Generator semnal PWM
Documentatie

Cioroiu Silvia, Grasu Costin-Alexandru, Tancu Miruna

November 2025

Cuprins

1 Bridge de comunicatie

1.1 Comportamentul protocolului SPI

Protocolul SPI este un protocol de comunicație sincronă, în care masterul controlează întregul schimb de date. În modul CPOL=0, CPHA=0, comportamentul semnalelor este următorul:

- Bitul de date este transmis de master pe `MOSI` pe **frontul descrescător** al semnalului `SCLK`.
- Modulul nostru citește bitul de intrare de pe `MISO` pe **frontul crescător** al lui `SCLK`, moment în care datele sunt valide.
- Modulul actualizează ieșirea `MOSI` pe **frontul descrescător** al lui `SCLK`.
- Comunicarea rămâne activă doar cât timp `CS_N` este **LOW (0)**.
- Transferul se face **MSB-first**.

1.2 Logica utilizată în proiect şi justificare

În implementarea noastră, semnalul `SCLK` nu provine dintr-un domeniu de clock extern, ci este generat în interiorul FPGA-ului şi este sincron cu `clk`. Acest lucru elimină necesitatea mecanismelor de sincronizare suplimentare și simplifică arhitectura generală.

1.2.1 1. Fără necesitatea sincronizării (CDC)

Într-un design SPI clasic, semnalele `SCLK`, `MOSI` și `CS_N` sunt asincrone față de ceasul intern și necesită sincronizare.

În cazul nostru:

- `SCLK` este generat intern și este sincron cu `clk`,
- nu sunt necesare registre de sincronizare,
- logica poate utiliza direct fronturile lui `SCLK`.

Consecință: design-ul este mai simplu, mai eficient și lipsit de probleme de metastabilitate.

1.2.2 2. Detectarea fronturilor lui SCLK

Detectarea fronturilor lui `SCLK` se face direct prin sensibilitatea blocurilor `always`:

- `always @(posedge sclk)` — recepția datelor,
- `always @(negedge sclk)` — transmisia datelor.

Această abordare respectă cerințele protocolului SPI în modul CPOL=0, CPHA=0.

1.2.3 3. Registre separate pentru recepție și transmisie

Design-ul utilizează două registre distincte:

- `r_slave` — registru de recepție în care sunt acumulate bit cu bit datele primite de pe `MISO`,
- `r_master` — registru de transmisie care conține byte-ul ce urmează a fi transmis pe `MOSI`.

Separarea registrelor permite recepția și transmisia simultană a datelor fără conflicte.

1.2.4 4. Contorul de biți și generarea unui byte complet

Pentru fiecare **front crescător** al lui `SCLK`, contorul intern este incrementat. Când acesta ajunge la valoarea `7`:

- byte-ul complet este format,
- valoarea este transferată către `data_in`,
- se generează un puls `byte_sync` activ pentru un singur front de `SCLK`.

Resetarea contorului se face la dezactivarea semnalului `CS_N`.

1.2.5 5. Controlul prin semnalul `CS_N`

Semnalul `CS_N` controlează durata unei tranzacții SPI. Când `CS_N` devine HIGH:

- transferul SPI este oprit,
- contorul de biți este resetat,
- registrul `r_master` este încărcat cu valoarea `data_out`,
- ieșirea `MOSI` este forțată la valoarea logică 0.

1.2.6 6. Avantajele arhitecturii folosite

- Nu necesită sincronizare CDC.
- Implementare precisă a protocolului SPI.
- Arhitectură simplă și robustă.

1.3 Implementarea în cod

Funcționarea bridge-ului SPI este rezumată astfel:

- Pe frontul crescător al lui `SCLK`, bitul de pe `MISO` este shiftat în registrul `r_slave`.
- După recepția a 8 biți, `data_in` este actualizat și se generează `byte_sync`.
- Pe frontul descrescător al lui `SCLK`, se transmite bitul curent din `r_master` pe `MOSI`.
- Registrul `r_master` este reîncărcat cu `data_out` la dezactivarea `CS_N` sau imediat după generarea `byte_sync`.

