

Departamento de Arquitectura y Tecnología de Computadores



UNIVERSIDAD DE SEVILLA

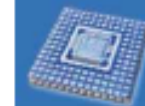


E.T.S. de Ingeniería Informática
Avda. Reina Mercedes, S/N.
41012 Sevilla, SPAIN



Escuela Universitaria Politécnica
C/ Virgen de África, 7.
41011 Sevilla, SPAIN

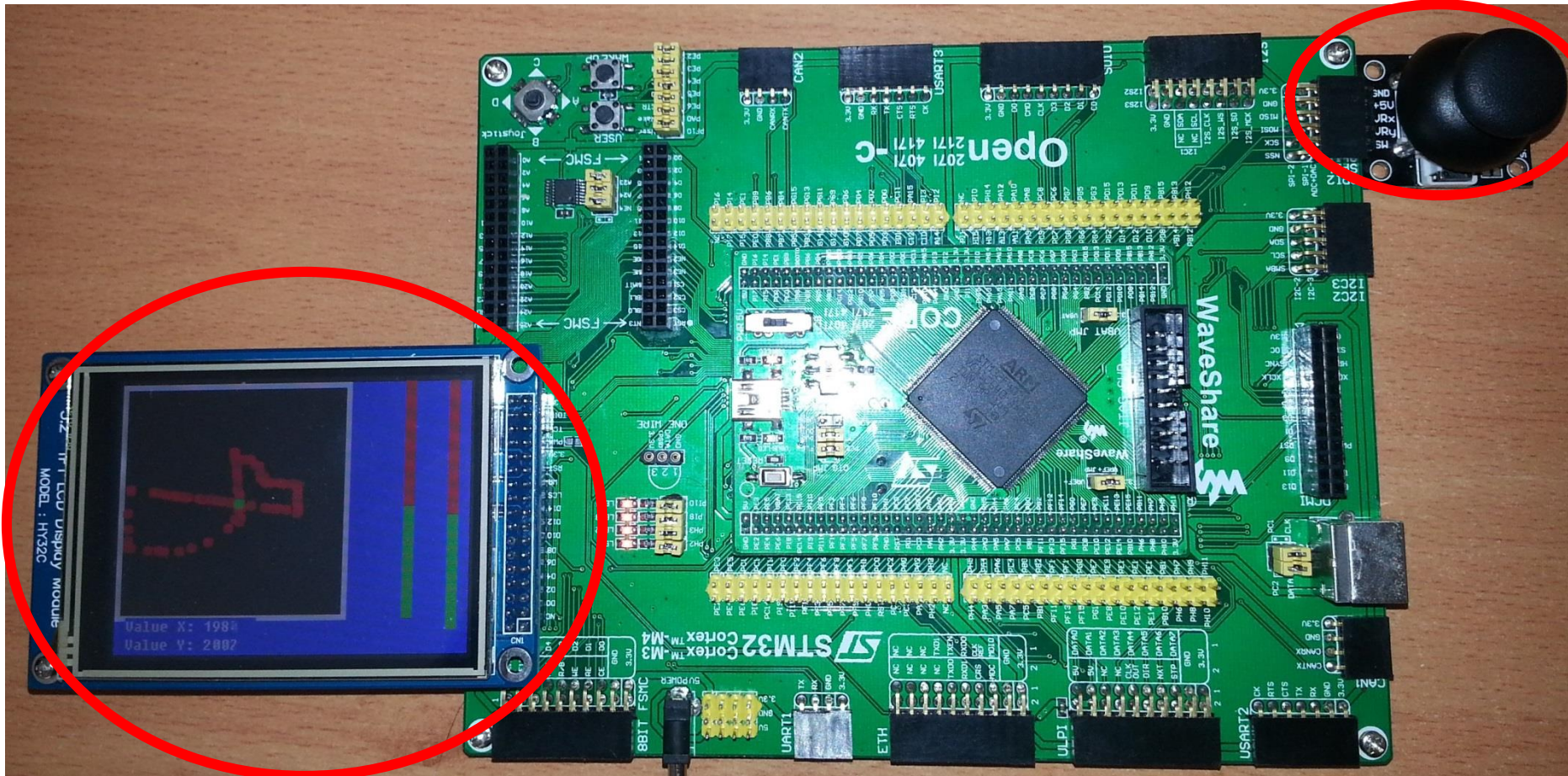
Displays LCD gráficos para sistemas empuotrados



Objetivos

- Controlar un display LCD usando el STM32:
 - Texto.
 - Gráficos estáticos.
 - Gráficos dinámicos.
- Diseño e implementación de tareas que muestren distintos gráficos interactivos.
- Creación y eliminación de tareas dinámicamente.
- Lectura del ADC mediante DMA.

Hardware Setup



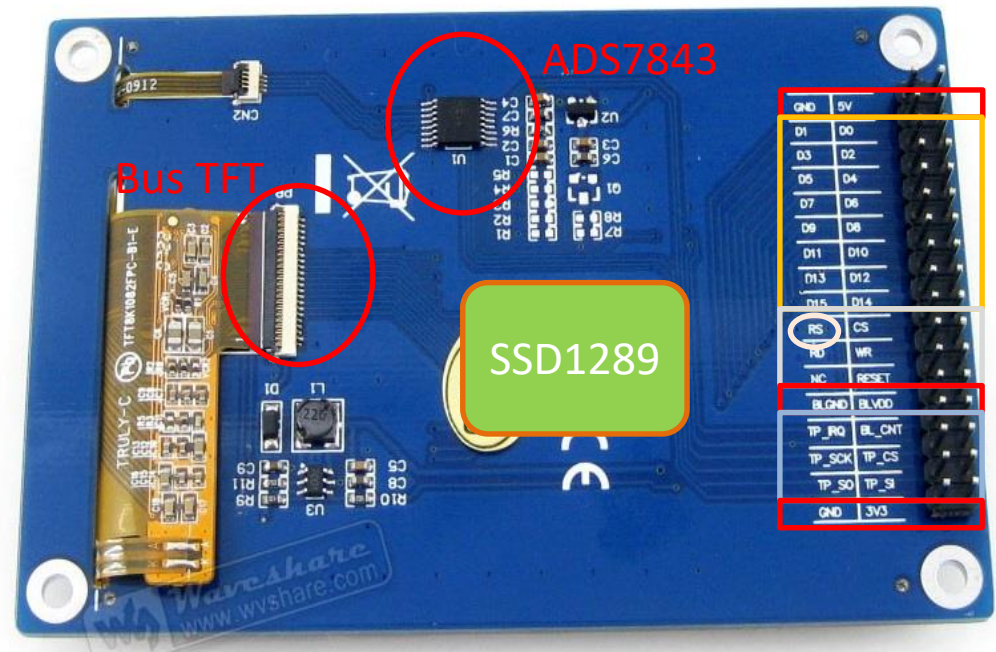
Display LCD

- El display LCD-TFT basado en el controlador SSD1289:
 - **Tamaño:** 3.2 pulgadas
 - **Resolución espacial:** 320 horizontal X 240 Vertical
 - **Resolución color:** 16bits (RGB565)
 - **Touch Panel:** Resistivo, con interfaz SPI (ADS7843)



