SISTEMAS EMPOTRADOS

PRACTICA 4.

Ignacio Ortega y Rocío Ruiz

Universidad Antonio de Nebrija

ÍNDICE

Mar, 27 oct. 2020

PRACTICA 4

En esta práctica hemos aprendido el uso de dos nuevos módulos, Pulse Width Module (PWM) e Input Capture (IC).

PWM es un tipo de modulación usado por muchas aplicaciones para variar el ciclo de trabajo de una máquina. Para implementar esto no se necesitaría un módulo hardware concreto, pero sería mucho mas ineficiente debido a los delays que se generan. En aplicaciones que requieren precisión es necesario el uso de este módulo.

IC es el receptor capaz de medir lo que dura el ciclo de trabajo de la señal. Para ello hace uso de un Timer. También podría hacerse sin usar un hardware específico, pero esto requeriría que el programa estuviera todo el rato pendiente de la señal que entra (polling).

En aplicaciones muy demandantes, o que necesitan mucha precisión, es recomendable hacer uso de estos módulos.

Enunciado

Se pide realizar un software que cumpla los siguientes objetivos:

- Módulos IC y OC configurados.
- Configurar el *Handler* para cada ISR de cada módulo IC.
- A través de la se deberá imprimir, al ritmo de 200ms, el tiempo medido de cada señal (*Duty Cycle*).
- Mediante Uart ISR Rx, se leerán hasta 4 teclas, que controlan el ancho de pulso variable para cada señal. (2 teclas para aumentar y disminuir el duty por cada señal PWM).

Configuración Output Compare

Output Compare (OC) es el módulo equivalente a PWM en dsPIC33fj32mc204. El primer paso es configurar este módulo para después gestionar sus interrupciones y comportamiento. Hay que configurar dos módulos OC para generar dos señales.

Hemos establecido los pines C2 y C3 para el OC1 y OC2 respectivamente. Para ello, los definimos como salida y remapeamos como salida el módulo OC1 con el pin remapeable asignado en el microcontrolador en Proteus.

Para configurar correctamente el OC hay que inicializar el subregistro OCM a cero. Este registro nos indica el modo de trabajo del OC. También configuramos el OC para que no se detenga si la CPU entra en modo reposo e indicamos que trabajamos con el Timer 2.

El módulo de OC tiene dos buffers. Uno primario de solo lectura y otro secundario para evitar posibles *glitch*. Tenemos que indicar el ciclo de trabajo de cada módulo OC. Para el preescaler más bajo, 1, y una frecuencia de CPU de 2MHz, si queremos un ciclo de trabajo de 1ms, el TMR será de 2000. Bueno, más precisamente 1999.

El segundo módulo lo hemos configurado para un ciclo de trabajo de 2ms con un TMR de 3999. Para alterar los ciclos de trabajos hay que modificar el segundo buffer OCxRS.

```
void output_compare_config(void)
{
    //Configurar pines con los que vamos a trabajar (RP:de remapeable en proteus)
    TRISCbits.TRISC2 = 0;   //Pin configurado como salida
    TRISCbits.TRISC3 = 0;   //Pin configurado como salida
    RPOR9bits.RP18R = 0x12; //Pin RC2 conectado al modulo OC1 subreg:RP18
    RPOR9bits.RP19R = 0x13; //Pin RC3 conectado al modulo OC2 subreg:RP19

    //Configurar PWM OC1
    OC1CONbits.OCM    = 0;
    OC1CONbits.OCSIDL = 0;  // el OC se detiene si la CPU entra en modo reposo
    OC1CONbits.OCFLT  = 0;
    OC1CONbits.OCTSEL = 0;  // Trabajamos el OC con Timer2
    OC1CONbits.OCM    = 6;  // Modo PWM sin pin de fallo

    //Configuro duty OC1
    //Duty OC1 = (Ton/T) = (1/5)*100 = 20% de trabajo ------------ Ton = el duty
    OC1R    = 2000;     //Ciclo de trabajo8(duty) 1ms con prescaler=1 y Fcpu= 2MHz. //Este es de solo lectura
    OC1RS   = 2000;     //Igual porque es un segundo buffer para evitar glitch. Este es el que se modifica

    //Configurar PWM OC2
    OC2CONbits.OCM    = 0;
    OC2CONbits.OCSIDL = 0;  // el OC se detiene si la CPU entra en modo reposo
    OC2CONbits.OCFLT  = 0;
    OC2CONbits.OCTSEL = 0;  // Trabajamos el OC con Timer2
    OC2CONbits.OCM    = 6;  // Modo PWM sin pin de fallo

    //Configuro duty OC2
    //Duty OC2 = (Ton/T) = (2/5)*100 = 40% de trabajo ------------ Ton = el duty
    OC2R    = 4000;     //Ciclo de trabajo8(duty) 1ms con prescaler=1 y Fcpu= 2MHz
    OC2RS   = 4000;     //Igual porque es un segundo buffer para evitar glitch.
```

