**Sensor de orientación automática para un monitor**

# Introducción

El objetivo de este proyecto es crear un sensor para cambiar automáticamente la orientación de la pantalla de un ordenador. Dicho cambio, se realizará una vez el ángulo de la pantalla pase de unos valores en concreto, momento cuando se mandará la señal por UART al ordenador para cambiar de orientación.

Para obtener el ángulo del monitor se usará un sensor de aceleración analógico para medir la dirección de la gravedad. Además, habrá un botón para recalibrar la orientación que se toma como 0 grados.

La conexión de los diferentes componentes al dspic se muestra en Ilustración 1.

Diagrama

Descripción generada automáticamente

Ilustración 1 Diagrama de bloques del proyecto

# Acondicionamiento del acelerómetro

El acelerómetro que se va a usar es el ADXL335, que es un acelerómetro de 3 ejes, el cual puede medir aceleraciones de hasta 3g en las tres direcciones del espacio, tanto en sentido positivo como negativo.

Para determinar el ángulo de la pantalla, se necesita medir la aceleración en dos ejes, en este caso se han elegido X e Y, cuyas direcciones se representan en el propio acelerómetro, tal como se puede ver en Ilustración 1.

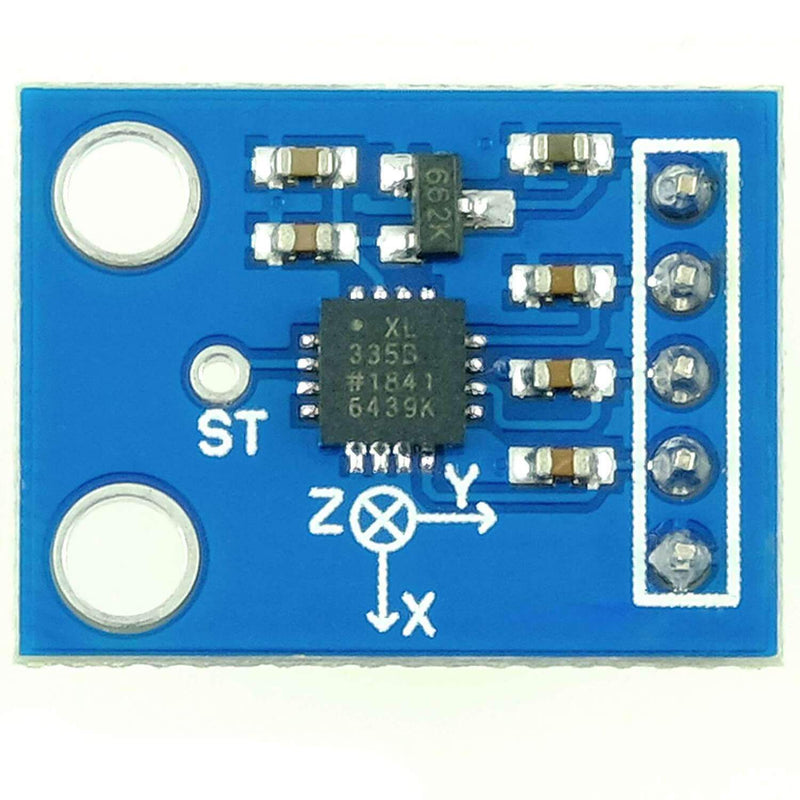


Ilustración 2 ADXL335

En el proyecto, el rango de medida de la aceleración va a ser ±g (± 10 m/s2), y la tensión que dan los pines en función de la aceleración que hay en su eje correspondiente es la que se ve en Ilustración 3, en la cual se ve que la relación entre la entrada y la salida es lineal.

Ilustración 3 Curva de calibración acelerómetro ADXL335

Como la sensibilidad es baja y no se aprovecha todo el rango de tensión del dsPIC, se va a acondicionar el acelerómetro haciendo un ajuste de ganancia utilizando el circuito mostrado en Ilustración 2, donde los valores de los componentes son:

* E = 1.65V
* R1 = 1 kΩ
* R2 = 5.6 kΩ
* Amplificadores operacionales MCP6002 alimentados entre 3.3V-0V.

