Documentație Proiect: Generator Semnal PWM

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

Cuprins

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,
6. modulul de top (`top.v`) care integrează componentele.

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 Regiștri (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).

Tratarea adreselor invalide:

Menționăm că această abordare crește robustețea 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 folosind o structură *case* pentru decodificarea adresei:

```
// In interiorul blocului case (addr)
6'h07: begin // COUNTER_RESET (Scriere-Doar) @ 0x07
    // Da un puls de un ciclu CLK catre logica contorului
    count_reset <= 1'b1;
    // Nu se stocheaza nicio valoare
end
```

2.4 Numărătorul (counter.v)

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

Prescaler

Prescalerul realizează divizarea frecvenței, numărând 2^{prescale} cicluri înainte de a incrementa contorul intern. Implementarea folosește un registru pe 32 de biți:

```
// (32'd1 << prescale) este echivalent cu 2^prescale
if (presc_cnt >= (32'd1 << prescale) - 1) begin
    presc_cnt <= 32'd0;
```

```
    presc_tick <= 1'b1;
end else begin
    presc_cnt <= presc_cnt + 32'd1;
end
```

Comportamentul de actualizare și COUNTER_EN

- Dacă **counter_en = 0**, contorul își menține valoarea (îngheață).
- Actualizarea registrelor (ex: PERIOD) are efect imediat asupra logicii combinaționale a contorului. Condiția de wrap (**count_val** $\geq$ **period**) asigură că, dacă noua perioadă este mai mică decât valoarea curentă a contorului, acesta se resetează sincron la 0 (pentru numărare în sus) sau la noua perioadă (pentru numărare în jos).

Wrap-around

Notă: Condiția **count_val** $\geq$ **period** a fost aleasă pentru a evita blocarea contorului în cazuri rare în care ar depăși perioada (de exemplu la modificarea dinamică a registrului PERIOD).

- Count up: reset la 0 dacă **count_val** $\geq$ **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 wrap-ului

Deoarece numărătorul și generatorul PWM sunt sincrone, detectarea momentului de wrap (final de perioadă) se face simplu prin compararea valorii curente a contorului cu perioada setată:

```
wire is_wrap = (count_val == period);
```

Această condiție este utilizată pentru a forța starea semnalului la începutul unui nou ciclu.

Mod aliniat (FUNCTIONS[1]=0)

La fiecare wrap:

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

Comutarea are loc la **compare1**.

Cazuri particulare (Edge Cases)

Un caz particular tratat în mod specific este situația în care **compare1** este 0 în modul aliniat la stânga. Conform specificațiilor hardware implementate, semnalul rămâne 0 (duty cycle 0%):

```
if (compare1 == 16'h0000) begin
    pwm_out <= 1'b0;
end
```

Mod nealiniat (FUNCTIONS[1]=1)

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

- 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** este setată asincron la 0 sau își menține starea în funcție de reset, asigurând un start curat.

2.6 Integrarea în Top Module (top.v)

Modulul `top.v` instanțiază și interconectează toate submodulele prezentate anterior. O modificare importantă realizată în acest fișier o reprezintă definirea corectă a direcției porturilor pentru interfața SPI în mod *Slave*:

- **mosi** (Master Out Slave In) este configurat ca **input**, primind date de la master.
- **miso** (Master In Slave Out) este configurat ca **output**, transmițând date către master.

Această configurare asigură compatibilitatea corectă cu testbench-ul și respectă specificațiile protocolului SPI pentru un dispozitiv slave.

2.7 Adaptarea Testbench-ului (testbench.v)

Pentru a valida funcționalitatea corectă a designului și a se alinia cu modificările aduse modulului `top.v`, testbench-ul a primit următoarele adaptări:

- **Maparea semnalelor SPI:** Instanțierea modulului `dut` a fost actualizată pentru a reflecta direcția corectă a porturilor. Semnalul `tb_mosi` (definit ca `reg`) este conectat la intrarea `mosi` a perifericului, iar ieșirea `miso` a perifericului este captată pe firul `tb_miso`.
- **Creșterea frecvenței de simulare:** Parametrul CLK_HALF a fost redus de la 50 la 10. Aceasta implică o creștere a frecvenței ceasului de sistem (**clk**) față de ceasul SPI (**sclk**), permițând o eșantionare mai precisă și verificarea stabilității la viteze mai mari.
- **Corecții:** A fost adăugată scrierea explicită în registrul REG_FUNCTIONS pentru a activa modul *Range Between Compares*. În versiunea inițială, această configurare lipsea, iar testul rula incorect folosind setările reziduale de la testul anterior.

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, cu tratarea cazurilor limită.