi generează un *tic* de incrementare/decrementare pentru contorul principal;

- dacă `upnotdown` = 1, contorul crește de la 0 la `period`, apoi revine la 0;
- dacă `upnotdown` = 0, contorul scade de la `period` la 0, apoi revine la `period`;
- semnalul `count_reset` forțează imediat contorul la 0 (și resetează prescaler-ul).

Frecvența efectivă de numărare este

$$f_{count} = \frac{f_{clk}}{(prescale + 1)}.$$

5.2 Generarea PWM

Generatorul PWM compară `count_val` cu pragurile `compare1` și `period`. Pentru modul de aliniere la stânga (cel mai folosit):

$$pwm\_out = \begin{cases} 1, & \text{dacă } count\_val < compare1, \\ 0, & \text{altfel.} \end{cases}$$

Factorul de umplere este

$$Duty\ Cycle = \frac{compare1}{period} \cdot 100\,\%.$$

Frecvența semnalului PWM este

$$f_{PWM} = \frac{f_{clk}}{(prescale + 1) \cdot period}.$$

6 Secvență minimă de configurare

Presupunem $f_{clk} = 10\,\text{MHz}$ și dorim un semnal PWM de aproximativ 1 kHz cu factor de umplere 50%.

6.1 Calcul parametri

- alegem $prescale = 0 \Rightarrow f_{count} = 10\,\text{MHz}$;
- perioada necesară: $period = 10\,000$ (10 MHz / 1 kHz);
- pragul PWM: $compare1 = period/2 = 5000$.

6.2 Pași prin SPI

1. Scriere `PERIOD_LSB` (0x00) cu 0x10, apoi `PERIOD_MSB` (0x01) cu 0x27 (0x2710 = 10000).
2. Scriere `COMPARE1_LSB` (0x03) cu 0x88, `COMPARE1_MSB` (0x04) cu 0x13 (0x1388 = 5000).
3. Scriere `PRESCALE` (0x0A) cu 0x00.
4. Scriere `COUNTER_EN` (0x02) cu 0x01 pentru a porni numărătorul.
5. Scriere `PWM_EN` (0x0C) cu 0x01 pentru a activa ieșirea PWM.
6. Opțional, scriere `FUNCTIONS` (0x0D) cu 0x00 pentru modul aliniere la stânga.

7 Fragmente esențiale de cod

7.1 Interconectarea din top.v

Listing 1: Fragment relevant din top.v

```
spi_bridge i_spi_bridge (
    .clk(clk),
    .rst_n(rst_n),
    .sclk(sclk),
    .cs_n(cs_n),
    .mosi(miso),
    .miso(mosi),
    .byte_sync(byte_sync),
    .data_in(data_in),
    .data_out(data_out)
);

instr_dcd i_instr_dcd (
    .clk(clk),
    .rst_n(rst_n),
    .byte_sync(byte_sync),
    .data_in(data_in),
    .data_out(data_out),
    .read(read),
    .write(write),
    .addr(addr),
    .data_read(data_read),
    .data_write(data_write)
);
```

7.2 Logica de bază din counter.v

Listing 2: Parte din implementarea contorului

```
always @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    count_val_reg    <= 16'd0;
    prescale_counter <= 8'd0;
  end else if (count_reset) begin
    count_val_reg    <= 16'd0;
    prescale_counter <= 8'd0;
  end else if (en) begin
    if (prescale_counter >= prescale) begin
      prescale_counter <= 8'd0;
      if (upnotdown) begin
        if (count_val_reg >= period)
          count_val_reg <= 16'd0;
        else
          count_val_reg <= count_val_reg + 16'd1;
      end else begin
        if (count_val_reg == 16'd0)
```

```
          count_val_reg <= period;
        else
          count_val_reg <= count_val_reg - 16'd1;
      end
    end else begin
      prescale_counter <= prescale_counter + 8'd1;
    end
  end
end
```

7.3 Compararea în pwm_gen.v

Listing 3: Logică simplă de generare PWM

```
always @(posedge clk or negedge rst_n) begin
  if (!rst_n)
    pwm_out_reg <= 1'b0;
  else if (!pwm_en)
    pwm_out_reg <= 1'b0;
  else begin
    case (functions[1:0])
      2'b00: pwm_out_reg <= (count_val < compare1);
      2'b01: pwm_out_reg <= (count_val >= (period - compare1));
      default: pwm_out_reg <= 1'b0;
    endcase
  end
end
```

8 Concluzii

Versiunea implementată respectă cerințele de arhitectură: există un bloc clar de registre, un numărător programabil, un generator PWM parametrizabil și un bridge SPI care permite configurarea completă din software. Codul este simplu, structurat pe module, și poate fi folosit ca bază pentru extinderi (mai multe canale PWM, întreruperi, moduri suplimentare).