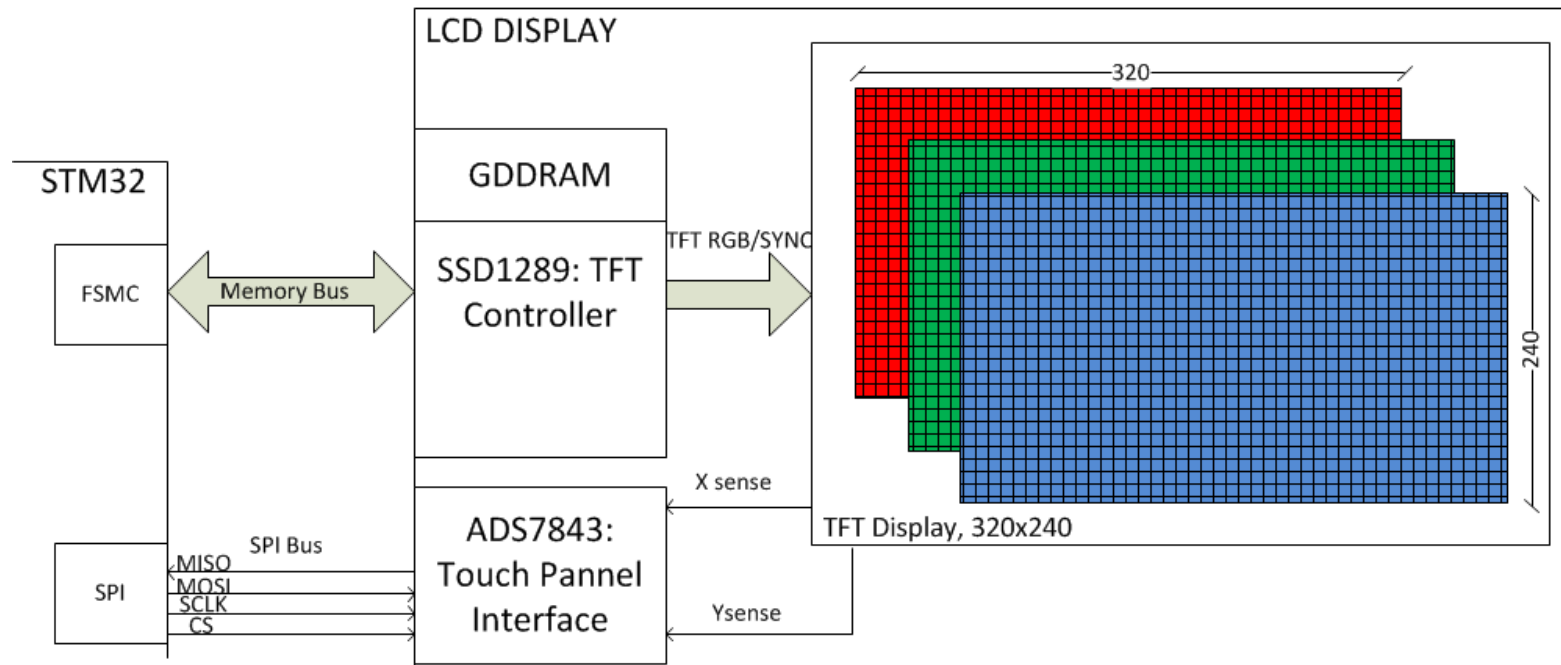
Display LCD

1. **Alimentaciones:** 3.3V / 5V
2. **Bus de datos:** D0..D15 (16bits)
3. **Un solo bit de dirección:** RS
4. **Líneas de control:**
 1. Chip Select (CS)
 2. Read/Write (RD y WR)
5. **Bus SPI:** touch panel.



Display LCD

- El módulo del display lo integran tres componentes:
 1. El display TFT con película resistiva.
 2. El controlador del display: SSD1289
 3. Interfaz Touch Panel: ADS7843



La codificación RGB-565

- El display tiene 16 bits de resolución de color y usa la codificación RGB-565.
- Los 16 bits del color empaquetan el color usando la codificación 565:
 - Los 5 bits más significativos contienen el rojo (32 niveles de rojo).
 - Los 6 bits siguientes contienen el verde (64 niveles de verde).
 - Los 5 bits menos significativos contienen el azul (32 niveles de azul).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R4				R0	G5					G0	B4				B0

Conexión con el STM32

- El display se conecta al Flexible Static Memory Controller (FSMC) integrado en el STM32.
- El FSMC permite incorporar memorias externas (SRAM, NAND...) al mapa de memoria del STM32, expandiendo su espacio de direcciones.
- El FSMC se configura para que el display pase a formar parte del mapa de memoria del STM32.



Conexión con el STM32

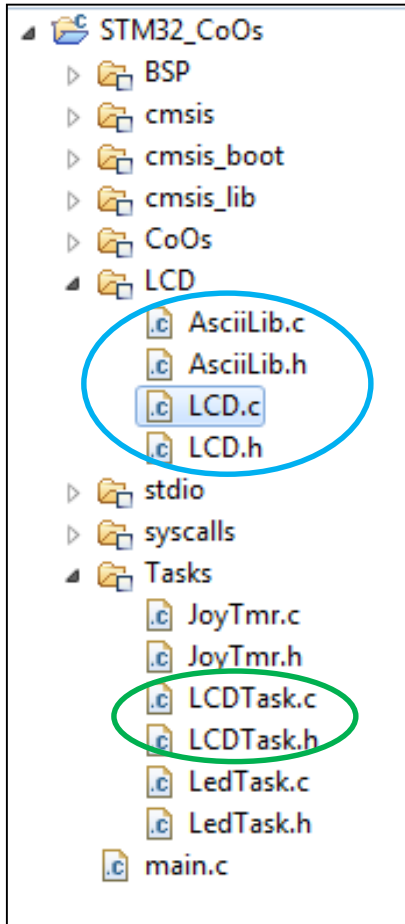
- Desde el punto de vista del FSMC, **RS** es un bit de dirección, y lo interpreta como un espacio de memoria con 2 posiciones:
 1. LCD_REG: Dirección de memoria de la GDDRAM.
 2. LCD_RAM: Dato a leer/escribir en la GDDRAM.
- Desde el punto de vista del software son dos variables globales.

```
#define LCD_REG      (*((volatile unsigned short *) 0x6F000000)) /* RS = 0 */  
#define LCD_RAM     (*((volatile unsigned short *) 0x6F010000)) /* RS = 1 */
```

- Se suministran las funciones para el manejo en alto nivel.


El proyecto de partida

- Se ha añadido al proyecto:
 1. Librerías para el manejo del display y un juego de caracteres ASCII.
 2. Esqueleto para las tareas que controlen el display.



Manejo del display

- Colores

- Predefinidos: 
- Definidos por el usuario:

```
uint16_t color = RGB565CONVERT ( R, G, B);|
```

```
/* LCD color */  
#define White      0xFFFF  
#define Black      0x0000  
#define Grey       0xF7DE  
#define Blue       0x001F  
#define Blue2      0x051F  
#define Red        0xF800  
#define Magenta    0xF81F  
#define Green      0x07E0  
#define Cyan       0x7FFF  
#define Yellow     0xFFE0
```

- Inicialización:

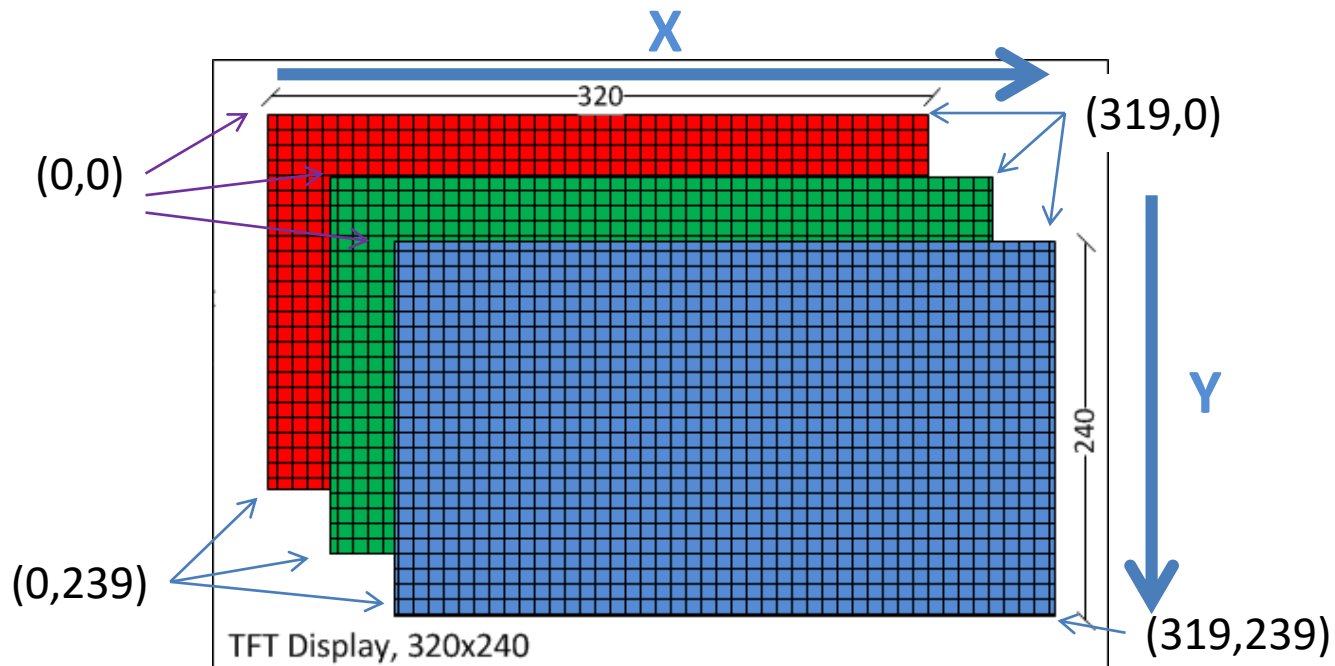
- `LCD_Initialization();` //Inicializa el FSMC y el display

- Acto seguido, limpiar la pantalla:

- `LCD_Clear (color);` //Limpia la pantalla en azul

Manejo del display

- **Rellenar un pixel:**
 - `LCD_SetPoint (x, y, color);`
- Las coordenadas del display (x, y):



Funciones para mostrar texto

- Mostrar un carácter suelto:
 - `LCD_PutChar (x, y, 'b', color, colorFondo);`
- Mostrar una cadena de texto:
 - `LCD_PrintText(x, y, "cadena",
color, colorFondo);`
- Se combinan normalmente con **sprintf** para incluir variables. **Ojo con el tamaño de la cadena vs. la pila de la tarea!**

```
char str[32];
uint16_t sensor;
//...
    sensor = readSensor();
    sprintf(str, "Valor: %d", sensor);
    LCD_PrintText(80,20, str, White, Black);
//...
```


Funciones gráficas

- Pintar una línea:

- `LCD_DrawLine(x1, y1, x2, y2, color)`

- Pintar un rectángulo:

- `LCD_DrawRectangle(x, y, ancho, alto,
pxborde, color)`

- Pintar un círculo completo con centro en (x, y):

- `LCD_DrawCircle(x, y, radio, color)`

- Rellenar una rectángulo:

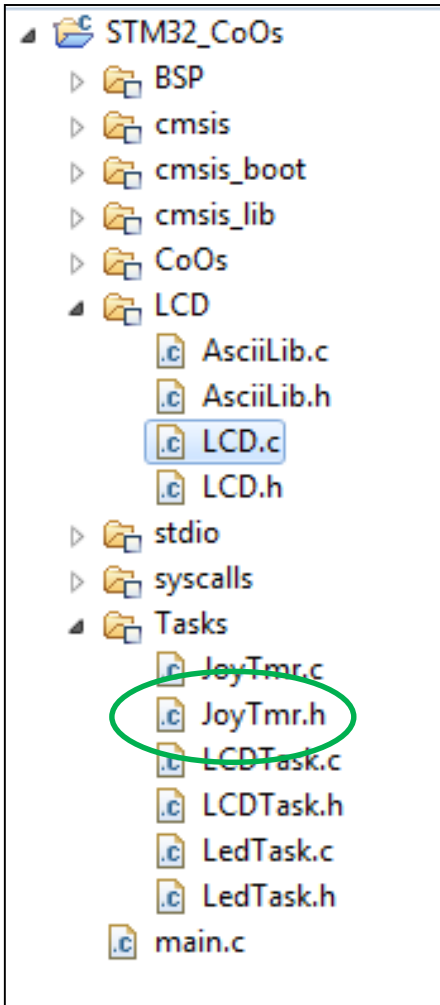
- `LCD_FillRectangle(x, y, ancho, alto, color)`

- Rellenar un círculo completo con centro en (x, y):

- `LCD_FillCircle (x, y, radio, color)`

Tarea para el control del display

- Inicializa y limpia el display.
- Crea una tarea controladora.



```
#define STACK_SIZE_LCD 1024          /*!< Define "taskA" task size */
OS_STK LCD_stk[2][STACK_SIZE_LCD]; /*!< Define "taskA" task stack */

void CreateLCDTask(void){
    Init_AnalogJoy();
    LCD_Initialization();
    LCD_Clear(Blue);
    CoCreateTask (LCDManagerTask,0,1,&LCD_stk[0][STACK_SIZE_LCD-1],STACK_SIZE_LCD);
}

void LCDManagerTask (void* pdata) {
    OS_TID lcdId;

    for (;;) {
        lcdId = CoCreateTask (LCDHelloWorldTask,0,1,&LCD_stk[1][STACK_SIZE_LCD-1],STACK_SIZE_LCD);
        waitForKey(5,0);
        CoDelTask(lcdId);

        lcdId = CoCreateTask (LCDGradientTask,0,1,&LCD_stk[1][STACK_SIZE_LCD-1],STACK_SIZE_LCD);
        waitForKey(5,0);
        CoDelTask(lcdId);

        lcdId = CoCreateTask (LCDDrawAreaTask,0,1,&LCD_stk[1][STACK_SIZE_LCD-1],STACK_SIZE_LCD);
        waitForKey(5,0);
        CoDelTask(lcdId);

        lcdId = CoCreateTask (LCDScopeTask,0,1,&LCD_stk[1][STACK_SIZE_LCD-1],STACK_SIZE_LCD);
        waitForKey(5,0);
        CoDelTask(lcdId);
    }
}
```

Tarea para el control del display

- Vamos a programar varias tareas para pintar las distintas pantallas. Cada tarea representa una pantalla.
- La idea es que se vayan ejecutando una a una estas tareas.
- Usamos la tarea **LCDManagerTask** para controlar la tarea que se está ejecutando, y poder cambiar de tarea/pantalla cuando se pulse el botón central del joystick.
- Se ha añadido una quinta bandera al timer del joystick, asociada al botón central.
- El proceso es el siguiente :
 - Se elimina la tarea actual y crea la siguiente.
 - **LCDManagerTask** se bloquea a la espera de una nueva pulsación del botón central: **WaitForKey(5, 0)**.
 - Una vez que ha mostrado todas las pantallas, vuelve a la primera.

Tarea de ejemplo

```
void LCDHelloWorldTask(void * parg){
    char str[32];
    uint16_t i=0;
    LCD_Clear(Red);
    LCD_PrintText(10,20, "Hola Mundo!", Blue, White);

    for(;;){
        sprintf(str,"Variable i: %d", i);
        LCD_PrintText(10,32,str,Blue,White);
        i++;
        CoTimeDelay(0,0,1,0);
    }
}
```

Ejercicio 1

- Modificar **LCDHelloWorldTask** para pintar los siguientes elementos antes de entrar en el bucle:
 - Rellenar rectángulo con origen 100, 100, y un tamaño de 100x20 de color amarillo.
 - Rellenar rectángulo con origen 150, 80, y un tamaño de 30x20 de color amarillo.
 - Dibujar un rectángulo simple en 155, 85 de 20x10, con 2 píxeles de borde, y color rojo.
 - Rellenar círculos:
 - En 120, 120 con radio 10 de color negro.
 - En 180, 120 con radio 10 de color negro.
 - En 120, 120 con radio 5 de color blanco.
 - En 180, 120 con radio 5 de color blanco.