Diagram

Description automatically generated

Ilustración 4 Acondicionamiento acelerómetro

Para poder distinguir entre aceleraciones positivas y negativas, se ha ajustado E para tener Vo=1.65V cuando no hay aceleración, y la ganancia (-R2/R1), se ha ajustado para tener Vo = 0 cuando la aceleración es máxima (1 g) y Vo = 3.3V cuando la aceleración es mínima (-1g), de forma que la ecuación de la tensión de salida del circuito de acondicionamiento es ( 1 ).

( 1 )

La tensión de salida es lineal respecto a la aceleración, pero la pendiente es negativa. Esto no supone ningún problema, ya que se hará la respectiva corrección cuando se obtenga de forma digital la aceleración a partir de la medida de Vo.

El acelerómetro se ha conectado al circuito de acondicionamiento a través de un buffer para aislarlo.

Se necesitan montar dos circuitos de acondicionamiento, uno para el pin correspondiente al eje X y otro para el pin correspondiente al eje Y.

# Parte digital

Una vez acondicionados los sensores, se procede a diseñar la parte digital del proyecto. Esta parte se encargará de leer los sensores, calcular la posición, enviar dicha posición por la UART y representar el cuadrante en el que se encuentra la posición en unos LEDs. Además, se incluirá un botón de posición 0, cada vez que el usuario lo pulse, el dspic tomará el ángulo que dan los sensores en ese instante como el ángulo 0.

Se utilizará el sistema operativo en tiempo real FreeRTOS, en el cual se incluirán 4 tareas:

* LeerAcel: Esta tarea se encarga de leer las entradas analógicas a las cuales están conectados los circuitos de acondicionamiento del sensor (a AN0 se conecta el eje X y al AN1 el eje Y) y se calcula el ángulo, el cual se guarda en una variable global. Si el ángulo cambia más de una cierta tolerancia con respecto a la medida anterior, se libera el semáforo que permite ejecutar la tarea ComputarOri. Por otro lado, se comprueba si se ha pulsado un botón, el cual ajusta el punto de ángulo 0 del acelerómetro.
* ComputarOri: Establece la orientación del sensor a partir del ángulo según Ilustración 4, y la guarda en una variable global. Si la orientación cambia respecto a la orientación de la computación anterior, se liberan los semáforos para ejecutar las tareas MostrarOri y EnviarOri.

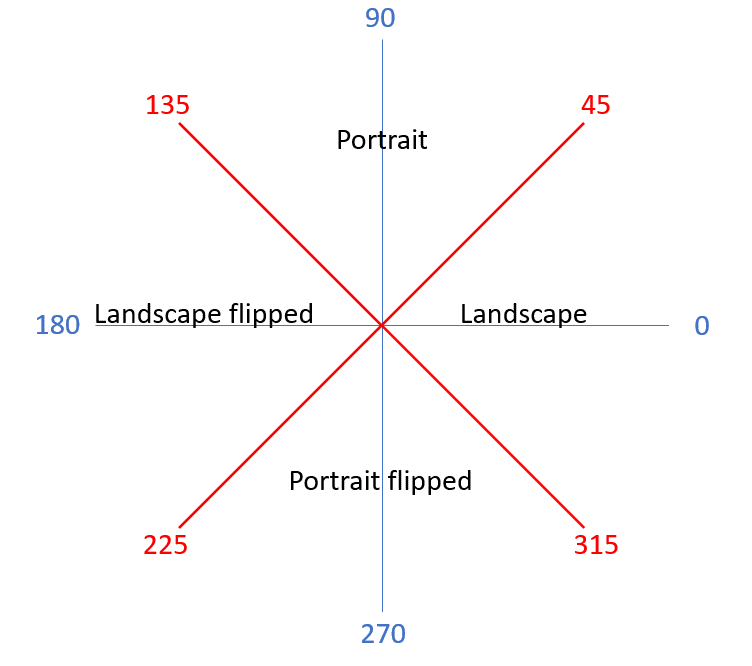


Ilustración 5 Orientaciones

* MostrarOri: Muestra en los LEDs conectados al dspic la orientación en ese momento, siguiendo Tabla 1.

|  |  |
| --- | --- |
| **Orientación** | **LED** |
| Landscape | RB12 |
| Portrait | RB13 |
| Landscape flipped | RB14 |
| Portrait flipped | RB15 |

Tabla 1 Asociación orientaciones con LEDs

* EnviarOri: Envía por la UART la orientación computada por ComputarOri.

