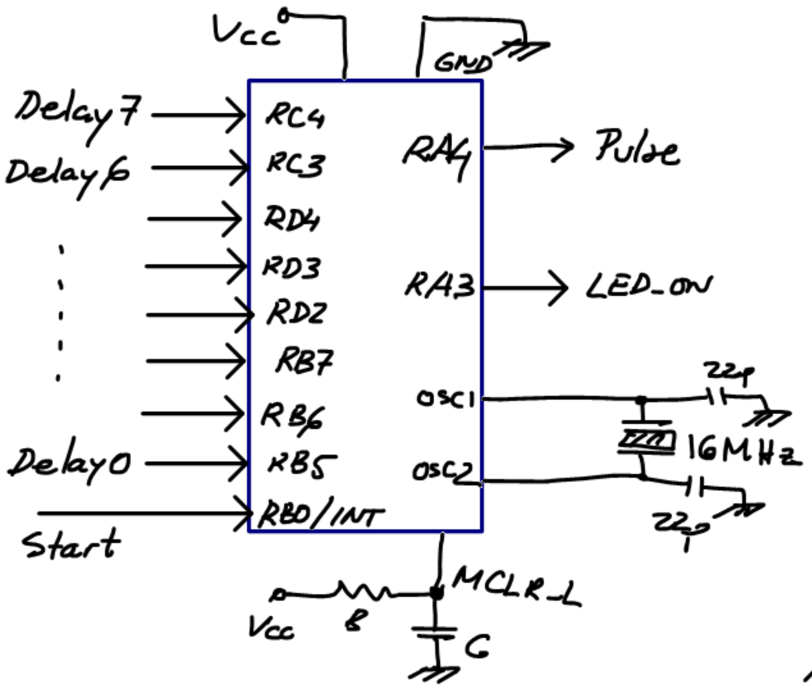
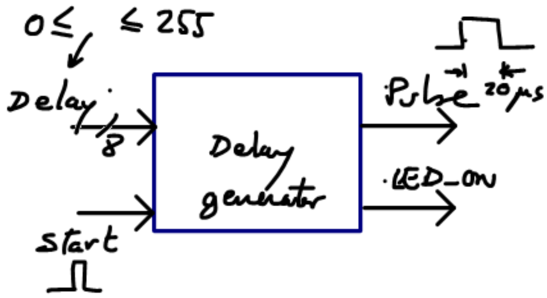
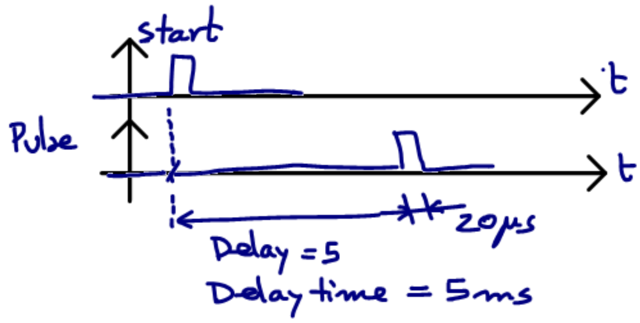
Example solutions for the Problem 2

a)

