Retardos, Macros y Subrutinas

Autor: Luis David Barahona Valdivieso

Fecha: 25/04/2025

índice

1. Macros

1.1. Planteamiento del problema e identificación de requerimientos

Crear una macro capaz de permitir el desplazamiento aritmético a la derecha. Recordemos que este operador permite realizar divisiones con números que son potencias de 2.

Por ejemplo: Dividir 1024 / 8 es igual a 1024>>7, pues 2^7=128. En ambos casos el resultado debe generar 8.

1.2. Diagrama de bloques y de flujo

1.3. Código en Ensamblador

https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/ASR_Arithmetic_Shift_Right/ASR_Code_Assembler/AssemblerApplication1/main.asm

1.4. Código en C

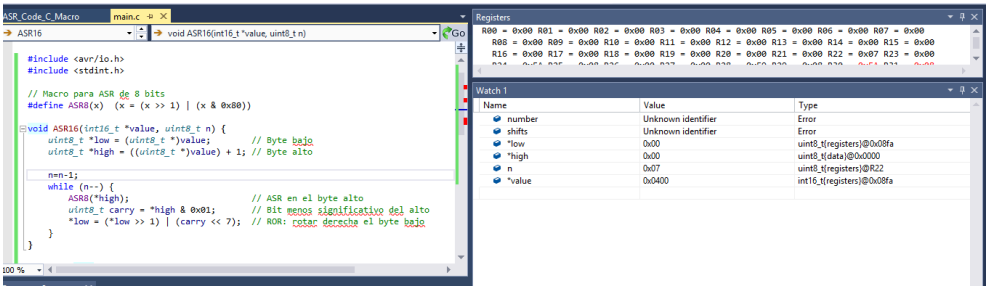
C Con macro:
https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/ASR_Arithmetic_Shift_Right/ASR_Code_C_Macro/ASR_Code_C_Macro/main.c

C con rutina:
https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/ASR_Arithmetic_Shift_Right/ASR_Code_C_Rutina/ASR_C_Code/main.c

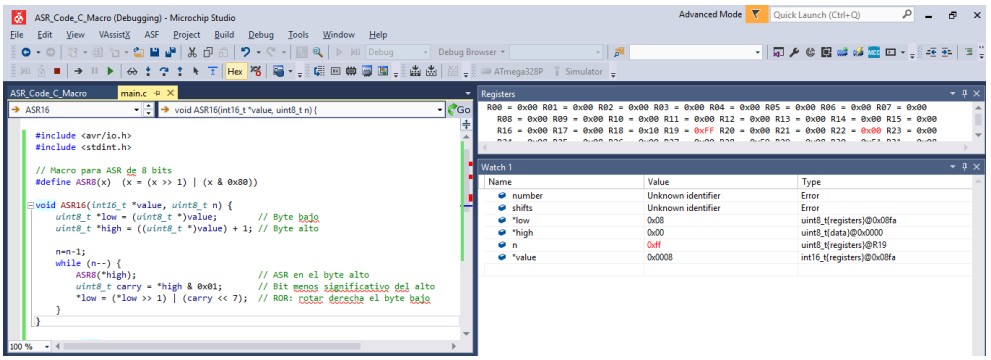
1.5. Simulación de los casos en Microchip y Proteus

SIMULACIÓN USANDO MACRO

CASO 1: Mientras n!=0 aún no hemos realizado los 7 desplazamientos necesarios

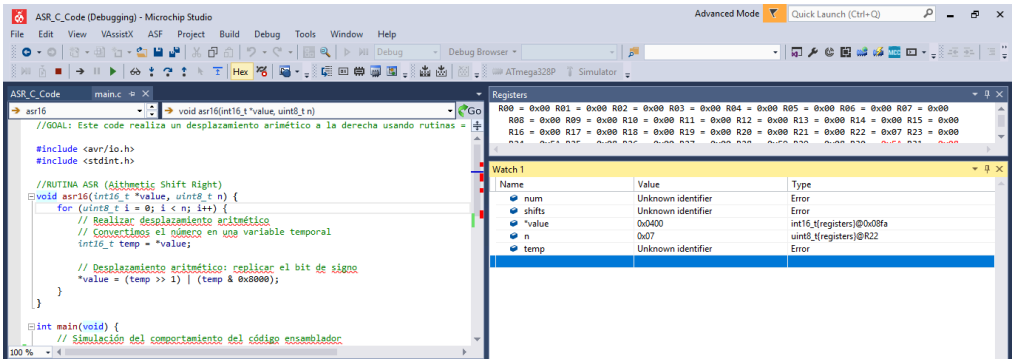


CASO 2: Cuando se n=0 (quiere decir que han pasado 7 desplazamientos), low debería ser 0x08, pues la división entre 1024 y 128 es 8.

