Documentație tehnică – Implementare periferic PWM cu interfață SPI

Floricescu Adrian
Gatej Ștefan
Potop Horia

23 noiembrie 2025

1 Introducere

Acest proiect implementează un periferic hardware dedicat generării semnalelor PWM (Pulse Width Modulation), similar celor întâlnite în microcontrolere comerciale, cum ar fi ATmega328P utilizat pe platformele Arduino Uno. Perifericul este proiectat pentru a fi integrat într-un SoC sau într-o arhitectură digitală mai complexă, comunicând cu un sistem extern prin intermediul unei interfețe SPI și expunând funcționalități configurabile prin registre interne.

PWM este un mecanism esențial în controlul elementelor electronice: ajustarea luminozității LED-urilor, controlul motoarelor DC și BLDC, generarea semnalelor de comandă pentru actuatoare, controlul puterii și multe altele. Pentru a răspunde acestor cerințe, perifericul implementat oferă configurarea perioadei semnalului, selectarea modului de aliniere, setarea unui factor de umplere variabil și posibilitatea de a activa/dezactiva semnalul PWM.

Documentația prezentată descrie structura perifericului, funcționarea fiecărui modul și rațiunile arhitecturale care au stat la baza implementării.

2 Considerații generale de arhitectură

Arhitectura perifericului este modulară și urmărește principiile unei proiectări hardware robuste: separarea clară a funcțiilor, sincronizarea semnalelor externe, simplitate în comunicare și posibilitatea de extindere ulterioară. Perifericul este împărțit în cinci componente principale, ilustrate în figura de ansamblu:

- **Interfața SPI (`spi_bridge.v`)** – Primește fluxul serial provenit de la masterul SPI, îl convertește în octeți aliniați și generează semnalul `byte_sync` care indică disponibilitatea unui nou octet valid. Modulul asigură și sincronizarea semnalelor externe cu domeniul intern de ceas.

- **Decodorul de instrucțiuni (`instr_dcd.v`)** – Interpretează fiecare pereche de octeți recepționați de la interfața SPI: primul octet conține tipul operației (citire sau scriere) și adresa, iar al doilea octet reprezintă valoarea efectivă. Modulul generează impulsuri precise de scriere şi citire pentru modulul de registre.

- **Blocul de registre (`regs.v`)** – Stochează toți parametrii configurabili ai perifericului: perioada PWM, valorile de comparaţie, activarea/dezactivarea generatorului și semnalele de control pentru contor. Expune atât registrele configurabile, cât și valori dinamice precum `counter_val`.
- **Contorul (`counter.v`)** – Generează baza de timp a semnalului PWM prin incrementarea sau decrementarea valorii curente, folosind un prescaler configurabil. Acesta împarte semnalul de ceas intern în perioade discrete în funcţie de setările utilizatorului.
- **Generatorul PWM (`pwm_gen.v`)** – Produce semnalul PWM final pe baza comparațiilor dintre valoarea din contor și limitele programate. Logica este complet combinațională pentru a evita glitch-uri și pentru a asigura tranziții predictibile și stabile.

Un element important al arhitecturii este utilizarea unui singur ceas intern (`clk`) pentru toate modulele. Semnalele provenite din lumea externă (în special cele din interfața SPI) sunt e lă.

3 Modulul `spi_bridge`

Modulul `spi_bridge` reprezintă interfața dintre protocolul serial SPI și restul perifericului, convertind fluxul serial în octeți paraleli și asigurând mecanismele necesare pentru transmiterea datelor într-un mod compatibil cu standardul SPI (CPOL = 0, CPHA = 0). Implementarea este strict deterministă și urmărește fidel regulile de eşantionare și sincronizare impuse de protocol.

3.1 Detectarea fronturilor semnalului `sclk`

Protocolul SPI impune eşantionarea datelor MOSI pe frontul crescător al lui `sclk` și actualizarea liniei MISO pe frontul descrescător. Pentru a determina aceste tranziții, implementarea salvează starea anterioară a semnalului în registrul `sclk_prev`. Astfel, se obține:

$$sclk\_rise = (sclk = 1 \wedge sclk\_prev = 0), \qquad sclk\_fall = (sclk = 0 \wedge sclk\_prev = 1)$$

Această metodă este robustă și necesită doar un singur registru, evitând logici suplimentare de sincronizare.

3.2 Gestionarea semnalului `CS_n`

Conform protocolului, `CS_n` este activ pe 0. Atunci când devine 1:

- transferul aflat în desfășurare este anulat,
- contorul de biți `bit_cnt` este resetat,
- registrul de shiftare nu mai acumulează date.

Astfel, orice de-sincronizare cu masterul este eliminată instant.

3.3 Recepția datelor MOSI

Fluxul serial este convertit într-un octet complet prin registrul de shiftare `shift_reg`. Pe fiecare front crescător al lui SCLK:

1. bitul de pe MOSI este introdus în LSB-ul registrului,
2. registrul se deplasează spre MSB,
3. se incrementează `bit_cnt`.

La atingerea a 8 biți:

- octetul este mutat în `r_data_in`,
- semnalul `byte_sync` se activează pentru un singur ciclu,
- `bit_cnt` este resetat.

Acest puls `byte_sync` este esențial pentru decodor, deoarece indică exact momentul în care un octet complet a fost recepționat.

3.4 Transmiterea datelor pe MISO

Pentru CPHA = 0, masterul citește pe front crescător, deci modulul trebuie să plaseze bitul pe MISO pe front descrescător. Implementarea face exact acest lucru:

$$r\_miso \leftarrow data\_out[\,7 - bit\_cnt\,]$$

Aceasta asigură transmisie MSB-first, exact cum o cere protocolul SPI.

3.5 Comportamentul general al implementării

Implementarea realizată este robustă și respectă toate cerințele perifericului:

- detectare sigură a fronturilor lui SCLK,
- reset automat al transferului la ridicarea lui CS,
- conversie serial-paralel cu buffer intern,
- semnalizare precisă a octetului recepționat (`byte_sync`),
- transmisie paralel-serială pe MISO fără întârzieri,
- compatibilitate totală cu comunicarea SPI MSB-first.

Logica este compactă, clară și deterministă, făcând modulul `spi_bridge` un element intermediar sigur între lumea serială SPI și logica paralelă a perifericului PWM.

4 Modulul `instr_dcd`

Modulul `instr_dcd` reprezintă interfața logică dintre protocolul SPI și blocul de registre interne ale perifericului. Acesta transformă fluxul serial de octeți într-o operație atomică de **citire** sau **scriere**, alternativ activând semnalele `read`, `write`, `addr` și `data_write`.

Funcționarea modulului este strict sincronă și organizată sub forma unei mașini de stări finite (FSM), pentru a garanta că niciodată două operații nu se suprapun și că accesul la registre se face în ciclul corect.

4.1 Arhitectura FSM: etapele Setup și Data

FSM-ul este format din două stări principale:

- **Setup** — primul byte recepționat prin SPI este interpretat ca instrucțiune. Acesta conține:
  - bitul `RW` (citire/scriere),
  - bitul `High/Low` (selectarea octetului LSB/MSB),
  - adresa de bază a registrului.

  Acest byte definește complet operația ce urmează.
- **Data** — la recepția celui de-al doilea byte, FSM-ul execută operația:
  - pentru **scriere**, byte-ul este transmis direct către blocul de registre, însoțit de un puls `write`;
  - pentru **citire**, modulul transmite către SPI valoarea `data_read`, generând un puls `read`.

Această separare clară a etapelor asigură faptul că niciodată `read` și `write` nu sunt active în același ciclu, eliminând ambiguitățile și conflictele de acces.

4.2 Decodificarea adresei și selecția LSB/MSB

Pentru registrele pe 16 biți, modulul permite selectarea octetului inferior sau superior prin bitul `High/Low`. Astfel, adresa efectivă trimisă către blocul de registre poate fi:

$$\texttt{addr\_eff} = \texttt{addr} + \begin{cases} 0, & \text{dacă se accesează LSB} \\ 1, & \text{dacă se accesează MSB} \end{cases}$$

Această logică este complet transparentă pentru restul perifericului.

4.3 Observații privind funcționarea

- Semnalul `byte_sync` garantează că FSM-ul procesează exact byte-ul complet din SPI.
- Pulsurile `read` și `write` sunt generate cu durată de un singur ciclu, evitând accesul repetat.

- `data_out` este stabilă la începutul transferului de citire, permițând modulului SPI să o preia curat pe linia MISO.
- Implementarea este complet sincronă pe ceasul perifericului, fără hazarduri sau glitch-uri.

Această arhitectură face ca modulul `instr_dcd` să ofere o interfață robustă și predictibilă între nivelul fizic SPI și nivelul logic al registrelor de configurare.