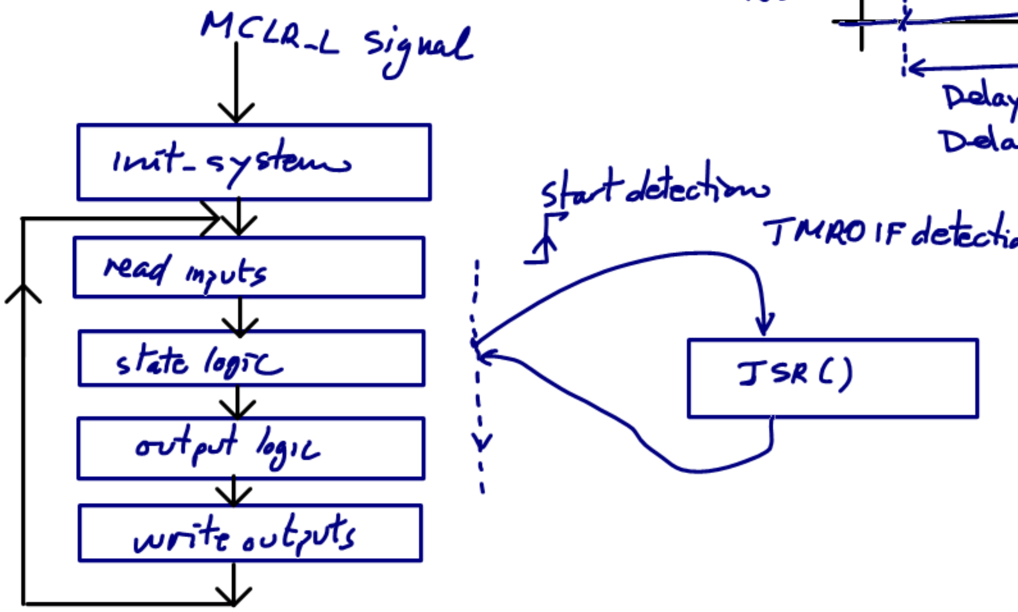




start the sequence to generate the pulse after the [Delay] value in ms



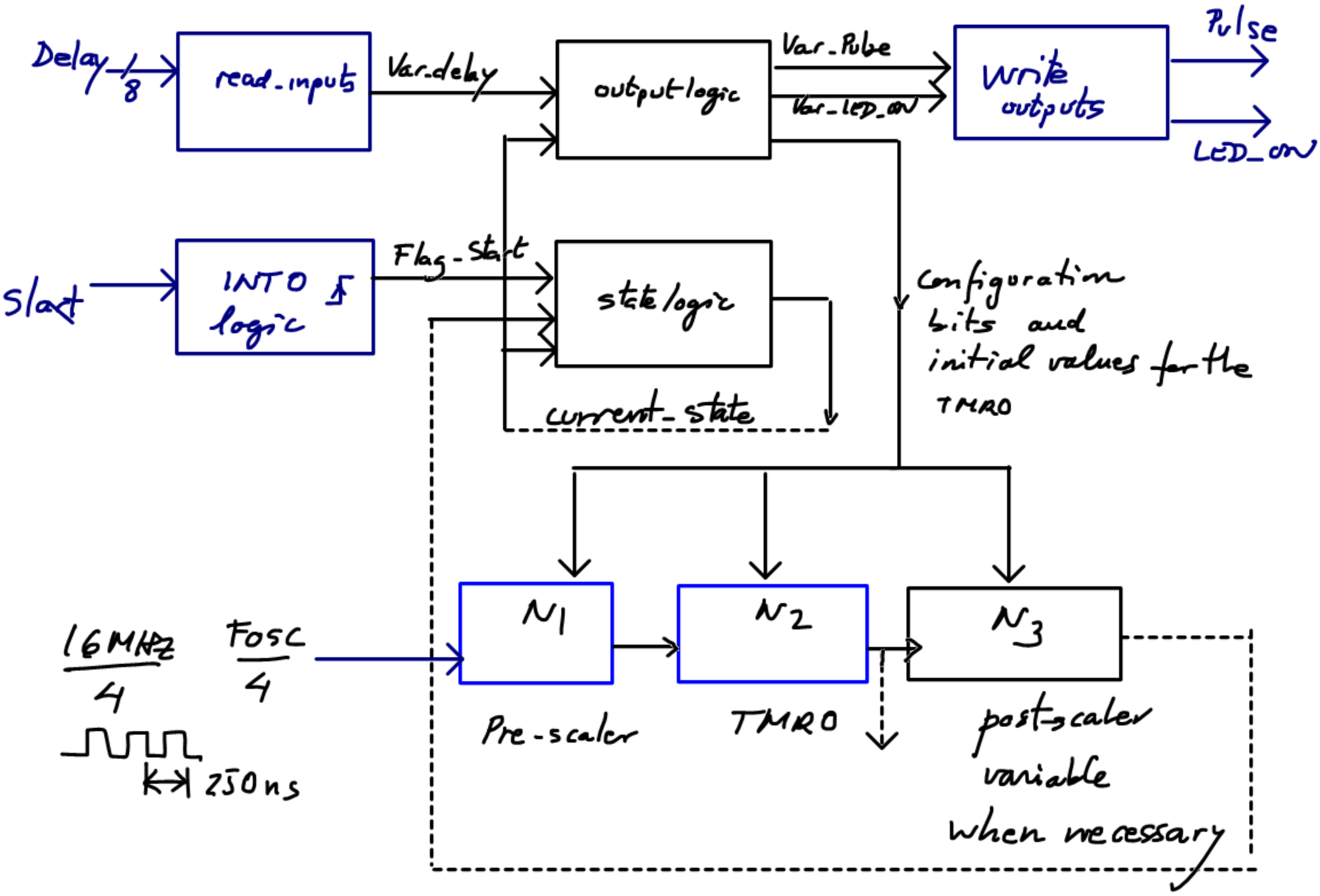
b)