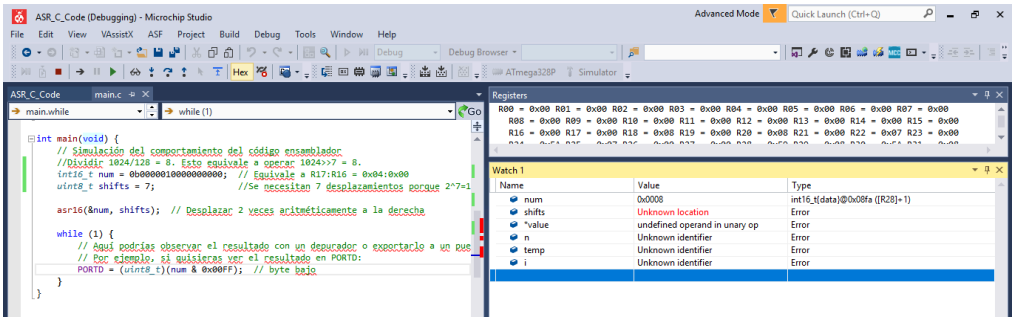


SIMULACIÓN USANDO RUTINA

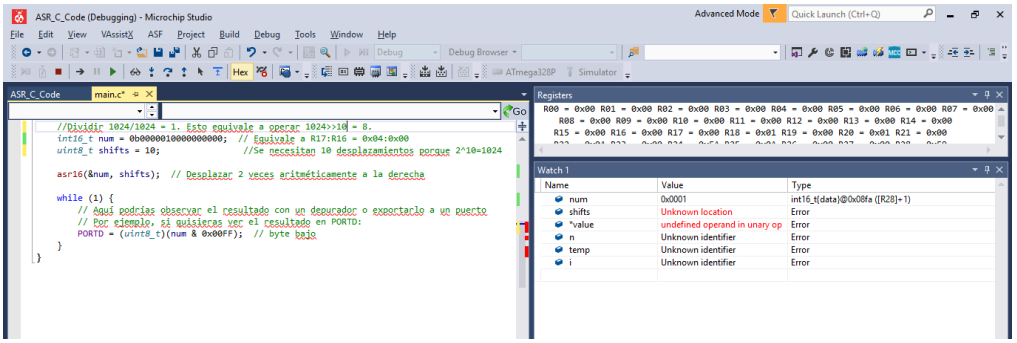
CASO 1: Dentro de la rutina, realizando los deplazamientos



CASO 2: Dividir 1024/128=1024>>7=8



CASO 3: Dividir 1024/1024=1024>>10=1



1.6. Implementación

1.7. Documentación en GITHUB

https://github.com/luisbarahona100/Mentorias/tree/main/Mentor%C3%ADa%204/ASR_Arithmetic_Shift_Right

