Universitatea Națională de Știință și Tehnologie Politehnica București

Facultatea de Automatică și Calculatoare

# Raport Tehnic de Implementare: Generator PWM Controlat prin Interfață SPI

**Echipa de implementare:**

*Vlăduțu Alexandru*

*Dragne Antonio*

*Vasilescu Alexandru*

**Grupa:** 333AA

14 decembrie 2025

# Cuprins

# 1 Introducere și Obiective

Prezentul document detaliază arhitectura, implementarea și verificarea unui sistem digital complex, proiectat pentru a genera semnale PWM (*Pulse Width Modulation*) configurabile. Sistemul este conceput ca un modul IP (*Intellectual Property*) care se interfațează cu un procesor gazdă prin magistrala serială SPI.

Obiectivul principal al proiectului a fost realizarea unui design robust, sincron, capabil să opereze în două domenii de ceas distincte și să gestioneze modificarea parametrilor de funcționare în timp real, fără a introduce instabilități în semnalul de ieșire.

# 2 Analiza Detaliată a Arhitecturii Hardware

Sistemul este divizat modular, respectând principiile de proiectare ierarhică. În continuare, este analizată funcționalitatea și implementarea fiecărui fișier sursă (`.v`).

## 2.1 Modulul Top-Level (`top.v`)

**Descriere Funcțională:** Acesta reprezintă nivelul superior al ierarhiei și are rolul exclusiv de a instanția submodulele și de a defini interconexiunile dintre acestea. Nu conține logică secvențială proprie, funcționând ca o schemă de legătură (*wrapper*).

**Modificări și Optimizări:** În versiunea finală, fișierul a suferit revizuiri critice pentru asigurarea integrității datelor:

1. **Interconectarea Magistralei Interne:** S-a corectat rutarea semnalelor `byte_sync`, `data_in` și `data_out` între *SPI Bridge* și *Instruction Decoder*. În versiunea anterioară, absența acestor legături izola logica de comunicație de cea de control.

2. **Maparea Pinilor SPI:** S-a rectificat asignarea porturilor `mosi` și `miso` pentru a corespunde direcției corecte a fluxului de date (Input vs Output).

3. **Optimizarea Resurselor:** Magistrala `functions` a fost redusă de la 8 biți la 2 biți, eliminând utilizarea inutilă a celulelor logice FPGA, întrucât sunt definite doar 3 moduri de operare.

## 2.2 Puntea de Comunicație (`spi_bridge.v`)

**Descriere Funcțională:** Acest modul realizează deserializarea datelor primite de la Master (MOSI) și serializarea datelor transmise de Slave (MISO).

**Provocarea Tehnică - Traversarea Domeniilor de Ceas (CDC):** Sistemul operează cu două ceasuri asincrone: `SCLK` (ceasul magistralei seriale) și `CLK` (ceasul sistemului). Utilizarea directă a semnalelor dintr-un domeniu în celălalt ar duce la metastabilitate.

**Soluția Implementată:** S-a introdus un sincronizator cu trei etaje de bistabile. Semnalul care indică recepția unui octet (`spi_done`) este trecut prin lanțul de registre `spi_done_r1` → `r2` → `r3`. Detectarea frontului (edge detection) se face comparând `r2` cu `r3` în domeniul de ceas local, garantând un semnal de control stabil (`byte_sync`).

## 2.3 Decodorul de Instrucțiuni (`instr_dcd.v`)

**Descriere Funcțională:** Acest modul implementează protocolul de comunicație proprietar. Funcționează ca o mașină de stări (FSM) care interpretează fluxul de octeți primit de la *SPI Bridge*.

**Protocolul Implementat:**

- **Ciclul 1 (Control):** Primul octet conține bitul de R/W și adresa registrului țintă (6 biți).
- **Ciclul 2 (Date):** Al doilea octet conține datele efective (pentru scriere) sau este ignorat (în cazul citirii, când perifericul trimite datele înapoi).

Decodorul gestionează semnalele de control `read`, `write` și selectează adresa corectă pe magistrală.

## 2.4 Bancul de Regiștri (`regs.v`)

**Descriere Funcțională:** Reprezintă memoria de configurare a perifericului. Implementează conceptul de *Memory Mapped I/O*, unde parametrii funcționali sunt accesibili la adrese specifice.

**Elemente de Design Notabile:**

- **Bit de Auto-Reset (Self-Clearing):** Registrul de la adresa `0x07` (`COUNTER_RESET`) are un comportament particular. La scrierea valorii '1', acesta generează un impuls pentru resetarea numărătorului, iar în următorul ciclu de ceas revine automat la starea '0'. Aceasta simplifică secvența de inițializare din software.
- **Acces Read/Write:** Modulul multiplexează datele de ieșire în funcție de adresa selectată, permițând procesorului să verifice configurația curentă (read-back).

## 2.5 Unitatea de Numărare (`counter.v`)

**Descriere Funcțională:** Generează baza de timp pentru semnalul PWM. Este un numărător binar care incrementează la fiecare front de ceas (sau divizat, în funcție de prescaler).

**Stabilitatea Sistemului - Shadow Registers:** O problemă critică în sistemele PWM este modificarea perioadei în timpul funcționării. Dacă noua perioadă este scrisă în timp ce numărătorul are o valoare ridicată, se poate produce o depășire (overflow)

nedorită. Soluția implementată constă în utilizarea "registrelor umbră". Valorile scrise de utilizator sunt stocate într-un buffer și transferate în registrele active (`active_period`) doar în momentul în care numărătorul ajunge la zero (sfârșitul ciclului curent).