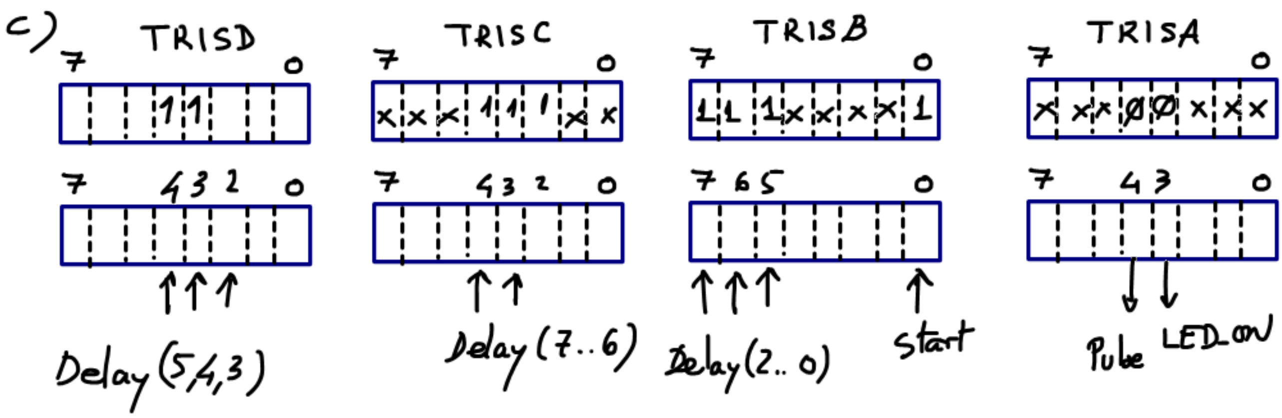
} interrupts to start the sequence and to detect the end of the delay time and the pulse time (20µs)

8 bit

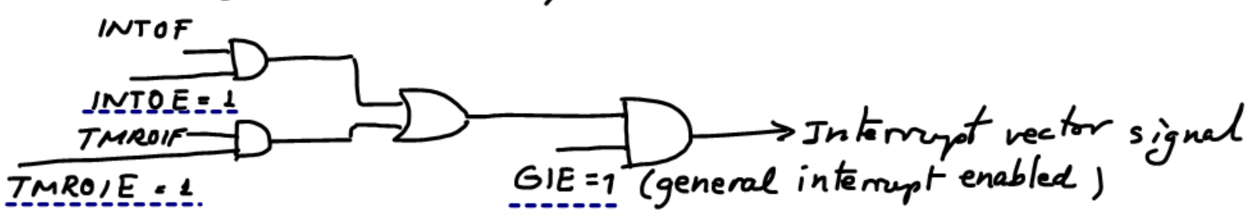
char		Var_Delay	(8 bit to program the delay)
char	0 - - - - - 0 1	Var_Pulse	→ '1' to generate the 20µs pulse
char	0 - - - - - - 0 1	Var_LED_ON	→ '1' when running
char	0 - - - - - - 0 1	Flag_Start	→ '1' when INT0IF = 1
char	0 - - - - - - 0 1	Flag_Timer	→ '1' when TMR0IF = 1
char		current_state	(8 bits to encode the states A, B, C, ...)



c)