Ejercicio 1

- Modificar el gráfico anterior para que cada 100mSeg. avance hacia la derecha 10 píxeles.

Ejercicio 2

- Modificar **LCDGradientTask** para pintar degradados.
- Rellenar las pantalla con **barras verticales de 10 píxeles de ancho** (rectángulos), de manera que cada una de ellas contenga **cada uno de los 32 tonos de rojo, dejando las otras componentes a 0**.
- Esperar 500mSeg, volver a rellenar la pantalla con barras verticales verdes (64 tonos).
- Repetir para el color azul (32 tonos).
- Usar función **RGB565CONVERT (R, G , B)**.



Ejercicio 2

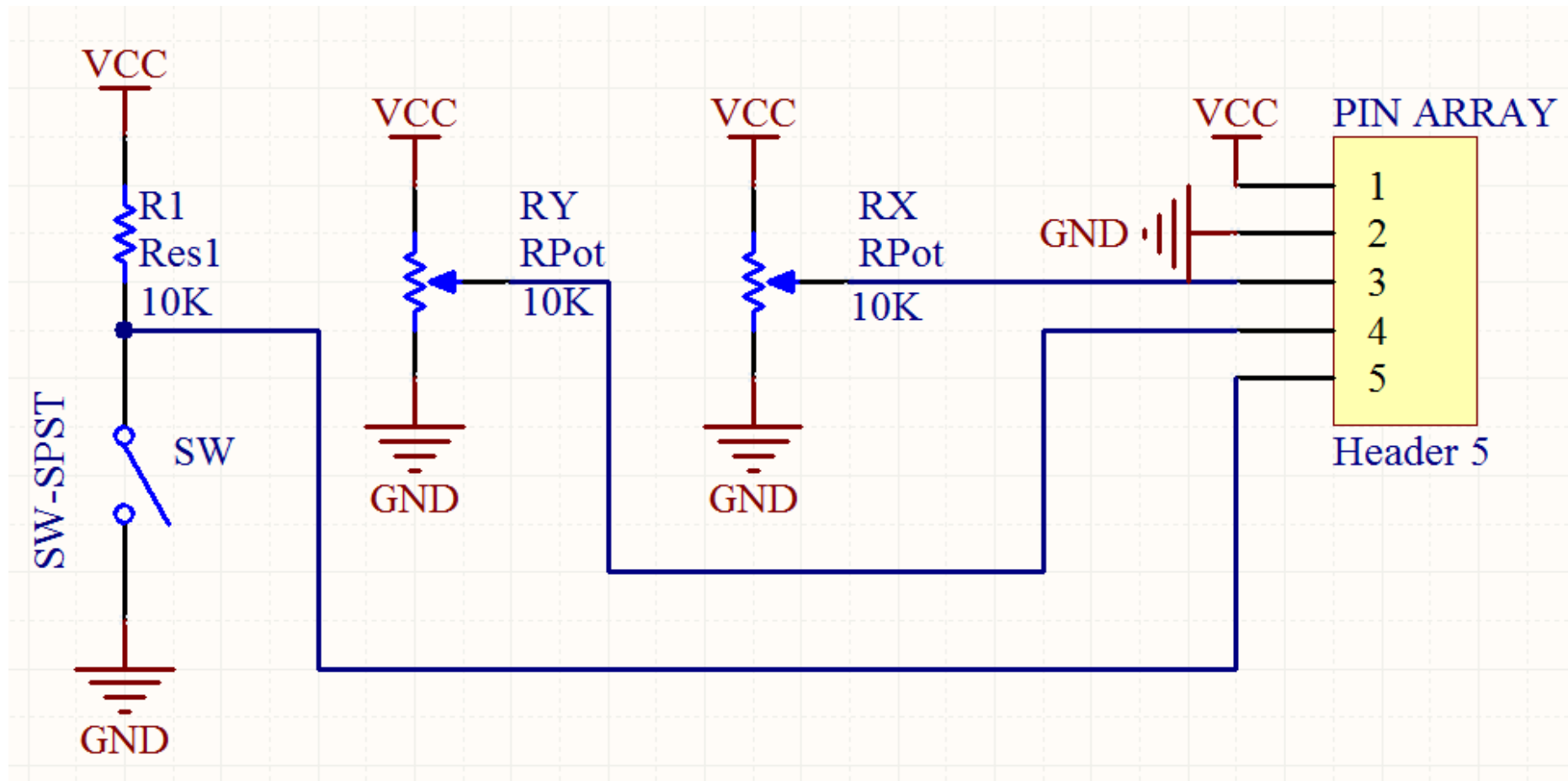
- Usar las funciones proporcionada por CoOs para medir cuanto tiempo tarda en pintarse cada uno de los gradientes.
- Mostrar el tiempo en pantalla, así como el tiempo que se toma en escribir cada pixel.

El Joystick Analógico

- Está compuesto por 2 potenciómetros conectados al stick.
- La rotación de cada potenciómetro depende de la posición del stick.
- El voltaje de salida es proporcional a la inclinación del stick.
- Además contiene un botón.