Configuración Timer 2

El OC va a trabajar con el Timer 2. La configuración de este Timer la hemos hecho dentro la configuración del OC, aunque podría hacerse en otra función aparte.

Para una correcta configuración, primero inicializamos a cero el registro TON que mantendrá el Timer apagado. Indicamos que el Timer es de 16 bits y no de 32 haciendo uso del Timer 3. El preescaler es 1, como ya habíamos indicado en la configuración del OC. El periodo de este Timer es de 10000 para que tenga un periodo de 5ns.

```
//Configuramos Timer2
T2CONbits.TON     = 0;
T2CONbits.T32     = 0;
T2CONbits.TGATE   = 0;
T2CONbits.TCKPS   = 0; //Prescaler mas bajo = 1.
T2CONbits.TCS     = 0;

//Configuro periodo
PR2 = 10000;        //Periodo = 5ms con Precaler=1 y Fcpu = 2MHz
```

Activación Timer 2 y OCx

Los registros que habíamos inicializado a cero para asegurar una correcta configuración ya pueden adquirir sus valores finales. Activando el Timer 2 y definiendo el modo de trabajo de los módulos OC (OCM) para generar señales PWM de ciclo de trabajo variable sin protección de falla.

```
// Activar los modulos OC1, OC2 y Timer
T2CONbits.TON     = 1;
OC1CONbits.OCM    = 6;  //Activar modulo OC1 por pin RC2
OC2CONbits.OCM    = 6;  //Activar modulo OC2 por pin RC3
```

Interrupción UxRX

El enunciado nos pide que cuando el usuario ingrese una tecla por la terminal virtual de Proteus, se realicen distintas operaciones sobre el ciclo de trabajo que OC esta generando.

Para ello lo que hacemos es reconocer si la tecla introducida por el usuario es alguno de los comandos predeterminados (+, -, p, m). Si este es el caso, cambia el valor de del registro OCxRS, el cual, como ya se ha comentado antes, indica cual es el ciclo de trabajo y el cual, se puede ir modificando.

COMANDOS:

- + : suma 100 al ciclo de trabajo del modulo Output Compare 2.
- - : resta 100 al ciclo de trabajo del modulo Output Compare 2.
- p : suma 100 al ciclo de trabajo del modulo Output Compare 1.
- m : resta 100 al ciclo de trabajo del modulo Output Compare 1.

```
void __attribute__((__interrupt__, no_auto_psv)) _U1RXInterrupt(void)
{

    dataCMD_ISR[0] = U1RXREG;
    if(dataCMD_ISR[0] == '+') dutyOC2 = dutyOC2 + 100;
    if(dataCMD_ISR[0] == '-') dutyOC2 = dutyOC2 - 100;
    if(dataCMD_ISR[0] == 'p') dutyOC1 = dutyOC1 + 100;
    if(dataCMD_ISR[0] == 'm') dutyOC1 = dutyOC1 - 100;

    OC2RS = dutyOC2;
    OC1RS = dutyOC1;

    IFS0bits.U1RXIF = 0;        // Reset Rx Interrupt
}
```

Configuración Input Capture

Input Capture (IC) es el módulo que se encarga de la recepción de señales PWM. Su función es medir cuánto mide el ciclo de trabajo, es decir, cuanto dura un uno lógico hasta que cambia de nuevo a un cero lógico. Esa duración es el Duty. Para calcularlo necesitamos que el módulo IC lance una interrupción cuando el sucede cada flanco de subida y cuando sucede un flanco de bajada. En estos dos momentos el buffer del módulo IC llamado ICxBUF guarda el tiempo que marca el Timer. Si cuando sucede el evento del flanco de bajada restamos el valor del buffer al que tenía en el flanco de subida anterior, tendremos el ciclo de trabajo.