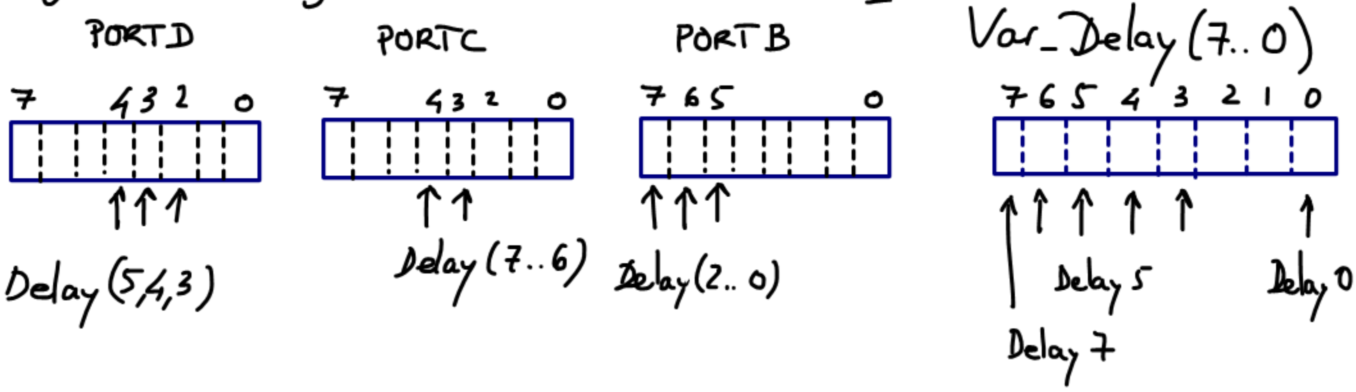
- When detecting an active edge (programmable) from INT0 (RB0) the hardware flag is set (INT0IF = 1) rising/falling



- when detecting TMR0 overflow → TMR0IF = 1

⇒ Both enabling masks must be set to allow interrupts)

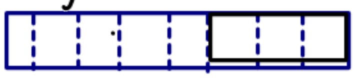
d) The objective is to generate the variable → char Var_Delay(7..0)



1. Read PORTD
2. Mask 0b00011100
3. Shift ← 1 bit and save in buffer Var_Buffer1

4. Read PORTC
5. Mask 0b 00011000
6. Shift ← 3 bit and save in Var_Buffer2

7. Read PORTB
8. Mask 0b11100000
9. shift → 5 bit and OR and save in Var_delay



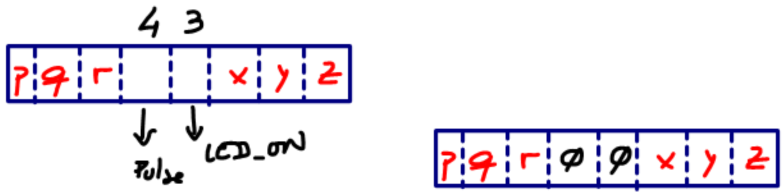
```
Var_delay = ((PORTB & 0b11100000) >> 5) | Var_Buffer2 | Var_Buffer1;
```

& → Bitwise AND

| → Bitwise OR

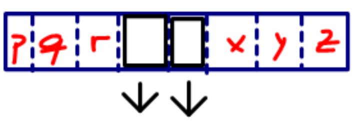
e) To write Var_Pulse and Var_LED_ON in the same port while preserving other port bits

1. Read the PORT A
2. Mask bits of no interest (to preserve them)
3. OR the Pulse and LED_ON bits after shifting them
4. Write the PORT A




```
Var_Buffer1 = PORTA & 0b11100111;
```

```
PORTA = Var_Buffer1 | (Var_Pulse << 4) | (Var_LED_ON << 3);
```



f) • Let's use the timer 0 to generate the delay. For example:

$$\text{Delay time} = \frac{4}{16\,MHz} \cdot N_1 \cdot N_2 \cdot N_3 \quad (ms)$$

- $\frac{4}{16\,MHz}$ → 250 ns
- N_1 → 4
- N_2 → (256 − Var_Delay)
- N_3 → 1000 (int variable)
- 250 ns · 4 → 1 µs
- 1 µs · N_2 · 1000: 1 µs · N_2 → µs; × 1000 → ms
- N_2: 1 → 1 ms … 255 → 255 ms

• Let's use the same timer 0 to generate 20 µs. For example:

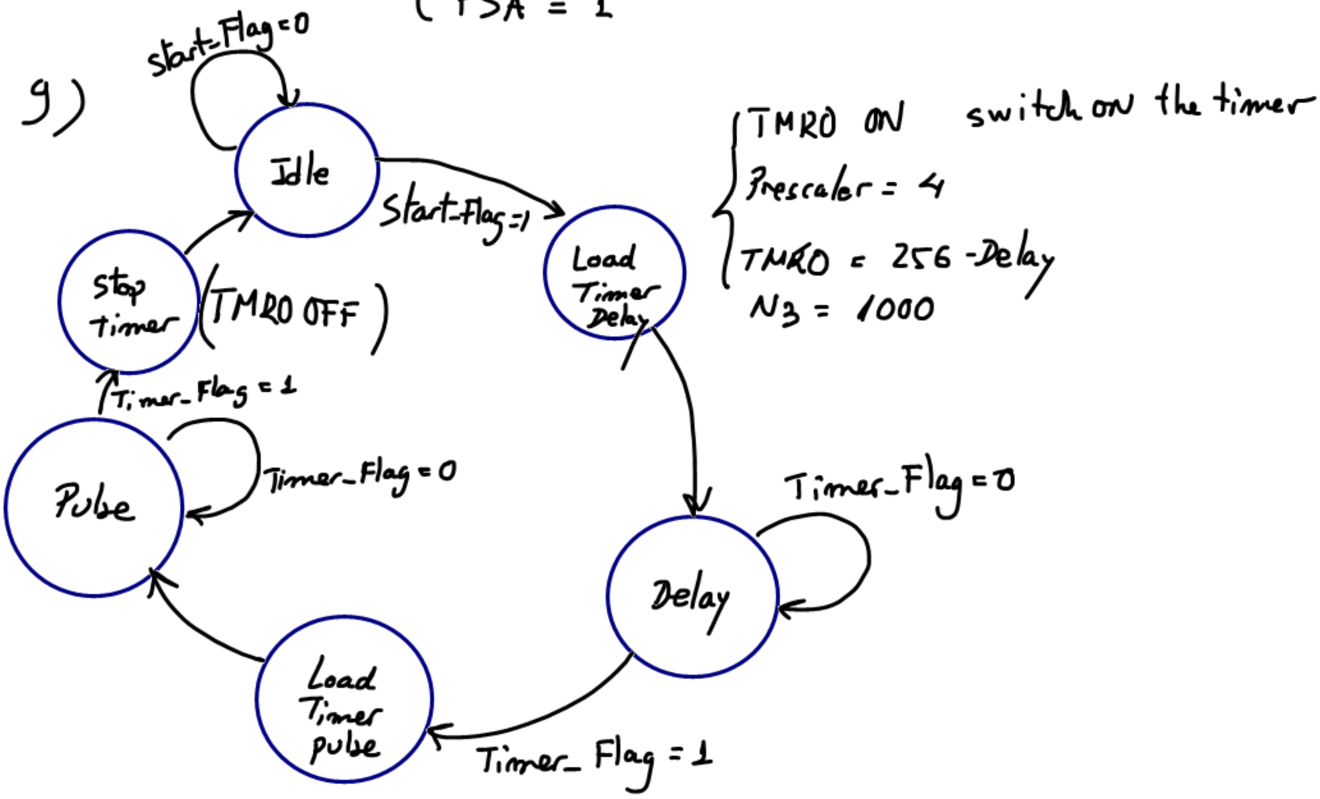
N_3 is not required

$$20\,\mu s = 250\,ns \cdot 4 \cdot 20$$

- 4 → N_1
- 20 → N_2 (256 − 20) TMR0

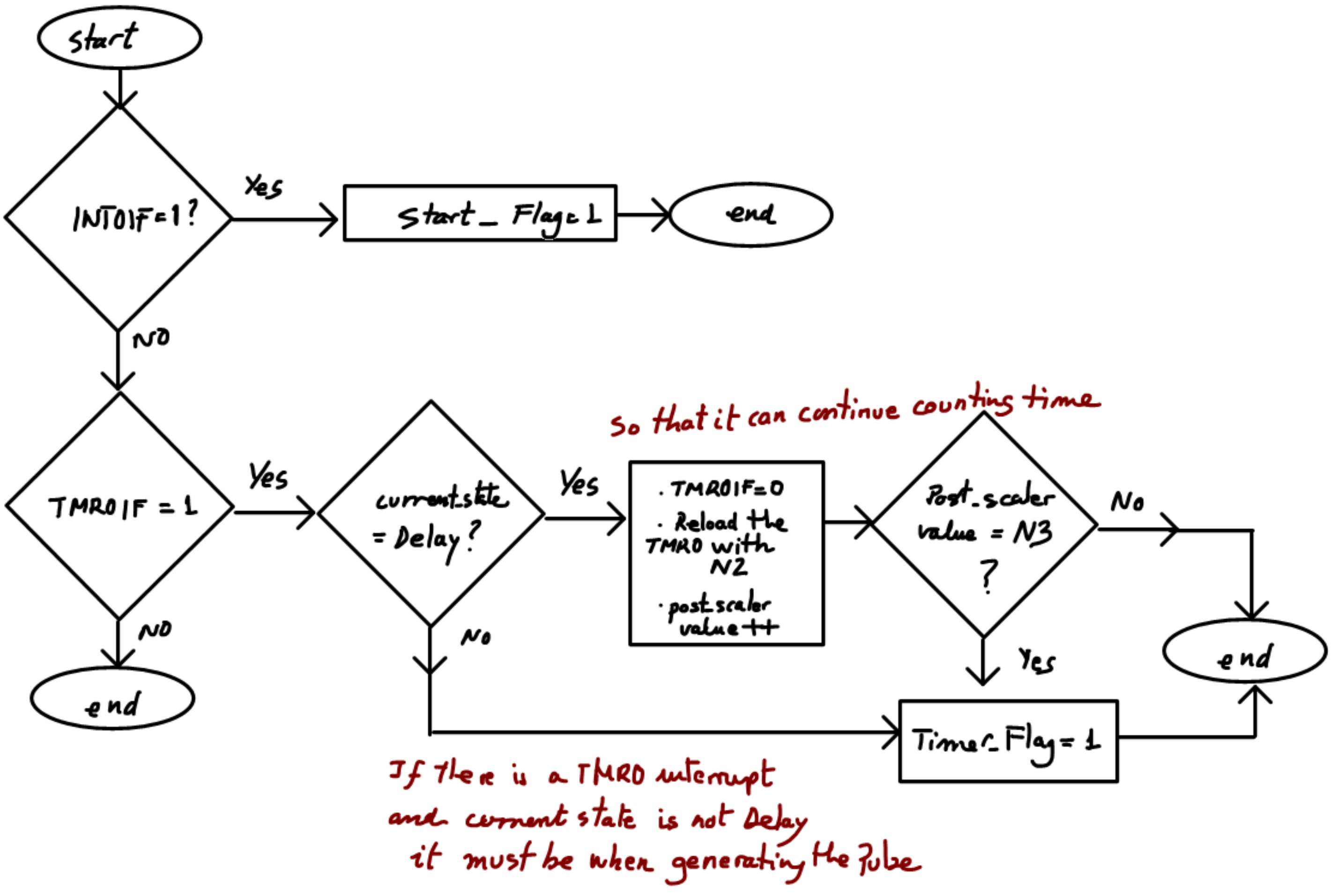
For both timings:
- T0SE = x
- T0CS = ∅
- T0PS(2..0) => Select ÷4 (N_1 prescaler)
- PSA = 1

g)