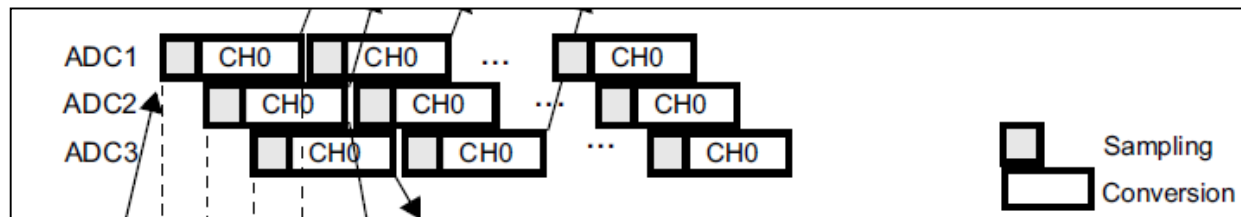


Esquema del Joystick Analógico

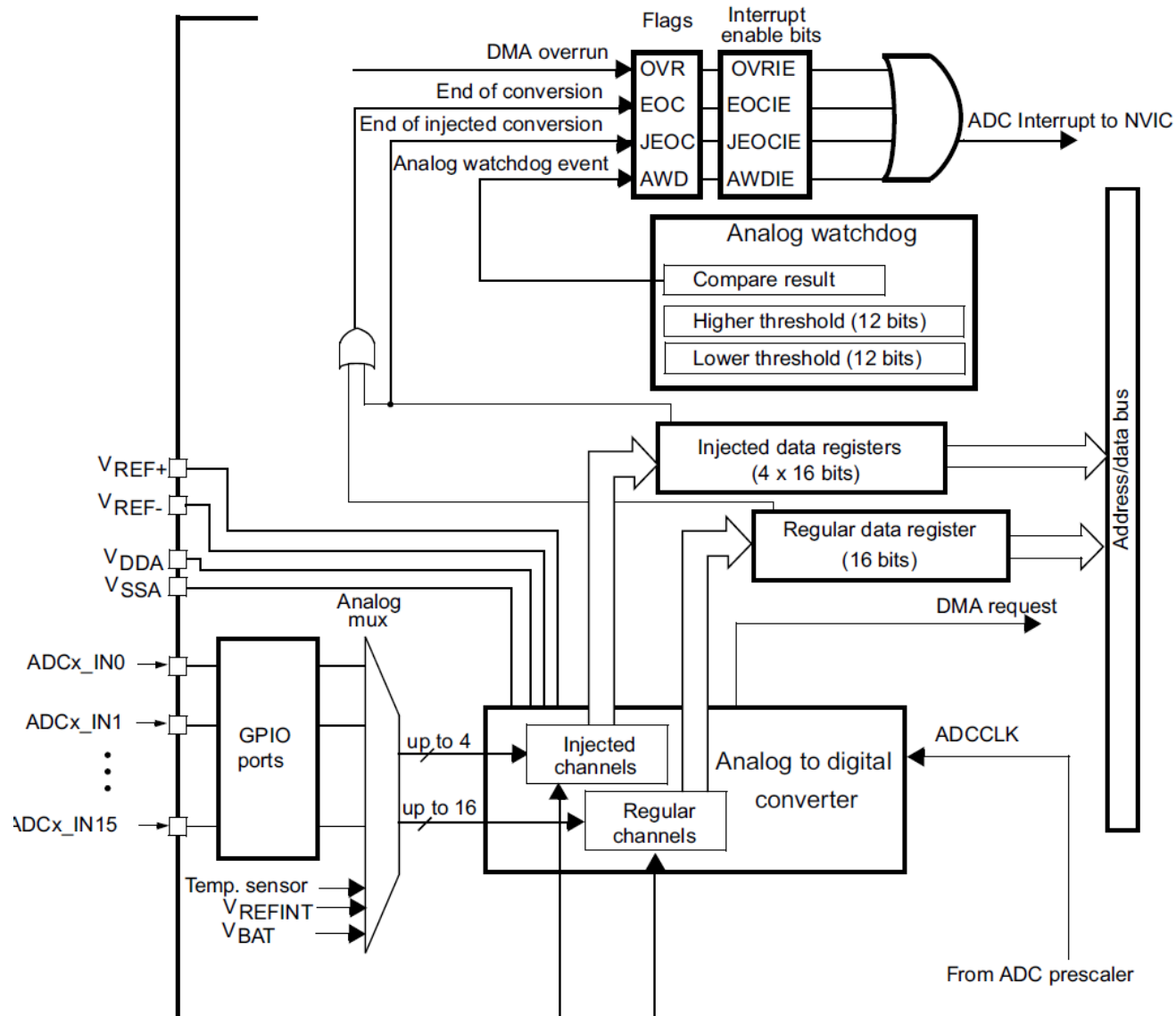


Los ADCs del STM32

- El STM32F4xx contiene 3 ADCs.
- Tienen una **resolución programable**, desde **6bits hasta 12bits**.
- Dependiendo de su modo de uso, el **tiempo de muestreo** está **entre 2.1 y 7.2 Msamples/Seg.**
- Multiplexores analógicos para **16 canales**.
- Modos automáticos de **escaneo y conversión continua**.
- **Recogida automática de muestras con DMA.**
- Los 3 ADCs pueden ser independientes, o hacer conversiones en cadena para aumentar la frecuencia de muestreo.



Arquitectura del ADC

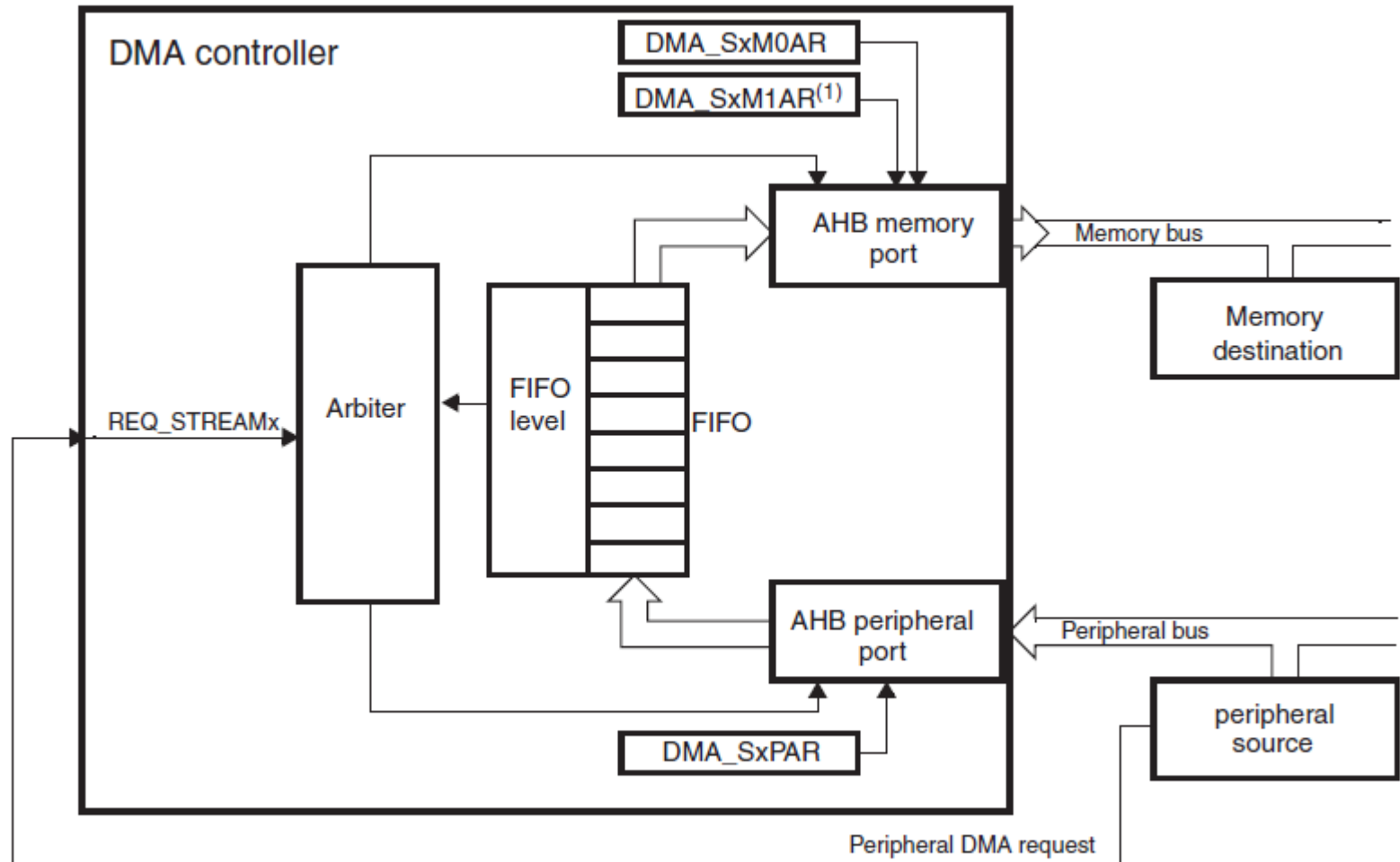


Digitalizando el Joystick Analógico

- **Las salidas del joystick X e Y se han conectado a PA6 y PA7 respectivamente.**
- **Para digitalizarlos vamos a usar el DMA.**
- **El ADC se ha configurado para que haga un escaneo continuo de PA6 y PA7.**
- **Cada vez que termina una conversión el DMA mueve automáticamente el resultado desde el registro datos del ADC a un vector, lanzándose la conversión del siguiente canal.**
- **El vector se ha declarado como una variable global, en la memoria RAM principal. Cada posición del vector corresponde a la información de un canal del ADC.**
- **El proceso es automático, sólo hay que consultar la variable global.**

