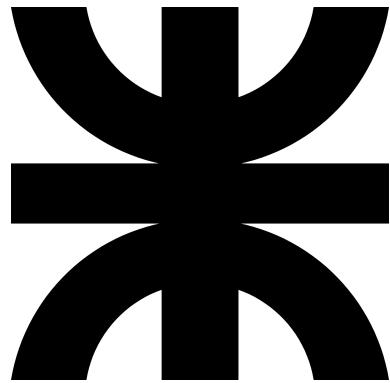


Universidad Tecnológica Nacional

Facultad Regional Córdoba



Carrera de Ingeniería en Electrónica

TÉCNICAS DIGITALES III

Profesor titular: Ing. Gutiérrez, Guillermo.

Jefe TP: Ing. Benasulin, Dimas.

TRABAJO PRÁCTICO N°1

“Familiarización con la tecnología de microcontroladores”

INTEGRANTES:

Campos, Luciano

Leg. : 70745

Delgado, Kevin

Leg. : 69708

Llovet, Mauricio

Leg. : 71573

Resumen

El presente informe corresponde al práctico N°1 de la cátedra de “Técnicas Digitales III” de la Universidad Tecnológica Nacional, Regional Córdoba. El mismo tiene como objetivo constituir una experiencia de aproximación práctica a la programación de microcontroladores con el uso de interrupciones, temporizadores y perro guardián.

El informe se divide en tres secciones y un anexo; en la primera sección se presenta la problemática de diseño a resolver. En la segunda se presenta la solución propuesta, mientras que la tercera y cuarta sección presentan, respectivamente, los entornos y métodos empleados para la verificación y validación de la solución. Por último, se tienen dos anexos: en el primero se expone la totalidad del código generado y en el segundo se demuestra el cálculo del período de la comunicación.

Todo el material expuesto en este informe puede consultarse en el siguiente repositorio: [Repo_GitHub_TDIII](#)

Contents

1	Objetivo	3
1.1	Generalidades	3
1.2	Comunicación serie	3
1.3	Interrupción por pulsador	3
1.4	Watchdog Timer	3
2	Solución propuesta	4
2.1	Estructura del código	4
2.2	void main (void):	9
2.3	Recepción por UART	11
2.4	Respuesta a PC por UART	13
2.5	Interrupciones y manejo de la salida PWM	14
2.5.1	Control de salida PWM	16
2.5.2	Interrupción por Pulsador	16
2.5.3	Reseteo por WatchDog	17
3	Verificación de software	20
3.1	Entorno de trabajo	20
3.2	Resultados de simulación en MPLAB	20
3.2.1	Configuración del estímulo	20
3.2.2	Temporización de UART	20
3.2.3	Salida PWM	22
3.2.4	Interrupción por pulsador	23
3.2.5	Reinicio por WDT	25
4	Validación del software	27
4.1	Entorno de trabajo	27
4.1.1	Programador	28
4.2	Prueba funcional	30
4.2.1	Comunicación por UART	30
4.2.2	Control de la salida PWM por UART	30
4.2.3	Control del ciclo de trabajo mediante pulsador	31
4.2.4	Reinicio del sistema por WDT	33
5	Conclusiones	35
6	Anexo A: Código para PIC12F629	36
7	Anexo B: Temporización de UART	45

1 Objetivo

1.1 Generalidades

Desarrollar un programa para un microcontrolador PIC de 8 bits que permita el control de la intensidad de un LED a través de una comunicación serie con una PC y un pulsador.

1.2 Comunicación serie

- La PC enviará un comando que representa el nivel de intensidad del LED conectado a la salida, siendo 255 el 100% de intensidad; el resto se distribuye linealmente.
- La comunicación será asíncrona a 9600 baudios en formato 8N1.
- Se utilizará una aplicación en la PC para mostrar la comunicación con el sistema.
- El micro enviará un ACK¹ de recepción de datos '0x1b'.
- El micro enviará el dato '0xEE' cuando faltan 20 (veinte) segundos para el reinicio por WDT².

1.3 Interrupción por pulsador

Se debe implementar una interrupción por hardware generada por un pulsador, la cual ajustará el valor del ciclo de trabajo del PWM del diodo LED. El ajuste será en pasos discretos de manera tal que el pulsador ajustará el ciclo de trabajo al próximo valor de la lista por encima el valor actual.

- PWM = 0%.
- PWM = 10%.
- PWM = 25%.
- PWM = 50%.
- PWM = 75%.
- PWM = 100%.

La escala presentada es cíclica.

1.4 Watchdog Timer

El sistema debe reiniciarse mediante el WDT y apagar el LED 2 (dos) minutos luego del último dato recibido desde la PC.

¹Acknowlege

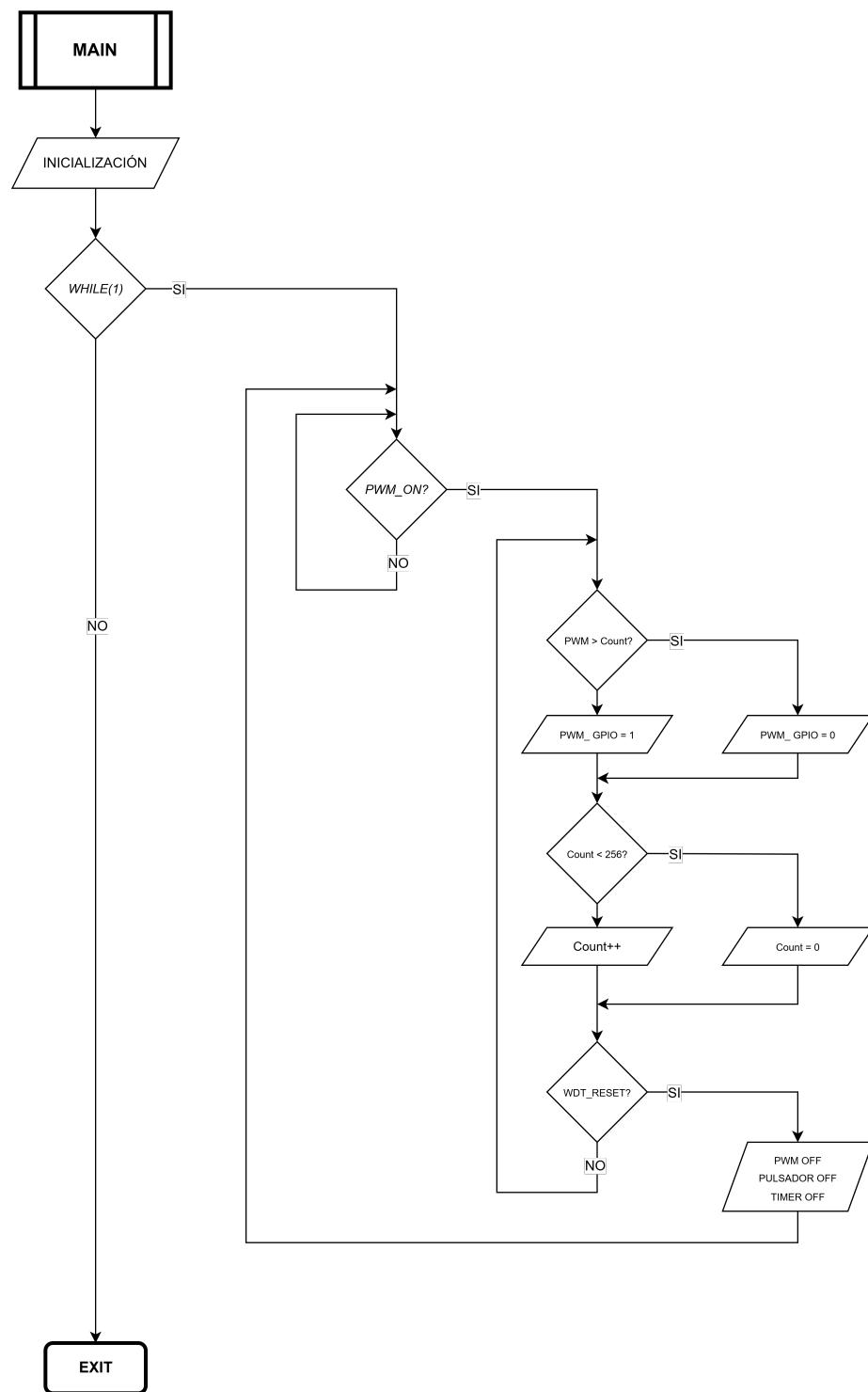
²Wacht Dog Timer

2 Solución propuesta

2.1 Estructura del código

Se plantea estructurar el código según los diagramas de flujo mostrados en [1](#), [2](#), [3](#) y [4](#). Posterior a ellos la estructura completa del código basado en los mismos. Las características principales del código según estos son las siguientes:

- el bloque *main* consta de un bucle *while* infinito que chequea el estado de variable que funciona como bandera para generar o no un PWM en la GPIO conectada al diodo LED.
- En todo momento la ejecución del *main* puede ser interrumpida por una interrupción de hardware generada por el cambio de estado del GPIO asociado a la recepción de la comunicación serie. En el inicio del sistema, o luego de un reinicio, esta interrupción por hardware es la única que puede interrumpir el *main* ya que las demás interrupciones están desactivadas.
- La interrupción por hardware en el pin de recepción habilita una rutina de recepción sincronizada con el período de bit en la comunicación serie. El dato recibido se valida y se procede a responder al PC según lo requerido en la consigna.
- La interrupción anterior establece, además, un contador para el reinicio del sistema a los dos minutos desde el momento en que se recibió el dato. Notar que por cada nuevo dato recibido el bloque de interrupción por hardware reinicia este contador.
- La interrupción por pulsador compara el valor del dato actual con una lista predefinida y asigna el nuevo valor según cuál sea el primer valor mayor al actual en la lista. Para el caso especial en que el ciclo de trabajo sea 100% el ciclo de trabajo se setea al próximo elemento en la lista que es 0%.
- Se emplea una interrupción por *timer* para avanzar un contador con un periodo conocido permitiendo controlar el reinicio por *Watchdog* en una marca de tiempo específico. Transcurridos 100 (cien) segundos desde la última comunicación se envía a la PC el dato especificado en consigna (0xEE), y tras 120 (ciento veinte) segundos de recibido el dato (20 segundos desde que se envía el mensaje 0xEE) se deja de alimentar el *watchdog* y se permite el reinicio del microcontrolador.

Figure 1: Diagrama de flujo *main*.

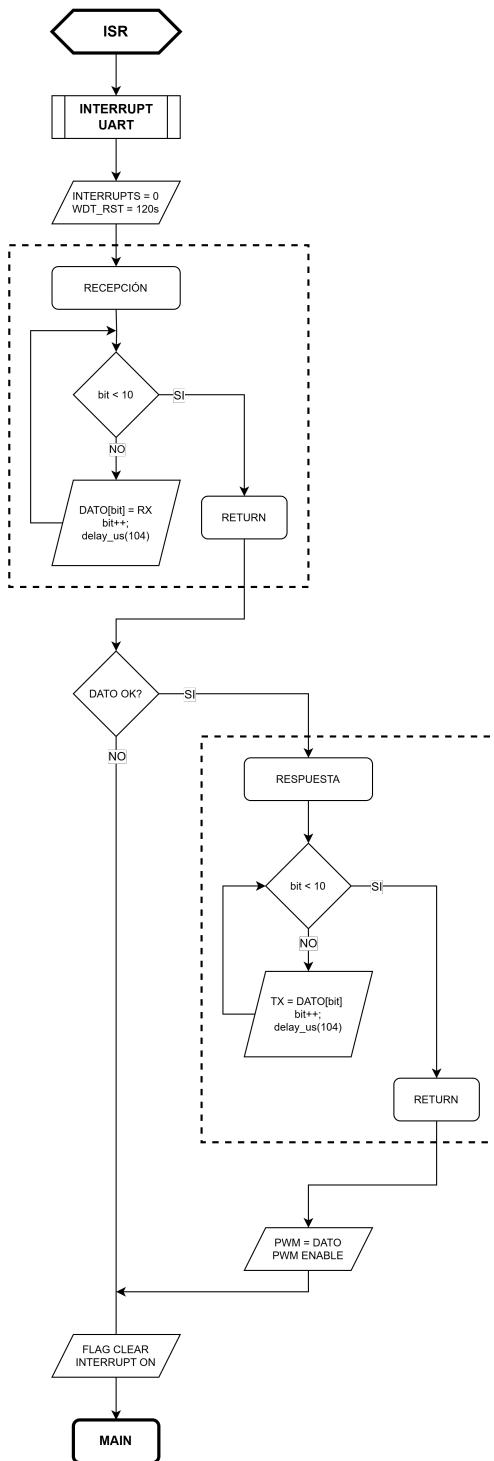


Figure 2: Diagrama de flujo interrupción por hardware.

