Controlul unui Servomotoras utilizand PS si PL

Romitan Flaviu

March 10, 2025

Contents

Rezumat

Acest raport descrie implementarea unui sistem integrat pentru controlul unui servomotoras utilizand resursele disponibile pe platforma Zybo. Proiectul exploreaza doua moduri principale: controlul directiei de rotatie printr-un potentiometru si controlul incremental al pasului de rotire folosind un numarator sus-jos activat de butoane. Solutia combina utilizarea componentelor hardware si software, incluzand XADC pentru citirea semnalelor analogice, generarea PWM in logica PL si procesarea datelor in PS.

1 Introducere

Controlul precis al servomotoarelor este fundamental in numeroase aplicatii, precum robotica si automatizarile industriale. Acest proiect utilizeaza arhitectura platformei Zybo, care imbina resursele PS si PL pentru a realiza un sistem eficient si flexibil.

Proiectul are urmatoarele obiective principale:

- Implementarea unui control bidirectional al motorasului utilizand XADC si potentiometrul;
- Realizarea unui sistem incremental pentru pasii motorasului folosind un numarator sus-jos actionat de doua butoane;
- Validarea functionarii sistemului prin masuratori si analize.

Aceasta documentatie prezinta fundamentele teoretice, implementarea practica si rezultatele experimentale care valideaza functionalitatea sistemului.

2 Fundamentare teoretica

2.1 Servomotorasul si PWM

Controlul unui servomotoras se bazeaza pe semnalele PWM care determina pozitia acestuia. Durata impulsurilor PWM este direct proportionala cu unghiul dorit. In cazul nostru, durata variaza intre 0.4 ms si 2.2 ms pentru un interval complet de 180 de grade. Am folosit formulele gasite pe acest site pentru a calcula frecventa necesara pentru divizorul de frecventa, dupa care am implementat asta in cod.

2.2 XADC

Convertorul Analog-Digital Extins (XADC) permite citirea semnalelor analogice generate de potentiometru. Aceste semnale sunt utilizate pentru a controla directia de rotatie a motorasului in modul automat. Am folosit PMOD-ul XADC de pe placa ca sa conectam potentiometrul, si am asigurat conexiunile corecte folosind schema din manualul de referinta al placii.

2.3 Numaratorul sus-jos

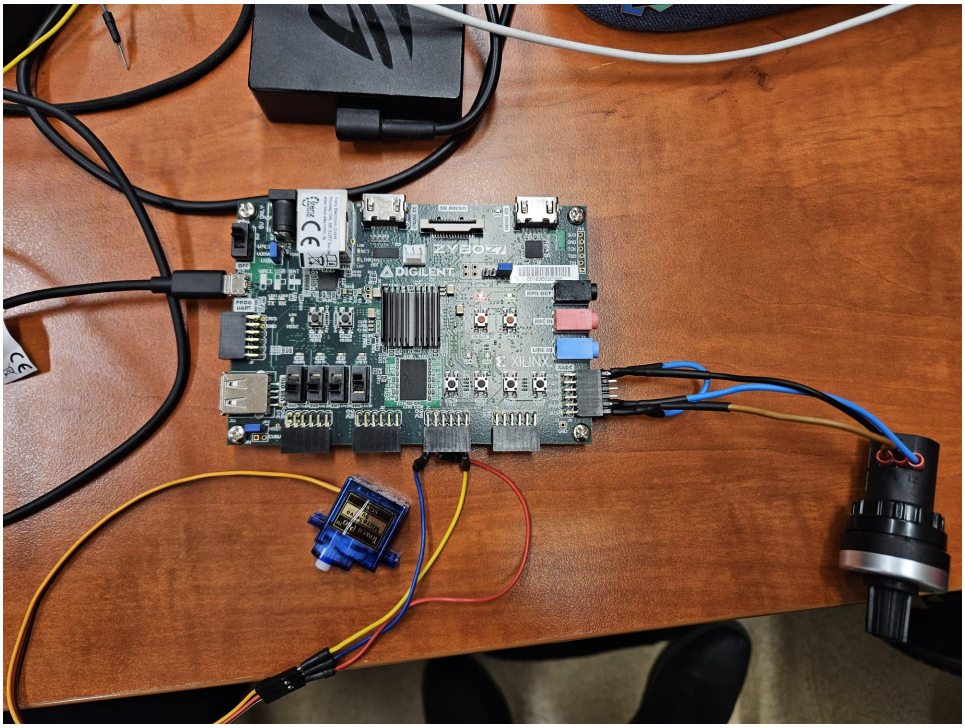
Numaratorul sus-jos este un circuit digital utilizat pentru incrementarea si decrementarea valorii unghiului. Acesta este controlat prin doua butoane, permitand utilizatorului sa ajusteze manual pozitia motorasului.