Configuramos dos pines como entrada, C5 y C6. Los remapeamos a los pines de entrada correspondientes a IC1 y IC2. Para configurar correctamente el IC hay que inicializar el subregistro ICM a cero. Este registro nos indica el modo de trabajo del IC. . También configuramos el IC para que no se detenga si la CPU entra en modo reposo e indicamos que trabajamos con el Timer 3. El módulo OC e IC deben trabajar con distintos Timer.

El subregistro ICI nos indica cada cuántos eventos queremos que el módulo IC realice su función. En este caso, en todos los eventos. Inicializamos el flag de interrupción a cero y activamos las interrupciones para el módulo ICx.

Hemos configurado IC1 y IC2 de la misma manera. Cada una recibirá las señales de OC1 y OC2 respectivamente.

```
void input_capture_config(void)
{
   //Configurar pines con los que vamos a trabajar (RP:de remapeable en proteus)
    TRISCbits.TRISC5 = 1;    //Pin configurado como entrada
    TRISCbits.TRISC6 = 1;    //Pin configurado como entrada
    RPINR7bits.IC1R = 21;    // R7 por IC1 y 21 por RP21 (remapping))
    RPINR7bits.IC2R = 22;    // R7 por IC2 y 22 por RP22 (remapping))

    //Configurar IC1
    IC1CONbits.ICM    = 0;
    IC1CONbits.ICSIDL = 0;  // el IC se detiene si la CPU entra en modo reposo
    IC1CONbits.ICTMR  = 0;  // Trabajamos con Timer 3
    IC1CONbits.ICI    = 0;  // salta en cada evento
    IFS0bits.IC1IF = 0;
    IEC0bits.IC1IE = 1;  // Enable IC1 interrupts

    //Configurar IC2
    IC2CONbits.ICM    = 0;
    IC2CONbits.ICSIDL = 0;  // el IC se detiene si la CPU entra en modo reposo
    IC2CONbits.ICTMR  = 0;  // Trabajamos con Timer 3
    IC2CONbits.ICI    = 0;  // salta en cada evento
    IFS0bits.IC2IF = 0;
    IEC0bits.IC2IE = 1;  // Enable IC2 interrupts
```

Configuración Timer 3

El IC va a trabajar con el Timer 3. La configuración de este Timer la hemos hecho dentro la configuración del IC, aunque podría hacerse en otra función aparte.

Para una correcta configuración, primero inicializamos a cero el registro TON que mantendrá el Timer apagado.

El preescaler es 1, y el periodo de este Timer es de 10000 para que tenga un periodo de 5ns como el OC.

```
//Configuramos Timer3
T3CONbits.TSIDL   = 0;
T3CONbits.TON     = 0;
T3CONbits.TGATE   = 0;
T3CONbits.TCKPS   = 0; //Prescaler mas bajo = 1.
T3CONbits.TCS     = 0;
PR3 = 10000;
```

Activación Timer 3 y ICx

Los registros que habíamos inicializado a cero para asegurar una correcta configuración ya pueden adquirir sus valores finales. Activando el Timer 3 y definiendo el modo de trabajo de los módulos IC (OCM) para que las interrupciones sucedan cada flanco de subida.

Es solo para flanco de subida porque la interrupción irá cambiando este valor entre flanco de subida y flanco de bajada cada vez que se entre en la interrupción.

```
// Activar los modulos IC1, IC2 y Timer3
T3CONbits.TON     = 1;  //Activar Timer3
IC1CONbits.ICM    = 3;  //Activar modulo OC1 por pin RC2
IC2CONbits.ICM    = 3;  //Activar modulo OC2 por pin RC3
```

Interrupción ICx

Necesitamos realizar dos interrupciones IC con el mismo comportmiento para IC1 e IC2. Los nombres de las interrupciones son "_IC1Interrupt" e "_IC2Interrupt" respectivamente.

IC ha sido configurado inicialmente para lanzar la interrupción IC cada flanco de subida. Cuando esto sucede, la variable "rise_pulso_IC" almacena el contenido del buffer IC, el cual contiene el tiempo que marcaba el Timer 3 en ese momento. A continuación, cambia el modo de trabajo del IC.

Ahora, esta interrupción saltará cada vez que haya un flanco de bajada. Cuando esto suceda, la variable "time_pulso_IC" almacenará la diferencia entre el tiempo que hay almacenado en el buffer IC y el tiempo que se había guardado el "rise_pulso_IC" en el flanco de subida. A continuación, se cambia el modo de trabajo de IC para que salte en el flanco de subida.