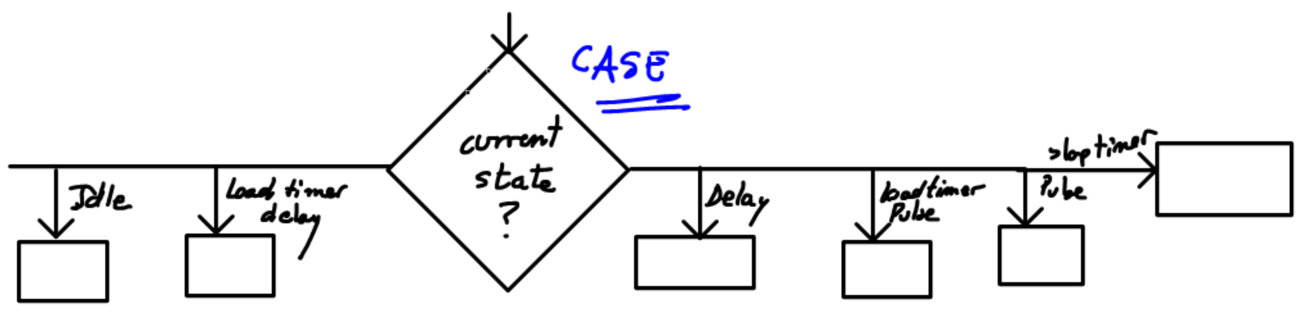
6 states are required to generate the delay and the pulse

h) The ISR must be in charge of setting the Start_Flag and the Timer_Flag variables



i) output_logic generates the variables Var_Pulse, Var_LED_ON and the values to configure the Timer0

