Tema Generator PWM

Tudor Ana Maria ,Ilie Ioana Laura

14 Decembrie 2025

1 Structura Functionala si Implementare Modulara

Designul generatorului PWM (Pulse Width Modulation) este realizat printr-o arhitectura separand functionalitatile de control si executie. Modulele implementate sunt: `spi_bridge`, `instr_dcd`, `regs`, `counter`, si `pwm_gen`.

1.1 Modulul `counter`

`counter` utilizeaza un registru de contorizare `count_reg` (16 biti) si un registru auxiliar `presc_cnt` pentru prescaling.

- **Prescaling**: Factorul de prescaling este determinat de bitii `prescale[3:0]`. Limita `presc_limit` este derivata prin operatia $2^{\text{prescale}[3:0]}$ (implementata ca shift logic: `16'd1` $\ll$ `prescale[3:0]`), definind frecventa efectiva de incrementare a `count_reg`.
- **Control directie si Limite**: Directia de numarare (UP/DOWN, `upnotdown`) si valoarea de resetare (`period`) sunt programabile:
  - UP: Ciclul se incheie la count_reg = period − 1, urmat de reset la 0 (Overflow).
  - DOWN: Ciclul se incheie la count_reg = 0, urmat de reset la period − 1 (Underflow).
- **Reset Asincron/Sincron**: Suporta reset asincron de sistem (`rst_n`) si reset sincron, pulsat (`count_reset`), ambele fortand `count_reg` si `presc_cnt` la 0.

1.2 Modulul `instr_dcd`

Acest modul implementeaza un FSM (Finite State Machine) cu doua stari pentru decodarea instructiunilor primite pe bus-ul de 8 biti.

- **Starea** `ST_SETUP`: Se primeste primul byte (`data_in`) al instructiunii, din care se extrag parametrii de control: R/W (`data_in[7]`), High/Low Byte (`data_in[6]`), si adresa de registru (`data_in[5:0]`).
- **Starea** `ST_DATA`: Executia operatiei.
  - Operatie Write (`instr_rw`=1): Se primeste al doilea byte, `data_in`, si se activeaza pulsul `write` (un singur ciclu).
  - Operatie Read (`instr_rw`=0): Se preia `data_read` de la `regs` si se activeaza pulsul `read` (un singur ciclu).
- **Adresare 16 biti**: Adresa efectiva *addr* este generata ca addr = instr_addr + instr_highlow, facilitand accesul secvential la LSB si MSB.

1.3 Modulul `regs`

Bancul de registri (Registers) asigura interfata de memorie intre `instr_dcd` si celelalte module.

- **Mapare Adrese**: Adresele registrelor (ex: `ADDR_PERIOD_L` - $6'h00$, `ADDR_PERIOD_H` - $6'h01$) sunt definite explicit, asigurand o mapare clara pe bus-ul de 8 biti.
- **Acces 16 biti**: Scrierile pe registrele de 16 biti se realizeaza pe doi octeti consecutivi, direct in blocurile LSB (`[7:0]`) si MSB (`[15:8]`).
- **Reset Pulsat**: Scrierile pe `ADDR_COUNTER_RESET` seteaza intern `count_reset` la 1, urmat de dezactivarea automata (*self-clearing*) la 0 in ciclul de ceas urmator, generand un puls unic.
- **Citire Status**: `counter_val` (valoarea curenta a contorului) este transmisa combinational prin `data_read` la adresele $6'h08/6'h09$.

1.4 Modulul `pwm_gen`

Acest modul genereaza semnalul PWM (`pwm_out`) pe baza comparatiilor si configuratiei.

- **Logica Combinationala**: Starea viitoare a semnalului (`pwm_next`) este calculata intr-un bloc combinational (`always @*`) pe baza `count_val` si a registrelor `compare1`/`compare2`.
- **Moduri de Operare (`functions[1:0]`)**:
  - 2'b00: Aliniere la Stanga (Edge-aligned): $pwm\_next = (\text{count\_val} \leq \text{compare1})$.
  - 2'b01: Aliniere la Dreapta: $pwm\_next = (\text{count\_val} \geq \text{compare1})$.
  - 2'b10: Mod Fereastra (Window/Double-Edge): $pwm\_next = (\text{count\_val} \geq \text{compare1}) \wedge (\text{count\_val} < \text{compare2})$.
- **Iesire Sincrona**: Semnalul final `pwm_out` este inregistrat in `pwm_reg` pe frontul de ceas pozitiv, conditionat de bitul de enable `pwm_en`.

1.5 Modulul `spi_bridge`

`spi_bridge` realizeaza conversia seriala SPI (dinspre Master) la bus paralel de 8 biti.

- **Protocol**: Utilizeaza ceasul periferic `sclk` pentru shiftare, operatiile fiind sincronizate pe frontul crescator al acestuia. Transmisia este MSB-first.
- **Shift In/Out**: `in_shift` preia datele seriale de pe `mosi`. `out_shift` furnizeaza datele seriale pe `miso`, preincarcate cu `data_out`.
- **Sincronizare Byte**: `byte_sync` este pulsat (activ pentru un ciclu `sclk`) cand `bit_cnt` ajunge la 7, indicand ca un byte complet este disponibil in `data_in`.
- **Clear pe CS_n**: Cand `cs_n` se dezactiveaza, `bit_cnt` se reseteaza la 0, iar `out_shift` se reincarca cu urmatoarele date de raspuns (`data_out`).

2 Analiza Aciclicitatii in *counter*

2.1 Conditia de Terminare a Ciclului

Corectitudinea functionarii modulului *counter* depinde integral de logica de resetare la limita (*period*).

- **UP Mode**: Contorul realizeaza tranzitia corecta de la $period - 1$ la 0.
- **DOWN Mode**: Contorul realizeaza tranzitia corecta de la 0 la $period - 1$.

2.2 Blocajul Contorului

Modulul *counter* ar putea deveni aciclic sau blocat daca registrul *period* este programat cu valori 0 sau 1:

- **Cazul *period* = 0**: In UP mode, expresia $period - 1$ devine $16'hFFFF$, fortand contorul sa numere pana la valoarea maxima , ignorand functionalitatea *period*.
- **Cazul *period* = 1**: Atat in UP cat si in DOWN mode, contorul se va reseta la 0 sau $period - 1 = 0$ imediat ce atinge sau porneste de la 0. Acest lucru creeaza un ciclu trivial $0 \rightarrow 0$ (blocaj), impiedicand generarea unui semnal PWM functional.