La variable "pulso" nos ayuda a distinguir cuando está entrando por evento de flanco de subida o flanco de bajada.

Por ultimo, reiniciamos el flag de la interrupción de Input Capture.

```
void __attribute__((__interrupt__, no_auto_psv)) _IC1Interrupt(void)
{


    if(pulsol == 0){
        rise_pulsoIC1 = IC1BUF;
        IC1CONbits.ICM = 2; // Capture next falling edge
        pulsol = 1;

    }else{
        time_pulsoIC1 = IC1BUF - rise_pulsoIC1;
        IC1CONbits.ICM = 3; // Capture next rising edge
        pulsol = 0;

    }

    IFS0bits.IC1IF = 0;          // Reset IC1 Interrupt
}
```

Programa Principal

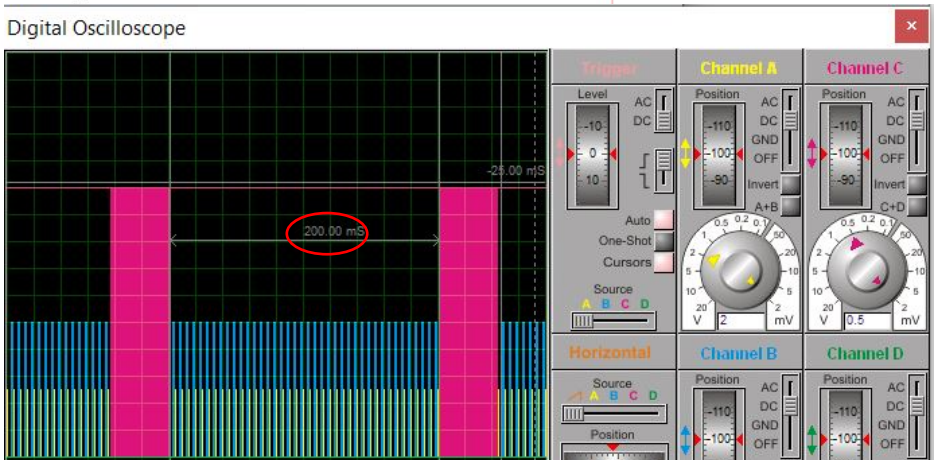
Lo primero que realiza el programa es configurar el oscilador para hacer funcional la CPU a 2MHz. A continuación, se configuran los pines como digitales, la UART, el Output Compare y el Input Compare. Entre las configuraciones se genera un delay de 10 ms para asegurarse de que todo se configura correctamente.

En el bucle infinito del programa utilizamos una función para extraer los tiempos. Para extraer los tiempos "tiempo_real_IC1" y "tiempo_real_IC2" utilizamos los tiempos "time_pulso_IC1" y "time_pulso_IC2", que son donde, en las interrupciones, hemos guardado el tiempo del ciclo de trabajo.

Después enviamos un mensaje mediante la UART para que se vayan imprimiendo los tiempos de ciclo que trabajo que tiene cada señal. Así, si el usuario modifica estos valores, podrá ser observado en la terminal virtual de Proteus.

Por último, ponemos un delay de aproximadamente 200ms que es lo que pide el enunciado. En este caso hemos puesto 194 ms ya que, midiendo en el oscilador, este es el valor que realmente hace que los datos se impriman a 200 ms. Lo hemos hecho de esta manera porque la impresión de los datos no es algo que necesite mucha precisión, y hemos considerado que configurar otro Timer para esta funcionalidad no era necesario. En caso de necesidad de precisión sí habría que hacerlo.

```
    PLLFBD = 0;                     // M  = 2
    CLKDIVbits.PLLPOST = 0;         // N1 = 2
    CLKDIVbits.PLLPRE  = 0;         // N2 = 2
    while(OSCCONbits.LOCK != 1);    // Esperar a un PLL estable
#endif

    AD1PCFGL          = 0xFFFF;     // Primer paso. Todos los pines configurados como pines digitales
    uart_config(baud_9600);
    delay_ms(10);
    output_compare_config();
    delay_ms(10);
    input_capture_config();
    delay_ms(10);

    while(1){

        tiempo_real_IC1 = 1.0*((double)time_pulsoIC1)/2000.0;
        tiempo_real_IC2 = 1.0*((double)time_pulsoIC2)/2000.0;

        sprintf(txbuffer,"TIME_IC1: %05.3fms    TIME_IC2: %05.3fms \r\n",tiempo_real_IC1, tiempo_real_IC2);
        EnviarString(txbuffer);

        delay_ms(194);
    }
}
```