En Ilustración 5 se muestra de forma gráfica las relaciones entre las tareas y su nivel de prioridad asociado. A LeerAcel se le ha asociado el nivel de prioridad más bajo porque es una tarea que se va a tener ejecutándose todo el rato, no se bloquea. A las tareas que sí se bloquean se les va asociando niveles de prioridad cada vez más altos para que se ejecuten según se desbloqueen y, cuando terminan de ejecutarse, ceden el semáforo correspondiente y se vuelven a bloquear.

Diagrama

Descripción generada automáticamente

Ilustración 6 Relación entre tareas y niveles de prioridad

# Parte PC

El ordenador recibe por UART la posición del sensor del micro, y cambia la rotación de la pantalla en consecuencia. Para ello, se ha creado nuestra propia clase Cpantalla, que se encarga de cambiar la orientación usando la librería Windows.h.

# Anexo de código

Main.c

#include <p33FJ128MC802.h>

#include <stdint.h> // define los tipos uint8\\_t, ...

#include <stdio.h>  // Para usar sprintf

#include "DriverPicTrainer/config.h"

#include "Interrupciones.h" // Enable y Disable

#include "TareaAcel.h"

#include "TareaEnviar.h"

#include "TareaCompOri.h"

#include "TareaLEDS.h"

#include "FreeRTOS.h" // Includes del Kernel

#include "semphr.h"

#include "queue.h"

#include "task.h"

#define PRIO\_LEDS 3

#define PRIO\_SERIE 3

#define PRIO\_ORI 2

#define PRIO\_ACEL 1

#define TAM\_PILA\_HORA      250 // Como utiliza sprintf, necesita 100 palabras más.

#define TAM\_PILA\_ENTRADAS  250 // Como utiliza sprintf, necesita 100 palabras más.

#define TAM\_PILA\_SERIE     250

SemaphoreHandle\_t sem\_ori = 0; // Semáforo para computar la orientación actual

SemaphoreHandle\_t sem\_serie = 0; // Semáforo para el acceso a la UART

SemaphoreHandle\_t sem\_leds = 0; // Semáforo para el actializar leds

float angulo = 0;

int orientacion =0;

int main(void)

{

    InicializarReloj();

    InitAcel();

    InitLeds();

    InitEnviar();

    // Se inicializa el semáforo a 0(bloqueado)

    vSemaphoreCreateBinary(sem\_ori);

    vSemaphoreCreateBinary(sem\_serie);

    vSemaphoreCreateBinary(sem\_leds);

    // Se crean las tareas

    xTaskCreate(LeerAcel,   "LeA", TAM\_PILA\_HORA, NULL,

                PRIO\_ACEL, NULL);

    xTaskCreate(ComputarOri, "ComOri", TAM\_PILA\_ENTRADAS, NULL,

                PRIO\_ORI, NULL);

    xTaskCreate(MostrarOri,   "MoOri", TAM\_PILA\_SERIE, NULL,

                PRIO\_LEDS, NULL);

    xTaskCreate(EnviarOri,   "EnO", TAM\_PILA\_SERIE, NULL,

                PRIO\_SERIE, NULL);

    vTaskStartScheduler(); // y por último se arranca el

                           // planificador.

    return 0; // En teoría no se debe llegar nunca aquí.

}

/\* Cuando está activa la verificación de desborde de pila, si FreeRTOS descubre

 \* un desbode en la pila ejecuta esta función. La función se limita a bloquearse

 \* en un bucle sin fin para que al pausar el programa con el debugger veamos que

 \* efectivamente algo va mal con las pilas. En estos casos suele bastar con

 \* aumentar su tamaño.

 \*/

#ifdef configCHECK\_FOR\_STACK\_OVERFLOW

void vApplicationStackOverflowHook( TaskHandle\_t xTask,

                                        char \* pcTaskName )

{

    static int var\_inc = 0;

    while(1){

        var\_inc++; // Para que el optimizador no quite el bucle.

    }

}

#endif

/\* Rutinas de atención a los traps. Vienen bien en depuración, para ver si está

 \* ocurriendo algún error de acceso a memoria o matemático. En ese caso el

 \* programa entrará continuamente en estas funciones y haciendo una pausa con

 \* el debugger veremos que ha ocurrido un error.

 \*/

static int a=0, b=0, c=0;

void \_\_attribute\_\_((interrupt,no\_auto\_psv)) \_AddressError(void)

{

    INTCON1bits.ADDRERR = 0; // Borramos el flag

    a++;

    return;

}

void \_\_attribute\_\_((interrupt,no\_auto\_psv)) \_StackError(void)

{

    INTCON1bits.STKERR = 0; // Borramos el flag

    b++;

    return;

}

void \_\_attribute\_\_((interrupt,no\_auto\_psv)) \_MathError(void)

{

    INTCON1bits.MATHERR = 0; // Borramos el flag

    c++;

    return;

}

TareaAcel.h

#ifndef TAREA\_ACEL\_H

#define TAREA\_ACEL\_H

void InitAcel();

void LeerAcel(void \*pvParameters);

#endif  /\* TAREA\_ACEL\_H \*/

TareaAcel.c

/\*

 \* File:   TareaAcel.c

 \* Author: franb

 \*

 \* Created on November 18, 2022, 5:59 PM

 \*/

#include <p33FJ128MC802.h>

#include <stdint.h>

#include <stdio.h>

#include "Interrupciones.h"

#include "DriverPicTrainer/adc.h"

#include "TareaAcel.h"

// Includes del Kernel.

#include "FreeRTOS.h"

#include "semphr.h"

#include "task.h"

#include <math.h>

#define PI 3.14159

#define PIN\_BOTTON 2

#define TOLERANCIA 5

void InitAcel()

{

    inicializarADCPolling(3); // Inicializo el ADC AN0 y AN1

    TRISB |= (1 << PIN\_BOTTON);

    PORTB &= ~(1 << PIN\_BOTTON);

}

void LeerAcel(void \*pvParameters)

{

    extern float angulo;

    extern SemaphoreHandle\_t sem\_ori;

    // Internal variables

    //Guardamos el angulo act y los dos anteriores

    float angulo\_act;

    float angulo\_ant = 0;

    float angulo\_ant\_ant =0;

    // lectura en cada eje en escala 1024 y pasada a aceleraciones

    unsigned int lectura\_y, lectura\_x;

    float l\_y, l\_x;

    //Detección de pulsador

    float puls\_act, puls\_ant;

    puls\_ant = 1;

    lectura\_y = leerADCPolling(1);

    lectura\_x = leerADCPolling(0);

    l\_y = (lectura\_y - 512.0) / 512.0 \* 10;

    l\_x = (lectura\_x - 512.0) / 512.0 \* 10;

    float offset\_angulo = (atan2(-l\_x, -l\_y) \* 180 / PI);

    float angulo\_calc;

    while (1)

    {

        lectura\_y = leerADCPolling(1);

        lectura\_x = leerADCPolling(0);

        l\_y = (lectura\_y - 512.0) / 512.0 \* 10;

        l\_x = (lectura\_x - 512.0) / 512.0 \* 10;

        if ((l\_x \* l\_x + l\_y \* l\_y) < 100) // si el módulo es muy grande es que hay aceleraciones externas (filtramos)

        {

            angulo\_calc = (atan2(-l\_x, -l\_y) \* 180 / PI);

            angulo\_act = angulo\_calc - offset\_angulo;

            while (angulo\_act < 0)

                angulo\_act += 360;

            while (angulo\_act >= 360)

                angulo\_act -= 360;

            puls\_act = PORTB & (1 << 2);

            if (puls\_act != puls\_ant && puls\_act == 0)

            {

                offset\_angulo = angulo\_calc;

            }

            puls\_ant = puls\_act;

            //Si se acaba de estabilizar en un valor

            if (fabs(angulo\_act - angulo\_ant) <= TOLERANCIA && fabs(angulo\_ant\_ant - angulo\_ant) >= TOLERANCIA)

            {

                vTaskSuspendAll(); // Suspend task scheduling

                angulo = angulo\_act;

                xTaskResumeAll();

                vTaskDelay(10);

                xSemaphoreGive(sem\_ori);

            }

            angulo\_ant\_ant = angulo\_ant;

            angulo\_ant = angulo\_act;

        }

    }

}

TareaCompOri.h

#ifndef TAREA\_COMP\_H

#define TAREA\_COMP\_H

void InitOrientacion();

void ComputarOri(void \*pvParameters);

#endif  /\* TAREA\_COMP\_H \*/

TareaCompOri.c

/\*

 \* File:   TareaCompOri.c

 \* Author: franb

 \*

 \* Created on November 18, 2022, 5:59 PM

 \*/

#include <p33FJ128MC802.h>

#include <stdint.h>

#include <stdio.h>

#include "Interrupciones.h"

#include "DriverPicTrainer/adc.h"

#include "TareaCompOri.h"

// Includes del Kernel.

