

UNIVERSITATEA NAȚIONALĂ DE ȘTIINȚĂ ȘI
TEHNOLOGIE POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL DE CALCULATOARE

Proiect Arhitectura Calculatoarelor

Documentație Tehnică: Generator PWM Controlat prin SPI

Echipa de proiect:

Vladutu Alexandru

Dragne Antonio

Vasilescu Alexandru

Grupa: 333AA

23 noiembrie 2025

Cuprins

1	Introducere	2
2	Descrierea Funcționării Modulelor	2
2.1	Bridge-ul SPI: Sincronizarea între două lumi	2
2.2	Decodorul de Instrucțiuni: "Creierul" simplificat	3
2.3	Registrii: Memoria de configurare	3
2.4	Numărătorul: Baza de timp și "Shadow Registers"	3
2.5	Generatorul PWM: Logica de ieșire	4
3	Concluzii	4

1 Introducere

Acest proiect prezintă implementarea unui modul hardware care generează semnale PWM (*Pulse Width Modulation*), controlat de un procesor extern printr-o interfață serială SPI.

Din punct de vedere al **Arhitecturii Calculatoarelor**, proiectul ilustrează relația clasică dintre un *Master* (procesorul) și un *Slave* (perifericul). Procesorul trimite comenzi pentru a configura parametrii perifericului (frecvență, factor de umplere), iar perifericul execută aceste comenzi independent, eliberând procesorul de sarcina de a comuta manual pinii de ieșire.

2 Descrierea Funcționării Modulelor

Arhitectura este împărțită în blocuri logice, fiecare având o responsabilitate clară. Mai jos explicăm teoria din spatele fiecărui modul și modul în care am rezolvat problemele specifice hardware-ului.

2.1 Bridge-ul SPI: Sincronizarea între două lumi

Teorie: Acest modul este "traducătorul" sistemului. El primește datele bit cu bit (serial) de la Master și le transformă în pachete de 8 biți (paralel) pe care le poate înțelege restul sistemului.

Provocarea tehnică: Cea mai mare problemă la periferice este că ceasul de comunicație (SCLK) și ceasul intern al sistemului (CLK) nu sunt sincronizate. Dacă am citi datele direct, am risca să citim valori instabile (metastabilitate), ceea ce ar duce la erori aleatoare.

Soluția noastră: Am implementat un mecanism de sincronizare cu **trei bistabile** (flip-flops). Semnalul care ne anunță că a venit un octet nou este trecut prin aceste bistabile pentru a-l "muta" în siguranță pe ceasul intern.

```
1 // Trecem semnalul prin 3 registre pentru a elimina
   metastabilitatea
2 always @(posedge clk) begin
3     spi_done_r1 <= spi_done_toggle;
4     spi_done_r2 <= spi_done_r1;
5     spi_done_r3 <= spi_done_r2;
6
7     // Detectam schimbarea doar cand semnalul este stabil
8     if (spi_done_r2 != spi_done_r3)
9         byte_sync <= 1'b1; // Acum putem citi datele sigur
10 end
```

Listing 1: Sincronizarea semnalului între domeniile de ceas

2.2 Decodorul de Instrucțiuni: "Creierul" simplificat

Teorie: Odată ce datele ajung în paralel, sistemul trebuie să știe ce să facă cu ele. Decodorul funcționează ca o Mașină cu Stări Finite (FSM) simplă.

Protocolul nostru are două etape: 1. **Faza de Setup:** Primul octet ne spune *ce facem* (citim sau scriem) și *unde* (la ce adresă). 2. **Faza de Date:** Al doilea octet conține informația propriu-zisă.

Practic, acest modul dirijează traficul: dacă e o scriere, trimite datele către regiștri; dacă e o citire, cere datele din regiștri și le trimite înapoi la SPI.

2.3 Regiștrii: Memoria de configurare

Teorie: În arhitectura calculatoarelor, perifericele sunt controlate prin ****Memory Mapped I/O****. Asta înseamnă că setările (perioada, activarea) sunt văzute ca niște adrese de memorie.