2. Subrutinas

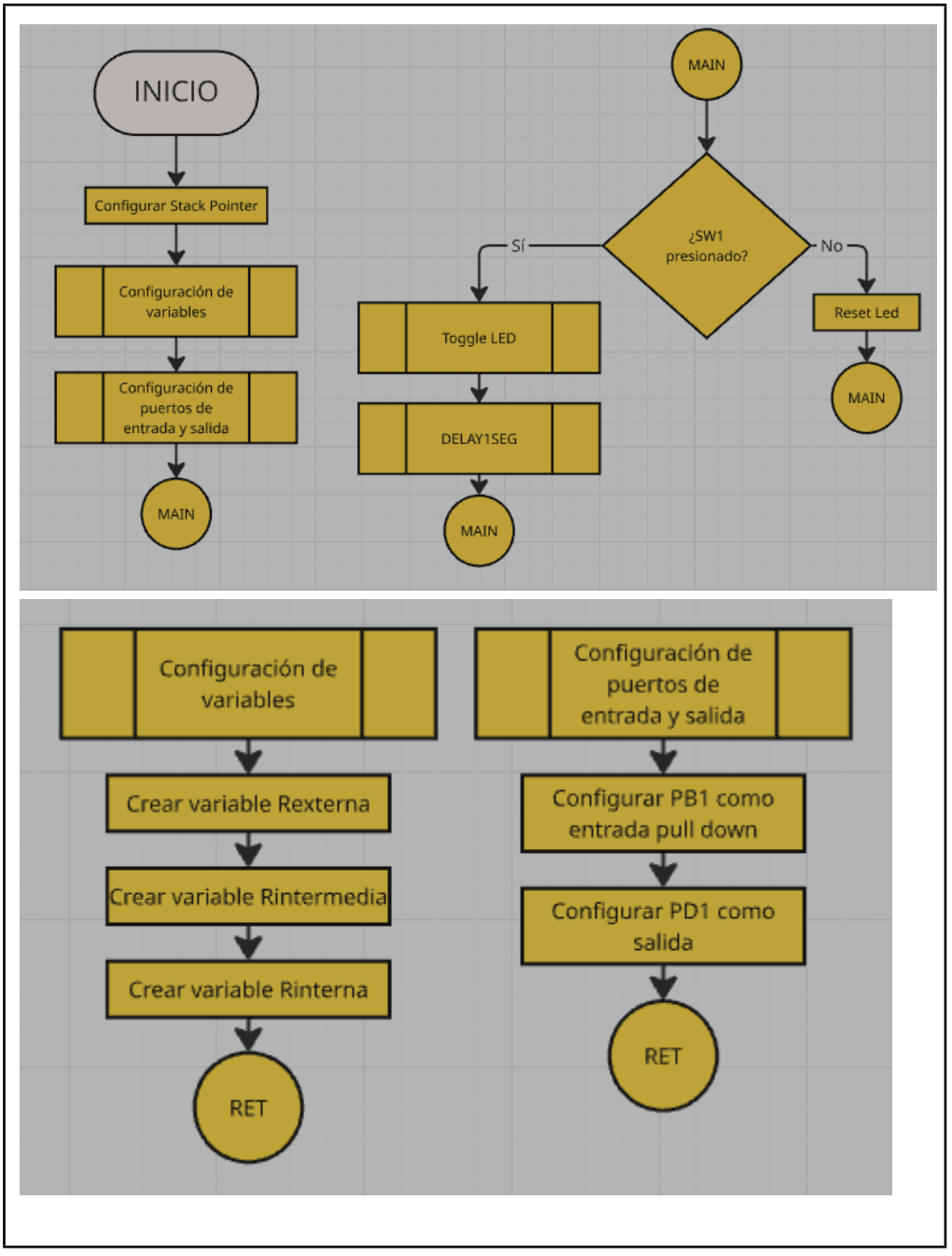
2.1. Planteamiento del problema

Prender y apagar un led cada segundo