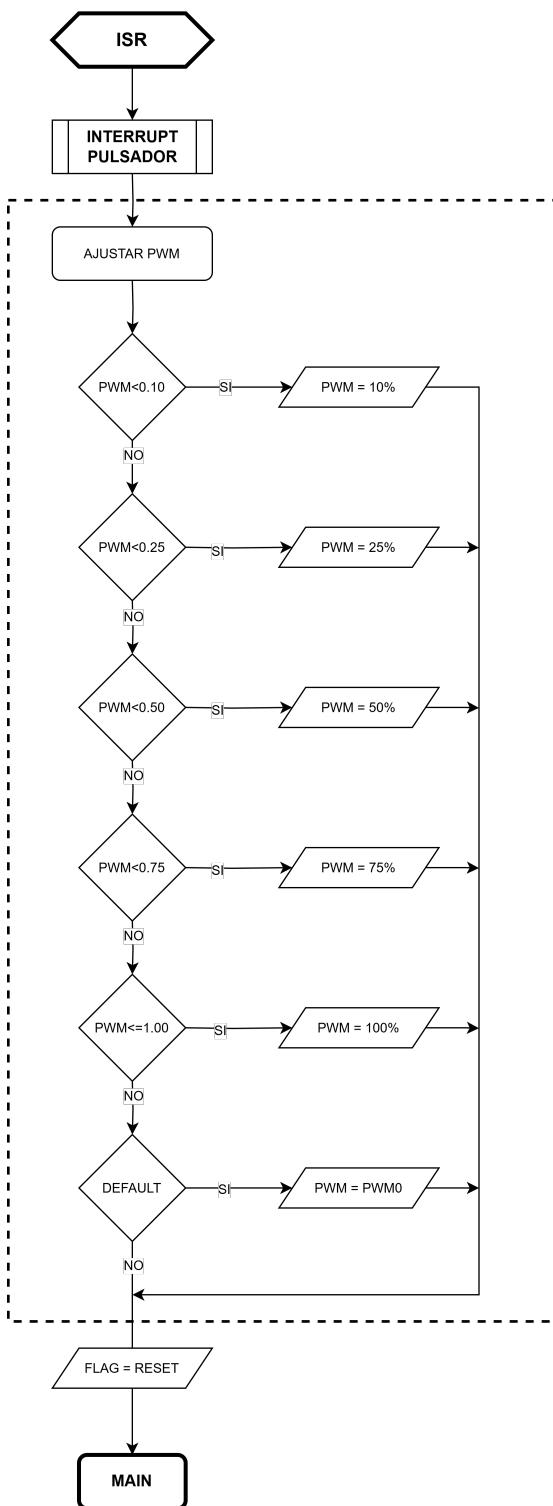
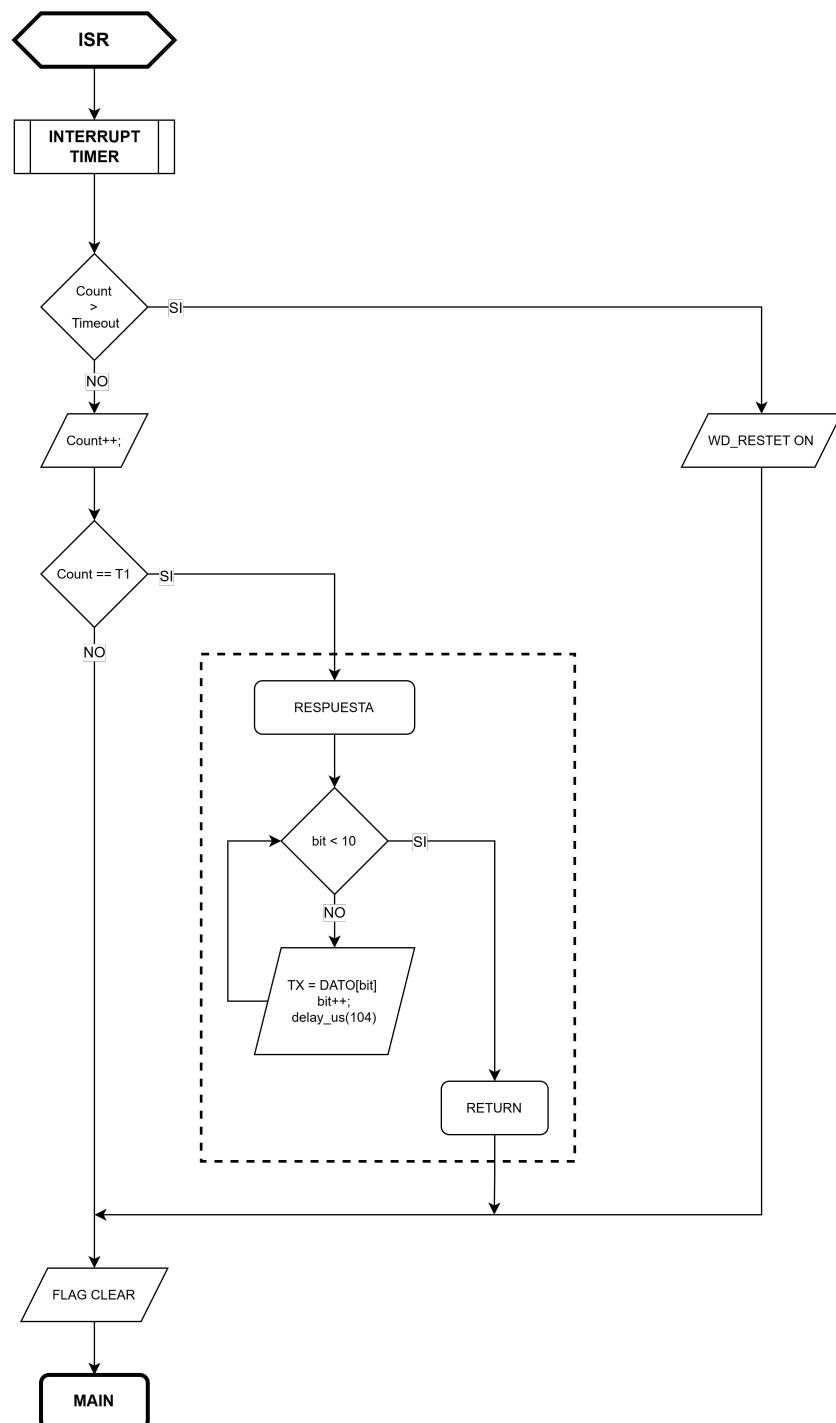


Figure 3: Diagrama de flujo interrupción por gpio.

Figure 4: Diagrama de flujo *timer*.

2.2 void main (void):

El código *main.c* consiste principalmente en:

- Manipulación de registros para la configuración de GPIOs e interrupciones.
- Bucle infinito para el manejo de la salida PWM.

La información referente a la configuración de registros puede encontrarse en la hoja de datos del **PIC12F629**, en este informe no se explicará en detalle este segmento del main. La configuración de pines definida para el práctico es la siguiente:

- *GP0* salida PWM.
- *GP1* entrada de pulsador.
- *GP2* RX UART
- *GP4* TX UART.

Mas adelante se volverá sobre esta porción del código para abordar el control de la salida PWM y el reinicio del sistema. Por ahora puede decirse que una vez habilitada la salida PWM, el tiempo de actividad queda controlado por una rutina de interrupción. Según sea el estado del puerto UART el resto del código decidirá, o no, dejar de refrescar el WDT y permitir el reinicio del sistema. La implementación del código se adjunta a continuación.

```
1 void main(void) {
2     // CONFIGURACIÓN GPIO
3     TRISIObits.TRISIO0=0;           // GP0 = Salida PWM
4     TRISIObits.TRISIO1=1;           // GP1 = Entrada Pulsador
5     TRISIObits.TRISIO2=1;           // GP2 = UART RX
6     TRISIObits.TRISIO4=0;           // GP4 = UART TX
7     CMCON = 0x07;                  // Desactivar comparadores
8     PWM_OUT = 0;                   // Inicialización de GPIO PWM
9     UART_TX = 1;                   // Inicialización de GPIO UART
10
11    // CONFIGURACIÓN TIMER 0
12    OPTION_REGbits.T0CS = 0;        // CLK para TMR0. T=4/fosc
13    TMR0 = 0;                      // Inicializo timer
14
15    // CONFIGURACIÓN WDT Y PRESCALER
16    OPTION_REGbits.PSA = 1;          // Asigna el prescaler al WDT
17    OPTION_REGbits.PS = 0b111;       // Prescaler a 1:32 (T=576ms)
18
19
20    // INTERRUPCIONES
21    INTCONbits.INTE = 1;            // Interrup. por GP2 (UART RX)
22    OPTION_REGbits.INTEDG = 0;       // Interrup. en flanco de bajada
23    INTCONbits.INTF = 0;            // Inicialización de bandera
24    IOCbis.IOC1 = 1;                // Interrup. GP1 (PULSADOR)
25    INTCONbits.T0IE = 0;             // Interrup. TMR0
26    INTCONbits.GIE = 1;              // Global Interrupt Enable
27    INTCONbits.GPIE = 0;             // Global Peripheral Interrupt
28
29    // VARIABLES
30    int Contador_PWM = 0;
31    CLRWDT();
32
33    // Main
34    while(1){
35        if(FLAG_PWM_ENABLE == 1){
36            if(PWM_DUTY_CYCLE == 255)
37                PWM_OUT = 1;
38            else if(Contador_PWM < PWM_DUTY_CYCLE)
39                PWM_OUT = 1;
40            else
41                PWM_OUT = 0;
42            if(Contador_PWM < 255)
43                Contador_PWM+=5;      // Resolución PWM ~ 2.34%
44            else
45                Contador_PWM = 0;
46        }
47        if (CONTADOR_REINICIOS_WDT < MAX_REINICIOS_WDT)
48            CLRWDT();
49        if(STATUSbits.nTO == 0){
50            FLAG_PWM_ENABLE = 0;
51        }
52    }
53    return;
54 }
```

2.3 Recepción por UART

La recepción comienza cuando se detecta un flanco de bajada en el terminal RX del pic (GP2). Por simplicidad, se empleó GP2 que posee una interrupción por hardware dedicada. Una vez se detecta el flanco de bajada correspondiente al start-bit de la comunicación serie se desactiva el resto de interrupciones del sistema y se utiliza el temporizador 0 (TMR0) para asegurar el sincronismo en el algoritmo de recepción. También se inicia el temporizador 1 que será el encargado, de forma indirecta, de reiniciar el sistema si transcurren 2 minutos sin que se reciba un nuevo dato.

La interrupción llama a una función externa *int RECEPCION_UART()* que sincroniza la lectura del puerto RX mediante interrupciones por TMR0 y almacena el resultado bit a bit en una variable intermedia. Contabilizados los 10 bits correspondientes a una comunicación serie 8N1, se verifica la presencia del bit de stop y se transfiere el dato (se valida) de la variable local de recepción a la variable global *DATO_UART*. Adicionalmente, esta función tiene retorno tipo entero para manejo de errores.

El código implementado se muestra a continuación:

```
1 //  INTERRUPCIONES -----
2 void __interrupt() isr(void) {
3
4     // INTERRUPCIÓN POR TMR0 *****
5     if (INTCONbits.T0IF) {
6         TMR0 = 0;
7         INTCONbits.T0IF = 0;          // Bandera de TMR0
8     }
9
10    // INTERRUPCIÓN BIT START UART *****
11    if (INTCONbits.INTF){
12        // se reinicia el WDT, interrupciones y PWM
13        CLRWD();                  // Reinicio del WDT
14        OPTION_REGbits.PSA = 1;    // Asigna el prescaler al WDT
15        OPTION_REGbits.PS = 0b111; // Prescaler a 1:32 (T=2048ms
16        )
17
18        TMR0 = 0;
19        INTCONbits.T0IF = 0;
20        INTCONbits.T0IE = 1;
21
22        INTCONbits.GPIF = 0;
23        INTCONbits.GPIE = 0;
24
25        FLAG_PWM_ENABLE = 0;
26        PIR1bits.TMR1IF = 0;
27        INTCONbits.PEIE = 0;       // Interrupciones perifé
28        ricas
29
30        // validación del dato leído
31        if (RECEPCION_UART()){
32            RESPONDER_UART(0x1B);      // 0b00011011
33            FLAG_PWM_ENABLE = 1;
34            PWM_DUTY_CYCLE = DATO_RECIBIDO_UART; // RESOLUCIÓN
35            PWM ~2.5%
36        }
37
38 }
```

```
35     TMR0 = 0;
36     INTCONbits.T0IF = 0;
37     INTCONbits.T0IE = 0;
38
39     CONTADOR_REINICIOS_WDT = 0;
40     T1CONbits.TMR1CS = 0;           // Reloj interno (Fosc/4)
41     T1CONbits.T1CKPS = 0b11;       // Prescaler a 1:8
42     TMR1 = PRECARGA_TMR1_WDT;    // Precarga TMR1
43     PIR1bits.TMR1IF = 0;
44     T1CONbits.TMR1ON = 1;         // Enciende TMR1
45     PIE1bits.T1IE = 1;
46
47     INTCONbits.GPIF = 0;
48     INTCONbits.INTF = 0;          // Limpieza de bandera
49
50     INTCONbits.PEIE = 1;
51     INTCONbits.GPIE = 1;          // Interrupción pulsador.
52 }
53 ...
54 }
55
56 // RECEPCIÓN POR UART
57
58 //Esta función guarda en la variable global DATO_RECIBIDO_UART
59 //el dato
60 //recibido por comunicación serie en el pin UART_RX
61
62 int RECEPCION_UART (void){
63
64     int desplazamiento_bit = 0;
65     int RECEPCION_EN_PROCESO = 1;
66     int RESULTADO_COMM = 0;
67     unsigned char CARACTER_RECIBIDO_UART = 0x00;
68
69     if(UART_RX == 0){
70         TMR0 = UART_PERIOD_FULL;
71         while(RECEPCION_EN_PROCESO == 1)
72         {
73             if(INTCONbits.T0IF){
74                 TMR0 = UART_PERIOD_FULL;
75                 INTCONbits.T0IF = 0;
76                 if(desplazamiento_bit <8)
77                     CARACTER_RECIBIDO_UART |= (UART_RX <<
78                     desplazamiento_bit);
79                 else{
80                     if (UART_RX == 1)
81                         RESULTADO_COMM = 1;
82                     RECEPCION_EN_PROCESO = 0;
83                 }
84                 desplazamiento_bit++;
85             }
86         }
87         _delay_us(UART_PERIOD_FULL);
88         if((RESULTADO_COMM == 1) & (UART_RX == 1))
89             DATO_RECIBIDO_UART= CARACTER_RECIBIDO_UART;
90     }
91 }
```

```
88         else
89             RESULTADO_COMM = 0;
90     }
91     return (RESULTADO_COMM);
92 }
```

2.4 Respuesta a PC por UART

Una vez recibido el dato y validado el mismo, se prosigue con la respuesta a la PC. Para esta operación se emplea la función *void RESPONDER_UART (unsigned int)*. Esta función emplea una lógica similar a la función de recepción, empleando interrupciones por TMR0 para sincronizar la escritura del puerto GP4 enviando bit a bit el dato que le fue pasado por valor.

La función *void RESPONDER_UART (unsigned int)* tiene una implementación diferente de aquella del código de recepción, se emplea un *switch case* en lugar de un ciclo *for* para lograr una mejor temporización sin necesidad de ajustar temporalmente el retardo entre el envío de cada bit.

