

# Aplicaciones de Sistemas Embebidos con Doble Núcleo

## Fundamentos de Programación para Sistemas Embebidos



UTN FRA  
Departamento de Ingeniería Electrónica  
Laboratorio de Sistemas Embebidos

1 de agosto de 2024

### 1 Periféricos

- Generalidades
- Configuración de pin

### 2 GPIO

- Configuración de entrada
- Configuración de salida

### 3 ADC

- Generalidades
- Uso del ADC

- Lectura de secuencia

### 4 DAC

- Generalidades
- Uso del DAC

### 5 SysTick Timer

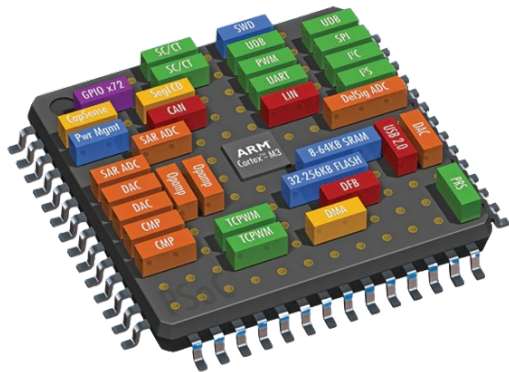
- Generalidades
- Uso del SysTick Timer

### 6 Ejercicios

### 7 Referencias

# Fundamentos de Programación para Sistemas Embebidos

## Generalidades



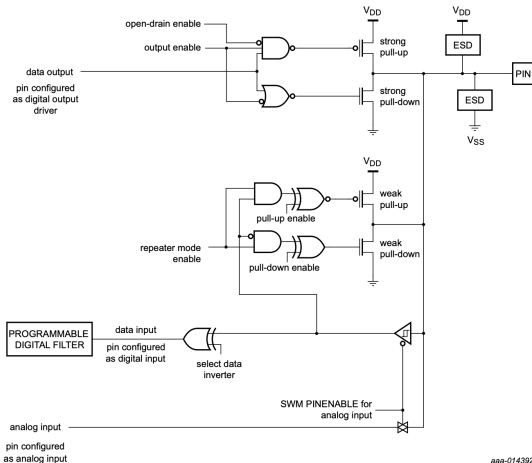
## Periféricos comunes

- General Purpose Input Output (GPIO)
- Analog to Digital Converter (ADC)
- Digital to Analog Converter (DAC)
- System Timer (Systick) y otros Timers
- Universal Synchronous Asynchronous Receiver Transmitter (USART)
- Inter-Integrated Circuit (I2C)
- Serial Peripheral Interface (SPI)
- Universal Serial Bus (USB)

# Fundamentos de Programación para Sistemas Embebidos

## Configuración de pin

Cada pin tiene múltiples funciones asociadas. Se selecciona solo una de ellas de acuerdo a lo que se defina en la SWM y el IOCON.



aaa-014392

### Uso típico

---

```
// Estructura de configuracion para entrada
gpio_pin_config_t in_config = { kGPIO_DigitalInput };

// Habilito el puerto 0
GPIO_PortInit(GPIO, 0);

// Configuro el pin 4 como entrada
GPIO_PinInit(GPIO, 0, 4, &in_config);

// Leo el estado del pin 4
uint32_t state = GPIO_PinRead(GPIO, 0, 4);
```

- Se usa la estructura del tipo `gpio_pin_config_t` con el valor `kGPIO_DigitalInput`.
- La función `GPIO_PortInit` permite habilitar el numero de puerto indicado.
- La función `GPIO_PinInit` configura un pin en particular.
- Se usa `GPIO_PinRead` para ver el estado del pin que será high (1) o low (0).

### Uso típico

---

- Se usa la estructura del tipo `gpio_pin_config_t` con el valor `kGPIO_DigitalOutput`.

- Se puede en la estructura anterior, dar un valor de salida por defecto.

- Se usa `GPIO_PinWrite` para escribir un valor en el pin que será high (1) o low (0).

```
// Configuración para salida, salida en 0 por defecto  
gpio_pin_config_t out_config = { kGPIO_DigitalOutput, 0 };
```

```
// Habilito el clock del GPIO 1  
GPIO_PortInit(GPIO, 1);
```

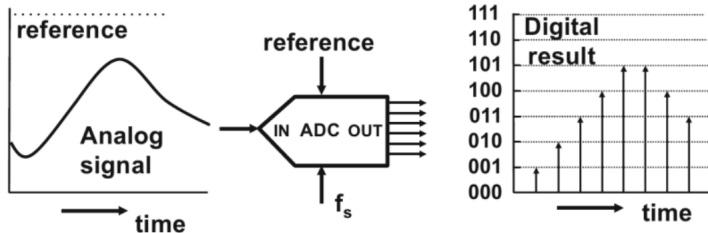
```
// Configuro el pin 0 del GPIO 1 como salida  
GPIO_PinInit(GPIO, 1, 0, &out_config);
```

```
// Escribo un 1 en el pin 0  
GPIO_PinWrite(GPIO, 1, 0, 1);
```

# Fundamentos de Programación para Sistemas Embebidos

## Generalidades del ADC

- Convierten señales analógicas a valores discretos.
- Una resolución de 12 bits implica  $2^{12}$  o 4096 valores discretos de resultado para una señal de 0 a 3,3V.
- Un total de 12 canales multiplexados.
- Velocidad de muestreo de 1,2Ms/s.



# Fundamentos de Programación para Sistemas Embebidos

## Inicialización del ADC

- Se elige la entrada analógica para el pin con `SWM_SetFixedPinSelect`.
- Se elige la fuente de clock y prescaler del ADC con `CLOCK_Select` y `CLOCK_SetClkDivider`.
- Se alimenta el hardware del ADC con `POWER_DisablePD`.
- Una calibración de clock es necesaria antes de inicializar el ADC, esto se hace con `ADC_DoSelfCalibration`.

```
// Habilito funcion analogica para el canal 5 (P0 21)  
CLOCK_EnableClock(kCLOCK_Swm);  
SWM_SetFixedPinSelect(SWM0, kSWM_ADC_CHN5, true);  
CLOCK_DisableClock(kCLOCK_Swm);
```

```
// Elijo clock desde el FRO con divisor de 1  
CLOCK_Select(kADC_Clk_From_Fro);  
CLOCK_SetClkDivider(kCLOCK_DivAdcClk, 1);
```

```
// Prendo el ADC  
POWER_DisablePD(kPDRUNCFG_PD_ADC0);
```

```
// Obtengo frecuencia deseada y calibro ADC  
uint32_t freq = CLOCK_GetFreq(kCLOCK_Fro);  
ADC_DoSelfCalibration(  
    ADC0,  
    CLOCK_GetClkDivider(kCLOCK_DivAdcClk)  
);
```



# Fundamentos de Programación para Sistemas Embebidos

## Inicialización de una secuencia

```
// Configuración por defecto del ADC
adc_config_t adc_config;
ADC_GetDefaultConfig(&adc_config);
ADC_Init(ADC0, &adc_config);

// Configuro y habilito secuencia A
adc_conv_seq_config_t adc_sequence = {
    .channelMask = 1 << 5, // Canal 5 habilitado
    .triggerMask = 0,
    .triggerPolarity = kADC_TriggerPolarityPositiveEdge,
    .enableSyncBypass = false,
    .interruptMode = kADC_InterruptForEachConversion
};

ADC_SetConvSeqAConfig(ADC0, &adc_sequence);
ADC_EnableConvSeqA(ADC0, true);
```

- Para por defecto el ADC usamos `ADC_GetDefaultConfig` y `ADC_Init`.
- Las secuencias son conversiones continuas de uno o muchos canales. Se configuran el/los canales en el campo `channelMask` de la estructura `adc_conv_seq_config_t`.
- Se pueden configurar fuentes de disparo y si las interrupciones son por canal o secuencia.
- La secuencia se configura y habilita con `ADC_SetConvSeqAConfig` y `ADC_EnableConvSeqA`.

# Fundamentos de Programación para Sistemas Embebidos

## Lectura de una secuencia

- Se inicia una secuencia de conversiones con `ADC_DoSoftwareTriggerConvSeqA`.
- La función que devuelve el resultado de la conversión cuando está listo es `ADC_GetChannelConversionResult` y requiere una estructura del tipo `adc_result_info_t`.
- El valor del resultado se encuentra en el campo `result` de la estructura.

```
// Inicio conversion por software
ADC_DoSoftwareTriggerConvSeqA(ADC0);

// Estructura de resultado de la conversion
adc_result_info_t adc_info;

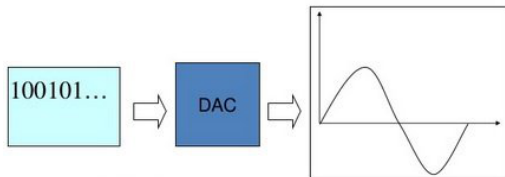
// Espero a terminar la conversion
while(!ADC_GetChannelConversionResult(
    ADC0, 5, &adc_info
));

// Resultado de la conversion
uint16_t adc_result = adc_info.result;
```

# Fundamentos de Programación para Sistemas Embebidos

## Generalidades del DAC

- Convierten valores digitales en un registro a valores de tensión analógicos.
- Una resolución de 10 bits implica  $2^{10}$  o 1024 valores analógicos posibles entre 0 y 3,3V.
- Dos salidas posibles con una tasa de refresco de hasta 1 MHz.



# Fundamentos de Programación para Sistemas Embebidos

## Uso del DAC

```
// Configuro la salida del DAC al P0 17
CLOCK_EnableClock(kCLOCK_Swm);
SWM_SetFixedPinSelect(SWM0, kSWM_DAC_OUT0, true);
CLOCK_DisableClock(kCLOCK_Swm);

// Habilito la funcion de DAC en el P0 17
CLOCK_EnableClock(kCLOCK_Iocon);
IOCON_PinMuxSet(IOCON, 0, IOCON_PIO_DACMODE_MASK);
CLOCK_DisableClock(kCLOCK_Iocon);

// Prendo el DAC
POWER_DisablePD(kPDRUNCFG_PD_DAC0);

// Configuro el DAC con 1us de refresco
dac_config_t dac_config = { kDAC_SettlingTime1us };
DAC_Init(DAC0, &dac_config);

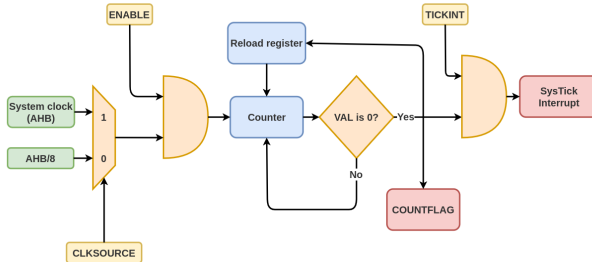
// Escribo un valor en el DAC (1,65V de salida)
DAC_SetBufferValue(DAC0, 512);
```

- Se usan funciones como [SWM\\_SetFixedPinSelect](#) y [IOCON\\_PinMuxSet](#) para elegir la función de DAC para un pin.
- El hardware DAC tiene que alimentarse con [POWER\\_DisablePD](#).
- Se configura el funcionamiento del DAC con [DAC\\_Init](#) a partir de alguna estructura de configuración.
- Se usa [DAC\\_SetBufferValue](#) para indicar un valor de 10 bits que se desea a la salida del DAC.

# Fundamentos de Programación para Sistemas Embebidos

## Generalidades del SysTick Timer

- 1 Estándar en todos los Cortex-M y generan interrupciones cuando llega a cero el contador.
- 2 Se pueden cargar valores de hasta  $2^{24}$  en el contador y el valor cargado responde a  $ticks = T_{systick} \times f_{systick}$ .



# Fundamentos de Programación para Sistemas Embebidos

## Configuración del SysTick Timer

```
// Configuro SysTick para 1 ms
SysTick_Config(SystemCoreClock / 1000);

/**
 * @brief Handler para interrupcion de SysTick
 */
void SysTick_Handler(void) {
    // Esta interrupcion se llama cada 1 ms

    // TODO
}
```

- Se puede usar la variable `SystemCoreClock` para independizarse del valor de clock ya que esta variable tiene el valor del clock del CPU.
- Se puede dividir por la frecuencia deseada del SysTick ( $1ms = 1/1000$ ) para evitar el uso de números con decimales.
- Cada vez que se vence el SysTick se llama al handler de una interrupción llamada `SysTick_Handler`. Esta función se define por fuera del main.

### Algunas propuestas para practicar

---

- 1 En un proyecto llamado **02\_systick\_blinky**, hacer un programa en el que el LED Azul parpadee cada 500ms y el LED D1 cada 1500ms.
  - 2 En un proyecto llamado **02\_custom\_blinky**, hacer un programa que haga que el LED Rojo parpadee de 100ms a 2s de acuerdo a lo indicado por el potenciómetro R22.
  - 3 En un proyecto llamado **02\_sine\_wave\_generator**, generar una senoidal de 1KHz de la mayor amplitud posible, con un offset de 1,65V en la salida PIO0\_29.
- 

Cada ejercicio que se resuelva, subirlo al repositorio personal del curso.

### Algunos recursos útiles

---

- Manual del LPC845
- Manual del LPC845 Breakout Board
- Documentación del SDK del LPC845
- Esquemático del kit del laboratorio
- Analog to Digital Converter (ADC) Basics
- SysTick Timer Armv7-M System Tick Timer