#include "FreeRTOS.h"

#include "semphr.h"

#include "task.h"

enum

{

    LANDSCAPE,

    PORTRAIT,

    LANDSCAPE\_FLIPPED,

    PORTRAIT\_FLIPPED,

};

void InitOrientacion()

{

}

void ComputarOri(void \*pvParameters)

{

    extern float angulo;

    extern int orientacion;

    extern SemaphoreHandle\_t sem\_serie;

    extern SemaphoreHandle\_t sem\_leds;

    extern SemaphoreHandle\_t sem\_ori;

    float angulo\_copia;

    int orientacion\_act;

    int orientacion\_ant = -1; // Para mandar el mensaje con el primer valor

    while (1)

    {

        if (xSemaphoreTake(sem\_ori, portMAX\_DELAY) == pdTRUE)

        {

            // Copiamos el valor de la variable compartida angulo. En la zona segura.

            vTaskSuspendAll();

            angulo\_copia = angulo;

            xTaskResumeAll();

            // We make sure the angle is between 0 and 360 degrees.

            while (angulo\_copia < 0)

                angulo\_copia += 360;

            while (angulo\_copia >= 360)

                angulo\_copia -= 360;

            switch ((int)angulo\_copia)

            {

            case 0 ... 45:

                orientacion\_act = LANDSCAPE;

                break;

            case 46 ... 135:

                orientacion\_act = PORTRAIT;

                break;

            case 136 ... 225:

                orientacion\_act = LANDSCAPE\_FLIPPED;

                break;

            case 226 ... 315:

                orientacion\_act = PORTRAIT\_FLIPPED;

                break;

            case 316 ... 360:

                orientacion\_act = LANDSCAPE;

                break;

            default:

                orientacion\_act = LANDSCAPE;

                break;

            }

            if (orientacion\_act != orientacion\_ant)

            {

                // Copiamos el valor a la variable compartida orientacion. En la zona segura.

                vTaskSuspendAll();

                orientacion = orientacion\_act;

                xTaskResumeAll();

                // Activamos los semáforos de las tareas que necesitan la orientación.

                xSemaphoreGive(sem\_serie);

                xSemaphoreGive(sem\_leds);

            }

            orientacion\_ant = orientacion\_act;

        }

    }

}

TareaEnviar.h

#ifndef TAREA\_ENVIAR\_H

#define TAREA\_ENVIAR\_H

void InitEnviar();

void EnviarOri(void \*pvParameters);

#endif  /\* TAREA\_ENVIAR\_H \*/

TareaEnviar.c

#include <p33FJ128MC802.h>

#include <stdint.h>

#include <stdio.h>

#include "Interrupciones.h"

#include "DriverPicTrainer/config.h"

// Includes del Kernel.

#include "FreeRTOS.h"

#include "queue.h"

#include "task.h"

#include "semphr.h"

void putsUART(char \*pcad);

void InicializarUART(unsigned long baudrate);

void ProcesaMensaje(char \*pmensaje);

QueueHandle\_t cola\_env;      // Cola para enviar caracteres a la UART

QueueHandle\_t cola\_rec;      // Cola para recibir de la UART

#define TAM\_COLA 100         // Tamaño de la cola

#define TAM\_MSG sizeof(char) // y de los elementos de la cola

void InitEnviar(void)

{

  // Como se van a usar los LEDs, se configura el puerto aquí

  TRISB = 0x0FFF;

  // Primero se crean las colas. Ambas del mismo tamaño

  cola\_rec = xQueueCreate(TAM\_COLA, TAM\_MSG);

  cola\_env = xQueueCreate(TAM\_COLA, TAM\_MSG);

  if ((cola\_rec == NULL) || (cola\_env == NULL))

  {

    // Error fatal: Nos bloqueamos

    while (1)

      ;

  }

  InicializarUART(115200); // Inicialización del puerto serie

  extern SemaphoreHandle\_t sem\_serie;

  // xSemaphoreTake(sem\_serie,100);

}

void EnviarOri(void \*pvParameters)

{

  extern SemaphoreHandle\_t sem\_serie;

  extern float angulo;

  extern int orientacion;

  float copia\_angulo;

  static char c\_angulo[18];

  while (1)