El código implementado se muestra a continuación:

```
1 // INTERRUPCIONES -----
2 void __interrupt() isr(void) {
3
4     // INTERRUPCIÓN POR TMR0 ****
5     if (INTCONbits.T0IF) {
6         TMR0 = 0;
7         INTCONbits.T0IF = 0;          // Bandera de TMR0
8     }
9
10    // INTERRUPCIÓN BIT START UART ****
11    if (INTCONbits.INTF){
12
13        // DESACTIVAR INTERRUPCIONES NO DESEADAS
14        ...
15
16        if (RECEPCION_UART()){
17            RESPONDER_UART(0x1B);      // 0b00011011
18            FLAG_PWM_ENABLE = 1;
19            PWM_DUTY_CYCLE = DATO_RECIBIDO_UART; // RESOLUCIÓN
20            PWM ~2.5%
21        }
22
23        ...
24        ...
25
26    }
27
28    ...
29
30    ...
31
32
33 // RESPUESTA POR UART -----
```

```
34
35 void RESPONDER_UART(unsigned char RESPUESTA)
36 {
37     int bit = 0;
38     int ENVIO_EN_PROCESO = 1;           // FLAG PARA CONTROL DE ENVÍO
39
40     TMR0 = UART_PERIOD_FULL;
41
42     while (ENVIO_EN_PROCESO && bit <= 10) {
43         while (!INTCONbits.T0IF);      // Bucle de espera
44             bloqueante
45
46         TMR0 = UART_PERIOD_FULL;
47         INTCONbits.T0IF = 0;
48
49         switch (bit) {
50             case 0: // Start bit
51                 UART_TX = 0;
52                 break;
53
54             case 1: case 2: case 3: case 4:
55             case 5: case 6: case 7: case 8:
56                 UART_TX = RESPUESTA & 1;
57                 RESPUESTA >>=1;
58                 break;
59
60             case 9:      // Stop bit
61                 UART_TX = 1;
62                 break;
63
64             case 10:    // Termina el envío
65                 UART_TX = 1;
66                 ENVIO_EN_PROCESO = 0;
67                 break;
68
69             default:
70                 break;
71         }
72         bit++;
73     }
74
75 }
```

2.5 Interrupciones y manejo de la salida PWM

Una vez recibido el dato por UART desde la PC, y enviada la correspondiente respuesta a la misma, el servicio de interrupción por hardware configura las diferentes interrupciones de interés para que la operación del sistema cumpla con lo especificado en la consigna, es decir:

- La inicialización del PWM al valor especificado por UART.
- La temporización de los primeros 100 (cien) segundos para el envío de alerta de reinicio a la PC.

- La temporización de los 120 (ciento veinte) segundos para el reinicio del sistema ante la falta de nuevos datos.

Terminado el envío, el código logra todo esto de la siguiente manera:

1. Se configura el TMR1 (prescaler, precarga y habilitación de interrupción) para el control de un contador que supervisa el reinicio del WDT.
2. Se modifica el valor de la bandera *FLAG_PWM_ENABLE* para que una vez finalizado el manejo de la interrupcion por hardware el código main encienda la salida PWM.

Estas operaciones se muestran en el código que sigue, y corresponden a las líneas después del bloque *if* (líneas 17 a 33).

```
1 //  INTERRUPCIONES _____
2 void __interrupt() isr(void) {
3     ...
4     // CODIGO
5     ...
6     // INTERRUPCIÓN BIT START UART *****
7     if (INTCONbits.INTF){
8         ...
9         // CODIGO
10        ...
11        if (RECEPCION_UART()){
12            RESPONDER_UART(0x1B);
13            FLAG_PWM_ENABLE = 1;
14            PWM_DUTY_CYCLE = DATO_RECIBIDO_UART;
15        }
16
17        TMR0 = 0;
18        INTCONbits.T0IF = 0;
19        INTCONbits.T0IE = 0;
20
21        CONTADOR_REINICIOS_WDT = 0;
22        T1CONbits.TMR1CS = 0;
23        T1CONbits.T1CKPS = 0b11;
24        TMR1 = PRECARGA_TMR1_WDT;
25        PIR1bits.TMR1IF = 0;
26        T1CONbits.TMR1ON = 1;
27        PIE1bits.T1IE = 1;
28
29        INTCONbits.GPIF = 0;
30        INTCONbits.INTF = 0;
31
32        INTCONbits.PEIE = 1;
33        INTCONbits.GPIE = 1;
34    }
35    ...
36    // CODIGO
37    ...
38 }
```

2.5.1 Control de salida PWM

La activación y control del ciclo de trabajo de la salida PWM es controlada desde *main.c*. Existe un bucle infinito que controla la salida PWM en función del estado de la bandera *FLAG_PWM_ENABLE*. Dado que no se especifica requisito para la salida PWM, no se emplea para la misma una temporización estricta que controle su periodo y frecuencia de refresco. De hecho, según puede verse en el servicio de manejo de interrupciones, la salida PWM tiene apenas una resolución de 2.5%; suficiente para cumplir con la consigna dada.

2.5.2 Interrupción por Pulsador

Dado que el *PIC12F629* posee solo un pin de interrupción externa por hardware se emplean las interrupciones por GPIO para el pulsador. Estas interrupciones se activan por cada cambio de estado en el pin, de manera que se definió que el estado por defecto de la entrada conectada al pulsador (GP4) es a "bajo" o 0 lógico, y que el pulsador introduce un "alto" ó 1 lógico en la misma para poder establecer una rutina de control y evitar que el ciclo de trabajo del PWM por cada flanco de la señal en GP4.

El servicio de manejo de interrupciones verifica el estado de GP4 para determinar si el pulsador ha sido accionado o liberado, y en función del resultado llama a la función *Ajustar_PWM()*. Esta última compara el valor actual del ciclo de trabajo de la salida PWM y lo actualiza según las reglas definidas en la consigna.

La implementación esta funcionalidad puede verse en el código adjunto. La línea 1 lee el registro GPIO y almacenar su valor en una variable temporal, para asegurar que la interrupción se maneje correctamente y se eviten problemas como interrupciones fantasma o no deseadas.

```
1 void __interrupt() isr(void) {
2     if (INTCONbits.GPIF) {      // Interrupción es cambio de estado en
        GPIO
3
4     volatile unsigned char dummy_reg = GPIO;
5     INTCONbits.GPIF = 0;
6
7     if (PULSADOR == 1 && FLAG_PWM_ENABLE == 1)
8         Ajustar_PWM();
9 }
10 ...
11 //CODIGO
12 ...
13 }
14 void Ajustar_PWM (void){
15
16     if (PWM_DUTY_CYCLE < 25)           // 10%
17         PWM_DUTY_CYCLE = 25;
18
19     else if (PWM_DUTY_CYCLE < 64)     // 25%
20         PWM_DUTY_CYCLE = 64;
21
22     else if (PWM_DUTY_CYCLE < 122)    // 50%
23         PWM_DUTY_CYCLE = 122;
24
25     else if (PWM_DUTY_CYCLE < 192)    // 75%
26         PWM_DUTY_CYCLE = 192;
```

```

27
28     else if (PWM_DUTY_CYCLE < 255) // 100%
29         PWM_DUTY_CYCLE = 255;
30
31     else if (PWM_DUTY_CYCLE == 255)
32         PWM_DUTY_CYCLE = 0;
33
34     return;
35 }
```

2.5.3 Reseteo por WatchDog

Por consigna, el sistema debe enviar a la PC el mensaje *0xEE* transcurridos 100 (cien) segundos luego de recibir el último dato por UART y transcurridos otros 20 (veinte) segundos debe reiniciarse si no se recibió otro dato desde la PC. Para esto se emplea el TMR1 en conjunto con el WDT.

El TMR1 y WDT se configuran apenas se recibe un dato desde la PC, en el código de manejo de la interrupción por hardware en GP2. El WDT se configura en 2048 [ms] y el TMR1 se precarga para que se desborde cada 500 [ms]. El desborde de TMR1 controla un contador que emplea una variable global para ser verificada desde *main.c*. Por cada desborde, TMR1 se precarga nuevamente para temporizar otros 500 [ms] y avanza el contador; una vez el contador llega a una cuenta máxima de 200 (correspondiente a 100 segundos) se envía el mensaje *0xEE* a la PC desde el código de manejo de interrupción de TMR1. Si no existen nuevos mensajes desde la PC el contador prosigue su marcha con cada nuevo desborde de TMR1 hasta llegar a 236 (118 segundos) en cuyo momento se deja de alimentar al perro guardián. El perro guardián comienza entonces su conteo de 2048 [ms] y termina reiniciando el sistema a los 2 (dos) minutos de recibido el primer dato desde la PC.

Si en cualquier instante se recibe un nuevo dato, el contador se reinicia y se vuelve a tener 2 minutos hasta el próximo reinicio.

El WDT se alimenta desde *main.c* que controla el estado del contador para deshabilitar, o no, la salida PWM.

```

1 // INTERRUPCIONES
2 void __interrupt() isr(void) {
3     ...
4     // CODIGO
5     ...
6     // INTERRUPCIÓN BIT START UART *****
7     if (INTCONbits.INTF){
8         ...
9         // CODIGO
10        ...
11        if (RECEPCION_UART()){
12            RESPONDER_UART(0x1B);
13            FLAG_PWM_ENABLE = 1;
14            PWM_DUTY_CYCLE = DATO_RECIBIDO_UART;
15        }
16
17        TMR0 = 0;
18        INTCONbits.T0IF = 0;
```

```
19         INTCONbits.T0IE = 0;
20
21         CONTADOR_REINICIOS_WDT = 0;
22         T1CONbits.TMR1CS = 0;
23         T1CONbits.T1CKPS = 0b11;
24         TMR1 = PRECARGA_TMR1_WDT;
25         PIR1bits.TMR1IF = 0;
26         T1CONbits.TMR1ON = 1;
27         PIE1bits.T1IE = 1;
28
29         INTCONbits.GPIF = 0;
30         INTCONbits.INTF = 0;
31
32         INTCONbits.PEIE = 1;
33         INTCONbits.GPIE = 1;
34     }
35 ...
36 // CODIGO
37 ...
38
39 if (PIR1bits.TMR1IF){
40
41     // Cuando se establece la comunicación serie y se recibe un
42     // dato
43     // se resetea CONTADOR_REINICIOS_WDT y este bloque
44     // temporiza 100 seg
45     // para enviar el mensaje a la PC. Luego temporiza otros
46     // 120 seg y
47     // bloquea el reseteo del watchdog pueda volver en el main
48
49     TMR1 = PRECARGA_TMR1_WDT;
50     PIR1bits.TMR1IF = 0;
51
52     if(FLAG_PWM_ENABLE == 1){
53         CONTADOR_REINICIOS_WDT++;
54
55         if (CONTADOR_REINICIOS_WDT == 200){ // 100 segs desde
56             ultima COMM
57             RESPONDER_UART(0xee);
58         }
59     }
60 }
61
62 ...
63 //CODIGO
64 ...
65 void main(void) {
66     ...
67     // CODIGO
68     ...
69
70     while(1){
```

```
71         if(FLAG_PWM_ENABLE == 1){
72             if(PWM_DUTY_CYCLE == 255)
73                 PWM_OUT = 1;
74             else if(Contador_PWM < PWM_DUTY_CYCLE)
75                 PWM_OUT = 1;
76             else
77                 PWM_OUT = 0;
78             if(Contador_PWM < 255)
79                 Contador_PWM+=5;      // Resolución PWM ~ 2.34%
80             else
81                 Contador_PWM = 0;
82         }
83         if (CONTADOR_REINICIOS_WDT < MAX_REINICIOS_WDT)
84             CLRWDT();
85         if(STATUSbits.nTO == 0){
86             FLAG_PWM_ENABLE = 0;
87         }
88     }
89     return;
90 }
```

3 Verificación de software

3.1 Entorno de trabajo

Para el desarrollo del software se empleó el entorno de desarrollo integrado³ de *MPLab X* en su versión gratuita. Por otra parte, para el depurado del código desarrollado, se utilizó el debugger incluido en el mismo IDE de *MPLab* en conjunto con Proteus v8.9 SP2.

3.2 Resultados de simulación en MPLAB

Se utilizó el debugger de *MPLab* como primera instancia de depurado del código. En este, mediante las herramientas: *estímulo*, *analizador lógico*, *Stopwatch* y *Variables*, se ajustó el código referente a la UART para cumplir con los requisitos de temporización impuestos por la comunicación (véase anexo).

A continuación se expone el ejemplo de uso el debugger para la verificación del código propuesto en la sección anterior.

3.2.1 Configuración del estímulo

Utilizando la herramienta estímulo se creó un estímulo sincrónico según se muestra en la imagen 5 que simula, en orden, el envío del dato *0x55* desde la PC hacia el PIC (estímulo aplicado a GP2) y el accionamiento del pulsador que controla el ciclo de trabajo del PWM (estímulo aplicado a GP3).

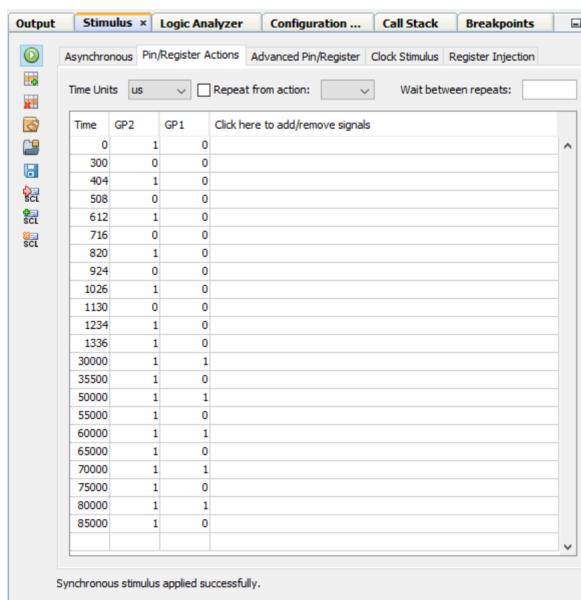


Figure 5: Configuración del estímulo para la simulación en MPLab.

3.2.2 Temporización de UART

Utilizando la herramienta estímulo se creó un estímulo sincrónico según se muestra en la imagen 5 (trazo rojo) que simula el envío del dato *0x55* desde la PC hacia el PIC. Luego, mediante el uso de breakpoints, el stopwatch, el analizador lógico y el monitor de variables de MPLab, se verificó que la temporización del código de recepción es adecuada dado que el dato recibido y almacenado en la variable *DATO_RECIBIDO_UART* se corresponde con el valor que genera el estímulo sincrónico, tal y como se muestra en la imagen 6.

³IDE

	Name	Address	Value
<input checked="" type="checkbox"/>	DATO_RECIBIDO_UART; file:D:\v 0x46	...	0x55
<input checked="" type="checkbox"/>	TMR0	0x1	238
<input checked="" type="checkbox"/>	PWM_DUTY_CYCLE; file:D:\Mis Dc 0x40	...	85
<input checked="" type="checkbox"/>	Contador_PWM; file:D:\Mis Docun 0x42	...	0
<input checked="" type="checkbox"/>	TMR1	0xE	0
<input checked="" type="checkbox"/>	WREG	...	0x00
<input checked="" type="checkbox"/>	DATO_RECIBIDO_UART; file:D:\v 0x46	...	0x55

Figure 6: Dato recibido por UART en MPLab.