DMA

Figure 28. Peripheral-to-memory mode



Configurando el ADC (1)

```
#define ADC1_DR_Address    (0x4001204C)
uint16_t ADC_ConvertedValue[2];

void Init_AnalogJoy(void)
{
    ADC_InitTypeDef        ADC_InitStructure;
    ADC_CommonInitTypeDef  ADC_CommonInitStructure;
    DMA_InitTypeDef        DMA_InitStructure;
    GPIO_InitTypeDef       GPIO_InitStructure;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2 | RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    /* DMA2 Stream0 channel0 configuration *****/
    DMA_InitStructure.DMA_Channel = DMA_Channel_0;
    DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)ADC1_DR_Address;
    DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&ADC_ConvertedValue;
    DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
    DMA_InitStructure.DMA_BufferSize = 2;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_High;
    DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
    DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
    DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
    DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
    DMA_Init(DMA2_Stream0, &DMA_InitStructure);
    DMA_Cmd(DMA2_Stream0, ENABLE);
}
```


Configurando el ADC (2)

```
/* Configure ADC1 Channel6 pin as analog input *****/
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
GPIO_Init(GPIOA, &GPIO_InitStructure);

/* ADC Common Init *****/
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
ADC_CommonInit(&ADC_CommonInitStructure);

/* ADC1 Init *****/
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ScanConvMode = ENABLE;
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 2;
ADC_Init(ADC1, &ADC_InitStructure);
```

Configurando el ADC (3)

```
/* ADC1 regular channe6 configuration *****/  
ADC_RegularChannelConfig(ADC1, ADC_Channel_6, 1, ADC_SampleTime_3Cycles);  
ADC_RegularChannelConfig(ADC1, ADC_Channel_7, 2, ADC_SampleTime_3Cycles);
```

```
/* Enable DMA request after last transfer (Single-ADC mode) */  
ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);  
  
/* Enable ADC1 DMA */  
ADC_DMACmd(ADC1, ENABLE);  
  
/* Enable ADC1 */  
ADC_Cmd(ADC1, ENABLE);
```

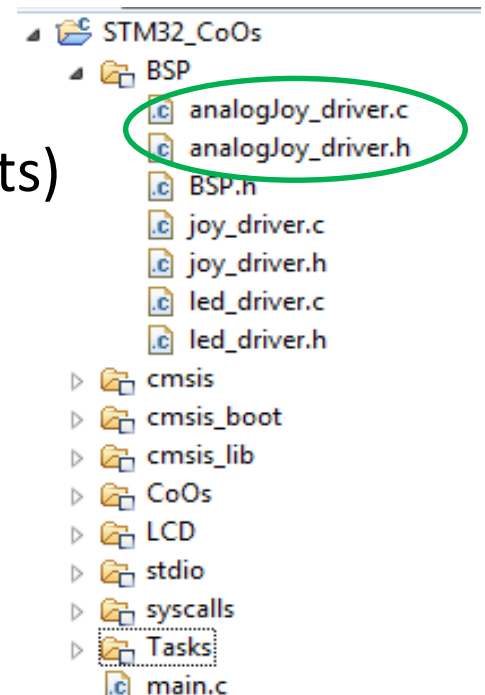
```
/* Start ADC1 Conversions */  
ADC_SoftwareStartConv(ADC1);
```

```
}
```

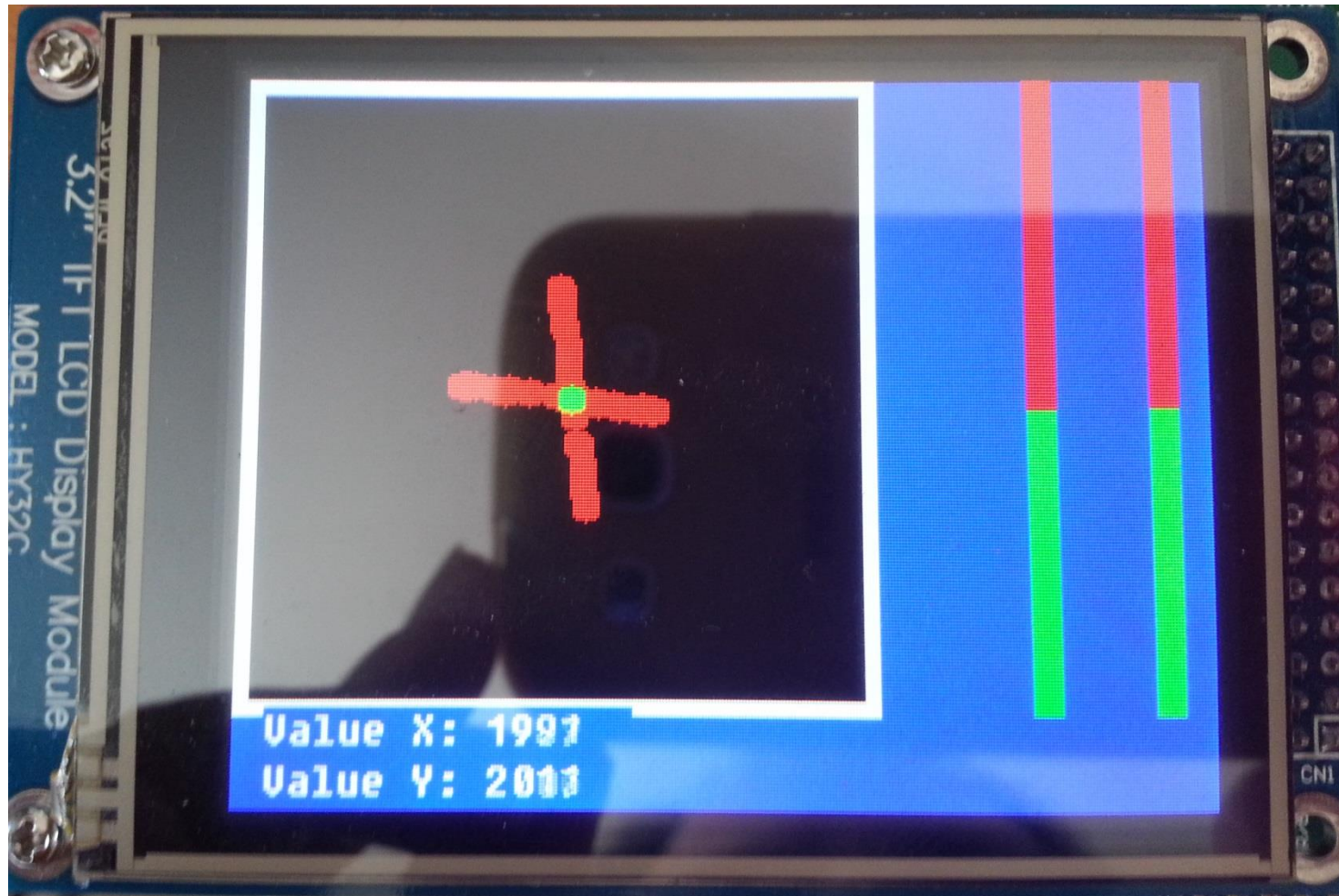
Driver del Joystick analógico

- `void Init_AnalogJoy (void)`
 - Inicializa el GPIO, el ADC, y el DMA.
- `uint16_t getAnalogJoy (uint8_t axxis)`
 - Devuelve el valor del eje pasado como parámetro:
 - 0: Eje X
 - 1: Eje Y
 - El valor devuelto está entre 0 y 4096 (12bits)

Izquierda	Centro	Derecha
4000-4080	2000	30 – 100
Arriba	Centro	Abajo
4000-4080	2000	30 – 100