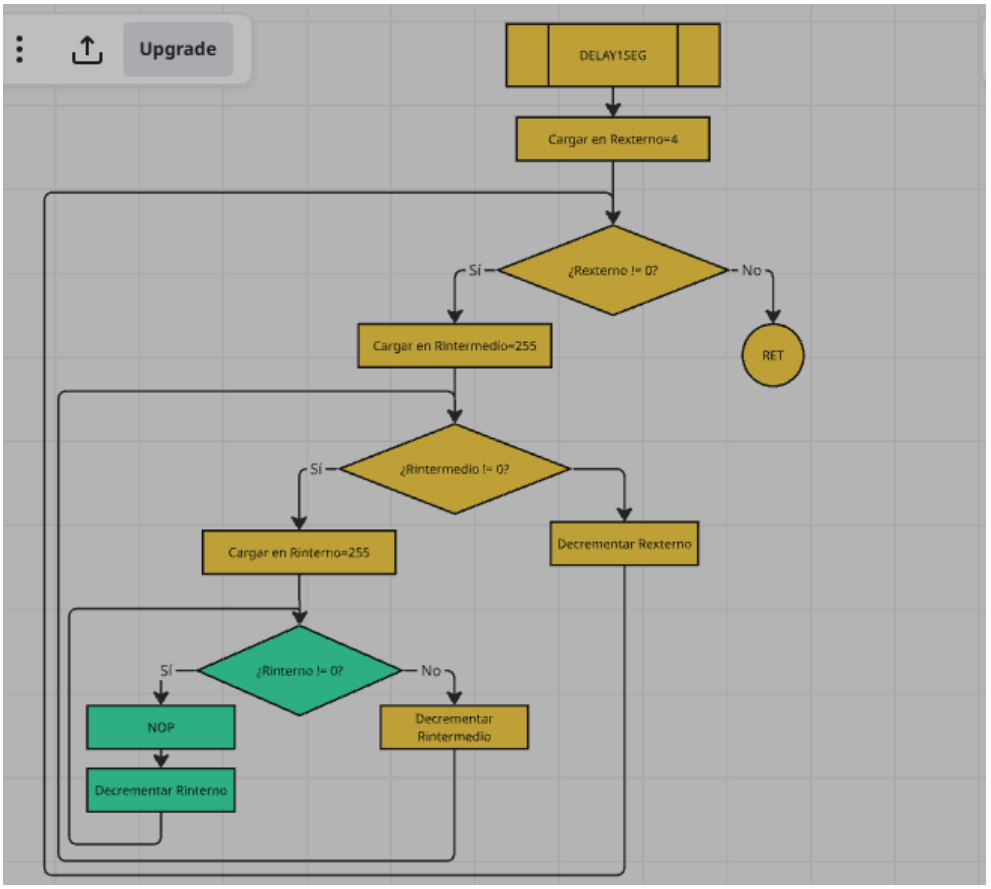
2.2. Identificación de requerimientos

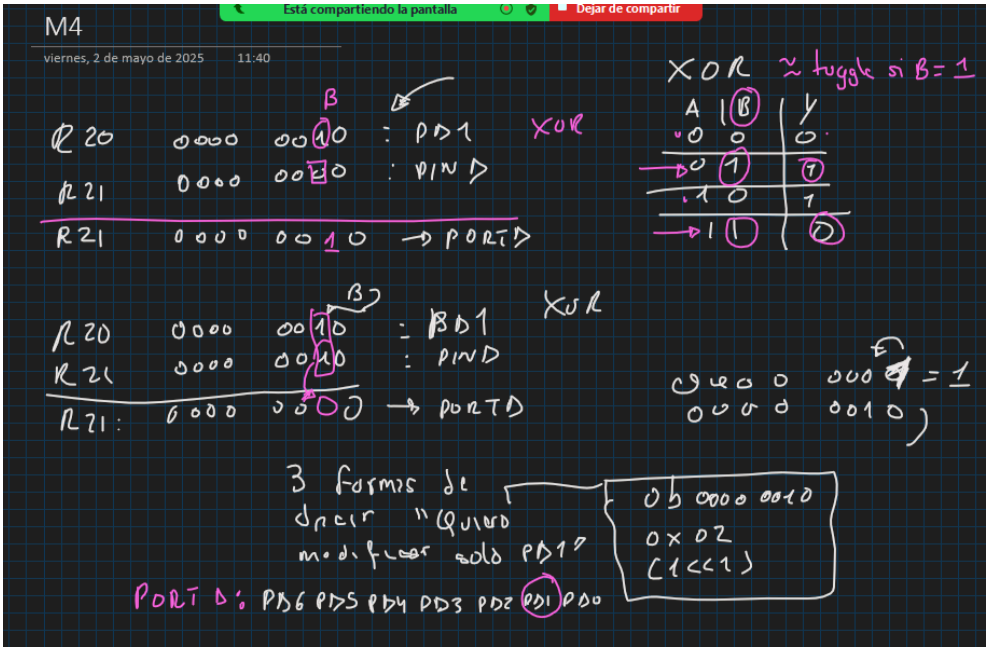
- Entrada : Switch en pull down
- Salida: Led

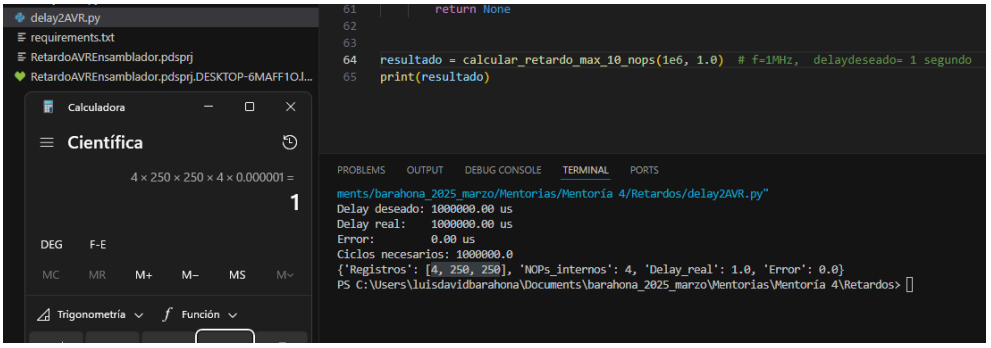
2.3. Diagrama de bloques y de flujo

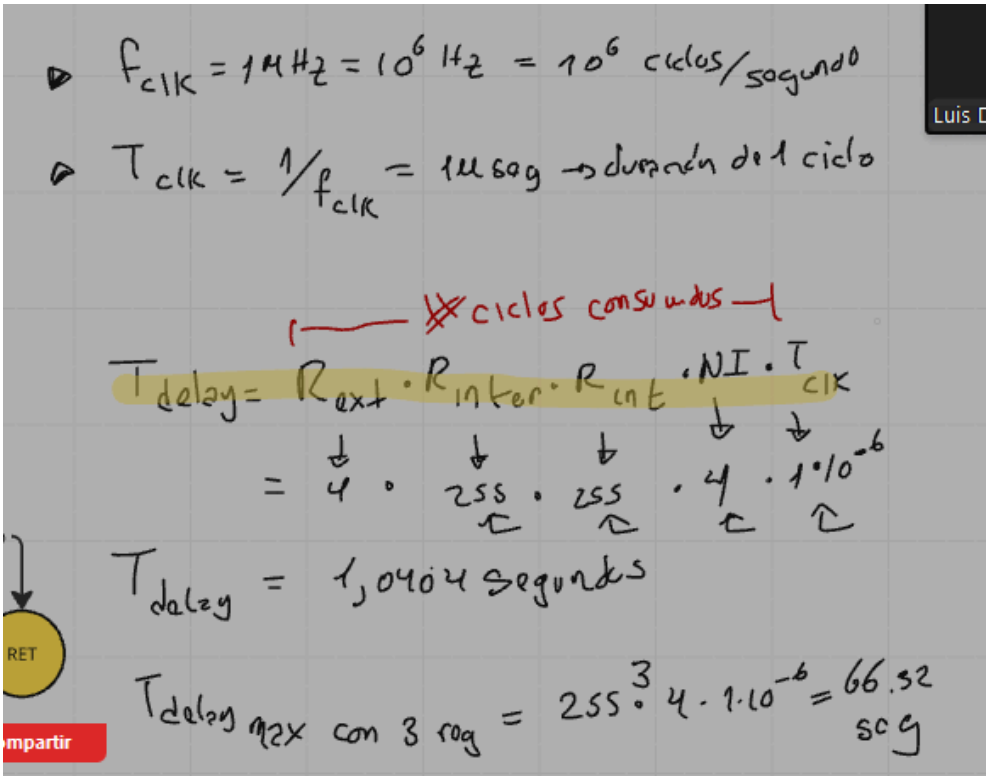



Toggle LED
Cargar en el R20 la posición deseada (0000 0010)
Cargar el estado del puerto D usando PIND
XOR entre PIND y R20, para ejecutar un toogle solo a PD1
RET


Upgrade
DELAY1SEG
Cargar en Rexterno=4
¿Rexterno != 0?
Sí
No
RET
Cargar en Rintermedio=255
¿Rintermedio != 0?
Sí
Decrementar Rexterno
Cargar en Rinterno=255
¿Rinterno != 0?
Sí
No
NOP
Decrementar Rintermedio
Decrementar Rinterno