De la misma manera, y con las mismas herramientas, se verificó que la temporización del código de envío de datos es correcta tal como se observa en la figura 7 (trazo verde) donde se envía la respuesta *0x1B* a la PC.

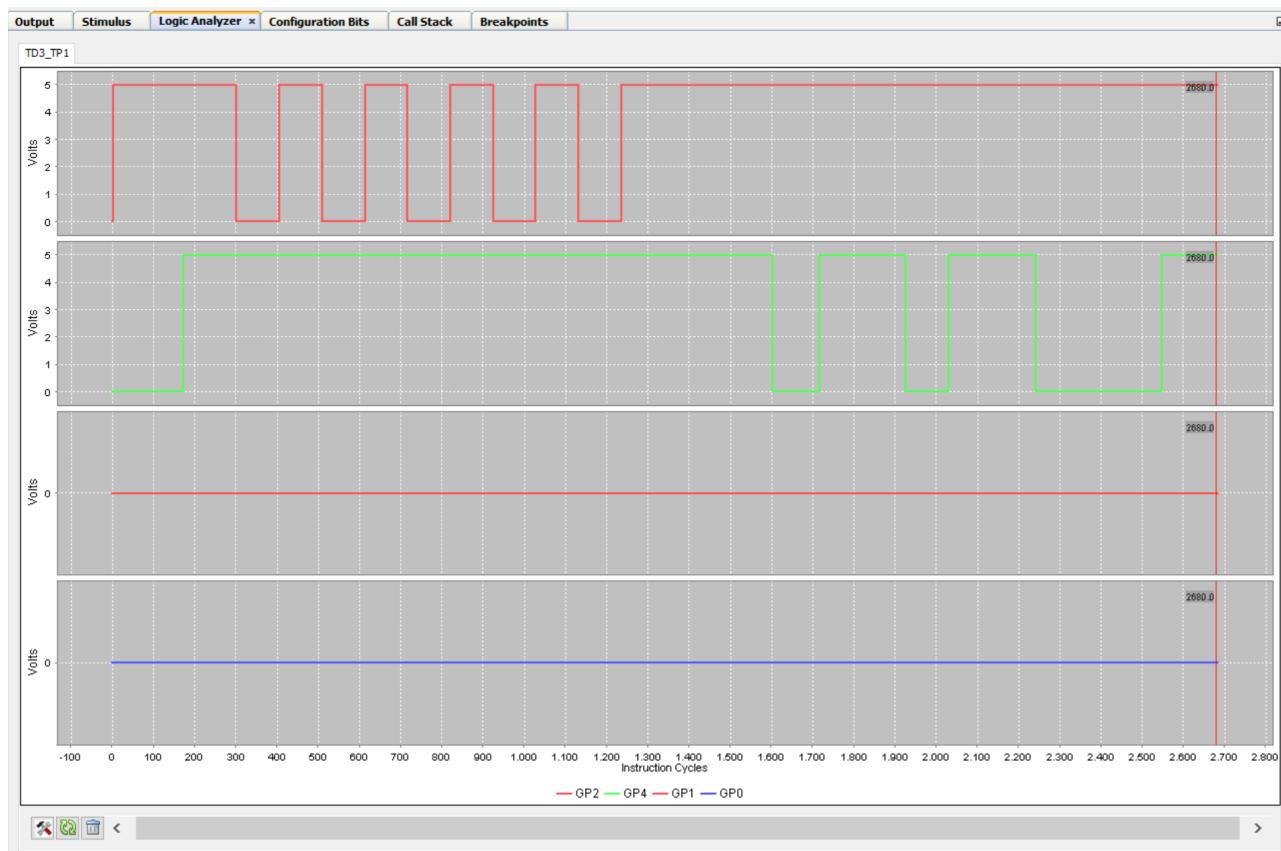


Figure 7: Datos recibido y enviado por UART en MPLab.

Finalmente, en la figura 8 se muestra que el tiempo de bit para el envío y recepción de datos por UART cumple con los requisitos de la sección 01 (el código de recepción se sincroniza para estar desfasado de los cambios de estado de los bits en la trama).

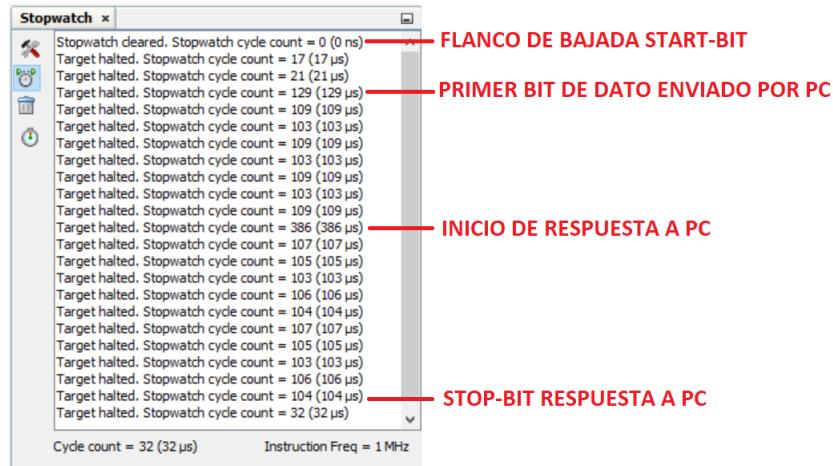


Figure 8: Temporización de la UART.

3.2.3 Salida PWM

Continuando con la simulación se verificó el funcionamiento de la salida PWM. Tal como se explicó en la sección anterior, el PWM se controla en pasos discretos de 2.5% según el valor del dato recibido por UART, según el cual *0x00* corresponde a un ciclo de trabajo del 0%, mientras que *0xFF* indica un 100%.

En las imágenes debajo, imagen 9 y 10, se muestra cómo la recepción y validación del dato enviado por la PC habilita la salida PWM (trazo azul) y configura el ciclo de trabajo según la siguiente ecuación:

$$\text{Duty Cycle} = d = \frac{T_{on}}{T} = \frac{\text{DATO_RECIBIDO_UART}}{255} \quad (1)$$

Conociendo el valor del dato enviado desde la pc *DATO_RECIBIDO_UART* = *0x55* (trazo rojo en las figuras mencionadas), y extrayendo el valor del período del PWM *T* = 3.544 [ms] y del tiempo de encendido *T_{on}* = 1.165 [ms] de las figuras 9 y 10; se verifica que el ciclo de trabajo del 33% en la simulación es correcto de acuerdo a la ecuación 1.

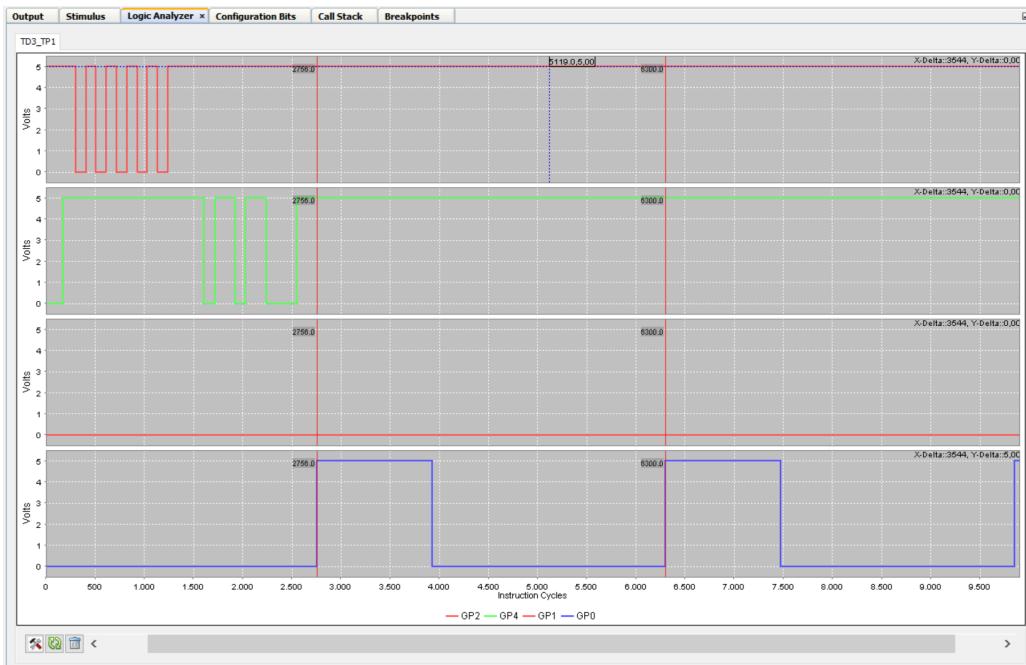


Figure 9: Período de la salida PWM.

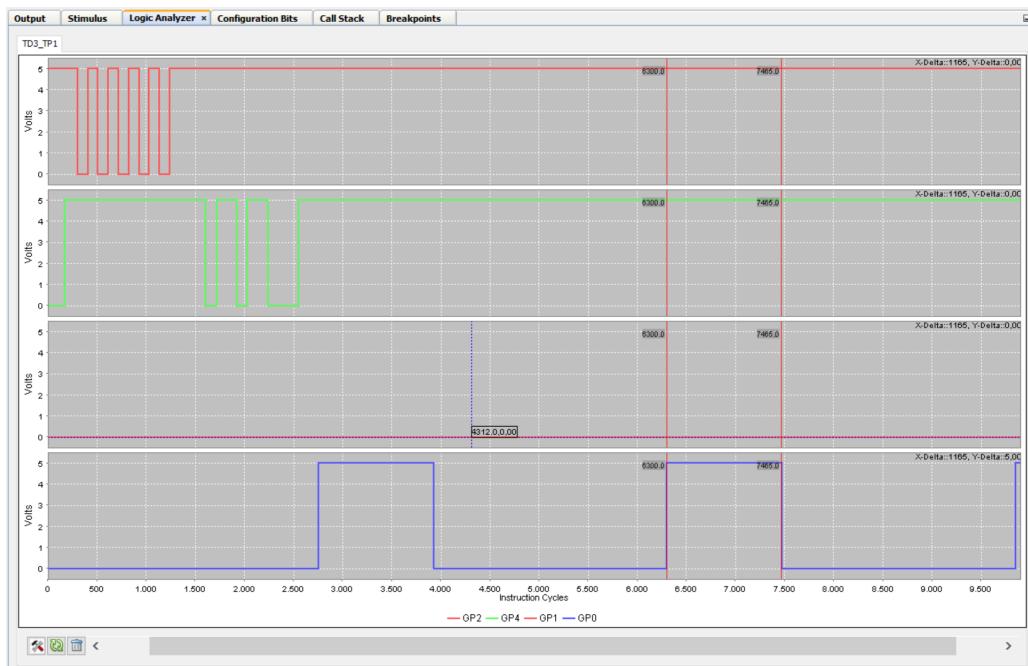


Figure 10: Tiempo de encendido de la salida PWM.

3.2.4 Interrupción por pulsador

Haciendo uso del estímulo expuesto en 5 se verifica que el cambio de estado de la señal asociada a GPIO3, correspondiente al pulsador, modifica el ciclo de trabajo de la salida PWM tal y como se muestra en las figuras 11 y 12. En estas figuras puede verse como en el próximo ciclo de la salida PWM luego de un cambio de estado de bajo a alto en la entrada GPIO3 asociada al pulsador externo (trazo rojo en la figura) el ciclo de trabajo se incrementa, en concreto el efecto es más visible si se analiza como el tiempo de apagado del PWM se reduce mientras el período se mantiene sin cambios respecto de las imágenes 9 y 10, en concreto el ciclo de trabajo se incrementa desde d_0 33% a $d = 50\%$.

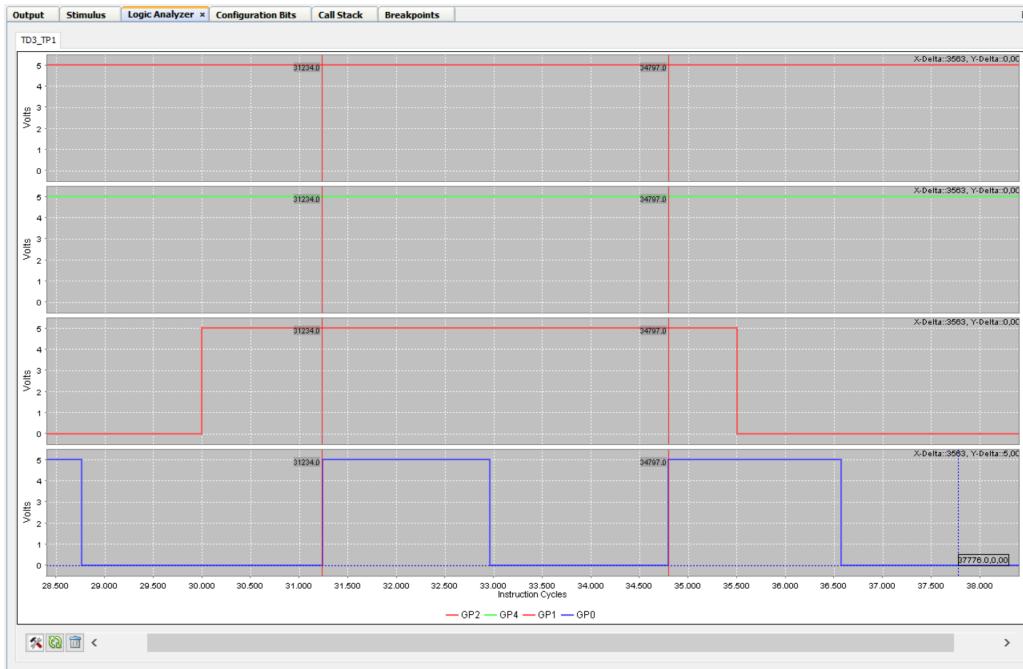


Figure 11: Cambio del ciclo de trabajo del PWM por cambio de estado en GPIO3.

Para la verificación de este resultado se introduce un segundo pulso positivo en la entrada GPIO3 y se observa que el ciclo de trabajo del PWM vuelve a incrementarse pasando de $d_0 = 50\%$ a $d = 75\%$ de acuerdo al requisito expuesto en la sección 01.

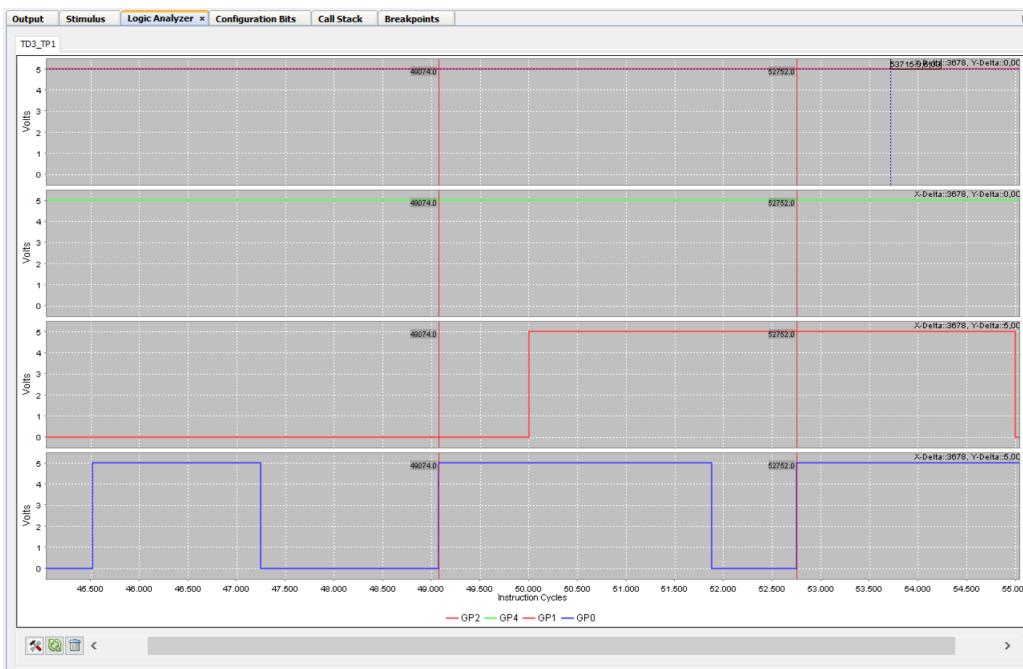


Figure 12: Cambio del ciclo de trabajo del PWM por cambio de estado en GPIO3.

Como comentario adicional, resta destacar que, como el ciclo PWM no está fuertemente temporizado mediante interrupciones, existe un ligero cambio del período cuando se procesa alguna interrupción; este cambio se midió en $\Delta T_{PWM} = 3\%$.

3.2.5 Reinicio por WDT

Por último, en base se verificó el funcionamiento del reinicio por WDT. Como se explicó en la sección anterior, una vez recibido el dato y validado el mismo se comienza una cuenta regresiva controlada por *TMR1* que, ante la ausencia de nuevos comandos enviados desde la PC, cuenta con dos hitos temporales relevantes:

- $t_1 = 100 [s]$: se envía la alerta *0xEE* a la PC que indica la próxima reinicialización a cero del sistema.
- $t_1 = 120 [s]$: se reinicia el sistema a su estado inicial desactivando la salida PWM.

En la imagen 13 se muestran el resultado del stop-watch con las siguientes marcas de tiempo:

1. La finalización de la recepción del primer (y único) dato enviado desde la PC.
2. El primer desborde de *TMR1* para el control del reinicio por WDT.
3. El envío del mensaje *0xEE* a la PC.

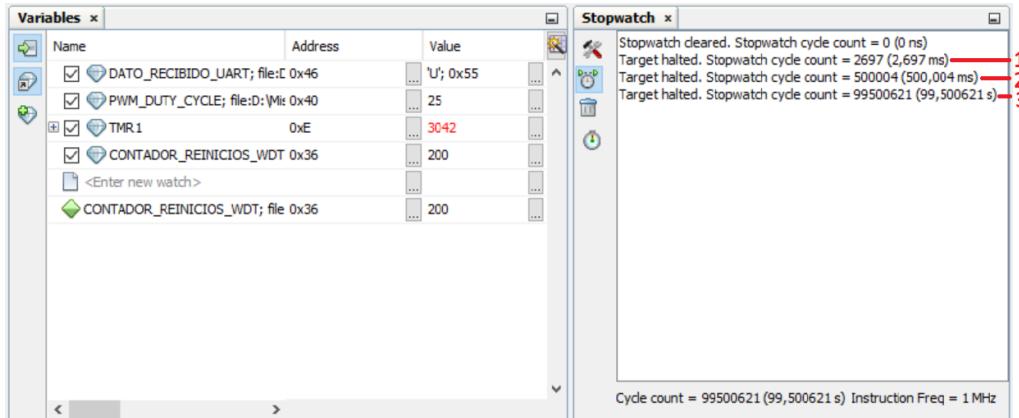


Figure 13: Marcas de tiempo relacionadas al reinicio por WDT.

Según se observa, el desborde de *TMR1* ocurre cada $T_{TMR1} = 500[ms]$ tal cual se había planeado; y el envío del dato *0xEE* a la PC se da a los:

$$t_{0xEE} = T_{TMR1} + t_{stop} = 0.500004[s] + 99.500621[s] = 100.00058[s] \quad (2)$$

Este resultado corresponde a un error relativo inferior al % 0.001 respecto del tiempo de respuesta indicado en la consigna de 100 (cien) segundos. Adicionalmente, puede verse que el contador de reinicios del timer está en 200 (doscientos) indicando que efectivamente se ha mantenido con un período de T_{TMR1} 500[ms]⁴. Por último, en la imagen 27, se muestra que transcurridos otros 20.001353[s] desde t_{0xEE} la bandera que habilita la salida PWM se coloca a cero y se detiene el PWM; en base a todo lo cual el código propuesta queda verificado mediante simulación.

⁴En el presente informe se presentan los datos de forma resumida, en la práctica se verificó que este período se cumple de forma incondicional con un error inferior al 0.1%.

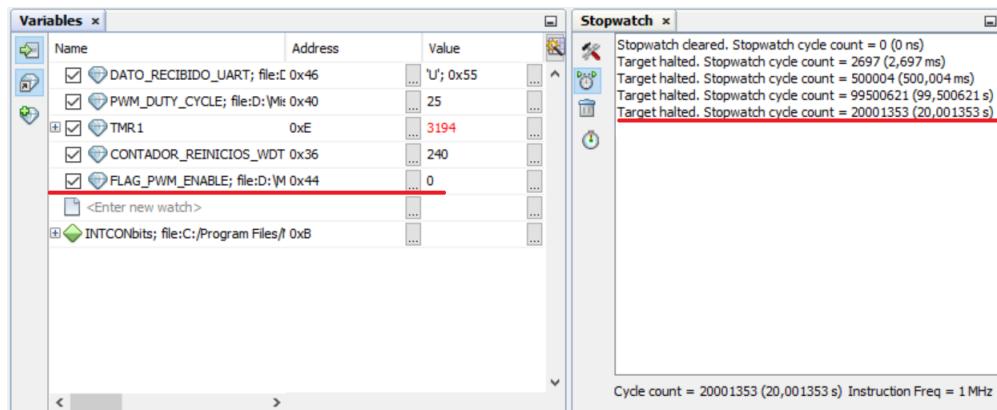


Figure 14: Marcas de tiempo relacionadas al reinicio por WDT.

4 Validación del software

Para la implementación física del sistema se diseñó el circuito de la figura 15; en el mismo se encuentran resaltados los diferentes bloques correspondientes a: la UART, el pulsador y la salida PWM. La implementación práctica se muestra en la figura 16.

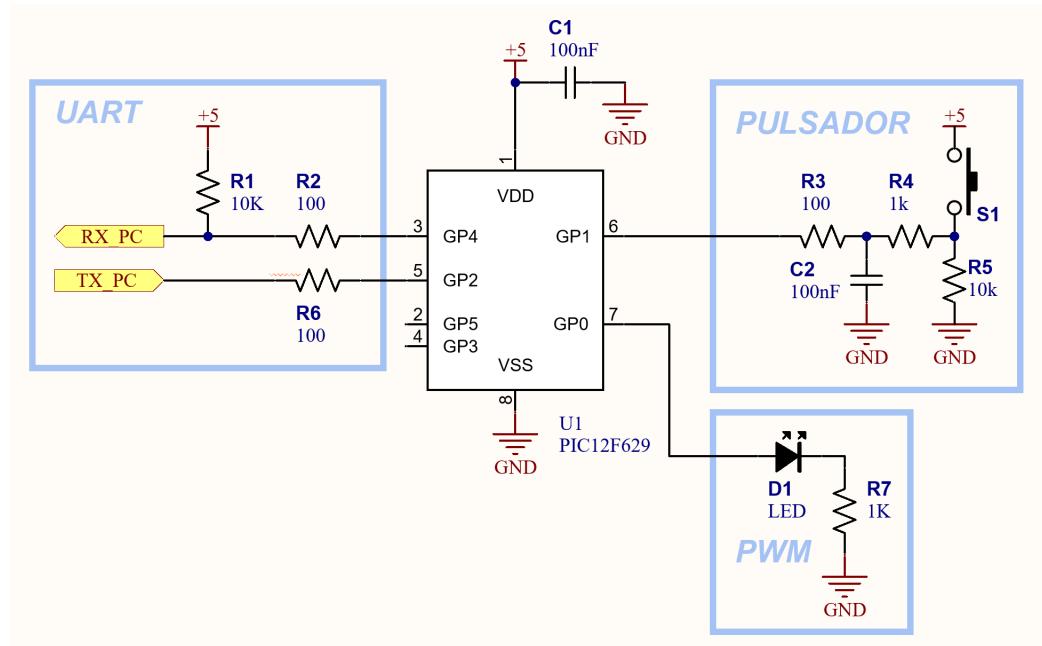


Figure 15: Circuito esquemático diseñado.

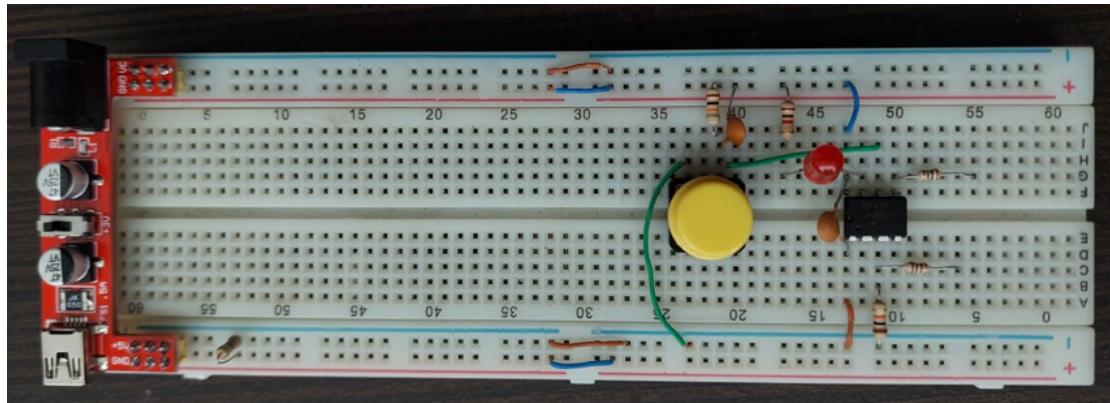


Figure 16: Implementación física del sistema.

4.1 Entorno de trabajo

Para la validación del diseño se utilizó el software RealTerm para generar y controlar una comunicación serie con una PC; para la interfaz física de la comunicación se empleó un adaptador USB a UART con salida TTL como lo es el CP2102. Debajo en las figuras se muestran una captura del software y el hardware empleados, imágenes 17 y 18 respectivamente.

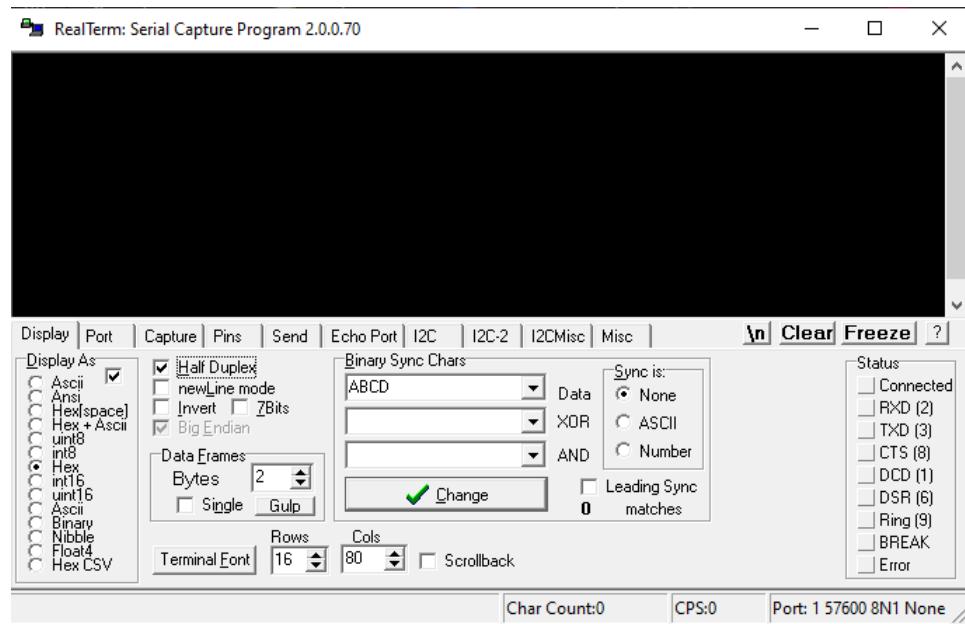


Figure 17: Software Realterm



Figure 18: adaptador USB-TTL cp2102.

4.1.1 Programador

Para la grabación del firmware en el PIC se empleó un programador Pickit 3 mediante el software oficial de Microchip para dicha herramienta. El código empleado para generar el firmware puede consultarse en el anexo A. En las figuras debajo se muestran la IDE de la herramienta de software empleada y una foto del programador Pickit 3, figuras 19 y 20.

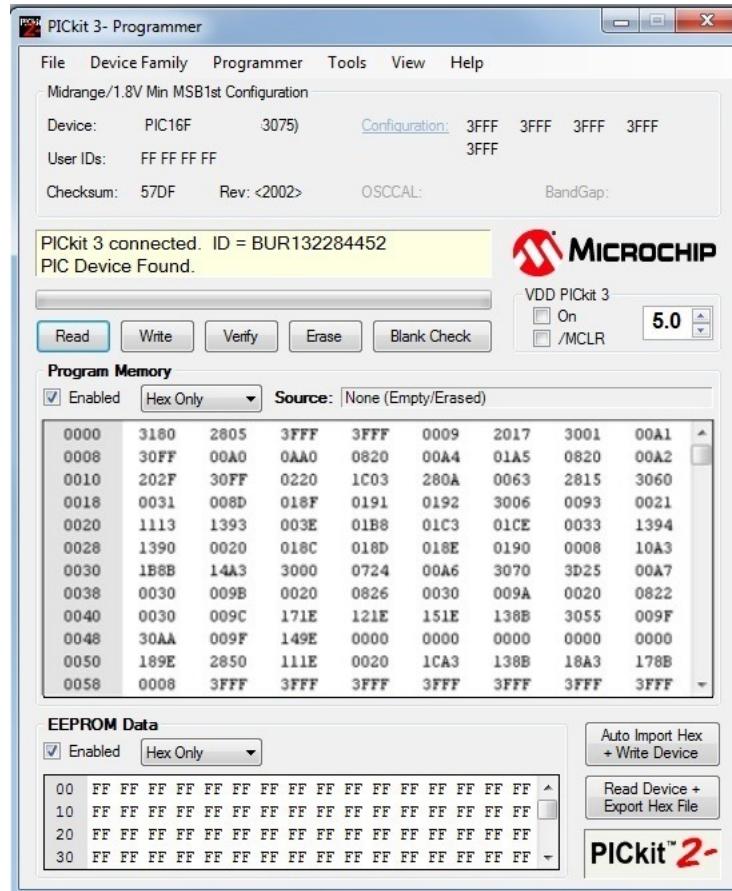


Figure 19: IDE para grabación de firmware con Pickit 3.



Figure 20: Programador Pickit 3.

4.2 Prueba funcional

Para la validación del firmware se realizaron diferentes mediciones sobre el circuito implementado para verificar el funcionamiento de cada aspecto del mismo. En la presente sección se muestran los resultados obtenidos y las formas de onda capturadas empleando un osciloscopio Tektronix modelo TBS1052B.

4.2.1 Comunicación por UART

Se verificó que ante un dato recibido desde la PC el PIC responde correctamente por UART con el mensaje *0x1B*. En la imagen 21 se muestra la entrada RX del PIC (Canal 1 / trazo amarillo) y la salida TX (Canal 2 / Trazo cian).

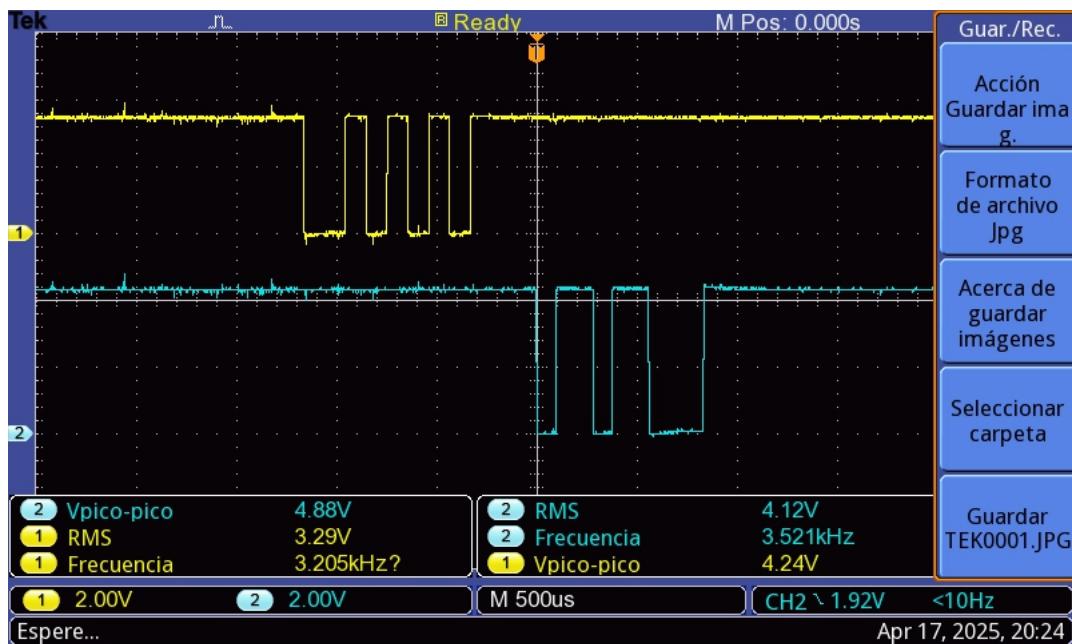


Figure 21: otra respuesta del Pc por tx a otro tipo de estímulo por uart, Rx(0xA1 en hexa).

4.2.2 Control de la salida PWM por UART

Se verificó que el dato recibido de la PC habilita correctamente la salida PWM del sistema. En la imagen 22 se muestra como se recibe el dato 0b01101111 correspondiente al 111 decimal (Canal 1 / trazo amarillo) y el ciclo de trabajo se configura en aproximadamente 42.5% (Canal 2 / Trazo cian).

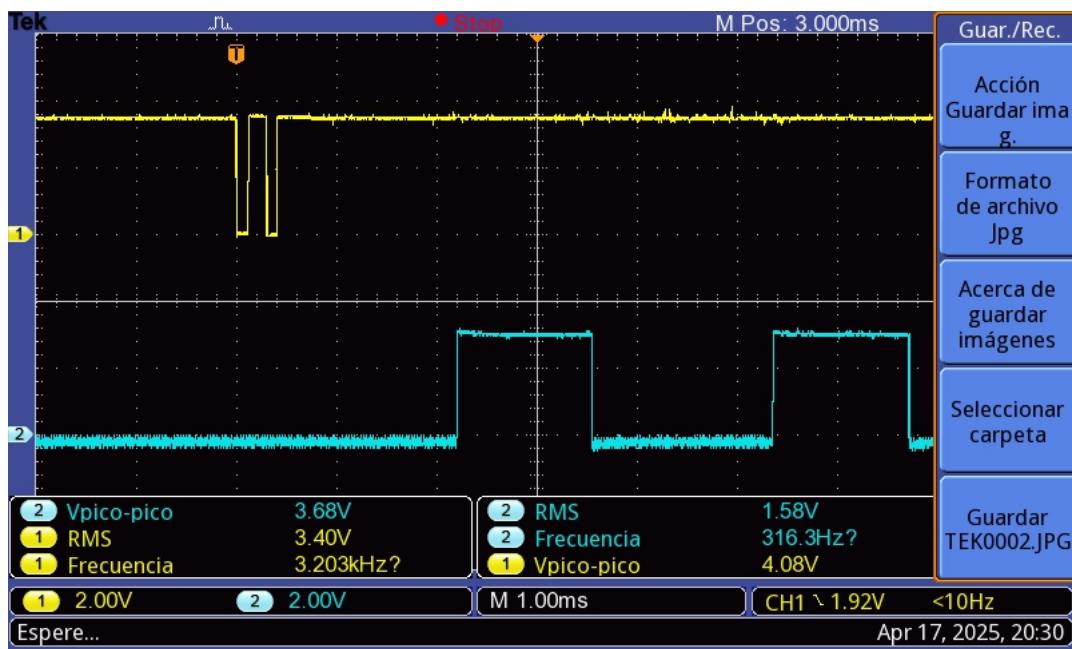


Figure 22: Control de la salida PWM mediante UART.

4.2.3 Control del ciclo de trabajo mediante pulsador

Se verificó como el pulsador efectivamente modifica el ciclo de trabajo de la salida PWM de acuerdo a lo planteado en la consigna. En las imágenes debajo puede verse como el accionamiento del pulsador (Canal 1/ trazo amarillo) modifica el ciclo de trabajo de la salida PWM (Canal 2 / trazo cian), respetando la secuencia: 0%, 10%, 25%, 50%, 75%, 100%.

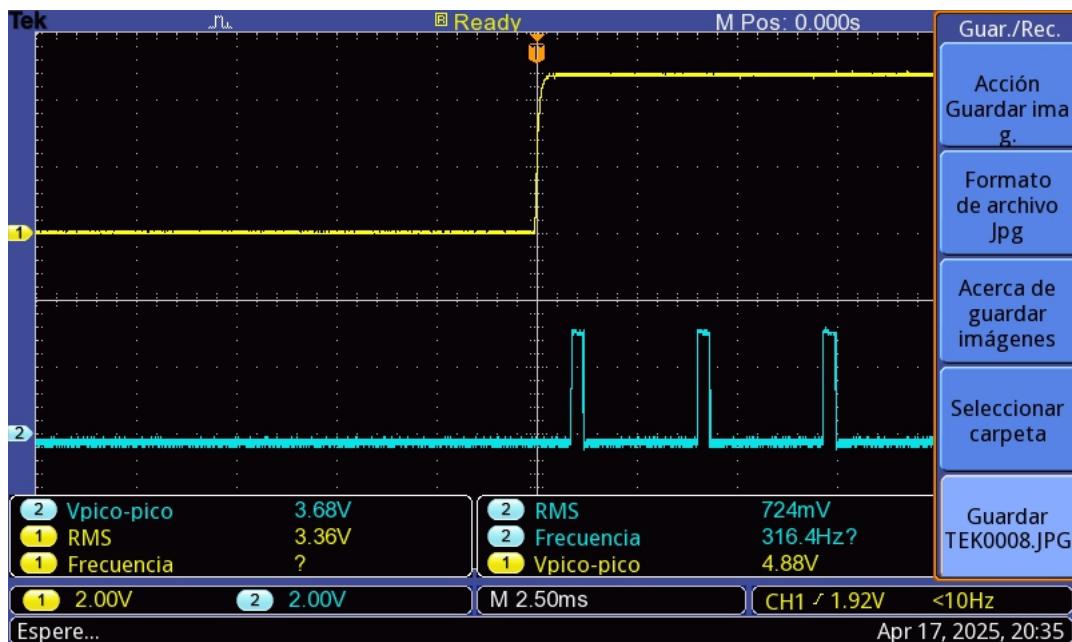


Figure 23: Modificación del ciclo de trabajo del PWM (Canal 2) mediante pulsador (Canal 1), 0% - 10%.

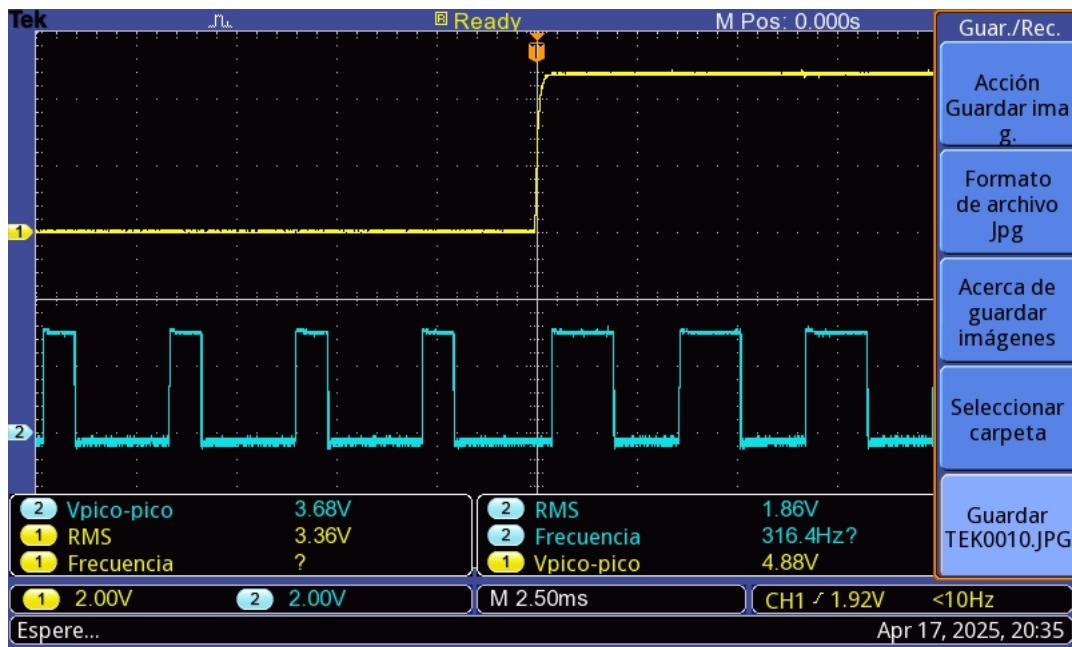


Figure 24: Modificación del ciclo de trabajo del PWM (Canal 2) mediante pulsador (Canal 1), 25% - 50%.

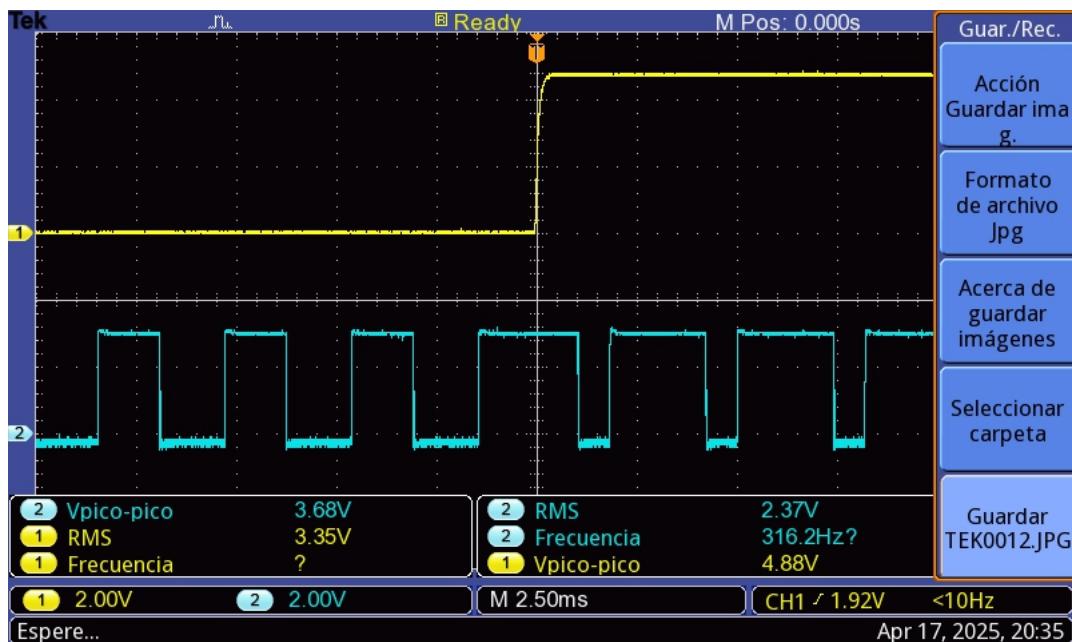


Figure 25: Modificación del ciclo de trabajo del PWM (Canal 2) mediante pulsador (Canal 1), 50% - 75%.

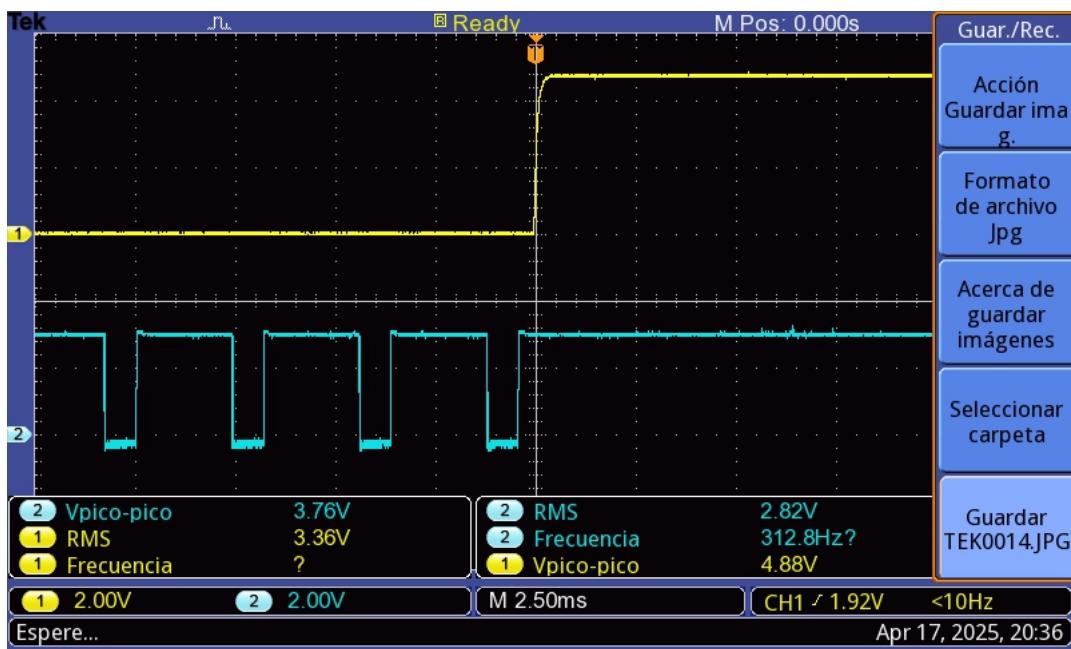


Figure 26: Modificación del ciclo de trabajo del PWM (Canal 2) mediante pulsador (Canal 1), 75% - 100%.

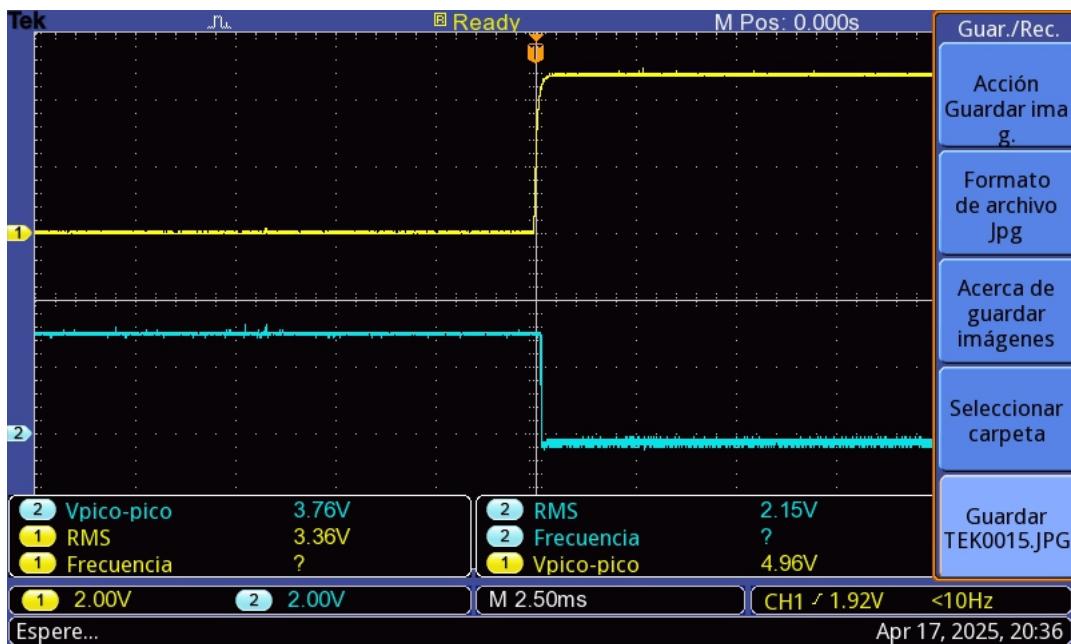


Figure 27: Modificación del ciclo de trabajo del PWM (Canal 2) mediante pulsador (Canal 1), 100% - 0%.

4.2.4 Reinicio del sistema por WDT

Se muestra el envío del mensaje de alerta a la PC (0xEE) y la consecuente desactivación de la salida PWM. El trazo amarillo (CH1) es la salida TX del PIC y el azul corresponde a la salida PWM.

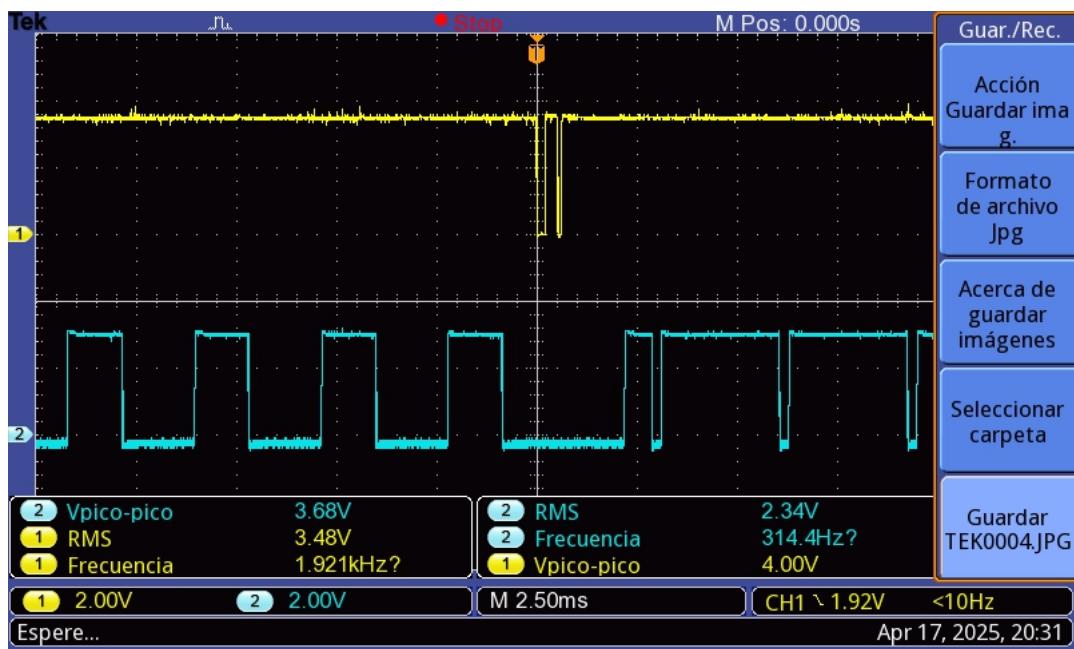


Figure 28: Envío del mensaje de alerta *0xEE* a la PC transcurridos los 100 segundos desde el inicio, y posterior desactivación del PWM.

5 Conclusiones

El presente trabajo práctico permitió consolidar un acercamiento práctico y efectivo a la programación de microcontroladores PIC. A través de la comunicación UART a 9600 baudios, se logró un intercambio bidireccional de datos entre la PC y el microcontrolador, validando la recepción de comandos para ajustar la intensidad del LED y enviando respuestas predefinidas, como el ACK (0x1B) y la alerta de reinicio (0xEE). La temporización crítica de este proceso se verificó mediante simulaciones en MPLAB y pruebas físicas con herramientas como el osciloscopio, confirmando la precisión en la sincronización de bits del protocolo implementado.

El control del ciclo de trabajo del PWM, esencial para modular la intensidad del LED, se gestionó mediante interrupciones generadas por un pulsador externo. Este mecanismo permitió transitar cíclicamente entre los niveles predefinidos (0%, 10%, 25%, 50%, 75%, 100%), demostrando una respuesta coherente con las especificaciones. Las mediciones realizadas con el osciloscopio evidenciaron un ajuste preciso de los tiempos de encendido y apagado, aunque se observaron variaciones mínimas en el período del PWM debido a la concurrencia de interrupciones.

La implementación del Watchdog Timer (WDT) reforzó la confiabilidad del sistema al garantizar un reinicio automático tras 120 segundos de inactividad en la comunicación UART. Este proceso, respaldado por el temporizador TMR1 y un contador de eventos, incluyó el envío de una señal de alerta (0xEE) a los 100 segundos, cumpliendo con el requisito de notificación previa. La validación de este módulo, tanto en simulaciones como en el hardware real, destacó su eficacia para recuperar el sistema ante fallos o bloqueos.

Finalmente, la integración de herramientas como MPLAB, Proteus y el programador Pickit 3 facilitó un ciclo de desarrollo riguroso, desde la depuración del código hasta la grabación del mismo en el PIC12F629. Las pruebas funcionales en el circuito físico, complementadas con instrumentación de medición, confirmaron la operatividad de todas las funcionalidades y la coherencia entre el diseño teórico y los resultados prácticos. En conjunto, el trabajo no solo cumplió con las exigencias técnicas de la consigna, sino que también demostró la viabilidad de integrar comunicación, gestión de interrupciones manejo de fallos en sistemas embebidos, ofreciendo una base práctica para aplicaciones similares en el futuro.

6 Anexo A: Código para PIC12F629

A continuación se expone el código de implementación junto con los tres archivos de cabecera desarrollados:

- configuration_bits.h
- funciones.h
- variables_definiciones.h
- main.c (programa principal)

configuration_bits.h:

```
1 // PIC12F629 Configuration Bit Settings
2
3 #pragma config FOSC = INTRCIO      // Selección oscilador interno
4 #pragma config WDTE = ON          // Watchdog deshabilitado
5 #pragma config PWRTE = ON         // Power-Up Timer habilitado
6 #pragma config MCLRE = OFF        // MCLR pin deshabilitado
7 #pragma config BOREN = ON         // Brown-out Detect habilitado (UVLO)
8 #pragma config CP = OFF           // Code Protection deshabilitado
9 #pragma config CPD = OFF          // Data Code Protection deshabilitado (protección
    de la memoria EEPROM para lectura con HW programador)
```

funciones.h:

```
1 // FUNCIONES
2
3 int RECEPCION_UART (void);                                // COMM. SERIE, RECEPCIÓN, 9600bps,
   8N1
4 void RESPONDER_UART (unsigned char RESPUESTA);           // COMM. SERIE, ENVÍO, 9600bps, 8N1
5 void set_timer_interrupt (unsigned char );                // HABILITA INTERRUPCIÓN POR TMR0 Y
   PRECARGA EL REGISTRO CORRESPONDIENTE
6 void Ajustar_PWM (void);                                  // CAMBIO DE CICLO DE TRABAJO
   MEDIANTE INTERRUPCIÓN POR PULSADOR
7 void LED_BLINK (int, int);                               // PARPADEO DE LED PARA
   INDICACIONES VISUALES
```

variables_definiciones.h (variables globales):

```
1 // VARIABLES GLOBALES PIC _____
2 #define _XTAL_FREQ 4000000          // Oscilador interno 12F629, 4MHz
3 #define PWM_OUT GPIObits.GP0        // Declaración de pines
4 #define PULSADOR GPIObits.GP1       // Declaración de pines
5 #define UART_RX GPIObits.GP2        // Declaración de pines
6 #define UART_TX GPIObits.GP4        // Declaración de pines
7
8
9 // VARIABLES GLOBALES PROGRAMA _____
10
11
12 // PWM _____
13
14 const unsigned char PRECARGA_TMR0_PWM = 185;      // Valor de precarga = 255 -
    Período del PWM en segundos
15
16 volatile unsigned int FLAG_PWM_ENABLE = 0;         // ON/OFF de PWM
17
18 volatile unsigned int Contador_PWM = 0;             // Variable de conteo para pwm
19 volatile unsigned int PWM_DUTY_CYCLE = 0;           // Variable de manejo pwm
20 const int NIVELES_PWM = 50;                         // Cantidad de niveles del PWM.
    Resolución 2.5%
21 volatile unsigned int FLAG_REFRESCAR_PWM;           // Refresco del PWM cada interrupci
    ón por timer 0
22
23
24 // UART _____
25 volatile char DATO_RECIBIDO_UART = 0x00;           // Respuesta UART
26 const char DATO_ENVIADO_UART = 'a';                 // Respuesta UART
27
28 const int UART_PERIOD_HALF = 52;                    // Valor de precarga de timer 0
    para delay de T/2 en comunicación serie 9600bps, 8N1
29 const int UART_PERIOD_FULL = 165;                   // Valor de precarga de timer 0
    para delay de T/2 en comunicación serie 9600bps, 8N1 (ideal 150; se compensa el
    tiempo de instrucción)
30
31 volatile unsigned int FLAG_UART = 0;                // Detección de bit de start
32 volatile unsigned int FLAG RECEPCION = 0;           // Validación de recepción
33 volatile unsigned int DESBORDE_TIMER0 = 0;           // Validación de recepción
34
35
36 // INTERCEPCIÓN DEL WDT _____
37 const long PRECARGA_TMR1_WDT = 3036;               // Precarga TMR1, 16 bits
38 const int MAX_REINICIOS_WDT = 236;                 // Variable para temporización de
    reinicio
39 volatile int CONTADOR_REINICIOS_WDT = 0;           // Variable para temporización de
    reinicio
40 volatile unsigned int WDT_RESET = 0;                 // Reseteo por desborde de WDT
```

main.c:

```
1 #include <xc.h>
2 #include <pic12f629.h>
3 #include "pic_config.h"
4 #include "funciones.h"
5 #include "variables_definiciones.h"

6
7 // INTERRUPCIONES _____
8 void __interrupt() isr(void) {
9     // INTERRUPCIÓN POR TMR0 *****
10    if (INTCONbits.T0IF) {
11        TMR0 = 0;
12        INTCONbits.T0IF = 0;
13    }
14    // INTERRUPCIÓN BIT START UART *****
15    if (INTCONbits.INTF){
16        CLRWDT();
17        OPTION_REGbits.PSA = 1;           // Asigna el prescaler al WDT
18        OPTION_REGbits.PS = 0b111;       // Prescaler a 1:32 (T=2048ms)

19
20        TMR0 = 0;
21        INTCONbits.T0IF = 0;
22        INTCONbits.T0IE = 1;

23
24        INTCONbits.GPIF = 0;
25        INTCONbits.GPIE = 0;

26
27        FLAG_PWM_ENABLE = 0;
28        PIR1bits.TMR1IF = 0;
29        INTCONbits.PEIE = 0;           // Interrupciones periféricas

30
31        if (RECEPCION_UART()){ // Verifica validez del dato
32            RESPONDER_UART(0x1B);
33            FLAG_PWM_ENABLE = 1;
34            PWM_DUTY_CYCLE = DATO_RECIBIDO_UART; // RESOLUCIÓN PWM ~2.5%
35        }

36
37        TMR0 = 0;
38        INTCONbits.T0IF = 0;
39        INTCONbits.T0IE = 0;

40
41        CONTADOR_REINICIOS_WDT = 0;
42        T1CONbits.TMR1CS = 0;          // Configuración TMR1
43        T1CONbits.T1CKPS = 0b11;      // prescaler a 1:8
44        TMR1 = PRECARGA_TMR1_WDT;    // Precarga TMR1
45        PIR1bits.TMR1IF = 0;
46        T1CONbits.TMR1ON = 1;
47        PIE1bits.T1IE = 1;

48
49        INTCONbits.GPIF = 0;
50        INTCONbits.INTF = 0;          // Limpieza de bandera

51
52        INTCONbits.PEIE = 1;          // Interrupciones periféricas
53        INTCONbits.GPIE = 1;          // Interrupción por pulsador.

54 }
```

```

55 // INTERRUPCIÓN TMR1 (WDT RESET) ****
56 if (PIR1bits.TMR1IF){
57
58     TMR1 = PRECARGA_TMR1_WDT ;
59     PIR1bits.TMR1IF = 0;
60
61     if(FLAG_PWM_ENABLE == 1){
62         CONTADOR_REINICIOS_WDT++;
63
64         if (CONTADOR_REINICIOS_WDT == 200){ // 100 segs desde ultima COMM
65             RESPONDER_UART(0xee);
66         }
67     }
68 }
69
70 else // Redundancia en desactivación de la interrupción
71     PIE1bits.TMR1IE = 0;
72 }
73
74 // INTERRUPCIÓN POR HW EN GP4 (PULSADOR) ****
75
76 if (INTCONbits.GPIF) { // Interrupción es cambio de estado en GPIO
77
78     volatile unsigned char dummy_reg = GPIO;
79     INTCONbits.GPIF = 0;
80
81     if (PULSADOR == 1 && FLAG_PWM_ENABLE == 1)
82         Ajustar_PWM();
83 }
84 }
85
86 // RECEPCIÓN POR UART -----
87 //Esta función guarda en la variable global DATO_RECIBIDO_UART el dato
88 //recibido por comunicación serie en el pin UART_RX
89
90 int RECEPCION_UART (void){
91
92     int desplazamiento_bit = 0;
93     int RECEPCION_EN_PROCESO = 1;
94     int RESULTADO_COMM = 0;
95     unsigned char CARACTER_RECIBIDO_UART = 0x00;
96
97     if(UART_RX == 0){ // Validación del start-bit
98         TMRO = UART_PERIOD_FULL;
99         while(RECEPCION_EN_PROCESO == 1)
100     {
101         if(INTCONbits.T0IF){
102             TMRO = UART_PERIOD_FULL;
103             INTCONbits.T0IF = 0;
104             if(desplazamiento_bit <8)
105                 CARACTER_RECIBIDO_UART |= (UART_RX << desplazamiento_bit);
106
107             else{
108                 if (UART_RX == 1) // Validación del carácter recibido
109                     RESULTADO_COMM = 1;
110             }
111         }
112     }
113 }
114 }
```

```
111             RECEPCION_EN_PROCESO = 0;
112         }
113     desplazamiento_bit++;
114 }
115 }
116
117 __delay_us(UART_PERIOD_FULL);
118 if((RESULTADO_COMM == 1) & (UART_RX == 1)) // Se valida el dato recibido
119     DATO_RECIBIDO_UART= CARACTER_RECIBIDO_UART;
120 else
121     RESULTADO_COMM = 0;
122 }
123 return (RESULTADO_COMM);
124 }
125
126
127 // RESPUESTA POR UART
128 //Esta función envía por pin UART_TX el carácter RESPUESTA según una
129 //comunicación serie 9600 bps, 8N1
130
131 void RESPONDER_UART(unsigned char RESPUESTA)
132 {
133     int bit = 0;
134     int ENVIO_EN_PROCESO = 1;           // FLAG PARA CONTROL DE ENVÍO
135
136     TMR0 = UART_PERIOD_FULL;
137
138     while (ENVIO_EN_PROCESO && bit <= 10) {
139         while (!INTCONbits.T0IF);      // Bucle de espera bloqueante
140
141         TMR0 = UART_PERIOD_FULL;
142         INTCONbits.T0IF = 0;
143
144         switch (bit) {
145             case 0: // Start bit
146                 UART_TX = 0;
147                 break;
148
149             case 1:
150             case 2:
151             case 3:
152             case 4:
153             case 5:
154             case 6:
155             case 7:
156             case 8:
157                 // En cada iteración se carga el bit 0 de la variable RESPUESTA
158                 // y se actualiza el valor de esa variable desplazando un bit
159                 // a la derecha, de esta manera todas las operaciones tienen
160                 // el mismo delay y el timing es más preciso
161                 UART_TX = RESPUESTA & 1;
162                 RESPUESTA >>=1;
163                 break;
164
165             case 9:      // Stop bit
166                 UART_TX = 1;
```

```
167         break;
168
169     case 10: // Termina el envío
170         UART_TX = 1;
171         ENVIO_EN_PROCESO = 0;
172         break;
173
174     default:
175         break;
176     }
177     bit++;
178 }
179
180 return;
181 }
182
183
184
185 // AJUSTAR PWM POR PULSADOR -----
186
187 void Ajustar_PWM (void){
188
189     if (PWM_DUTY_CYCLE < 25)           // 10%
190         PWM_DUTY_CYCLE = 25;
191
192     else if (PWM_DUTY_CYCLE < 64)      // 25%
193         PWM_DUTY_CYCLE = 64;
194
195     else if (PWM_DUTY_CYCLE < 122)     // 50%
196         PWM_DUTY_CYCLE = 122;
197
198     else if (PWM_DUTY_CYCLE < 192)     // 75%
199         PWM_DUTY_CYCLE = 192;
200
201     else if (PWM_DUTY_CYCLE < 255)    // 100%
202         PWM_DUTY_CYCLE = 255;
203
204     else if (PWM_DUTY_CYCLE == 255)
205         PWM_DUTY_CYCLE = 0;
206
207 return;
208 }
209
210
211 // CONFIGURACIÓN DE INTERRUPCIONES POR TMR0 -----
212 void set_timer_interrupt (unsigned char VALOR_PRECARGA_TIMER){
213
214     TMR0 = VALOR_PRECARGA_TIMER;      // Precarga del registro
215     INTCONbits.T0IF = 0;             // Reset bandera
216     INTCONbits.T0IE = 1;             // Habilitar interrupción por TMR0
217     return;
218 }
219
220
221 // PARPADEO DEL LED PARA INDICACIONES VISUALES -----
222 void LED_BLINK(int N, int periodo){
```

```
223
224     for (int i=0; i<N; i++){
225         PWM_OUT = ~PWM_OUT ;
226         for(int j=0; j<periodo; j++){
227             __delay_ms(1);
228         }
229     }
230 }
231
232
233 // ***** MAIN *****
234
235 void main(void) {
236     // CONFIGURACIÓN GPIO
237     TRISIObits.TRISIO0=0;           // GP0 = Salida PWM
238     TRISIObits.TRISIO1=1;          // GP1 = Entrada Pulsador
239     TRISIObits.TRISIO2=1;          // GP2 = UART RX
240     TRISIObits.TRISIO4=0;          // GP4 = UART TX
241
242     CMCON = 0x07;                 // Desactivar comparadores
243     PWM_OUT = 0;                  // Inicialización de GPIO PWM
244     UART_TX = 1;                  // Inicialización de GPIO UART
245
246
247     // CONFIGURACIÓN TIMER 0
248     // TMR0 controla el PWM
249     OPTION_REGbits.T0CS = 0;       // CLK interno para TMR0. T=4/fosc
250     TMR0 = 0;                     // Inicializo timer
251
252     // CONFIGURACIÓN WDT Y PRESCALER
253     OPTION_REGbits.PSA = 1;        // Asigna el prescaler al WDT
254     OPTION_REGbits.PS = 0b111;     // Prescaler a 1:32 (T=576ms)
255
256
257     // INTERRUPCIONES
258     INTCONbits.INTE = 1;           // Interrupción externa por GP2 (UART RX)
259     OPTION_REGbits.INTEDG = 0;      // Interrupción en flanco de bajada en GP2 (UART RX)
260     )
261     INTCONbits.INTF = 0;           // Inicialización de bandera
262
263     IOCbis.IOC1 = 1;              // Interrupción por cambio en GP1 (PULSADOR)
264
265     INTCONbits.T0IE = 0;           // Inicialmente deshabilitado (se habilita en la
266                                   función)
267
268     INTCONbits.GIE = 1;            // Global Interrupt Enable. Permite atender
269                                   las
270     INTCONbits.GPIE = 0;           // interrupción por cambio en GPIOs deshabilitada
271
272     // VARIABLES
273     int Contador_PWM = 0;
274
275     CLRWDT();
276
277     // Main
278     while(1){
```

```
276
277     if(FLAG_PWM_ENABLE == 1){
278
279         if(PWM_DUTY_CYCLE == 255)
280             PWM_OUT = 1;
281
282         else if(Contador_PWM < PWM_DUTY_CYCLE)
283             PWM_OUT = 1;
284         else
285             PWM_OUT = 0;
286
287         if(Contador_PWM < 255)
288             Contador_PWM+=5;      // Resolución PWM ~ 2.34%
289         else
290             Contador_PWM = 0;
291     }
292
293     if (CONTADOR_REINICIOS_WDT < MAX_REINICIOS_WDT)
294         CLRWDT();
295
296     if(STATUSbits.nT0 == 0){
297         FLAG_PWM_ENABLE = 0;
298     }
299 }
300
301 }
```

7 Anexo B: Temporización de UART

A continuación se expone el cálculo de período de la comunicación serie; este tiempo es utilizado en el código expuesto en el Anexo A para sincronizar la recepción y envío de datos por puerto serie a la PC.

Dado que se pide una velocidad de 9600 baudios por segundo, el período de bit resulta:

$$T_{\text{bit}} = \frac{1}{\text{Velocidad}} = \frac{1}{9600 \text{ bps}} = 0.00010416 \text{ s} = 104.16 \mu\text{s} \quad (3)$$

La figura debajo, figura 29, se muestra la trama de una comunicación serie 8N1.

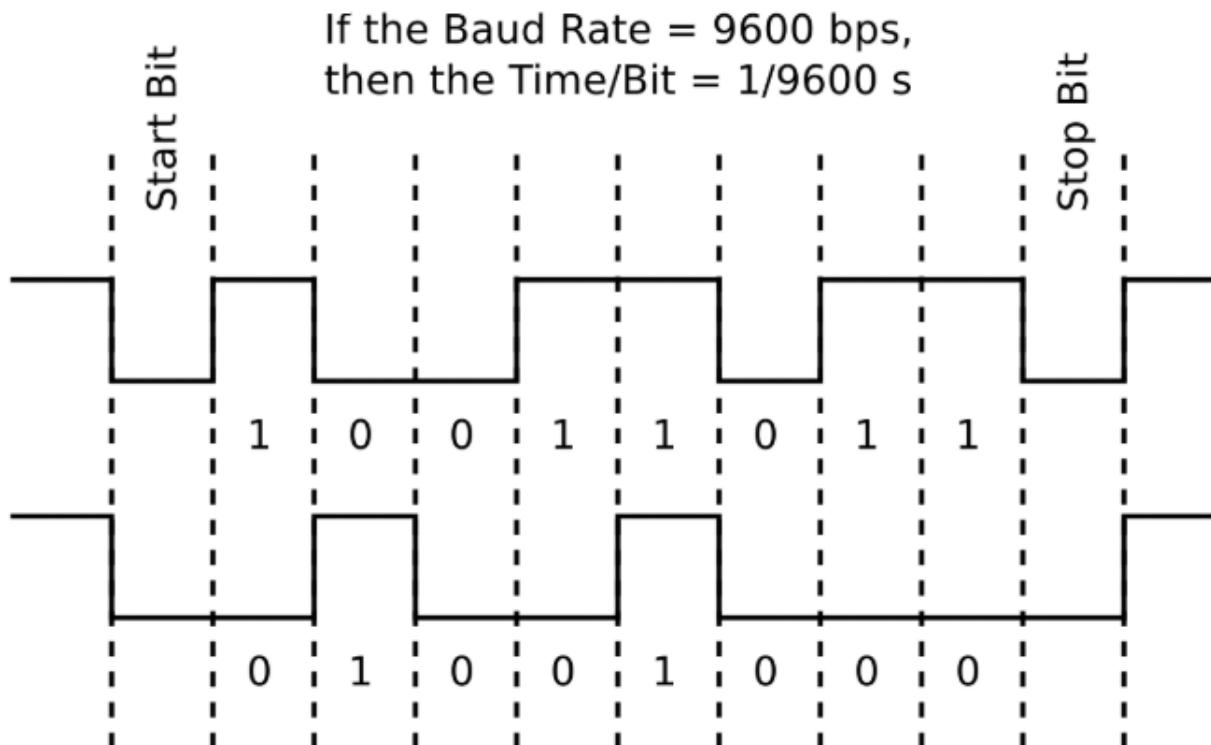


Figure 29: Diagrama de una comunicación serie 8N1.