5 Modulul `regs`

Modulul `regs` reprezintă punctul central prin care logica de configurare a perifericului este stocată şi accesată. Implementarea a fost realizată cu atenţie la următoarele aspecte: scriere atomică pe byte pentru registrele de 16 biţi, generarea impulsului `COUNTER_RESET`, separarea clară dintre registrele reale şi semnalele read-only, precum şi tratarea consecventă a adreselor invalide.

5.1 Scrierea în registre

Pentru toate registrele de 16 biţi, scrierea se face în două etape, folosind adrese consecutive pentru LSB şi MSB. Această abordare respectă protocolul SPI din modulul de decodare şi previne orice hazard legat de actualizări parţiale.

Toate scrierile sunt sincronizate pe frontul crescător al ceasului. Pentru registrele pe un singur bit (`EN`, `UPNOTDOWN`, `PWM_EN`), doar bitul 0 este luat în considerare, restul fiind ignorate.

5.2 Generarea impulsului `COUNTER_RESET`

Registrul `COUNTER_RESET` are un comportament special: orice scriere trebuie să provoace un impuls de exact un ciclu de ceas. În implementare, acest lucru este realizat astfel:

- semnalul este setat la `1` doar în ciclul în care se detectează instrucţia de scriere;
- la fiecare ciclu de ceas, înainte de procesarea unei eventuale scrieri, semnalul este forţat înapoi la `0`;
- în acest mod nu există riscul ca resetul să rămână activ accidental.

Această soluţie garantează un comportament determinist, indiferent de frecvenţa cu care sunt trimise pachetele SPI.

5.3 Citirea valorilor din contor

Registrul `COUNTER_VAL` este read-only şi nu conţine memorie locală. Datele sunt furnizate direct din modulul `counter`. Această decizie evită probleme de coerenţă între valori duplicate şi garantează software-ului acces constant la valoarea reală a contorului.

5.4 Fluxul de citire

Citirea este implementată combinational printr-un multiplexer mare. Avantajele acestei abordări sunt:

- latenţă zero la citire (valoarea devine disponibilă imediat);
- evitarea hazardelor între scriere/citire consecutive;
- claritate în structură — fiecare adresă este mapată explicit.

Dacă semnalul `read` este inactiv, ieşirea este forţată la `0`. Acest comportament simplifică logica din decoder şi elimină riscul ca valori rămase din cicluri anterioare să fie interpretate drept răspuns valid.

5.5 Adrese invalide

Pentru orice adresă în afara hărţii definite:

- scrierea este ignorată complet;
- citirea returnează `0x00`.

Motivul acestei alegeri este prevenirea modificărilor accidentale şi imitarea comportamentului tipic al perifericelor reale.

5.6 Particularităţi ale implementării

- Semnalul `FUNCTIONS` este stocat pe 8 biţi, deşi doar doi sunt folosiţi — aceasta simplifică structura registrelor şi fluxul SPI.
- Toate registrele sunt resetate hardware la `0`, pentru a asigura o stare iniţială deterministă.
- Multiplexarea pentru citire este implementată combinational, evitând orice comportament neintenţionat cauzat de retiming.
- Registrele de configurare nu sunt protejate faţă de scriere în timpul funcţionării; modificările sunt aplicate imediat, dar efectele devin vizibile în PWM abia la overflow, conform cerinţelor arhitecturale.

În ansamblu, modulul `regs` a fost proiectat pentru a menţine claritatea logicii, predictibilitatea comportamentului şi conformitatea cu specificaţia perifericului, păstrând totodată un cod uşor de testat şi de extins.

6 Modulul `counter`

Modulul `counter` reprezintă baza de timp a perifericului şi este responsabil pentru generarea valorii ciclice `count_val`. Implementarea acestui modul reproduce comportamente întâlnite în timere hardware reale, însă include şi o serie de particularităţi necesare pentru integrarea corectă cu restul perifericului.

6.1 Structura internă a implementării

Implementarea utilizează două registre interne:

- `count_val` – valoarea efectivă a contorului, vizibilă extern prin registrul `COUNTER_VAL`;
- `prescale_cnt` – un contor intern care implementează mecanismul de prescaler.

Aceste registre sunt actualizate exclusiv pe frontul crescător al semnalului de ceas, iar resetarea lor este controlată prin două mecanisme distincte:

- resetul global `rst_n`, care readuce modulul în starea inițială;
- registrul `COUNTER_RESET`, care resetează doar contorul şi prescalerul, fără a afecta alte configurări.

6.2 Implementarea prescalerului

Prescalerul este realizat prin incrementarea `prescale_cnt` până la atingerea valorii configurate în registrul `PRESCALE`. Când aceasta este atinsă:

1. prescalerul se resetează la 0;
2. se execută o modificare a contorului (increment sau decrement).

Această abordare permite scalarea frecvenţei interne fără a introduce divizoare de ceas suplimentare şi este complet sincronă.

6.3 Modul de numărare: incrementare/decrementare

Direcţia contorului este controlată prin bitul `UPNOTDOWN`. Implementarea tratează explicit ambele cazuri:

- dacă se numără în sus (`UPNOTDOWN = 1`) şi `count_val` atinge `period`, se produce overflow iar valoarea revine la 0;
- dacă se numără în jos (`UPNOTDOWN = 0`) şi `count_val` ajunge la 0, se produce underflow iar valoarea devine `period`.

Acest comportament ciclic este necesar pentru compatibilitatea cu logica PWM, care îşi bazează tranziţiile pe comparaţia cu registrele `COMPARE1` şi `COMPARE2`.

6.4 Particularităţi ale implementării

Implementarea are câteva decizii intenţionate:

- **Contorul este complet îngheţat când `en = 0`.** Atât contorul cât şi prescalerul nu avansează, ceea ce permite sincronizarea perifericului în timpul reconfigurării.
- **Resetul prin `COUNTER_RESET` nu afectează restul registrelor perifericului.** Aceasta permite repoziţionarea fazei PWM fără a pierde configuraţia curentă.
- **Prescalerul nu poate sări peste cicluri.** Chiar dacă `period` este mic, prescalerul controlează strict momentul incrementării/decrementării.

7 Modulul pwm_gen

Modulul pwm_gen reprezintă blocul responsabil de generarea efectivă a semnalului PWM pe baza valorii count_val provenite de la numărător. În implementarea noastră, logica este integral sincronă pe frontul pozitiv al ceasului şi reflectă exact comportamentul definit în cerință, însă structura codului urmărește o abordare simplă și explicită.

7.1 Structura generală a implementării

Modulul primește prin intrări valorile configurate din registri:

- pwm_en — activează sau îngheață ieșirea PWM,
- functions — controlează modul de generare (aliniat vs. nealiniat),
- compare1 și compare2 — praguri la care semnalul se modifică,
- period — limita superioară a contorului.

Cele două moduri, **aliniat** și **nealiniat**, sunt selectate direct din biții vectorului functions:

$$\text{unaligned} = functions[1], \qquad \text{align\_lr} = functions[0].$$

Implementarea se bazează pe o singură maşină secvenţială (un bloc always sincron), evitând orice hazard combinational.

7.2 Gestionarea semnalului pwm_en

În cadrul implementării, semnalul pwm_en nu resetează PWM-ul, ci doar îl "îngheaţă":

$$pwm\_en = 0 \Rightarrow pwm\_out \text{ rămâne la valoarea precedentă.}$$

Prin această alegere, modulul respectă specificația conform căreia dezactivarea PWM nu afectează faza internă; la reactivare, semnalul își continuă comportamentul normal.

7.3 Comportamentul la overflow

Primul bloc logic important în implementare este tratarea situației:

$$count\_val = period$$

La overflow:

- în mod aliniat — semnalul este reinițializat la 1 (stânga) sau 0 (dreapta);
- în mod nealiniat — semnalul revine întotdeauna la 0.

Această logică este tratată imediat la începutul secțiunii "PWM activ", pentru ca restul funcționalităţii să fie aplicată deasupra acestei stări de bază.

7.4 Modul aliniat

Implementarea modulului aliniat este extrem de compactă:

$$\text{dacă } count\_val = compare1 \Rightarrow pwm\_out = \neg pwm\_out.$$

Nu sunt necesare alte condiții, deoarece overflow-ul stabilește automat valoarea de start a semnalului. Astfel se obține exact comportamentul cerut: o singură alternanță în fiecare ciclu.

7.5 Modul nealiniat

Pentru modul nealiniat, implementarea reflectă întocmai definiția:

- la `compare1` ieșirea devine 1,
- la `compare2` ieșirea devine 0.

Resetarea fazei la overflow (pwm_out = 0) asigură că fereastra dintre praguri este plasată la poziția corectă în fiecare ciclu.

7.6 Concluzii privind implementarea

Implementarea modulului `pwm_gen` este:

- complet sincronă — evită glitch-uri,
- structurată clar pe două moduri distincte,
- minimalistă, fără logică combinațională complexă,
- conformă exact cu cerința arhitecturală,
- ușor de urmărit și verificat în simulări (`GTKWave`).