3 Proiectare si implementare

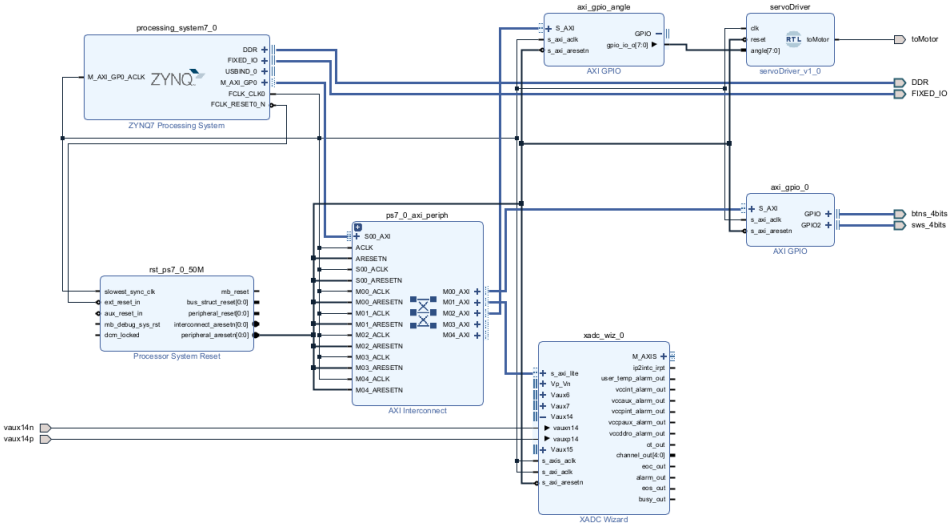
3.1 Metoda experimentala

Sistemul este compus din urmatoarele elemente principale:

- **Hardware**:
  - Zybo Board pentru integrarea PS si PL;
  - Potentiometru conectat la intrarea XADC, PMOD JA;
  - Servomotoras controlat conectat la pinul 1 de la PMOD JC.



- **Software**:
  - Programarea PS in C pentru gestionarea datelor;
  - Design logic in Vivado pentru generarea semnalului PWM in PL.



Avem, in plus fata de modulele necesare pentru procesorul ZYNQ, entitatea noastra top-level numita servoDriver, care preia unghiul din procesor, transmis printr-un AXI GPIO, il proceseaza si il transmite mai departe la entitatea care genereaza semnalul PWM. In plus, avem o entitate care ia semnalul de clock de la placa si il transforma intr-un clock valid pentru generarea semnalului PWM. Semnalul este apoi transmis la motor prin pinul PMOD pus in fisierul de constrangeri.

3.2 Controlul directiei de rotire

In modul automat, adica cand switchul este pe 1, potentiometrul controleaza unghiul motorasului. Semnalul analogic preluat prin XADC este convertit digital si mapat la intervalul de unghiuri permis. Implementarea utilizeaza formula:

$$angle = \frac{potentiometer\ value \times 180}{4095}$$

unde valoarea potentiometrului variaza intre 0 si 4095.

