

Implementare Generator Semnal PWM

Ciobanu Florinela-Bianca

Călimoceanu Răzvan

Grupa 333AA

1 Introducere

Proiectul vizează dezvoltarea unui Periferic Generator de Semnal PWM (Pulse Width Modulation) format din cinci componente majore: **Bridge-ul SPI**, **Decodorul de Instrucțiuni**, **Fișierul de Registri**, **Numărătorul** și **Generatorul PWM**. Această documentație prezintă implementarea modulară a perifericului, concentrându-se pe interfața de comunicație și pe structura logică a întregului sistem.

2 Modificări realizate în fișierul `top.v`

Pentru funcționarea corectă a implementării am realizat următoarele modificări în scheletul inițial:

- Eliminarea redeclarărilor de tip `wire` pentru semnalele `clk`, `rst_n`, `sclk`, `cs_n`, `miso` și `mosi`, deoarece acestea sunt deja declarate ca porturi de intrare/ieșire în antetul modulului.
- La instanțierea modulului `spi_bridge`, am inversat conectarea semnalelor `miso` și `mosi` pentru a respecta direcția datelor (input la input, output la output) și am adăugat parametri lipsă: `byte_sync`, `data_in`, `data_out`.
- La instanțierea modulului `instr_dcd`, am adăugat parametrul `byte_sync`.

3 Bridge-ul de Comunicație SPI (`spi_bridge`)

Modulul `spi_bridge` acționează ca interfață **Slave SPI**, convertind datele seriale primite (MOSI) în date paralele și pregătind datele interne paralele pentru transmisia serială (MISO).

3.1 Reguli SPI Implementate

Implementarea respectă următoarele reguli de timing:

- **CPOL=0, CPHA=0:** Datele sunt **scrise** pe frontul descrescător al ceasului (`sclk`) și sunt **citite** pe frontul crescător.
- **MSB First:** Bitul cel mai semnificativ este transmis primul.
- **Sincronizare ceas:** S-a considerat că ceasul SPI (`sclk`) și ceasul perifericului (`clk`) sunt **sincrone** (10MHz).

3.2 Logica de Recepție (MOSI)

Logica de recepție rulează pe frontul crescător al ceasului SPI.

- **Shift Register (shift_in):** La fiecare front crescător al ceasului (`always @(posedge sclk)`), bitul `mosi` este preluat și shiftat la stânga.
- **Generare Sincronizare Byte (byte_sync):** După 8 biți primiți, semnalul `byte_sync` este pulsat timp de un ciclu.
- **Ieșirea Paralelă (data_in):** Byte-ul recepționat este pus la dispoziția decodurului.

3.3 Logica de Transmitere (MISO)

Logica de transmisie rulează pe frontul descrescător al ceasului SPI.

- **Tri-state:** `miso` este deconectat (`1'bZ`) când `cs_n` este High.
- **Preload:** Când slave-ul este inactiv, registrul `shift_out` este încărcat cu `data_out`.
- **Transmisie:** La fiecare negedge de `sclk`, se transmite MSB-ul curent.

4 Decodorul de Instrucțiuni (instr_dcd)

Decodorul funcționează ca un **Automat cu Stări Finite (FSM)** cu două stări (Setup și Data), fiind responsabil de interpretarea comenzilor primite de la `spi_bridge`.

4.1 Sincronizarea Semnalului `byte_sync`

Deoarece semnalul `byte_sync` generat de `spi_bridge` are o durată nedeterminată raportată la ceasul de sistem (fiind generat în domeniul de ceas SPI lent), utilizarea sa directă ar putea cauza tranziții multiple eronate în FSM.

Pentru a remedia acest aspect, am implementat un mecanism de **detectie a frontului crescător** folosind un sincronizator cu 3 niveluri de regiștri:

- `byte_sync_d1`: Primul nivel preia semnalul asincron, protejând sistemul împotriva metastabilității.
- `byte_sync_d2`: Al doilea nivel stabilizează semnalul în domeniul de ceas curent.
- `byte_sync_d3`: Al treilea nivel memorează starea anterioară a semnalului.

FSM-ul avansează doar atunci când este detectat un front crescător valid, definit prin condiția logică:

$$\text{byte_sync_d2} == 1 \wedge \text{byte_sync_d3} == 0$$

Aceasta garantează procesarea unei singure instrucțiuni per byte recepționat.

4.2 Structura FSM

Automatul ciclează între două stări:

1. **Starea Setup (0):** Decodifică primul byte pentru a extrage tipul operației (R/W), zona de registru (High/Low) și adresa.
2. **Starea Date (1):** Execută efectiv transferul de date (citire sau scriere) și revine în Setup.

5 Fișierul de Regiștri (regs)

Modulul **regs** implementează fișierul configurabil de registre al perifericului. Acesta reprezintă interfața principală între software (prin SPI) și hardware (counter și generatorul PWM).

5.1 Responsabilități

- Interpretarea operațiilor de citire și scriere primite de la decodor.
- Maparea fiecărui registru la adrese pe un byte (inclusiv utilizarea segmentelor LSB/MSB).
- Expunerea valorilor către modulele **counter** și **pwm_gen**.
- Implementarea registrului **COUNTER_RESET** cu auto-clear.
- Menținerea semnalului read-only **COUNTER_VAL**.

5.2 Comportament

- Scrierile se efectuează doar când **write = 1**.
- Citirile returnează valoarea corectă, altfel 0 pentru adrese invalide.
- Resetul global readuce toate registrele la valorile implicite.

6 Numărătorul (counter)

Modulul **counter** furnizează baza de timp pentru generarea semnalului PWM, implementând numărătoarea controlată prin registrele configurabile.

6.1 Funcționalitate

- Numărare crescătoare sau descrescătoare în funcție de **UPNOTDOWN**.
- Limitarea prin valoarea **PERIOD**.
- Gestionarea overflow-ului și underflow-ului:
 - Dacă se numără în sus și se atinge **PERIOD**, contorul revine la 0.

- Dacă se numără în jos și se atinge 0, contorul revine la `PERIOD`.
- Prescaler implementat ca exponent: update la fiecare $2^{prescale}$ cicluri.
- Resetare instantanee a contorului la activarea `COUNTER_RESET`.

6.2 Comportament

- `EN = 0` oprește contorul.
- `EN = 1` permite avansarea contorului pe baza prescalerului.

7 Generatorul PWM (`pwm_gen`)

Modulul `pwm_gen` sintetizează semnalul de ieșire pe baza configurării registrelor și a valorii curente a numărătorului. Implementarea utilizează o logică de decizie ierarhică (bazată pe priorități) pentru a asigura un comportament determinist în toate situațiile.

7.1 Logica de Prioritate și Cazuri Limită

Pentru a preveni comportamente nedefinite, am implementat reguli de siguranță cu prioritate maximă:

- **Protecție la Egalitate:** Dacă `COMPARE1 == COMPARE2`, ieșirea este forțată la 0 logic. Aceasta previne conflictele de comutare simultană.
- **Protecție la Zero:** Dacă `COMPARE1 == 0`, ieșirea este menținută la 0 pentru a asigura un factor de umplere (duty cycle) de 0% corect.

7.2 Moduri de Funcționare Standard

Dacă nu sunt întâlnite condițiile de mai sus, generatorul operează conform modului selectat:

7.2.1 Mod Aliniat (`FUNCTIONS[1] = 0`)

- **Aliniere la Stânga** (`FUNCTIONS[0] = 0`):
 - Semnalul devine 0 la atingerea pragului `COMPARE1`.
 - Semnalul devine 1 la finalul perioadei (când `count_val == PERIOD`), pregătind startul noului ciclu.
- **Aliniere la Dreapta** (`FUNCTIONS[0] = 1`):
 - Semnalul devine 1 la atingerea pragului `COMPARE1`.
 - Semnalul este resetat la 0 la începutul perioadei (când `count_val == 0`).

7.2.2 Mod Nealinat (FUNCTIONS[1] = 1)

În acest mod, prioritățile sunt esențiale pentru definirea formei de undă:

1. **Resetare:** Prioritate 1 - Semnalul devine 0 la COMPARE2.
2. **Setare:** Prioritate 2 - Semnalul devine 1 la COMPARE1.
3. **Default:** Prioritate 3 - Semnalul este resetat la 0 la începutul ciclului (`count_val == 0`).

7.3 Controlul Ieșirii

Semnalul PWM_EN acționează ca un comutator global. Când este inactiv (0), ieșirea își păstrează ultima stare cunoscută (latch), iar logica de generare este ignorată.