2 Decodorul de instrucțiuni

2.1 Descriere Generală

Modulul `instr_dcd` (Instruction Decoder) realizează interfața logică dintre bridge-ul SPI și blocul de regiștri. Rolul său este de a interpreta fluxul de octeți primit de la masterul SPI și de a genera semnalele de control necesare pentru operațiile de citire și scriere în spațiul de regiștri.

Decodorul implementează un protocol simplificat, în care fiecare comandă este formată din **două octeți consecutivi**:

- un octet de comandă (instrucțiune),
- un octet de date (valoare scrisă sau dummy în cazul citirii).

Arhitectura modulului este bazată pe o **mașină de stări finite (FSM)** care asigură separarea clară între recepția instrucțiunii și executarea efectivă a accesului la regiștri.

2.2 Interfața Modulului

2.2.1 Interfața către SPI Bridge

Semnal	Direcție	Descriere
`byte_sync`	Input	Puls de un ciclu generat de bridge la recepția completă a unui octet SPI.
`data_in`	Input	Octetul recepționat de la master prin SPI.
`data_out`	Output	Octetul transmis înapoi către master în urma unei operații de citire.

2.2.2 Interfața către Blocul de Regiștri

Semnal	Direcție	Descriere
`read`	Output	Puls de citire activ timp de un ciclu de ceas.
`write`	Output	Puls de scriere activ timp de un ciclu de ceas.
`addr`	Output	Adresa registrului accesat (6 biți).
`data_write`	Output	Octetul ce va fi scris în registru la operații de WRITE.
`data_read`	Input	Octetul citit din registru, furnizat de blocul de regiștri.

2.3 Structura Instrucțiunii

Primul octet transmis de master conține informațiile de comandă:

Bit	Semnificație	Descriere
7	Read/Write	1 = operație de scriere, 0 = operație de citire
6	Neutilizat	Rezervat (setat la 1 de master, ignorat de decodor)
5:0	Address	Adresa registrului (6 biți)

2.4 Mașina de Stări (FSM)

Decodorul utilizează o FSM cu trei stări:

- **S_IDLE** – așteaptă octetul de instrucțiune,
- **S_SETUP** – așteaptă octetul de date,
- **S_ACCESS** – execută operația de citire.

2.4.1 S_IDLE – Recepția Instrucțiunii

La activarea semnalului `byte_sync`, decodorul:

- salvează octetul de instrucțiune,
- extrage bitul Read/Write,
- stabilește adresa registrului (`addr`),
- trece în starea `S_SETUP`.

2.4.2 S_SETUP – Recepția Datelor

În această stare:

- pentru **WRITE**: datele sunt preluate din `data_in`, se generează un puls `write`, iar FSM revine în `S_IDLE`,
- pentru **READ**: FSM avansează în starea `S_ACCESS`.

2.4.3 S_ACCESS – Operație de Citire

În această stare:

- se generează un puls `read` de un ciclu,
- valoarea `data_read` este capturată în `data_out`,
- FSM revine în starea `S_IDLE`.

2.5 Gestionarea Semnalelor de Control

Semnalele `read` și `write` sunt impulsuri curate, active exact un ciclu de ceas, ceea ce:

- previne accesări multiple accidentale,
- asigură compatibilitatea cu logica sincronă a blocului de regiștri,
- simplifică analiza temporală a sistemului.

3 Blocul de Regiștrii

3.1 Descriere Generală

Modulul `regs` implementează spațiul de memorie intern al perifericului PWM. Acesta conține registrele de configurare necesare controlului numărătorului și generatorului PWM, precum și un registru de stare read-only care reflectă valoarea curentă a numărătorului.