3.3 Controlul incremental al pasului de rotire

In modul manual, adica cand switchul este pe 0, cele doua butoane permit ajustarea incrementala a unghiului in pasi de 5 grade. Unghiul este actualizat conform urmatorului algoritm:

- BTN0 decrementeaza unghiul daca acesta este mai mare decat 0;
- BTN1 incrementeaza unghiul daca acesta este mai mic decat 180.

Valorile actualizate sunt transmise catre PL pentru a genera semnalul PWM corespunzator.

4 Rezultate experimentale

4.1 Observatii generale

Desi codul a fost implementat si s-a rulat cu succes in mediile de dezvoltare Vitis si Vivado, rezultatele experimentale nu au confirmat functionalitatea dorita pe placa Zybo. Sistemul s-a initializat corect, insa nu s-au observat actiuni pe hardware-ul conectat.

4.2 Probleme intampinate

- Mesajele de diagnosticare trimise catre consola prin `xil_printf` nu au fost afisate in consola `hterm`, desi codul includea instructiunile corespunzatoare.
- Nu s-a detectat nicio schimbare in pozitia servomotorasului, indiferent de valorile introduse prin potentiometru sau butoane.
- Semnalele generate in logica PL nu au fost confirmate ca fiind transmise corect catre componentele externe.

4.3 Concluzie asupra testelor

Aceste rezultate sugereaza o problema in comunicarea dintre PS si PL sau in configurarea hardware a sistemului. Debugging-ul suplimentar este necesar pentru a verifica atat conexiunile hardware, cat si configuratiile software. De asemenea, trebuie analizate potentiale erori in configurarea interfetelor GPIO si XADC.

5 Concluzii si dezvoltari viitoare

Proiectul a realizat integrarea eficienta a componentelor PS-PL pentru controlul precis al unui servomotoras. Totusi, rezultatele obtinute pe hardware nu au fost cele asteptate. Dezvoltari viitoare pot include:

- Investigarea configuratiilor hardware si software pentru a elimina problemele de comunicare;

- Adaugarea unui sistem de feedback mai robust pentru diagnosticare;
- Extinderea interfetei pentru utilizatori prin conectivitate wireless.

Bibliografie

1. Zybo Z7 Reference Manual
2. Controlarea unui servo motor cu semnal PWM pe platforma codeproject

Anexe

Anexa A: Codul sursa

main.c

```
#include <stdint.h>
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xil_types.h"
#include "xparameters.h"
#include "xadcps.h"
#include "xgpio.h"
#include <unistd.h>

#define ANGLE_MIN 0
#define ANGLE_MAX 180
#define ANGLE_STEP 5

#define GPIO_ANGLE_BASEADDR
    XPAR_AXI_GPIO_ANGLE_BASEADDR
#define GPIO_SW_BTN_BASEADDR      XPAR_AXI_GPIO_0_BASEADDR
#define XADC_DEVICE_ID            XPAR_XADC_WIZ_0_DEVICE_ID

#define XADCPS_SEQ_CH_VAUX14 (1 << 30)

static XAdcPs xadc;
XGpio gpio_angle, gpio_sw_btn;
int current_angle = 90;

int main(void) {
    init_platform();

```

```
    XAdcPs_Config *xadc_cfg = XAdcPs_LookupConfig(0);
    if (xadc_cfg == NULL) {
        xil_printf("Error: XADC config failed\r\n");
        return XST_FAILURE;
    }

    if (XAdcPs_CfgInitialize(&xadc, xadc_cfg, xadc_cfg->
       BaseAddress) != XST_SUCCESS) {
        xil_printf("Error: XADC initialization failed\r\n");
        return XST_FAILURE;
    }

    XAdcPs_SetSequencerMode(&xadc, XADCPS_SEQ_MODE_CONTINPASS
       );
    XAdcPs_SetSeqChEnables(&xadc, XADCPS_SEQ_CH_VAUX14);

    XGpio_Initialize(&gpio_angle, 1);
    XGpio_Initialize(&gpio_sw_btn, 0);

    XGpio_SetDataDirection(&gpio_angle, 1, 0x00000000);
    XGpio_SetDataDirection(&gpio_sw_btn, 1, 0xFFFFFFFF);
    XGpio_SetDataDirection(&gpio_sw_btn, 2, 0xFFFFFFFF);

    xil_printf("Initialization complete. Starting main loop
       ...\r\n");

    while (1) {
        int switch_state = (XGpio_DiscreteRead(&gpio_sw_btn,
           1) & 0x1);

        if (switch_state) {
            u32 raw_value = XAdcPs_GetAdcData(&xadc, 30);
            u32 pot_value = raw_value >> 4;
            current_angle = (pot_value * 180) / 4095;
            xil_printf("Potentiometer mode - Raw: %lu, Angle:
                %d\r\n", pot_value, current_angle);
        } else {
            int buttons = (XGpio_DiscreteRead(&gpio_sw_btn,
               2) & 0x3);

            if (buttons & 0x1) {
                if (current_angle > ANGLE_MIN) {
                    current_angle -= ANGLE_STEP;
                    xil_printf("Button Left - Angle: %d\r\n",
                        current_angle);
                }
            }
            if (buttons & 0x2) {
                if (current_angle < ANGLE_MAX) {
```

```
                    current_angle += ANGLE_STEP;
                    xil_printf("Button Right - Angle: %d\r\n"
                        , current_angle);
                }
            }
        }

        if (current_angle < ANGLE_MIN) current_angle =
            ANGLE_MIN;
        if (current_angle > ANGLE_MAX) current_angle =
            ANGLE_MAX;

        XGpio_DiscreteWrite(&gpio_angle, 1, (u8)current_angle
            );

        usleep(10000);
    }

    cleanup_platform();
    return 0;
}
```

servoClock.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity servoClock is
    Port(clk : in std_logic;
         rst : in std_logic;
         clk_out : out std_logic);
end servoClock;

architecture Behavioral of servoClock is

signal aux_clk : std_logic := '0';
signal counter : integer := 0;

begin

    process(clk,rst)
    begin
        if rst = '1' then
            aux_clk <= '0';
            counter <= 0;
        elsif rising_edge(clk) then
            if counter = 1000000 then
```

```
                aux_clk <= not(aux_clk);
                counter <= 0;
            else
                counter <= counter + 1;
            end if;
        end if;
    clk_out <= aux_clk;
    end process;

end Behavioral;
```

servoSignal.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity servoSignal is
    Port(clk : in  STD_LOGIC;
         reset : in  STD_LOGIC;
         angle : in  integer range 0 to 180;
         pwm : out STD_LOGIC);
end servoSignal;

architecture Behavioral of servoSignal is

signal pulse_width : integer := 0;
signal counter : integer range 0 to 130000 := 0;

begin

    process (clk, reset)
    begin
        if reset = '1' then
            counter <= 0;
            pwm <= '0';
        elsif rising_edge(clk) then
            pulse_width <= 20000 + ( angle / 180 * 110000 );
            if counter < pulse_width then
                pwm <= '1';
            else
                pwm <= '0';
            end if;
            if counter >= 130000 then
                counter <= 0;
            else
                counter <= counter + 1;
```

```
            end if;
        end if;
    end process;

end Behavioral;
```

servoDriver.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity servoDriver is
  Port(clk : in std_logic;
       reset: in std_logic;
       angle: in std_logic_vector(7 downto 0);
       toMotor: out std_logic);
end servoDriver;

architecture Behavioral of servoDriver is

signal div_clk : std_logic;
signal angle_int: integer;
signal pwm: std_logic;

begin

clkDiv: entity work.servoClock port map(clk, reset, div_clk);
angle_int <= to_integer(unsigned (angle));
pwmGen: entity work.servoSignal port map(div_clk, reset,
   angle_int, pwm);

toMotor <= pwm;

end Behavioral;
```