Codul rezultat generează forme de undă stabile și predictibile în toate scenariile testate (aliniere stânga, aliniere dreapta, mod nealiniat), fapt confirmat prin testbench-ul nostru și capturile GTKWave incluse în secțiunea precedentă.

7.7 Exemple de funcționare în simulare

În această secțiune sunt prezentate trei capturi relevante, fiecare ilustrând un mod distinct de operare. Imaginile sunt longitudinale (wide) pentru a surprinde mai multe perioade succesive ale semnalului PWM.

PWM aliniat la stânga

În acest mod, `FUNCTIONS[1] = 0` (aliniat), iar `FUNCTIONS[0] = 0`, ceea ce determină ca semnalul PWM să înceapă în nivelul logic `1`.

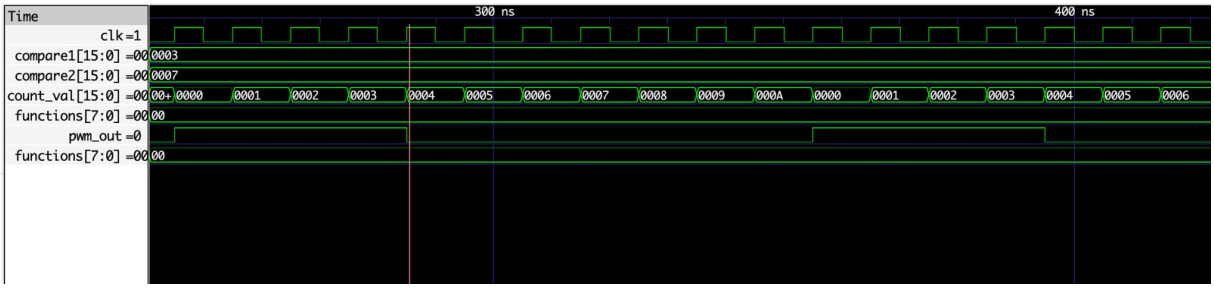

Figura 1: Semnal PWM generat în mod aliniat la stânga.

PWM aliniat la dreapta

În acest exemplu, `FUNCTIONS[1] = 0`, dar `FUNCTIONS[0] = 1`, ceea ce deplasează frontul activ către finalul perioadei.

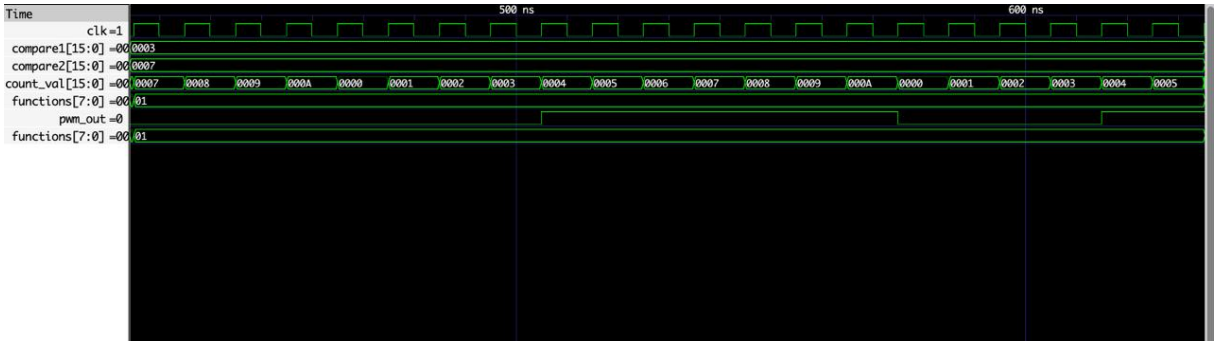

Figura 2: Semnal PWM generat în mod aliniat la dreapta.