Está compartiendo la pantalla
Dejar de compartir
M4
viernes, 2 de mayo de 2025 11:40
XOR ≈ toggle si B=1
R20 0000 0010 : PD1
R21 0000 0010 : PIND
R21 0000 0010 → PORTD
XOR
R20 0000 0010 : PD1
R21 0000 0010 : PIND
R21: 0000 0000 → PORTD
3 formas de decir "Quiero modificar solo PD1"
PORTD: PD6 PD5 PD4 PD3 PD2 PD1 PD0
0b 0000 0010
0x02
(1<<1)


delay2AVR.py
requirements.txt
RetardoAVREnsamblador.pdsprj
RetardoAVREnsamblador.pdsprj.DESKTOP-6MAFF1OJ...
Calculadora
Científica
4 × 250 × 250 × 4 × 0.000001 =
1
DEG F-E
MC MR M+ M− MS M∨
Trigonometría Función
return None
resultado = calcular_retardo_max_10_nops(1e6, 1.0) # f=1MHz, delaydeseado= 1 segundo
print(resultado)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ments/barahona_2025_marzo/Mentorias/Mentoría 4/Retardos/delay2AVR.py"
Delay deseado: 1000000.00 us
Delay real: 1000000.00 us
Error: 0.00 us
Ciclos necesarios: 1000000.0
{'Registros': [4, 250, 250], 'NOPs_internos': 4, 'Delay_real': 1.0, 'Error': 0.0}
PS C:\Users\luisdavidbarahona\Documents\barahona_2025_marzo\Mentorias\Mentoría 4\Retardos>


f_clk = 1MHz = 10^6 Hz = 10^6 ciclos/segundo
T_clk = 1/f_clk = 1 µseg → duración de 1 ciclo
# ciclos consumidos
T_delay = R_ext · R_inter · R_int · NI · T_clk
= 4 · 255 · 255 · 4 · 1·10^-6
T_delay = 1,0404 segundos
T_delay max con 3 reg = 255^3 · 4 · 1·10^-6 = 66.32 seg
RET
mpartir
Luis

2.4. Código en Ensamblador

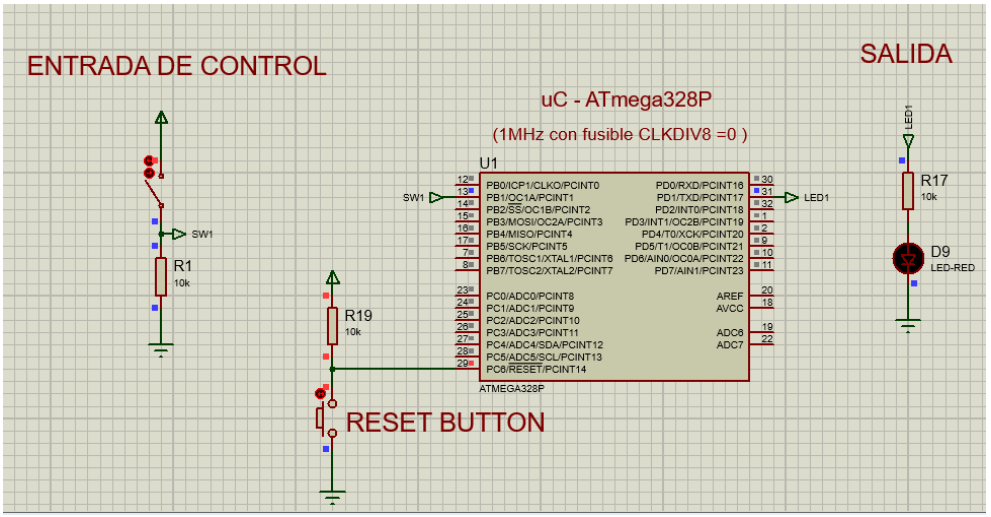
https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/Retardos/RetardoAVREnsamblador/RetardoAVREnsamblador/main.asm