Ejercicio 3: Barras y área de pintado

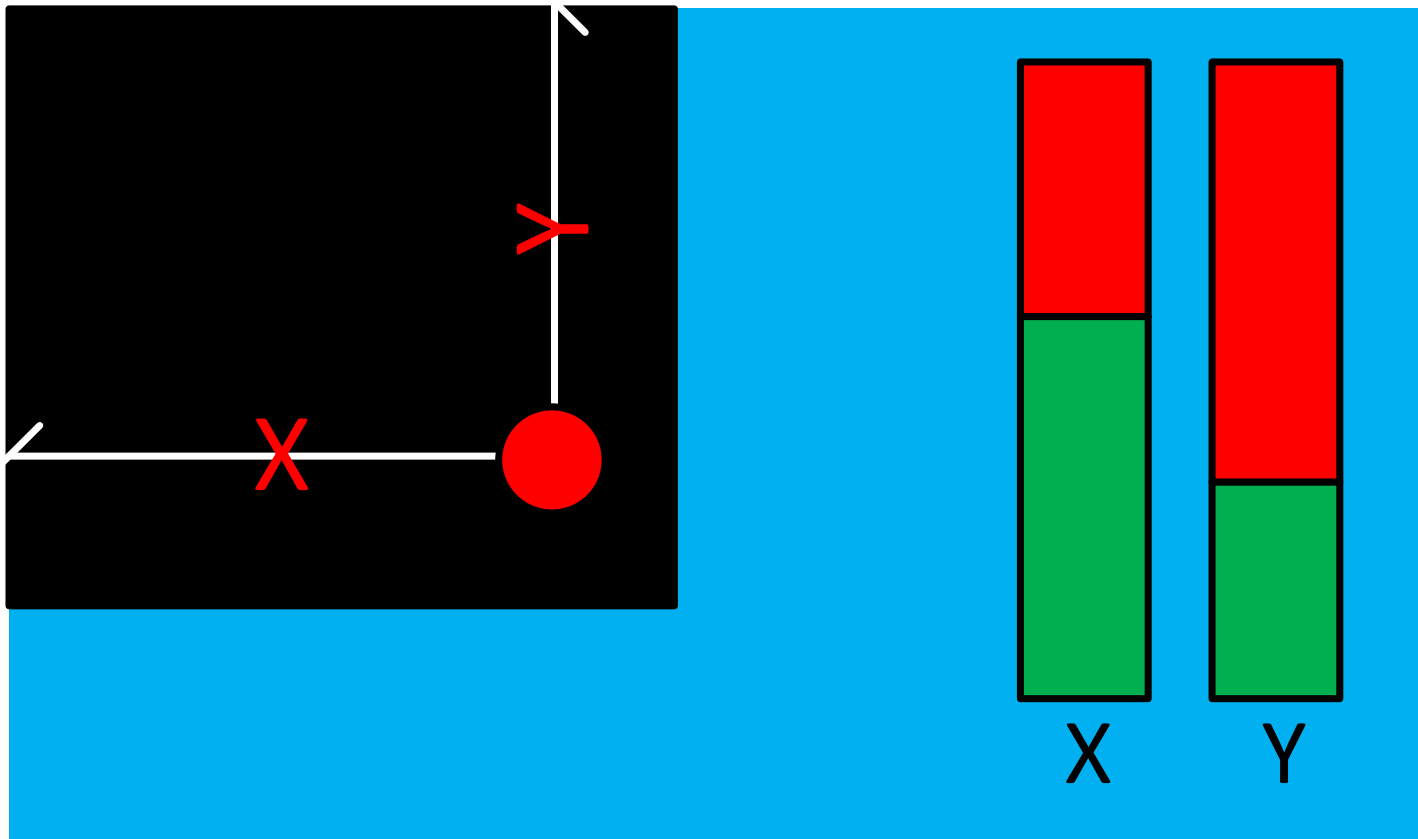


Ejercicio 3

- Modificar **LCDDrawAreaTask** para mostrar en pantalla el valor de ambos ejes como texto y con una tasa de refresco de 200mSeg.
- Usar `getAnalogJoy(axxis)` para leer los valores de los ejes del joystick
- Imprimir en pantalla:
 - Declarar cadena de texto: `char cad[32];`
 - Formatear la cadena usando `sprintf`
 - Imprimirla en el display con `LCD_PrintText`
- **Consejo:** al final de la cadena añadir dos espacios en blanco.

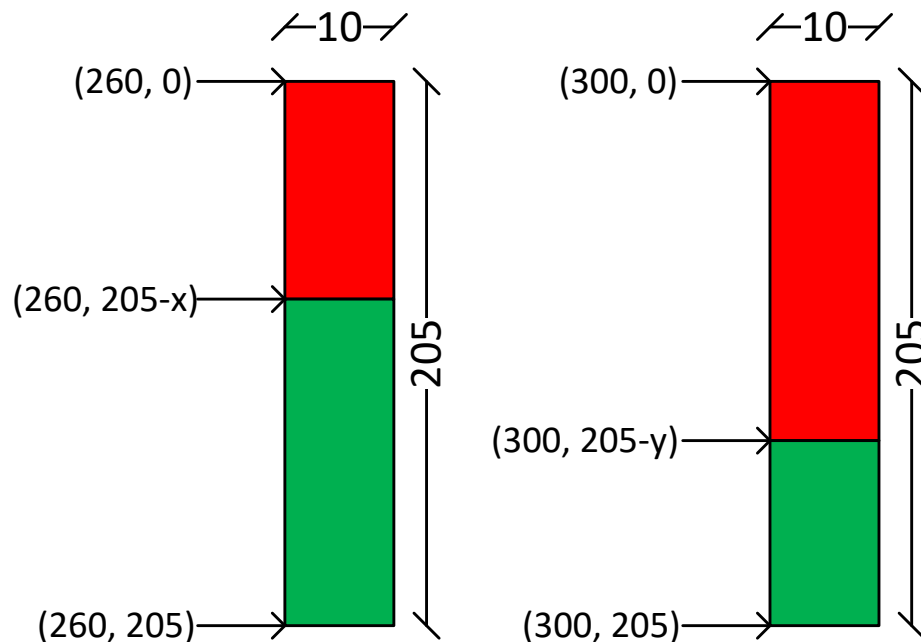
Ejercicio 3

- **3 Elementos:**



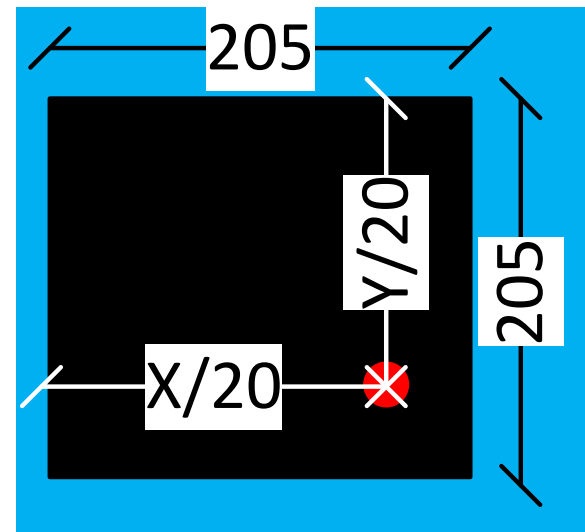
Ejercicio 3-1: Barras verticales

- **Pintar dos barras verticales con dos colores:**
 - Con una altura total del 205 pix. y anchura de 10 pix.
 - Los colores serán rojo y verde, y la cantidad de cada uno debe ser proporcional a cada eje del joystick.
 - El valor de cada eje tiene un rango entre [0, 4095].
 - Si los dividimos entre 20 nos queda un rango entre [0, 205]



Ejercicio 3-2: Área de pintura

- **Área de pintado libre:**
 - Rellenar un rectángulo desde (0, 0) de 205 x 205 de color negro **antes de entrar en el bucle de la tarea.**
 - Pintar un círculo relleno de algún color en coordenadas proporcionales a los ejes del joystick, con un radio de 5 píxeles.