### 2.6 Generatorul PWM (`pwm_gen.v`)

**Descriere Funcțională:** Acesta este un circuit combinațional care compară valoarea curentă a numărătorului cu pragurile setate (`COMPARE1`, `COMPARE2`).

**Moduri de Operare:** Pe baza intrării `functions` (2 biți), modulul selectează logica de ieșire:

- **Left Aligned:** Ieșirea este '1' de la start până la atingerea pragului `COMPARE1`.
- **Right Aligned:** Ieșirea devine '1' după depășirea pragului `COMPARE1` și rămâne activă până la final.
- **Window Mode:** Ieșirea este activă strict în intervalul definit între `COMPARE1` și `COMPARE2`.

## 3 Strategia de Verificare (`testbench.v`)

Fișierul `testbench.v` a fost complet rescris pentru a trece de la o simplă generare de semnale la o verificare automată a corectitudinii sistemului.

### 3.1 Automatizarea Validării

A fost introdusă procedura (task-ul) `check_pwm_duty`. Aceasta:

1. Măsoară numărul de cicli de ceas în care ieșirea `pwm_out` este activă pe durata mai multor perioade.
2. Compară valoarea măsurată cu valoarea teoretică așteptată.
3. Afișează mesaje de tip `[PASS]` sau `[FAIL]`, eliminând necesitatea inspectării vizuale manuale a formelor de undă.

### 3.2 Scenarii de Test Acoperite

Noul testbench validează scenarii complexe:

- **Configurare Secvențială:** Scriere registre → Reset Counter → Verificare Output.
- **Moduri de Lucru:** Testarea tuturor celor 3 moduri de aliniere.

- **Colțuri (Corner Cases):** Comportamentul când `COMPARE1` este egal cu `COMPARE2` (ieșirea trebuie să fie stabilă) și pornirea cu pragul zero.
- **Verificare Bidirecțională:** Testarea operației de citire SPI pentru confirmarea valorilor interne.

# 4 Concluzii

Proiectul a demonstrat implementarea cu succes a unui periferic complex, punând accent pe corectitudinea transferului de date și stabilitatea semnalelor. Modificările arhitecturale aduse fișierelor `top.v` și `spi_bridge.v` au fost esențiale pentru funcționalitatea sistemului, rezolvând problemele de conectivitate și sincronizare. De asemenea, utilizarea registrelor tampon (*shadow registers*) asigură o funcționare robustă în aplicații de control în timp real.

Testarea automată implementată în testbench confirmă respectarea specificațiilor pentru toate modurile de operare definite.

# Bibliografie


[1] M. Morris Mano, Michael D. Ciletti, *Digital Design: With an Introduction to the Verilog HDL.* O carte fundamentală pentru noi, care explică bazele circuitelor digitale (porți logice, bistabile, numărătoare) și cum se descriu acestea în Verilog. Ne-a ajutat să înțelegem structura internă a numărătorului și a registrelor.

[2] Pong P. Chu, *FPGA Prototyping by Verilog Examples.* Această carte a fost sursa noastră principală de inspirație pentru exemple practice. Conține șabloane de cod pentru module standard (precum interfețe seriale și generatoare de semnal) pe care le-am adaptat pentru proiectul nostru.

[3] David Harris, Sarah Harris, *Digital Design and Computer Architecture.* O referință excelentă pentru a înțelege cum interacționează un procesor cu perifericele sale (conceptul de Memory Mapped I/O), esențial pentru partea de registre a proiectului nostru.

[4] Nandland, *SPI Interface - Verilog VHDL.* [Online]. Disponibil la: `https://nandland.com/spi-serial-peripheral-interface/` Un tutorial pas-cu-pas care ne-a clarificat modul în care funcționează cele 4 moduri SPI și cum să scriem mașina de stări pentru a recepționa datele bit cu bit.

[5] SparkFun Electronics, *Pulse Width Modulation (PWM) Basics.* [Online]. Disponibil la: `https://learn.sparkfun.com/tutorials/pulse-width-modulation` Explicații

vizuale foarte bune despre factorul de umplere (duty cycle) și despre cum influențează frecvența semnalului comportamentul dispozitivelor conectate (ex: LED-uri sau motoare).

[6] HDLBits, *Verilog Practice*. [Online]. Disponibil la: `https://hdlbits.01xz.net/` O platformă cu exerciții interactive pe care am folosit-o la începutul proiectului pentru a ne reaminti sintaxa Verilog și modul de funcționare al blocurilor `always` și al asignărilor non-blocante.

[7] Analog Devices, *SPI Implementation Guide*. [Online]. Disponibil la: `https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html` Documentație tehnică detaliată care explică avantajele și dezavantajele SPI față de alte protocoale (precum I2C), utilă pentru justificarea alegerii acestui protocol în documentație.

[8] Ben Eater (YouTube Channel), *Digital Logic  Computer Design*. Deși lucrează pe breadboard, videoclipurile lui ne-au ajutat enorm să vizualizăm cum funcționează semnalele de ceas și cum se propagă datele într-un sistem digital înainte de a scrie codul.

[9] Digi-Key Electronics (YouTube), *Introduction to FPGA with Shawn Hymel*. O serie video modernă care explică conceptele de bază ale FPGA-urilor și scrierea primului modul Verilog, oferind o perspectivă practică asupra toolchain-ului de simulare.