2.5. Código en C

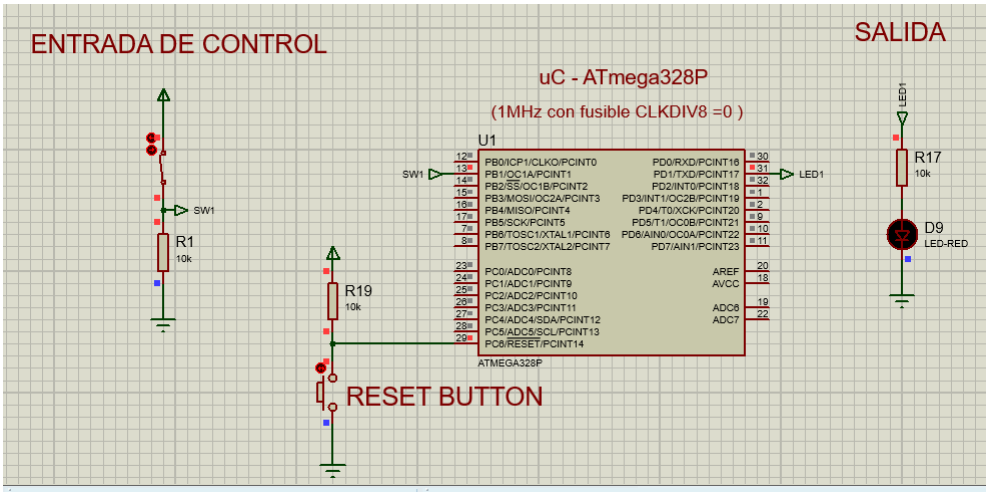
https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/Retardos/RetardoAVR_CodeC/RetardoAVR_CodeC/main.c

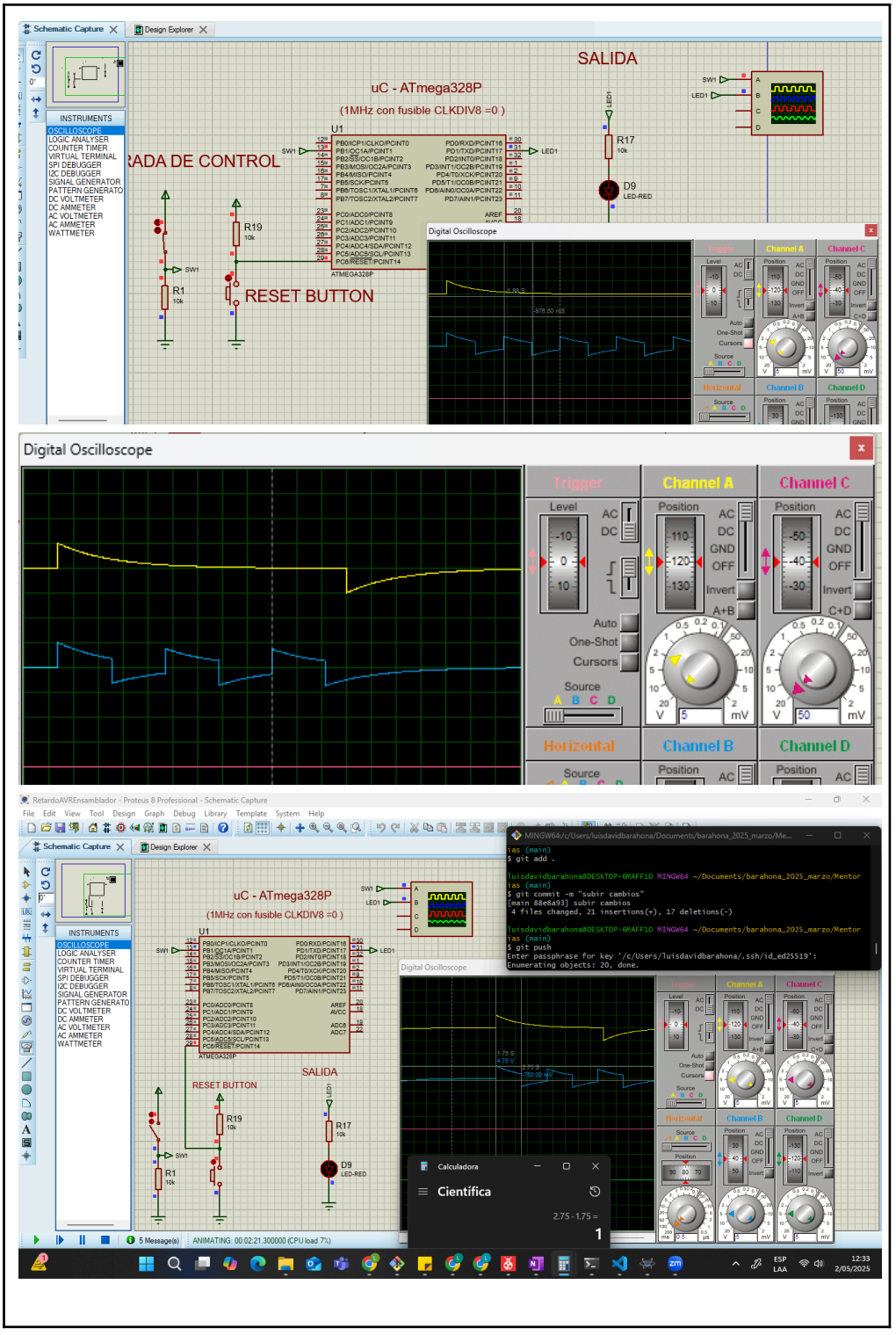
2.6. Simulación de los casos en Microchip y Proteus

CASO 1: SWITCH NO PRESIONAO



CASO 2: SWITCH PRESIONADO





2.7. Implementación

2.8. Documentación en GITHUB

CODE AVR ENSAMBLADOR:
https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/Retardos/RetardoAVREnsamblador/RetardoAVREnsamblador/main.asm

CODE AVR C:
https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/Retardos/RetardoAVR_CodeC/RetardoAVR_CodeC/main.c

SCRIPT PYTHON PARA CALCULAR EL VALOR DE LOS R:
https://github.com/luisbarahona100/Mentorias/blob/main/Mentor%C3%ADa%204/Retardos/delay2AVR.py

3. Anexo

3.1. Plantilla base de un código en AVR Ensamblador

3.1.1. Plantilla general de un programa en AVR Ensamblador

```
None
; --- Directivas iniciales ---
.include "m328Pdef.inc"

; --- Definción de macros ---

; --- Redireccionamiento al posición 0x00 de la memoria de programa
.org 0x00
    rjmp RESET

; --- Variables temporales ---
.def temp = r16

; --- Configuración de puertos e inicialización ---
RESET:

; --- Programa principal ---
MAIN:

; --- Subrutinas ---
OTRAS_ETIQUETAS:
```

3.1.2. Plantilla de una macro en AVR Ensamblador

```
None
;CREACIÓN DE MACRO
.macro toggle_bit
```

```
.endm

;USO DE MACRO
toggle_bit
```

- Es definido antes del MAIN

3.1.3. Plantilla de una rutina en AVR Ensamblador

```
None
;CONFIGURAR STACK POINTER
.include "m328Pdef.inc"
.org 0x00
    rjmp RESET
RESET:
    LDI R19, LOW(RAMEND)
    OUT SPL, R19
    LDI R19, HIGH(RAMEND)
    OUT SPH, R19


; USAR RUTINA
MAIN:
      RCALL RUTINA_SUMAR
      RJMP MAIN

;DEFINIR RUTINA
RUTINA_SUMAR:
      ;Lógica de la suma
RET
```

3.2. Plantilla base de un código en C

3.2.1. Plantilla general de un programa en C

```
C/C++
#include <avr/io.h>
```

```
#include <util/delay.h>
#include <avr/interrupt.h>
#include "DEF_ATMEGA328P.h"
#include <avr/sfr_defs.h>

int main(){


while (1){

}

}
```

- Se recomienda abrir el archivo debuggs/.lss para poder observar el código ensamblador equivalente al code C escrito.

3.2.2. Plantilla de una macro en C

```C/C++
// Macro para alternar el estado de un bit en un puerto
#define TOGGLE_BIT(port, pin) ((port) ^= (1 << (pin)))

//USO
TOGGLE_BIT(PORTB, PB0); // Alterna PB0
```

3.2.3. Plantilla de un subrutina en C

```C/C++
int function rutina (param1, param2){
      #LOGICA: SUMAR param1 y param2

}

#uso

int resultado=0;
resultado = rutina(1,2);
```

- En C, una rutina es la función que conocemos de toda la vida.