  {

    if (xSemaphoreTake(sem\_serie, (portTickType)1000) == pdTRUE)

    {

      vTaskSuspendAll();

      copia\_angulo = angulo;

      xTaskResumeAll();

      vTaskSuspendAll();

      //        sprintf(c\_angulo, " %f;\n", copia\_angulo);

//      snprintf(c\_angulo, sizeof c\_angulo, "Angulo: %.2f\n", copia\_angulo);

      snprintf(c\_angulo, sizeof c\_angulo, "%d\n", orientacion);

      xTaskResumeAll();

      putsUART(c\_angulo);

    }

  }

}

void ProcesaMensaje(char \*pmensaje)

{

  // Saca el primer carácter por los LEDS de la placa

  PORTB &= 0x0FFF;

  PORTB |= ((~(\*pmensaje)) & 0x0F) << 12;

}

void putsUART(char \*pcad)

{

  // Primero se copia la cadena a la cola de envío

  while (\*pcad != '\0')

  {

    xQueueSend(cola\_env, pcad, (TickType\_t)1000);

    pcad++;

  }

  IEC0bits.U1TXIE = 1; // Habilitar las interrupciones de transmisión

  IFS0bits.U1TXIF = 1; // Provocar una interrupción para emprezar a transmitir

}

/\*\*

 \* Inicializa la UART.

 \*

 \* Se inicializa la UART para usar una trama de 8 bits de datos sin paridad y

 \* con un bit de stop. El módulo usa interrupciones tanto para la recepción como

 \* para la transmisión. La comunicación con las rutinas de interrupción se

 \* realiza mediante dos colas.

 \*

 \* @param baudrate Baudrate de la uart en baudios

 \*/

#define PR\_INT\_TX 4

#define PR\_INT\_RX 4

void InicializarUART(unsigned long baudrate)

{

  U1BRG = (FCY / baudrate) / 16 - 1; // Velocidad de transmisión

  U1MODEbits.STSEL = 0; // Bits de parada: 1

  U1MODEbits.PDSEL = 0; // Bits de datos: 8, Paridad: Ninguna

  U1MODEbits.ABAUD = 0; // Deshabilitar la detección de velocidad

  U1MODEbits.BRGH = 0;  // Low Speed mode

  U1MODEbits.UEN = 0;   // Usar sólo U1TX y U1RX

  U1MODEbits.USIDL = 0; // Mantener el módulo activo en modo reposo

  U1STAbits.UTXISEL0 = 0; // Interrupcion de Tx: Cuando se vacía U1TXREG

  U1STAbits.UTXISEL1 = 1;

  U1STAbits.URXISEL = 0; // Interrupcion de Rx: Al recibir un caracter

  IPC3bits.U1TXIP = PR\_INT\_TX; // Prioridad de la interrupción de transmisión

  IPC2bits.U1RXIP = PR\_INT\_RX; // Prioridad de la interrupción de recepción

  IFS0bits.U1TXIF = 0; // Borrar la bandera de la interrupción Tx

  IFS0bits.U1RXIF = 0; // Borrar la bandera de la interrupción Rx

  IEC0bits.U1TXIE = 0; // Deshabilitar las interrupciones de transmisión

  IEC0bits.U1RXIE = 1; // Habilitar las interrupciones de recepción

  U1MODEbits.UARTEN = 1; // Habilitar el módulo UART

  U1STAbits.UTXEN = 1;   // Habilitar la transmisión (Sólo si UARTEN = 1)

  // Se configuran los pines del micro por los que se conecta la UART USB

  \_\_builtin\_write\_OSCCONL(OSCCON & 0xBF); // Desbloquear el PPS

  RPINR18bits.U1RXR = 5;                  // Asignar U1RX al pin 14 que es RP5

  RPOR2bits.RP4R = 3;                     // Asignar U1TX al pin 11 que es RP4

  \_\_builtin\_write\_OSCCONL(OSCCON | 0x40); // Bloquear el PPS

}

/\*\*

 \* Rutina de atención a la interrupción de la UART asociada a la recepción.

