Documentație Proiect: Generator Semnal PWM

Cosmulete Ion-Cosmin, Dinca Marta, Sova Ioan-Rares
Grupa 332AA

Contents

1 Introducere **2**

2 Particularități de implementare pe submodule **2**
2.1 Bridge-ul SPI (spi_bridge.v) 2
2.2 Decodorul de Instrucțiuni (instr_dcd.v) 3
2.3 Blocul de Registri (regs.v) 3
2.4 Numărătorul (counter.v) 3
2.5 Generatorul PWM (pwm_gen.v) 4

3 Concluzii **5**

1 Introducere

Scopul modulului este de a genera un semnal PWM configurabil prin SPI, replicând funcționalitatea unui periferic întâlnit în microcontrolere.

Implementarea este realizată în Verilog și urmează o arhitectură modulară compusă din: (1) interfața SPI, (2) decodorul de instrucțiuni, (3) blocul de regiștri, (4) numărătorul cu prescaler, (5) generatorul PWM și un ultim modul top.v care integrează celelalte cinci submodule.

2 Particularități de implementare pe submodule

2.1 Bridge-ul SPI (spi_bridge.v)

Implementarea protocolului SPI respectă CPOL=0, CPHA=0. Având în vedere că **sclk** și **clk** sunt sincrone (conform cerinței), sincronizarea se rezumă la detectarea frontului crescător al semnalului **byte_ready_s**.

Sincronizarea datelor între domeniile de ceas

> **Notă:** Deși ceasurile sunt sincrone, folosirea detectării de muchie garantează că pulsul **byte_sync** are durata exactă a unui ciclu **clk** și nu depinde de întârzierile interne de propagare.

Un mecanism de double-flop este folosit pentru a transfera semnalul **byte_ready_s** în domeniul **clk**, urmat de detectarea unei tranziții pentru generarea pulsului **byte_sync**:

```
reg byte_ready_s_sync1, byte_ready_s_sync2;
reg byte_ready_s_clkprev;

always @(posedge clk or negedge rst_n) begin
    byte_ready_s_sync1 <= byte_ready_s;
    byte_ready_s_sync2 <= byte_ready_s_sync1;

    if (byte_ready_s_sync2 & ~byte_ready_s_clkprev) begin
        data_in <= shiftr_in_s;
        byte_sync <= 1'b1;
    end

    byte_ready_s_clkprev <= byte_ready_s_sync2;
end
```

Pulsul **byte_sync** este consumat ulterior de decodorul de instrucțiuni.

2.2 Decodorul de Instrucțiuni (instr_dcd.v)

Decodorul este un FSM cu două stări: Setup și Data. Starea este menținută într-un singur bit boolean.

Fluxul de citire

- În faza Setup, primul byte (setup byte) este analizat. Dacă bitul R/W este 0, se citește registrul specificat.
- În faza Data, byte-ul primit este ignorat (dummy cycle). Se revine în starea Setup.

Fluxul de scriere

- Faza Setup latchează adresa și direcția.
- În faza Data, al doilea byte constituie payload-ul, iar decodorul generează un puls **write** de un singur ciclu.

2.3 Blocul de Registri (regs.v)

Blocul de regiștri stochează valorile de configurare ale perifericului. Implementarea suportă adresare pe byte pentru registrii pe 16 biți (ex.: PERIOD la 0x00 și 0x01).

Adrese invalide
Menționăm că această abordare crește robustetea și securitatea perifericului, evitând scrieri accidentale în zone nevalide.:

- la scriere: sunt ignorate;
- la citire: întorc 0, conform cerințelor.

Registrul COUNTER_RESET

Implementat ca write-only, generând un puls:

```
always @(posedge clk or negedge rst_n) begin
    count_reset <= 1'b0;
    if (write && addr == 6'h07)
        count_reset <= 1'b1;
end
```

2.4 Numărătorul (counter.v)

Acest modul oferă baza de timp printr-un prescaler și un contor configurabil.

Prescaler

Prescalerul numără $2^{prescale}$ cicluri înainte de incrementarea contorului:

```
if (presc_cnt >= (1 << prescale) - 1) begin
    presc_cnt <= 0;
    presc_tick <= 1'b1;
end else begin
    presc_cnt <= presc_cnt + 1;
```

```
end
```

Comportamentul cu COUNTER_EN

- Dacă **counter_en = 0**, contorul își menține valoarea.
- Dacă se modifică registrele în timpul funcționării, actualizarea are loc doar la overflow/underflow.

Wrap-around

> **Notă:** Condiția **count_val >= period** a fost aleasă pentru a evita blocarea contorului în cazuri rare în care ar depăși perioada.

- Count up: reset la 0 dacă **count_val period**.
- Count down: reset la **period** dacă **count_val = 0**.

2.5 Generatorul PWM (pwm_gen.v)

Generează semnalul PWM pe baza valorilor de comparare și a modului selectat.

Detectarea overflow/underflow

Realizată comparând contorul curent cu cel anterior:

```
assign is_wrap = (count_val == 0 && prev_count == period) ||
                 (count_val == period && prev_count == 0);
```

Mod aliniat (FUNCTIONS[1]=0)

La fiecare wrap:

- **FUNCTIONS[0]=0** → semnalul începe pe 1 (aliniere stânga)
- **FUNCTIONS[0]=1** → semnalul începe pe 0 (aliniere dreapta)

Comutarea are loc la **compare1**.

Mod nealiniat (FUNCTIONS[1]=1)

> **Notă:** Garda **compare1 < compare2** previne comportamente ambigue, asigurând existența unei porțiuni de semnal 'off'.

- semnalul începe la 0
- devine 1 la **compare1**
- revine la 0 la **compare2** (doar dacă **compare1 < compare2**)

Comportamentul cu PWM_EN

Dacă **pwm_en = 0**, ieșirea **pwm_out** își menține ultima stare.

3 Concluzii

Implementarea realizată respectă integral cerințele temei, incluzând:

- comunicarea SPI cu sincronizare corectă,
- decodificarea completă a instrucțiunilor,
- gestionarea registrilor și a adresării multi-byte,
- contor configurabil cu prescaler și direcție,
- generarea PWM în mod aliniat și nealiniat.