Arhitectura este complet sincronă și este optimizată pentru acces pe octeți, permițând gestionarea registrelor de 16 biți prin două accesări consecutive de 8 biți.

3.2 Tipuri de Registre

Blocul de regiștri conține:

- registre **R/W** (citire și scriere),
- registre **W** (doar scriere),
- registre **R** (doar citire).

3.3 Logica de Scriere

Scrierea este realizată într-un bloc secvențial sincron cu ceasul `clk`. Când semnalul `write` este activ, adresa este decodificată printr-o instrucțiune `case`, iar registrul corespunzător este actualizat.

- Registrele de 16 biți sunt mapate pe două adrese consecutive (LOW / HIGH byte).
- Registrele de 1 bit utilizează exclusiv bitul 0 din `data_write`.

Registru cu auto-clear:
Registrul `COUNTER_RESET` generează un puls de reset sincron pentru numărător și este șters automat în ciclul următor de ceas, asigurând un semnal de reset de durată exactă.

3.4 Logica de Citire

Citirea este implementată combinational, permițând acces instant la conținutul registrelor atunci când semnalul `read` este activ.

- Registrele de 1 bit sunt extinse la 8 biți prin zero-padding.
- Registrul `COUNTER_VAL` este read-only și reflectă direct valoarea intrării `counter_val`, fără stocare internă.
- Accesările la adrese neimplementate returnează valoarea `0x00`.

3.5 Ieșiri Continue către Modulele Funcționale

Valorile registrelor sunt conectate permanent către modulele `counter` și `pwm_gen` prin atribuiri continue (`assign`). Această abordare permite actualizarea imediată a parametrilor de funcționare, fără latență suplimentară.

3.6 Considerații de Design

- Toate actualizările sunt sincrone cu ceasul sistemului.
- Logica de citire combinațională minimizează latența de acces.
- Separarea clară între stocare, decodificare și interfață crește modularitatea și ușurința extinderii.

4 Numărătorul

4.1 Descriere Generală

Modulul `counter` reprezintă inima perifericului generator de PWM, oferind baza de timp necesară funcționării acestuia. Deoarece ceasul sistemului (`clk`) are o frecvență fixă, perifericul necesită un mecanism flexibil pentru a controla durata perioadei semnalului PWM și rezoluția acestuia.

Arhitectura modulului este compusă din două elemente funcționale:

1. **Prescaler-ul:** Un divizor de frecvență programabil care generează un semnal de validare (*tick*) la intervale specifice de timp.
2. **Numărătorul Principal:** Un registru pe 16 biți care evoluează (crescător sau descrescător) doar la primirea semnalului de la prescaler.

4.2 Interfața Modulului

Semnalele de intrare și ieșire sunt descrise în tabelul de mai jos:

4.3 Descrierea funcționării detaliate

Funcționarea modulului se bazează pe interacțiunea dintre un contor intern de prescalare și contorul principal expus către restul sistemului.

4.3.1 Prescaler

Pentru a obține perioade lungi ale semnalului PWM fără a mări dimensiunea numărătorului principal, folosim un prescaler exponențial. Valoarea țintă a prescaler-ului este calculată hardware prin operația de deplasare logică la stânga (*shift left*), implementând formula $2^{\texttt{prescale}}$:

$$\texttt{prescale\_target} = 1 \ll \texttt{prescale} \tag{1}$$

Nume Port	Direcție	Lățime	Descriere
`clk`	Input	1	Ceasul sistemului. Toate tranzițiile sunt sincrone cu acest ceas.
`rst_n`	Input	1	Reset asincron global (Active Low). Aduce registrele la valoarea 0.
`count_val`	Output	16	Valoarea curentă a numărătorului, utilizată de generatorul PWM pentru comparare.
`period`	Input	16	Valoarea maximă de numărare (TOP). Definește perioada semnalului PWM.
`en`	Input	1	Semnal de activare. Dacă este 0, numărătorul principal îngheață la valoarea curentă, iar prescaler-ul este resetat.
`count_reset`	Input	1	Reset sincron. Resetează forțat valoarea numărătorului la 0 la următorul front de ceas.
`upnotdown`	Input	1	Direcția de numărare: 1 = Crescător (0 → Period) 0 = Descrescător (Period → 0)
`prescale`	Input	8	Factorul de divizare a frecvenței (exponent al lui 2).

Table 1: Interfața modulului Counter

Mecanismul de Tick: Un contor intern (`prescale_counter`) se incrementează la fiecare ciclu de ceas. Când acesta atinge valoarea `prescale_target - 1`, modulul generează un semnal intern de tip puls numit `prescale_tick`.

- Dacă `prescale` = 0, ținta este 1, deci tick-ul se generează la fiecare ciclu (frecvență maximă).
- Dacă `prescale` = 2, ținta este 4, deci tick-ul se generează o dată la 4 cicli.

4.3.2 Logica Numărătorului Principal

Numărătorul principal (`count_val`) este actualizat **doar** atunci când semnalul `prescale_tick` este activ. Acest lucru asigură sincronizarea perfectă cu ceasul sistemului, indiferent de factorul de divizare.

Comportamentul de numărare depinde de intrarea `upnotdown`:

- **Mod Crescător (`upnotdown` = 1):** Numărătorul evoluează de la 0 la `period`. Când `count_val` este egal cu `period`, la următorul tick valid, valoarea se resetează la 0 (Overflow).

$$0 \rightarrow 1 \rightarrow ... \rightarrow \texttt{period} \rightarrow 0...$$

- **Mod Descrescător (`upnotdown` = 0):** Numărătorul evoluează de la valoarea curentă în jos. Când `count_val` atinge 0, la următorul tick valid, se reîncarcă cu valoarea din `period` (Underflow).

$$... \rightarrow 1 \rightarrow 0 \rightarrow \texttt{period} \rightarrow ...$$

4.3.3 Control și Resetare

Modulul implementează o ierarhie strictă a semnalelor de control:

1. **Reset Asincron (`rst_n`):** Are cea mai mare prioritate. Când este activ (0), toate registrele sunt șterse instantaneu.
2. **Reset Sincron (`count_reset`):** Dacă este activ, aduce numărătorul principal la 0 pe frontul crescător al ceasului. De asemenea, resetează prescaler-ul pentru a realinia faza semnalului.
3. **Enable (`en`):**
   - Dacă `en = 0`: Prescaler-ul este ținut în reset (valoarea 0), ceea ce blochează generarea tick-urilor. Numărătorul principal își păstrează valoarea curentă ("îngheață").
   - Dacă `en = 1`: Prescaler-ul începe să numere, permițând actualizarea numărătorului principal.

5 Generatorul de PWM

5.1 Descriere Generală

Modulul `pwm_gen` este responsabil pentru sinteza efectivă a semnalului digital modulat în durată (PWM - Pulse Width Modulation). Acesta primește starea curentă a numărătorului (baza de timp) și configurația definită în regiștri, comparându-le continuu pentru a decide starea logică a ieșirii `pwm_out`.

Acest semnal este destinat controlului dispozitivelor externe (LED-uri, motoare, drivere), iar flexibilitatea sa este asigurată de multiplele moduri de operare (aliniere stânga, dreapta sau definită între două praguri).

5.2 Interfața Modulului (Porturi)

5.2.1 Semnale de sistem

Nume Port	Direcție	Descriere
`clk`	Input	Ceasul sistemului. Folosit pentru a sincroniza ieșirea `pwm_out`, evitând glitch-urile.
`rst_n`	Input	Reset asincron (Active Low). Aduce ieșirea în starea 0.