 \*/

void \_\_attribute\_\_((interrupt, no\_auto\_psv))

\_U1RXInterrupt(void)

{

  char car\_rec;

  BaseType\_t xHigherPriorityTaskWoken;

  car\_rec = U1RXREG;

  xQueueSendFromISR(cola\_rec, &car\_rec,

                    &xHigherPriorityTaskWoken);

  IFS0bits.U1RXIF = 0; // Borra la bandera de interrupción

  if (xHigherPriorityTaskWoken == pdTRUE)

  {

    taskYIELD(); // Si el envío a la cola ha despertado

                 // una tarea, se fuerza un cambio de

                 // contexto

  }

}

/\*\*

 \* Rutina de atención a la interrupción de la UART asociada a la transmisión.

 \*/

void \_\_attribute\_\_((interrupt, no\_auto\_psv)) \_U1TXInterrupt(void)

{

  char car\_emi;

  BaseType\_t xHigherPriorityTaskWoken;

  IFS0bits.U1TXIF = 0; // Borrar la bandera de la interrupción

  if (xQueueReceiveFromISR(cola\_env, &car\_emi,

                           &xHigherPriorityTaskWoken) == pdTRUE)

  {

    U1TXREG = car\_emi;

  }

  else

  {

    // Cuando se vacía la cola hay que inhabilitar las interrupciones de

    // transmisión, pues si no, nos metemos en un bucle

    IEC0bits.U1TXIE = 0;

  }

}

TareaLeds.h

#ifndef TAREA\_LEDS\_H

#define TAREA\_LEDS\_H

void InitLeds();

void MostrarOri(void \*pvParameters);

#endif  /\* TAREA\_LEDS\_H \*/

TareaLeds.c

/\*

 \* File:   TareaLEDS.c

 \* Author: franb

 \*

 \* Created on November 18, 2022, 5:59 PM

 \*/

#include <p33FJ128MC802.h>

#include <stdint.h>

#include <stdio.h>

#include "Interrupciones.h"

#include "DriverPicTrainer/adc.h"

#include "TareaLEDS.h"

// Includes del Kernel.

#include "FreeRTOS.h"

#include "semphr.h"

#include "task.h"

void InitLeds()

{

    // PORTB RB12, RB13, RB14, RB15 como salidas (0 = salida)

    TRISB &= ~0xF000;

    // PORTB RB12, RB13, RB14, RB15 apagados (1 = apagado)

    PORTB |= 0xF000;

}

void MostrarOri(void \*pvParameters)

{

    extern int orientacion;

    extern SemaphoreHandle\_t sem\_leds;

    while (1)

    {

        if (xSemaphoreTake(sem\_leds, portMAX\_DELAY) == pdTRUE)

        {

            PORTB |= 0xF000;                 // Apago todos los leds

            PORTB &= ~(1 << 12 + orientacion); // Enciendo el led correspondiente

        }

    }

}

# Anexo código PC

#include <iostream>

#include <windows.h>

#include <winuser.h>

#include "serial.h"

#include "CPantalla.h"

#include <string>

#include <ctime>

using namespace std;

enum

{

    LANDSCAPE,

    PORTRAIT,

    LANDSCAPE\_FLIPPED,

    PORTRAIT\_FLIPPED,

};

int main(void)

{

    std::cout << "Welcome to Screen Rotator. " << std::endl;

    std::cout << "To close the program keep pressed the return or enter key. " << std::endl;

    CSerial port;

    CPantalla pantalla;

    unsigned char buffer[10];

    unsigned long resultado;

    //pantalla.setLandscape Flipped();

    while (1)

    {

        if (port.PortRead(10, buffer, &resultado))

        { // buffferlength bufferpointern\_Elem\_read pointer

            //cout << (int)buffer[0] << endl;

            cout << buffer[0] << endl;

            switch ((int)buffer[0]-48 )

            {

            case LANDSCAPE:

                pantalla.setLandscape();

                break;

            case PORTRAIT:

                pantalla.setPortraitFlipped();

                break;

            case LANDSCAPE\_FLIPPED:

                pantalla.setLandscapeFlipped();

                break;

            case PORTRAIT\_FLIPPED:

                pantalla.setPortrait();

                break;

            default:

                break;

            }

            //     if (buffer[0] == 0) pantalla.counterclockwise();

            //    if (buffer[0] == 1) pantalla.clockwise();

        }

        ////https://docs.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes

        uint8\_t a = GetKeyState(VK\_ESCAPE) & 0x80;

        uint8\_t b = GetKeyState(VK\_RETURN) & 0x80;

        if (a == 0x80 || b == 0x80)

        {

            cout << "\nProgram is closing by user action";

            // End program, call destructor

            return 0;

        }

    }

    return 0;

}