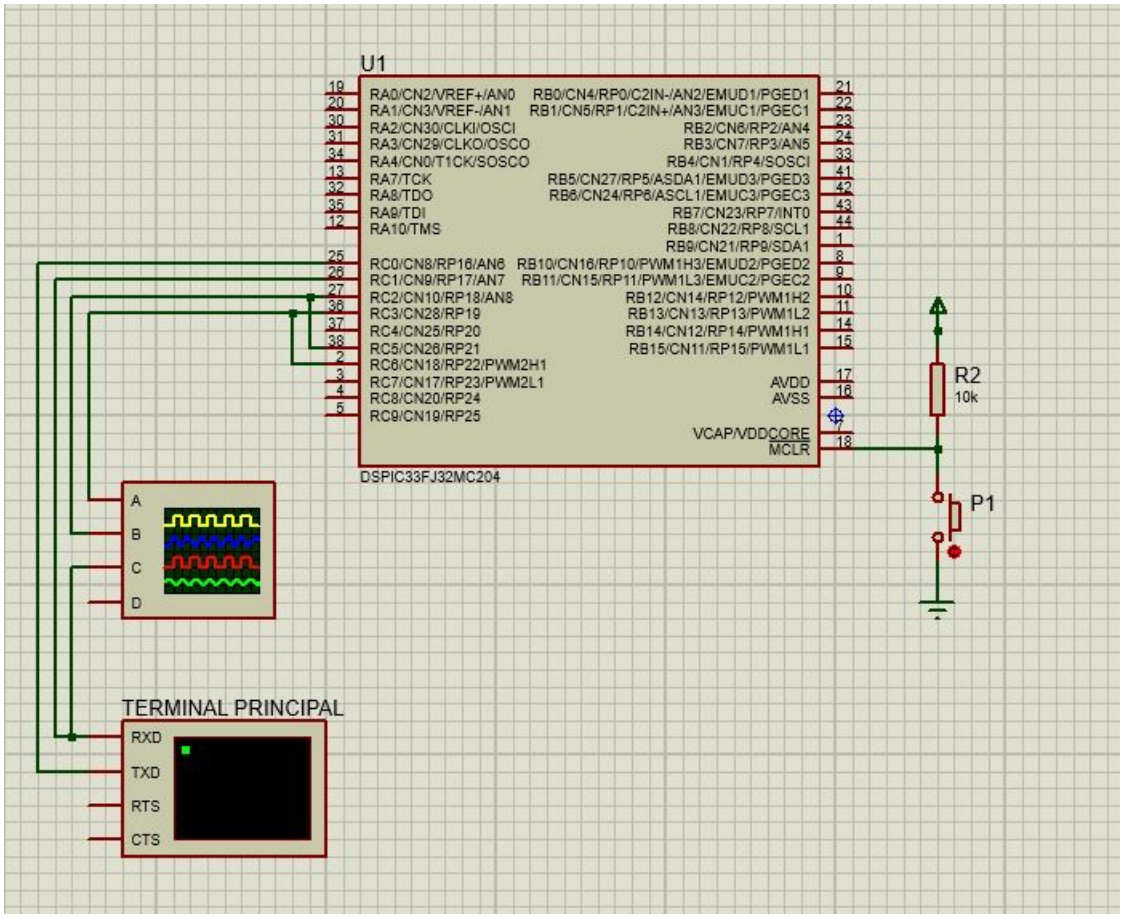


Proteus

Los pines C0 y C1 están conectados a los pines de Transmisión y Recepción de la Terminal Virtual respectivamente.

Los pines C3 y C4 correspondientes a OC1 y OC2, están conectados al osciloscopio para poder observar el pulso de las señales, y como van creciendo y disminuyendo según las indicaciones del usuario.

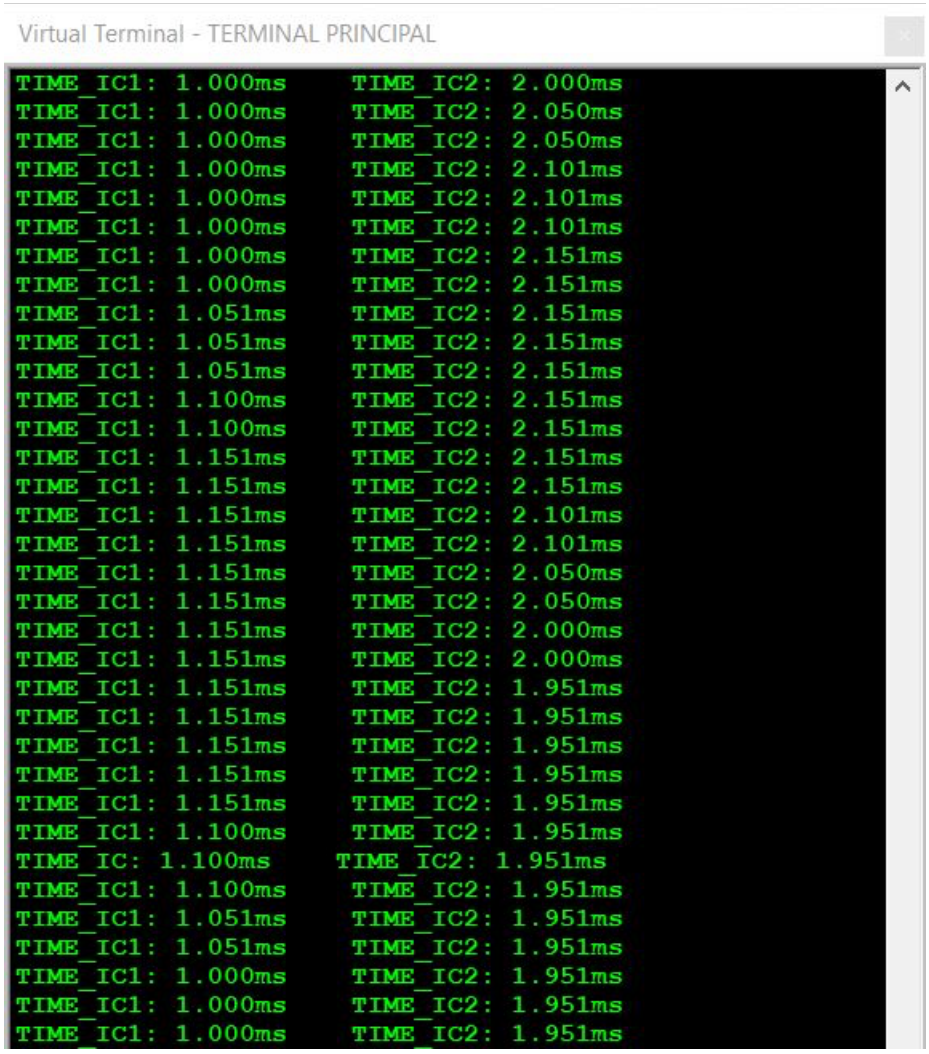
Los pines C5 y C6 correspondientes a IC1 e IC2, están conectados a los cables que sales de los pines C3 y C4 respectivamente, ya que la señal que está generando el Output Compare es la que queremos medir con Input Capture.



Demostración

La terminal virtual nos muestra los tiempos de ciclo de trabajo, que están siendo capturados por el Input Capture, cada 200 ms.

Como se puede ver en el ejemplo, el ciclo de trabajo recibido en el IC2 empieza aumentando a medida que el usuario lo indica pulsando la tecla '+'. A continuación, empieza a aumentar el ciclo de trabajo recibido en IC1 debido a que el usuario pulsa la tecla 'p'. Después, el ciclo de trabajo recibido en el IC2 empieza a disminuir a medida que el usuario lo indica pulsando la tecla '-'. A continuación, empieza a disminuir el ciclo de trabajo recibido en IC1 debido a que el usuario pulsa la tecla 'm'.



Conclusiones

Durante la realización de esta práctica, hemos podido comprobar la gran precisión de los módulos PWC e IC. Lo más fascinante es que tienen una fácil implementación, y muchísimos casos de uso en los que se los requiere.

Además, son mucho más eficientes que otro tipo de técnicas usadas para medir tiempos, entre otras, cosas como el polling.

Nos ha parecido muy interesante el ejemplo del dron sugerido en la práctica porque realmente, al haber realizado la practica, nos damos cuenta de todas las aplicaciones que haciendo uso de estos módulos son capaces de cubrir multitud de funcionalidades.