

# Documentatie periferic PWM controlat prin SPI

Alin-Marian Mirica

Alex-Florin MARIN-SIRBU

Pavel-Mihai BADEA

## 1 Ideea solutiei

Perifericul este format din mai multe blocuri care comunica intre ele. Comunicarea cu exteriorul se face prin SPI, iar in interior datele sunt prelucrate pe un bus paralel.

Fluxul de functionare este urmatorul:

- datele vin serial prin SPI
- sunt transformate in octeti in modulul SPI bridge
- comenzile sunt interpretate de decodor
- registrele sunt citite sau scrise
- numaratorul si PWM-ul folosesc valorile din registre

Activarea numaratorului (COUNTER\_EN) si activarea PWM-ului (PWM\_EN) sunt complet separate, conform cerintei. Doar atunci cand ambele sunt active, la iesirea pwm\_out apare un semnal PWM.

## 2 SPI Bridge

Comunicarea cu exteriorul se face prin protocolul SPI, in modul CPOL = 0 si CPHA = 0, cu transmiterea datelor MSB-first. Perifericul este configurat ca *slave*.

In acest mod de functionare:

- masterul pune datele pe linie pe **frontul descrescator** al semnalului SCLK;
- slave-ul (perifericul nostru) citeste datele pe **frontul crescator** al semnalului SCLK.

Desi enuntul considera ceasul SPI (`sclk`) si ceasul perifericului (`clk`) ca fiind sincrone (ambele 10 MHz), implementarea `spi_bridge` foloseste **direct semnalul sclk** pentru receptia si transmisia bitilor, pentru a evita erori la detectia fronturilor.

## 2.1 Receptie (MOSI)

Pentru receptie:

- pe fiecare front crescator al lui `sclk` se citește un bit de pe linia `MOSI`;
- după 8 biti receptionati se formează un byte complet;
- se generează un semnal intern de tip *toggle* (in domeniul `sclk`), care este apoi sincronizat pe `clk`;
- în urma sincronizării se obține un puls `byte_sync` de un singur ciclu de `clk`, folosit de decodorul de instrucțiuni;
- în momentul pulsului, `data_in` conține byte-ul receptionat.

## 2.2 Transmisie (MISO)

Pentru transmisie:

- datele sunt scoase pe linia `MISO` pe fiecare front descrescator al lui `sclk` (MSB primul);
- la activarea tranzacției (când `cs_n` devine 0), primul bit (MSB) este pus imediat pe `MISO`, astfel încât să fie stabil înainte de primul front crescator (când masterul citește);
- cât timp `cs_n` este 0, pot exista mai multe secvențe consecutive de câte 8 biti, iar modulul continuă să genereze `byte_sync` pentru fiecare byte complet receptionat.

Această implementare respectă cerințele temei pentru SPI simplificat (CPOL=0, CPHA=0, MSB-first) și asigură o funcționare robustă a comunicării.

## 3 Decodorul de instrucțiuni

Modulul `instr_dcd` interpretează comenzile primite prin SPI. Implementarea folosește o mașină de stări simplă, cu două stări.

În prima stare se primește comanda, care conține:

- tipul operației (read / write)
- adresa registrului

Dacă operația este de tip read, semnalul `read` este activat imediat, astfel încât registrul să poată pune datele pe magistrală înainte de al doilea byte SPI.

În a doua stare se primesc datele propriu-zise, folosite doar în cazul unei operații de write. Semnalul `write` este activ un singur ciclu de ceas, pentru a evita scrieri multiple.

## 4 Registri

Modulul `regs` contine toate registrele configurabile ale perifericului.

Scrierea in registre se face sincron cu ceasul `clk`, doar atunci cand semnalul `write` este activ. Citirea este combinationala si depinde de semnalul `read` si de adresa selectata.

Registrul `COUNTER_VAL` este read-only si reprezinta valoarea curenta a numaratorului. Registrul `COUNTER_RESET` reseteaza doar numaratorul si este implementat ca un puls de un singur ciclu, fara a afecta ceilalti parametri.

## 5 Numaratorul

Modulul `counter` implementeaza un numarator pe 16 biti, care poate numara crescator sau descrescator. Incrementarea sau decrementarea se face in functie de semnalul `upnotdown`.

Pentru controlul vitezei de numarare, am implementat un *prescaler*. Acesta este un contor separat, care ajunge pana la valoarea  $2^{prescale} - 1$ . Doar cand prescalerul ajunge la aceasta valoare, numaratorul principal isi modifica valoarea.

Numaratorul functioneaza doar atunci cand `COUNTER_EN` este activ. Valorile din registre pot fi modificate in timpul functionarii, dar efectul lor se vede doar la overflow, underflow sau reset.

## 6 Generatorul PWM

Modulul `pwm_gen` foloseste valoarea curenta a numaratorului pentru a genera semnalul PWM. Modul de functionare este ales prin registrul `FUNCTIONS`.

Sunt implementate urmatoarele moduri:

- PWM aliniat la stanga
- PWM aliniat la dreapta
- PWM activ intre doua valori de compare

PWM-ul este generat doar daca `PWM_EN` este activ. In cazuri speciale, cum ar fi `compare1 == compare2` sau un interval invalid, iesirea este fortata la 0 pentru a evita comportamente neasteptate.

## 7 Testare

Am adaugat un test suplimentar pentru dezactivarea PWM-ului in timpul functionarii, pentru a verifica faptul ca iesirea este fortata la 0 indiferent de starea numaratorului.

In mare parte, testele din testbench-ul dat au trecut - ce am facut pana acum functioneaza conform cerintei (*hopefully*).