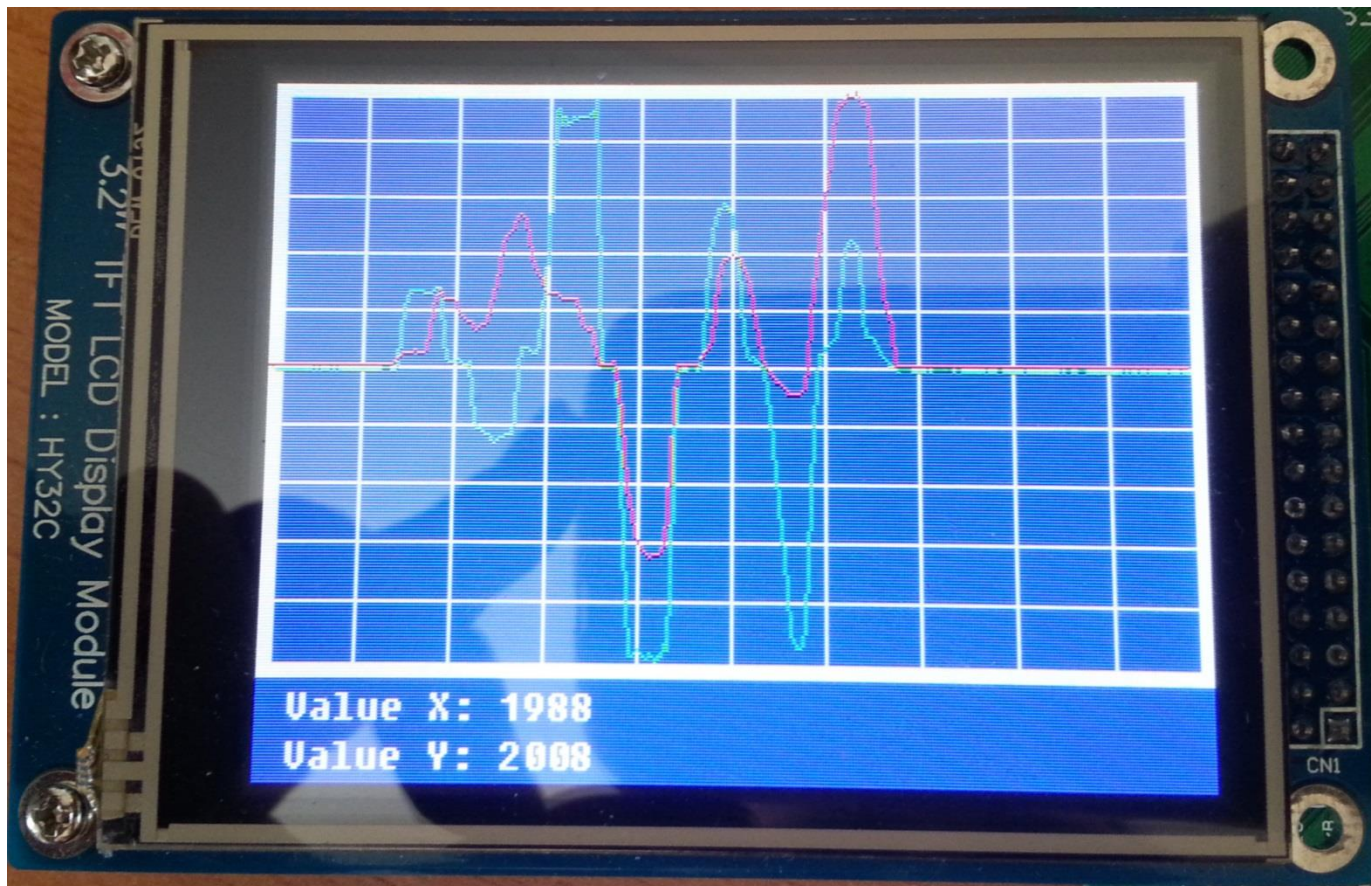


Ejercicio 3-3: Área de pintura

- Comprobar el sentido de cada eje respecto al área de pintura.
- ¿Qué está ocurriendo?
- Modificar la tarea para que el círculo del área de pintura siga la trayectoria real del joystick.

Ejercicio 4: Osciloscopio

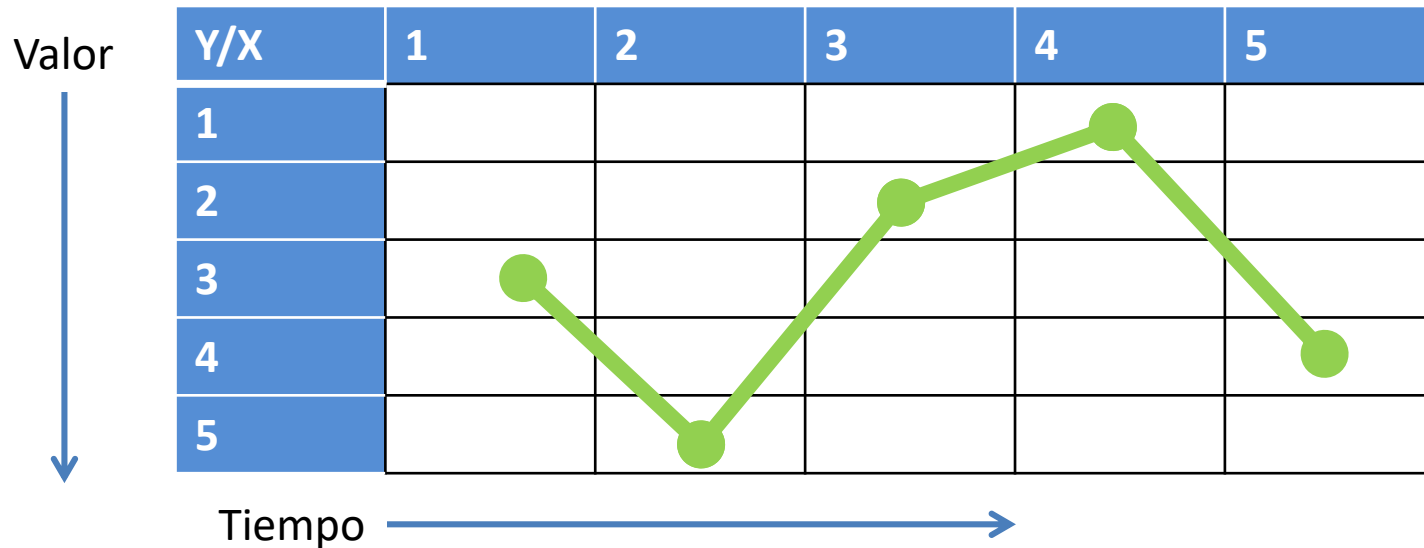
- Modificar la tarea **LCDScopeTask** para que simule un osciloscopio de dos canales.



Ejercicio 4: Osciloscopio

- Visualizar el eje X del joystick analógico en el tiempo.
- Las coordenadas x de cada línea representarán el tiempo, y las coordenadas y han de ser proporcionales al valor del eje.
- Incrementar la coordenada x con el tiempo.
- Pintar una línea entre el valor actual del eje y el último valor leído en el instante de tiempo anterior.
- Usar un área de visualización de 320 x 205.

Ejercicio 4: Osciloscopio



- Tiempo = 1, Valor = 3
- Tiempo = 2, Valor = 5
- Tiempo = 3, Valor = 2
- Tiempo = 4, Valor = 1
- Tiempo = 5, Valor = 4

Ejercicio 5: Osciloscopio

1. Modificar la tarea anterior para controlar el desborde de la pantalla.
2. Modificar la tarea anterior para muestre ambos ejes del joystick analógico.
3. Añadir un marco y una cuadrícula con 10 líneas horizontales y 10 líneas verticales homogéneamente repartidas.