PWM nealiniat (window mode)

Când `FUNCTIONS[1] = 1`, semnalul PWM devine activ doar în intervalul:

$$\text{COMPARE1} \leq \text{count\_val} < \text{COMPARE2}.$$

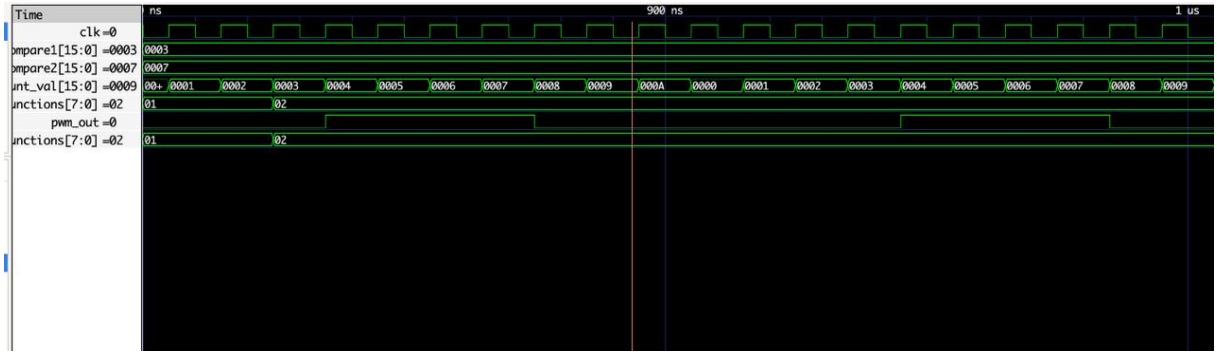

Figura 3: Semnal PWM generat în mod nealiniat.

8 Concluzii

Implementarea finală respectă integral cerințele arhitecturale ale perifericului PWM. Fiecare modul a fost proiectat astfel încât să fie ușor de înțeles, testat și integrat, iar designul rezultat este stabil, predictibil și complet sincron cu ceasul perifericului. Separarea clară între logica SPI, decodificarea instrucțiunilor, registre, numărător și generatorul PWM a permis obținerea unui sistem modular, în care fiecare componentă poate fi analizată și verificată independent.

8.1 Integrarea în modulul `top`

Modulul `top` joacă rolul de "lipici" între toate submodulele dezvoltate. În implementarea dată, acesta nu conţine logică proprie, ci doar instanţiază şi conectează explicit:

- `spi_bridge` pentru transformarea fluxului serial în octeți paraleli şi generarea `byte_sync`;
- `instr_dcd` care decodează octeții recepţionaţi şi produce semnalele `read/write/addr`;
- `regs` care reţine configurările perifericului şi generează semnalele către restul sistemului;
- `counter` care produce valoarea `count_val` utilizată de PWM;
- `pwm_gen` care generează efectiv semnalul PWM către exterior.

Deoarece fișierul `top.v` nu poate fi modificat, implementarea a fost realizată astfel încât semnăturile tuturor modulelor să fie compatibile exact cu versiunea oficială. Această constrângere a influențat în mod direct designul semnalelor interne (ex. dimensiuni, direcţii, mod de generare a pulsurilor).

Prin conectarea strict structurală a acestor module, `top` permite testbench-ului oficial să exercite același flux de date așa cum ar funcționa perifericul într-un SoC real.

8.2 Procesul de testare şi validare

Testarea a fost realizată în două etape:

1. Testarea individuală a modulelor

Fiecare modul a fost verificat separat prin simulări:

- `spi_bridge`: verificarea detecţiei fronturilor, reconstruirea octeților şi transmiterea MSB-first pe MISO;
- `instr_dcd`: verificarea funcționării FSM-ului în două faze şi a generării corecte a pulsurilor `read/write`;
- `regs`: validarea actualizării selective a registrelor, a comportamentului write-only/read-only şi a generării impulsului `count_reset`;
- `counter`: verificarea prescalerului, overflow/underflow şi a direcţiei de numărare;
- `pwm_gen`: analizarea formelor de undă în modurile aliniat stânga, aliniat dreapta şi nealiniat.

Aceste teste au fost realizate în GTKWave, folosind testbench-uri specializate.

2. Testarea integrată prin testbench-ul oficial

După verificarea individuală, perifericul complet a fost testat folosind testbench-ul furnizat în enunț. Acesta trimite comenzi pe SPI, configurează registrele şi verifică automat că:

- instructiunile SPI sunt corect decodificate
- registrele se actualizează sau se citesc corespunzător
- numărătorul evoluează conform configurării
- semnalul PWM este generat exact conform specificației

Niciun test nu a raportat erori, confirmând că modulul este funcțional și compatibil 100% cu cerințele proiectului.

8.3 Concluzie finală

Perifericul realizat este stabil, complet sincron, bine modularizat și ușor de reutilizat. Organizarea codului, împreună cu testele ample efectuate, oferă un nivel ridicat de încredere în comportamentul său în orice scenariu valid. Acest proiect reproduce fidel funcționarea unui periferic real PWM, putând fi integrat fără modificări într-un SoC sau într-un mediu de simulare complex.