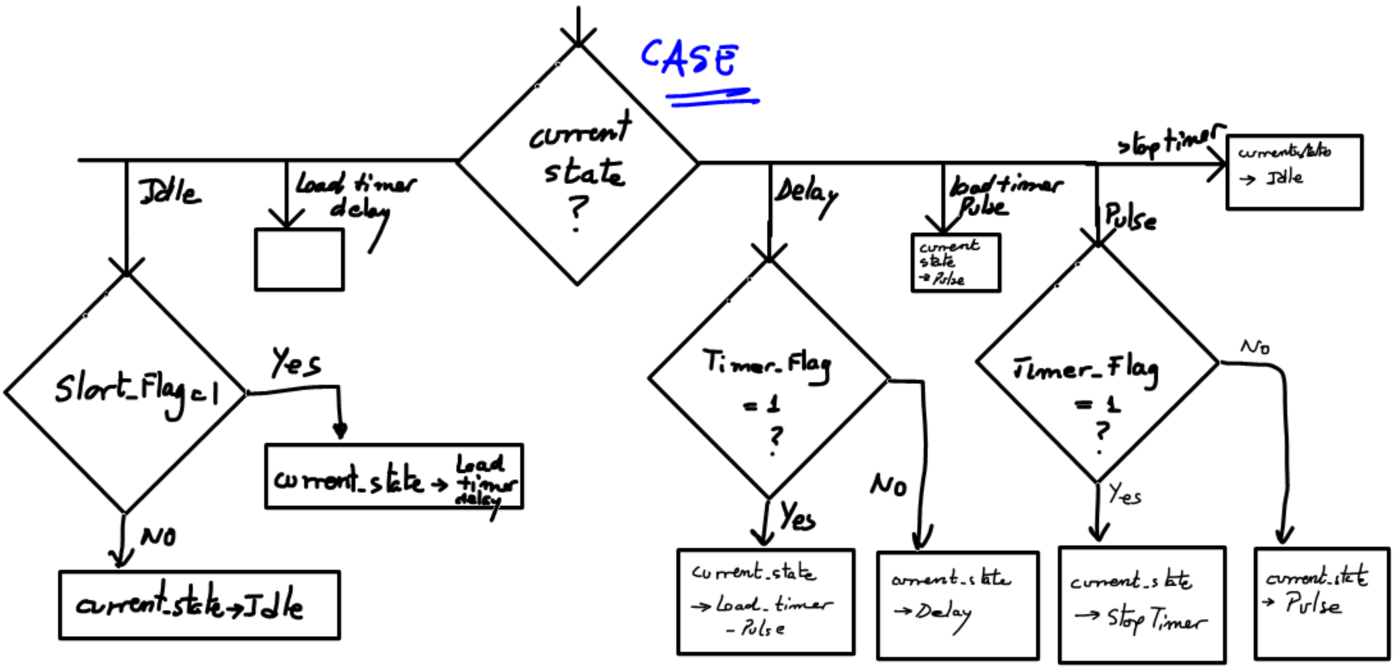
Delay	Current_state	Var_LED ON	Var_Pulse	Timer0 configuration bits and variables
x	Idle	0	0	Timer off
Delay	Load Timer delay	1	0	$N_1 = 4$; $N_2 =$ Delay
x	Delay	1	0	
x	Load pulse	1	1	$N_1 = 4$; $N_2 = 20$
x	Pulse	1	1	
x	Stop timer	0	0	Timer off



j)

Timer_Flag	Start_Flag	current_state	current_state+
x	0	Idle	Idle
x	1	Idle	Load timer delay
x	x	Load timer delay	Delay
0	x	Delay	Delay
1	x	Delay	Load timer pulse
x	x	Load timer Pulse	Pulse
0	x	Pulse	Pulse
1	x	Pulse	Stop timer
x	x	Stop timer	Idle

This function generates all the state transitions (arrows) and it is also interpreted in a behavioural way to generate the C code. The important statement is also the switch _ case



9 operations to set the new value of the current_state variable

As usual, with all this a, b, ... j plannings, now is time to start developing the project in the lab.

→ Take an example from P10 - P11 - P12 and copy & adapt it step by step.