Table 2: Semnale de sistem PWM

5.2.2 Intrări de configurare și stare

Nume Port	Direcție	Descriere
`pwm_en`	Input	Bit de activare a ieșirii PWM. Acționează ca un întrerupător principal (”Master Switch”).
`period`	Input	Perioada semnalului. Folosită pentru a detecta momentul de resetare (wrap-around) al numărătorului.
`functions`	Input	Registru de configurare a modului: Bit 0: Aliniere (0=Stânga, 1=Dreapta) Bit 1: 0=Aliniat, 1=Nealiniat.
`compare1`	Input	Valoare de prag principală. Determină factorul de umplere (Duty Cycle) sau momentul comutării.
`compare2`	Input	Valoare de prag secundară. Folosită doar în modul ”Nealiniat” pentru a marca sfârșitul pulsului activ.
`count_val`	Input	Valoarea curentă a numărătorului (primită de la modulul `counter`).

Table 3: Semnale de configurare PWM

5.2.3 Ieșire

Nume Port	Direcție	Descriere
pwm_out	Output	Semnalul PWM final, sintetizat și sincronizat cu ceasul sistemului.

Table 4: Semnale de ieșire PWM

5.3 Descrierea funcționării detaliate

O provocare majoră în proiectarea acestui modul a fost lipsa semnalului de direcție (upnotdown) la intrare, deși comportamentul PWM depinde de momentul începerii unui nou ciclu. Soluția adoptată este o arhitectură "bazată pe evenimente" (*Event-Based*), care permite funcționarea corectă indiferent dacă numărătorul urcă sau coboară.

5.3.1 Detecția începutului de ciclu

Pentru a inițializa corect starea semnalului PWM la începutul fiecărei perioade, modulul trebuie să detecteze momentul de "wrap-around" al numărătorului. Acest lucru se realizează prin monitorizarea istorică a valorii count_val:

- Se folosește un registru intern count_val_prev pentru a memora valoarea din ciclul anterior.
- Se generează un semnal intern cycle_start_event activ pentru un ciclu de ceas dacă:
  - **Overflow:** Valoarea anterioară era period și valoarea curentă este 0.
  - **Underflow:** Valoarea anterioară era 0 și valoarea curentă este period.

De asemenea, se generează semnalele compare1_event și compare2_event prin compararea directă a valorii curente cu pragurile setate.

5.3.2 Logica Next-State (Combinațională + Secvențială)

Logica de generare a semnalului este implementată în două etape pentru a asigura stabilitatea și a preveni comportamentele nedorite:

A. Logica Combinațională (always @(*))
Calculează starea viitoare (pwm_out_next) pe baza stării curente, a evenimentelor detectate și a modului de operare.

Gestionarea semnalului pwm_en: Dezactivarea PWM-ului trebuie să "lase blocată linia în starea în care se afla". Acest lucru este implementat prin feedback implicit:

```
pwm_out_next = pwm_out_reg; // Default: menține starea anterioară
if (pwm_en) begin
    // ... logica de modificare a stării doar dacă en este activ
end
```

B. Logica Secvențială (`always @(posedge clk)`)
Actualizează registrul de ieșire `pwm_out_reg` cu valoarea calculată, sincron cu ceasul sistemului. Această separare elimină glitch-urile care ar putea apărea în logica combinațională pură.

5.3.3 Moduri de operare implementate

Comportamentul este dictat de registrul `functions` și de evenimentele de comparare:

1. **Aliniere Stânga (`functions` = 2'b00):**
    - La `cycle_start_event`: Ieșirea devine **1** (începe activ).
    - La evenimentul `compare1`: Ieșirea comută în **0**.
2. **Aliniere Dreapta (`functions` = 2'b01):**
    - La `cycle_start_event`: Ieșirea devine **0** (începe inactiv).
    - La evenimentul `compare1`: Ieșirea comută în **1**.
3. **Nealiniat (`functions` = 2'b1x):**
    - La `cycle_start_event`: Ieșirea devine **0**.
    - La evenimentul `compare1`: Ieșirea comută în **1**.
    - La evenimentul `compare2`: Ieșirea comută în **0**.