Detaliu de implementare (Bitul Auto-Reset): Un aspect interesant este registrul COUNTER_RESET. Acesta nu memorează valoarea '1'. Când scriem '1', el dă un impuls de reset număratorului, iar în ciclul următor se șterge singur. Aceasta este o tehnică standard hardware pentru a crea semnale de comandă de tip "puls" din software.

```
1 // Default este mereu 0
2 count_reset <= 1'b0;
3
4 // Daca procesorul scrie 1, devine 1 doar pentru acest ciclu de
   ceas
5 if(write && addr == 6'h07)
6     count_reset <= 1'b1;
```

Listing 2: Logica bitului care se șterge singur (Self-Clearing)

2.4 Numărătorul: Baza de timp și "Shadow Registers"

Teorie: Numărătorul este inima sistemului PWM. El incrementează o valoare la fiecare impuls de ceas (sau divizat prin prescaler).

Provocarea tehnică: Ce se întâmplă dacă schimbăm perioada (limita de numărare) exact când numărătorul este la jumătate? *Exemplu:* Numărăm până la 100. Suntem la 80. Software-ul schimbă limita la 50. Numărătorul (fiind la 80) va continua să numere până la maximum posibil (65535) înainte să o ia de la capăt. Acest lucru creează un "glitch" (eroare) vizibil pe ieșire.

Soluția noastră: Am folosit ****Shadow Registers**** (registre umbră). Valoarea nouă scrisă de utilizator este ținută într-o zonă de așteptare. Ea devine activă doar când numărătorul termină ciclul curent (ajunge la 0). Astfel, tranziția este mereu curată.

```

1 // Update_event apare doar cand numaratoul a terminat ciclul
2 if (update_event) begin
3     // Abia acum mutam datele din zona de asteptare in cea activa
4     active_period    <= period;
5     active_prescale  <= prescale;
6 end

```

Listing 3: Actualizarea parametrilor doar la final de ciclu

2.5 Generatorul PWM: Logica de ieşire

Teorie: Acest modul este cel mai simplu: este un comparator. El verifică continuu dacă valoarea numărătorului este mai mică sau mai mare decât pragul setat (**COMPARE**).

Am implementat o logică flexibilă care permite nu doar PWM standard (activ la începutul perioadei), ci şi PWM "nealiniat" (activ între două valori arbitrare, A şi B). Acest lucru se realizează printr-un simplu lanţ de decizii **if-else** în hardware.

3 Concluzii

Prin realizarea acestui proiect, echipa a aprofundat conceptele fundamentale de **Arhitectura Calculatoarelor**:

1. **Sincronizarea:** Cum transferăm date între domenii de ceas diferite fără erori.
2. **Registre de control:** Cum un software poate controla hardware-ul scriind la adrese specifice.
3. **Stabilitatea:** Importanţa registrelor "shadow" pentru a preveni comportamente nedorite în timp real.

Implementarea în Verilog ne-a obligat să gândim în paralel (cum funcţionează hardware-ul), spre deosebire de programarea secvenţială clasică.

Bibliografie

- [1] Nandland, *FPGA Verilog Tutorials - SPI Interface*. Explicaţii clare despre cum funcţionează SPI la nivel de bit.
Disponibil la: <https://nandland.com/spi-serial-peripheral-interface/>
- [2] ChipVerify, *Verilog Tutorial*. Sursa principală pentru sintaxa Verilog şi exemple de maşini cu stări.
Disponibil la: <https://www.chipverify.com/verilog/verilog-tutorial>

- [3] SparkFun Electronics, *Pulse Width Modulation (PWM) Basics*. Teoria generală despre semnalele PWM.
Disponibil la: <https://learn.sparkfun.com/tutorials/pulse-width-modulation/all>
- [4] FPGA4Student, *Verilog Code for PWM Generator*. Exemple de implementare a numărătoarelor.
Disponibil la: <https://www.fpga4student.com/>
- [5] Microchip Technology, *ATmega328P Datasheet*. Documentația de referință pentru cum funcționează timerele în procesoare reale.
Disponibil la: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf