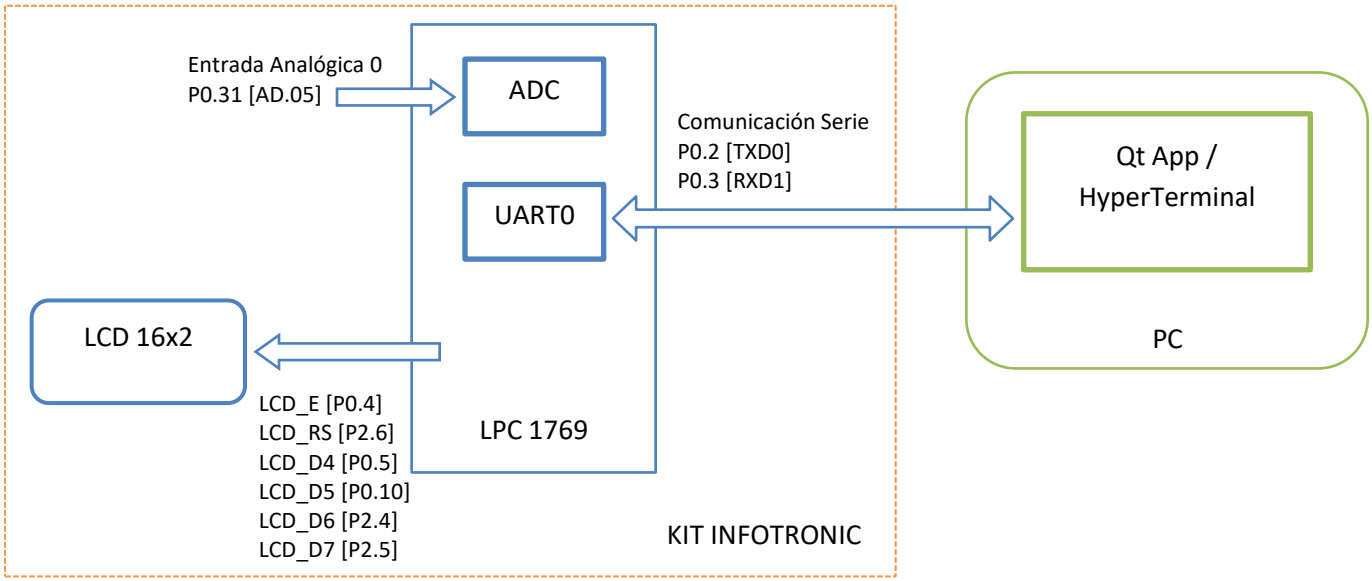
EJEMPLO DE APLICACIÓN - ADC + UART + LCD - INFORMÁTICA II - UTN FRBA

1. OBJETIVOS

- Establecer una Comunicación Serie entre el LPC1769 y la PC.
- Definir un formato de trama para la transmisión de datos (protocolo de comunicación).
- Utilizar el Conversor Analógico Digital (ADC) para obtener muestras de una señal.
- Controlar desde la PC el inicio y detención de la toma de muestras del ADC.
- Transmitir los valores obtenidos a la PC.
- Visualizar en el LCD el estado del sistema y el valor actual de la conversión del ADC.

2. DIAGRAMA EN BLOQUES

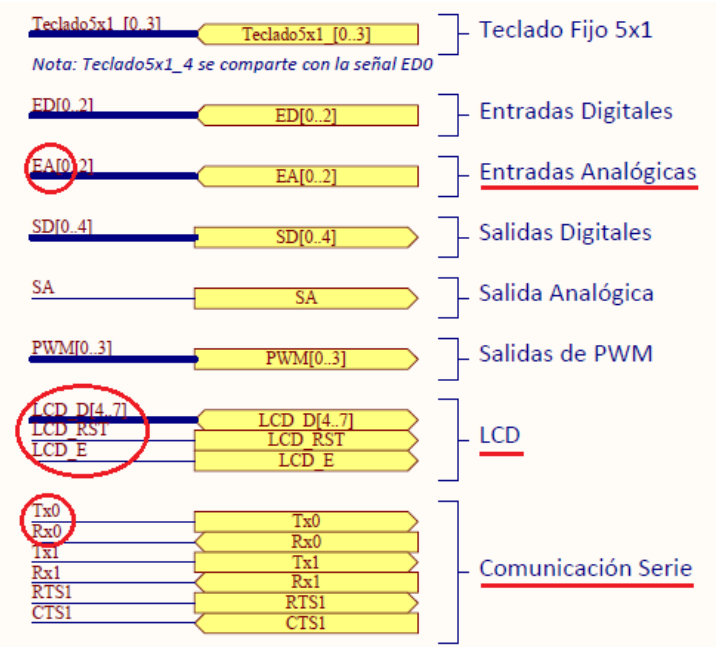


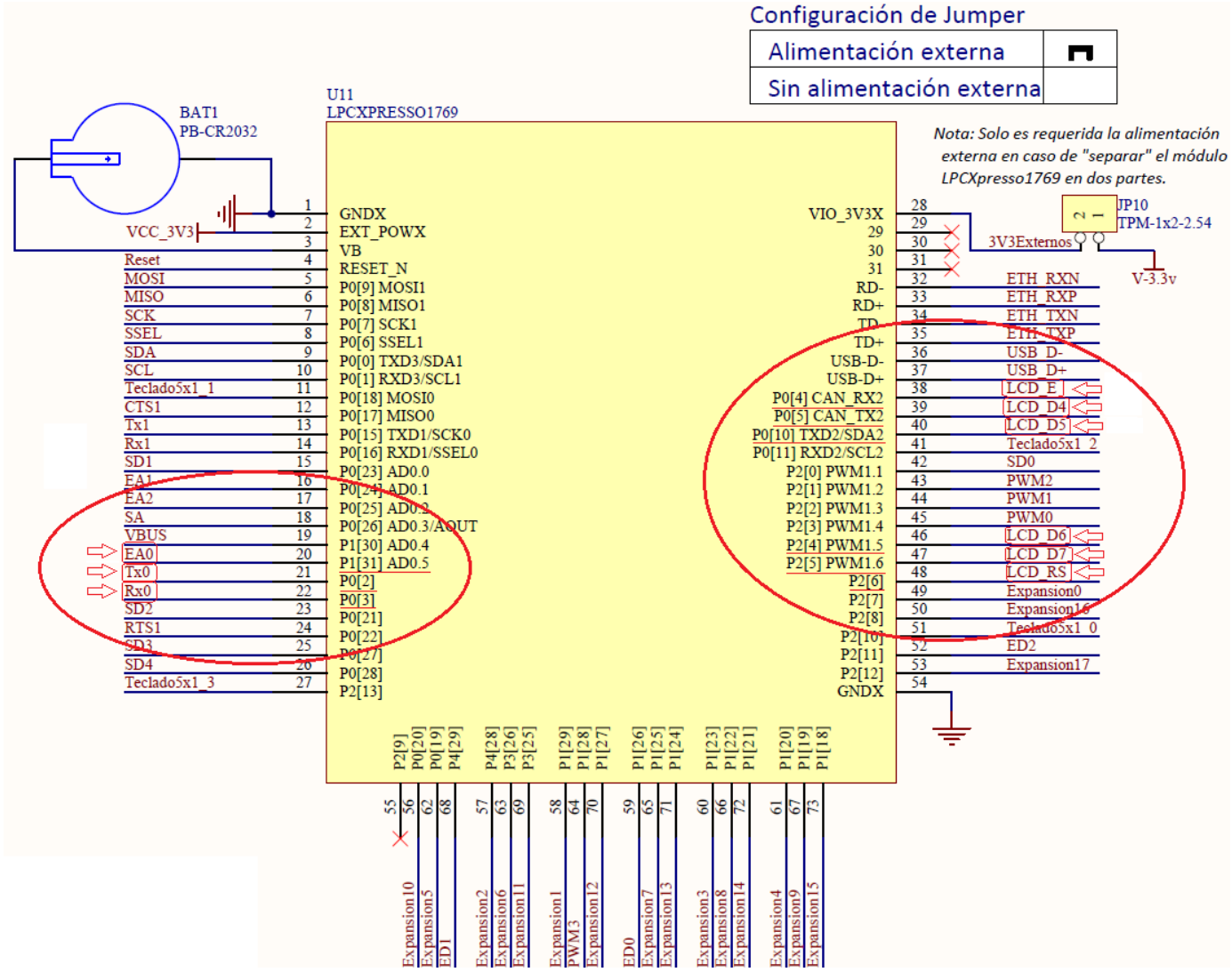
3. HARDWARE UTILIZADO (puntos de interés en el kit)

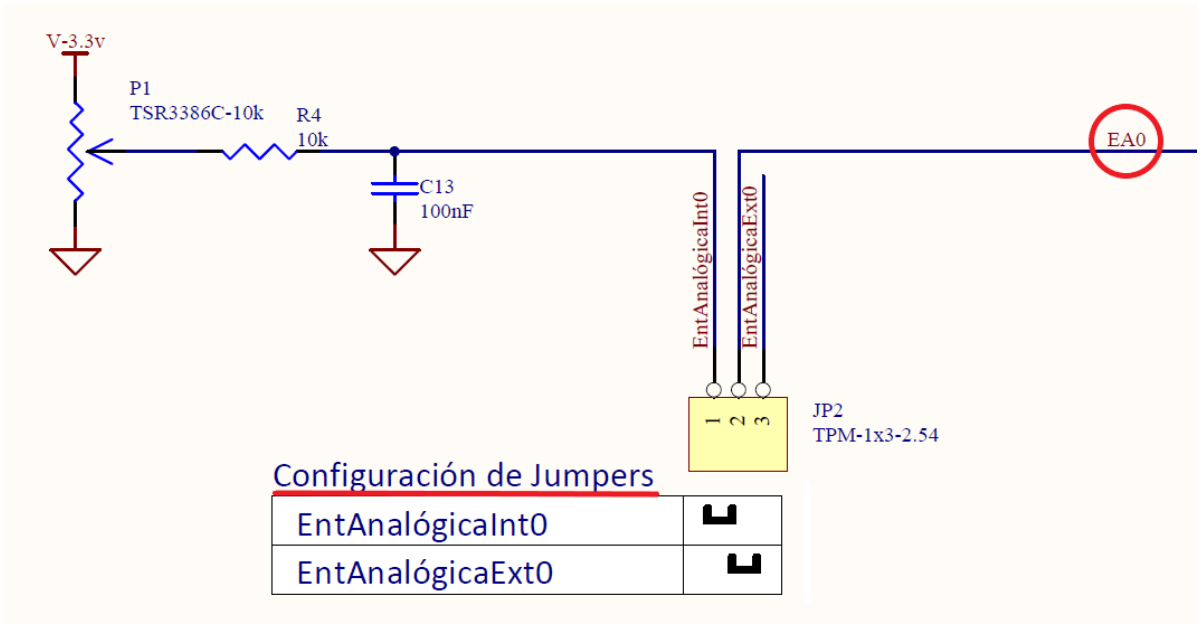
> Entrada Analógica (EA0)
EA0 -> P0.31 [AD0.5]
(canal 5 del ADC)

> Comunicación Serie (UART0)
Tx0 -> P0.2 [TXD0]
Rx0 -> P0.3 [RXD0]

> LCD (salidas GPIO)
LCD_E -> P0.4
LCD_RS -> P2.6
LCD_D4 -> P0.5
LCD_D5 -> P0.10
LCD_D6 -> P2.4
LCD_D7 -> P2.5




Configuración de Jumper
Alimentación externa
Sin alimentación externa
BAT1
PB-CR2032
U11
LPCXPRESSO1769
Nota: Solo es requerida la alimentación externa en caso de "separar" el módulo LPCXpresso1769 en dos partes.
JP10
TPM-1x2-2.54
3V3Externos
V-3.3v
VCC_3V3
GNDX
EXT_POWX
VB
RESET_N
P0[9] MOSI1
P0[8] MISO1
P0[7] SCK1
P0[6] SSEL1
P0[0] TXD3/SDA1
P0[1] RXD3/SCL1
P0[18] MOSI0
P0[17] MISO0
P0[15] TXD1/SCK0
P0[16] RXD1/SSEL0
P0[23] AD0.0
P0[24] AD0.1
P0[25] AD0.2
P0[26] AD0.3/AOUT
P1[30] AD0.4
P1[31] AD0.5
P0[2]
P0[3]
P0[21]
P0[22]
P0[27]
P0[28]
P2[13]
VIO_3V3X
RD-
RD+
TD-
TD+
USB-D-
USB-D+
P0[4] CAN_RX2
P0[5] CAN_TX2
P0[10] TXD2/SDA2
P0[11] RXD2/SCL2
P2[0] PWM1.1
P2[1] PWM1.2
P2[2] PWM1.3
P2[3] PWM1.4
P2[4] PWM1.5
P2[5] PWM1.6
P2[6]
P2[7]
P2[8]
P2[10]
P2[11]
P2[12]
GNDX
Reset
MOSI
MISO
SCK
SSEL
SDA
SCL
Teclado5x1_1
CTS1
Tx1
Rx1
SD1
EA1
EA2
SA
VBUS
EA0
Tx0
Rx0
SD2
RTS1
SD3
SD4
Teclado5x1_3
ETH_RXN
ETH_RXP
ETH_TXN
ETH_TXP
USB_D-
USB_D+
LCD_E
LCD_D4
LCD_D5
Teclado5x1_2
SD0
PWM2
PWM1
PWM0
LCD_D6
LCD_D7
LCD_RS
Expansion0
Expansion16
Teclado5x1_0
ED2
Expansion17
P2[9] P0[20] P0[19] P4[29] P4[28] P3[26] P3[25] P1[29] P1[28] P1[27] P1[26] P1[25] P1[24] P1[23] P1[22] P1[21] P1[20] P1[19] P1[18]
55 56 62 68 57 63 69 58 64 70 59 65 71 60 66 72 61 67 73
Expansion10 Expansion5 ED1 Expansion2 Expansion6 Expansion11 Expansion1 PWM3 Expansion12 ED0 Expansion7 Expansion13 Expansion3 Expansion8 Expansion14 Expansion4 Expansion9 Expansion15


V-3.3v
P1
TSR3386C-10k
R4
10k
C13
100nF
EA0
EntAnalógicaInt0
EntAnalógicaExt0
JP2
TPM-1x3-2.54
Configuración de Jumpers
EntAnalógicaInt0
EntAnalógicaExt0

4. DESARROLLO

Comunicación Serie (UART0)

Se eligió la siguiente configuración: velocidad 9600bps - 8 bits de DATOS - paridad PAR (EVEN) - 1 bit de STOP

```
void UART0_Init (void)
{
	//1.- Registro PCONP - bit 3 en 1 habilita la UART0
	PCONP |= 0x01 << 3;
	//2.- Registro PCLKSEL0 - bits 6 y 7 en 0 seleccionan que el clk de la UART0 sea 25MHz
	PCLKSEL0 &= ~(0x03 << 6);
	//3.- Registro U0LCR - trama de 8 bits, 1 bit de stop, paridad par, sin break cond. (DLAB = 1)
	U0LCR = 0x9B;
	//4.- Registros U0DLL (0x40010000) y U0DLM (0x40010004) - 9600 baudios
	U0DLM = 0;
	U0DLL = 0xA3;
	//5.- Habilito las funciones especiales de los pines TX0 y RX0
	//TX0D : PIN ?? -> P0[2]
	SetPINSEL(P0, 2, PINSEL_FUNC1);
	//RX0D : PIN ?? -> P0[3]
	SetPINSEL(P0, 3, PINSEL_FUNC1);
	//6.- Registro U0LCR, pongo DLAB(bit7) en 0
	U0LCR &= ~(0x01 << 7);
	//7. Habilito las interrupciones de RX y TX en la UART0 (Registro U0IER)
	U0IER = 0x03;
	//8. Habilito la interrupción de la UART0 en el NVIC (Registro ISER0)
	ISER0 |= (1<<5);
}
```

Rutina de atención de interrupción de la UART0:

```
void UART0_IRQHandler (void)
{
	uint8_t iir, dato;

	do {
		//Para limpiar los flags de IIR hay que leerlo, una vez que lo leí se resetea.
		iir = U0IIR;
		//THRE (Interrupción por TX)
		if (iir & 0x02) {
			if(!PopTx(&dato))
				U0THR = dato;  // hay un dato en el bufferTx para enviar
			else
				UART0_txEnCurso = 0; // si no hay más datos a enviar, limpio el flag
		}
		//Data Ready (Interrupción por RX)
		if ( iir & 0x04 ) {
			PushRx((uint8_t)U0RBR);	// guardo el dato recibido en el bufferRx
		}
	} while(!(iir & 0x01));		/* me fijo si cuando entré a la ISR había otra
					int. pendiente de atención: b0=1 (ocurre únicamente
					si dentro del mismo espacio temporal llegan dos
					interrupciones a la vez) */
}
```

Función para forzar la transmisión del primer dato (cuando no hay una TX en curso):

```
// Flag de TX en curso
volatile uint8_t UART0_txEnCurso;

void UART0_StartTx(void)
{
	uint8_t dato;

	if(!PopTx(&dato))
		U0THR = dato;  //fuerzo la transmisión del primer dato
}
```

Funciones Primitivas: PushTx(), PopTx(), PushRx() y PopRx()

```
// Buffer de Transmisión
uint8_t bufferTx[TXBUFFER_SIZE];
// Buffer de Recepción
uint8_t bufferRx[RXBUFFER_SIZE];

uint8_t PushTx(uint8_t dato)
{
	if(tx_buffer_full)
		return 1;	//buffer lleno

	bufferTx[index_tx_in] = dato;
	index_tx_in++;
	index_tx_in %= TXBUFFER_SIZE;
	tx_buffer_empty = 0;  //si agrego un dato el buffer ya no está vacío

	if(index_tx_in == index_tx_out)
		tx_buffer_full = 1;   //se llenó el buffer

	if (UART0_txEnCurso == 0) {
		UART0_txEnCurso = 1;  //si no había una TX en curso,
		UART0_StartTx();      //fuerzo el inicio de la TX
	}

	return 0;	//dato agregado al buffer
}

uint8_t PopTx(uint8_t *dato)
{
	if(tx_buffer_empty)
		return 1;	//buffer vacío

	*dato = (uint8_t) bufferTx[index_tx_out];
	index_tx_out++;
	index_tx_out %= TXBUFFER_SIZE;
	tx_buffer_full = 0;   //si saco un dato, el buffer ya no está lleno

	if(index_tx_out == index_tx_in)
		tx_buffer_empty = 1;  //se vació el buffer

	return 0;	//dato retirado del buffer
}

uint8_t PushRx(uint8_t dato)
{
	if(rx_buffer_full)
		return 1;	//buffer lleno

	bufferRx[index_rx_in] = dato;
	index_rx_in ++;
	index_rx_in %= RXBUFFER_SIZE;
	rx_buffer_empty = 0;  //si agrego un dato, el buffer ya no está vacío

	if(index_rx_in == index_rx_out)
		rx_buffer_full = 1;   //se llenó el buffer

	return 0;	//dato agregado al buffer
}

uint8_t PopRx(uint8_t *dato)
{
	if(rx_buffer_empty)
		return 1;	//buffer vacío

	*dato = (uint8_t) bufferRx[index_rx_out];
	index_rx_out++;
	index_rx_out %= RXBUFFER_SIZE;
	rx_buffer_full = 0;   //si saco un dato, el buffer ya no está lleno

	if(index_rx_out == index_rx_in)
		rx_buffer_empty = 1;  //se vació el buffer

	return 0;	//dato retirado del buffer
}
```

Desde el punto de vista de la aplicación, se adoptaron (a modo de ejemplo) los siguientes formatos de trama para la transmisión de datos:

Formato de la trama del LPC1769 a la PC (para enviar los valores del resultado del ADC)

'#'	valorADC_MSB	valorADC_LSB	'$'
1 byte	1 byte	1 byte	1 byte

Suponiendo que "valorADC" sea una variable del tipo uint32_t o uint16_t que contenga el resultado de la última conversión, lo que queremos es poder transmitir ese valor (de 12 bits, 0 a 4095) a la PC. Como solo podemos enviar de 8 bits por vez, lo separamos en parte alta y parte baja. En este ejemplo enviamos primero la parte alta y luego la baja, pero el orden es indistinto, siempre y cuando se respete un orden para poder volver a armar el valor en la PC.

- (most significant byte) valorADC_MSB = (valorADC >> 8) & 0xFF; // desplazo 8 bits y me quedo con la parte alta

- (least significant byte) valorADC_LSB = valorADC & 0xFF; // me quedo solo con la parte baja

Formato de la trama de la PC al LPC1769 (para enviar los comandos de iniciar/detener la medición)

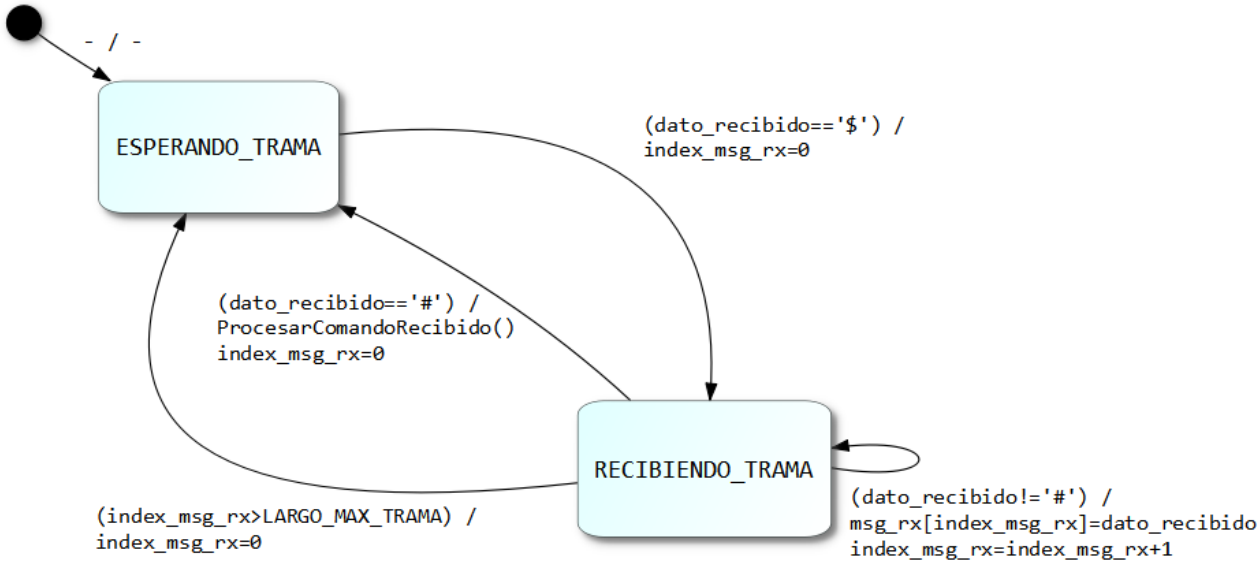
'$'	'M'	'0' o '1'	'#'
1 byte	1 byte	1 byte	1 byte

En este caso, el carácter 'M' indica que es un comando para el control de la medición.

'M''1' para iniciar y 'M''0' para detener.

Máquina de Estados: RX_Mensajes()

Espera que lleguen datos por el puerto serie, analiza la trama, identifica los comandos recibidos y da aviso a la máquina de estados que controla la medición según corresponda.



Implementación de la máquina de estados: RX_Mensajes()

```
uint8_t flag_IniciarMedicion = 0;
uint8_t flag_DetenerMedicion = 0;

void RX_Mensajes(void)
{
	uint8_t dato;
	static uint8_t index_msg_rx = 0;
	static char msg_rx[MAX_TRAMA_RX] = {0};
	static uint32_t estado_rx = ESPERANDO_TRAMA;

	// Chequeo si llegó un msje...
	if (!PopRx(&dato)) {
		// MdE: Análisis de la trama recibida
		switch (estado_rx) {
			case ESPERANDO_TRAMA: // Espero el caracter de inicio de la trama ('$')
				if ((char)dato == '$') {
					index_msg_rx = 0;
					estado_rx = RECIBIENDO_TRAMA;
				}
				break;
			case RECIBIENDO_TRAMA:// Recibo y almaceno la trama completa
				// Chequeo si llegó el final de la trama ('#')
				if ((char)dato != '#') {
					msg_rx[index_msg_rx] = (char)dato;
					index_msg_rx++;
					if (index_msg_rx > MAX_TRAMA_RX)
						estado_rx = ESPERANDO_TRAMA;
				} else {
					// msg_rx[0]: 'M' (comando medición ADC)
					// msg_rx[1]: '0' detener, '1' iniciar
					if ((msg_rx[0]) == 'M') {
						if((msg_rx[1]) == '0') {
							flag_DetenerMedicion = 1;
						}
						else if ((msg_rx[1]) == '1') {
							flag_IniciarMedicion = 1;
						}
					}
					estado_rx = ESPERANDO_TRAMA;
				}
				break;
			default:
				estado_rx = ESPERANDO_TRAMA;
				break;
		}
	}
}
```

Conversor Analógico Digital (ADC, 12bits)

Se configuró para tomar muestras de un solo canal (canal 5, entrada AD0.5, pin P0.31) y trabajar en modo BURST, es decir iniciando automáticamente sucesivas conversiones una a continuación de la otra. *NOTA: La otra opción (No BURST) consiste en iniciar manualmente cada conversión de manera individual, especificando el canal del cual se quiere tomar una muestra.* Se habilitó la interrupción del canal 5 para que interrumpa al finalizar cada conversión. Con respecto a la frecuencia del ADC, al tratarse de una señal de ejemplo tomada desde un potenciómetro, se seleccionó una frecuencia de aproximadamente 1,5kHz. Cambiando el valor cargado en el campo CLKDIV del registro de configuración, se puede aumentar la frecuencia del muestreo.

Frecuencia de Muestreo: fADC = fCLK_ADC / (65 * (CLKDIV + 1))

Resultado_ADC = (SEÑAL_ANALOGICA[V] / VREF[V]) * ($2^{n\_bits\_ADC}$ - 1) = (SEÑAL_ANALOGICA[V] / 3,3V) * (4095)

A continuación se pueden ver los registros del ADC que se utilizan para su inicialización:

Table 532: A/D Control Register (AD0CR - address 0x4003 4000) bit description

Bit	Symbol	Value	Description	Reset value
7:0	SEL		Selects which of the AD0.7:0 pins is (are) to be sampled and converted. For AD0, bit 0 selects Pin AD0.0, and bit 7 selects pin AD0.7. In software-controlled mode, only one of these bits should be 1. In hardware scan mode, any value containing 1 to 8 ones is allowed. All zeroes is equivalent to 0x01.	0x01
15:8	CLKDIV		The APB clock (PCLK_ADC0) is divided by (this value plus one) to produce the clock for the A/D converter, which should be less than or equal to 13 MHz. Typically, software should program the smallest value in this field that yields a clock of 13 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable.	0
16	BURST	1	The AD converter does repeated conversions at up to 200 kHz, scanning (if necessary) through the pins selected by bits set to ones in the SEL field. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1-bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion that's in progress when this bit is cleared will be completed. **Remark:** START bits must be 000 when BURST = 1 or conversions will not start. If BURST is set to 1, the ADGINTEN bit in the AD0INTEN register (Table 534) must be set to 0.	0
		0	Conversions are software controlled and require 65 clocks.	
20:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
21	PDN	1	The A/D converter is operational.	0
		0	The A/D converter is in power-down mode.	
23:22	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
26:24	START		When the BURST bit is 0, these bits control whether and when an A/D conversion is started:	0
		000	No start (this value should be used when clearing PDN to 0).	
		001	Start conversion now.	
		010	Start conversion when the edge selected by bit 27 occurs on the P2.10 / EINT0 / NMI pin.	

Table 534: A/D Interrupt Enable register (AD0INTEN - address 0x4003 400C) bit description

Bit	Symbol	Value	Description	Reset value
0	ADINTEN0	0	Completion of a conversion on ADC channel 0 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 0 will generate an interrupt.	
1	ADINTEN1	0	Completion of a conversion on ADC channel 1 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 1 will generate an interrupt.	
2	ADINTEN2	0	Completion of a conversion on ADC channel 2 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 2 will generate an interrupt.	
3	ADINTEN3	0	Completion of a conversion on ADC channel 3 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 3 will generate an interrupt.	
4	ADINTEN4	0	Completion of a conversion on ADC channel 4 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 4 will generate an interrupt.	
5	ADINTEN5	0	Completion of a conversion on ADC channel 5 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 5 will generate an interrupt.	
6	ADINTEN6	0	Completion of a conversion on ADC channel 6 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 6 will generate an interrupt.	
7	ADINTEN7	0	Completion of a conversion on ADC channel 7 will not generate an interrupt.	0
		1	Completion of a conversion on ADC channel 7 will generate an interrupt.	
8	ADGINTEN	0	Only the individual ADC channels enabled by ADINTEN7:0 will generate interrupts. **Remark:** This bit must be set to 0 in burst mode (BURST = 1 in the AD0CR register).	1
		1	Only the global DONE flag in ADDR is enabled to generate an interrupt.	
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Rutina de Inicialización del ADC:

```
void ADC_Init(void)
{
	//1.- Activo la alimentación del ADC desde el registro PCONP
	PCONP |= 1<<12;
	//2.- Selecciono el clock del ADC como 25MHz, PCLK_ADC = 100 Mhz / 4 = 25MHz
	PCLKSEL0 &= ~(0x03<<24);
	//3.- Y el divisor como 255 -> clk_ADC = 25MHz/256 = 98kHz
	// NOTA: PCLK_ADC0 is divided by (this value plus one) to produce the clock for the A/D converter
	AD0CR = 0xFF00;
	//4.- Configuro los pines del ADC0
	//AD0.5 (pote): P1[31]->PINSEL3: 30:31
	SetPINSEL(P1, 31, PINSEL_FUNC3);
	//5.- Activo interrupción del canal 5
	AD0INTEN = 0x00000020;
	//6.- Selecciono que voy a tomar muestras del canal AD0.5
	AD0CR |= 0x00000020;
	//7.- Activo el ADC (PDN = 1)
	AD0CR |= 1<<21;
	//8.- Como vamos a trabajar en modo BURTS, pongo los bits de START = 000
	AD0CR &= ~(0x0F<<24);
	// NOTA: El BURST no lo activo ahora, lo hago desde la aplicación.
	//AD0CR |= 1<<16;
	//9.- Habilito interrupción en el NVIC
	ISER0 |= (1<<22);
}
```

Dado que se va a controlar el funcionamiento del ADC desde la aplicación, se desarrollaron funciones para poder iniciar y detener el modo BURST (ráfaga).

```
void ADC_BurstEnable(void)
{
	//Activo el Burst, empieza a convertir.
	AD0CR |= 1<<16;
}

void ADC_BurstDisable(void)
{
	//Deshabilito el Burst, detiene la conversión.
	AD0CR &= ~(0x01 << 16);
}
```

El handler de interrupción, se encarga de leer el registro de datos del canal 5, obtener el resultado de la conversión y promediar una cantidad de muestras (a modo de ejemplo). Cuando calcula el promedio, notifica a la aplicación de que hay un nuevo valor disponible.

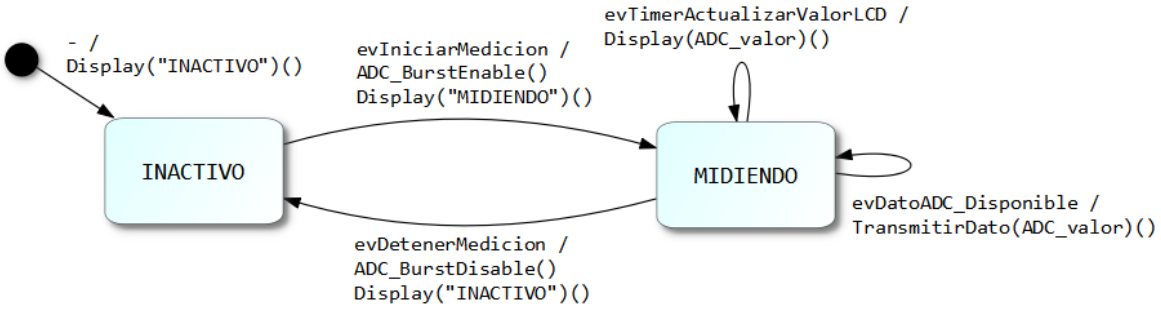
```
volatile uint32_t ADC_valor;
volatile uint8_t ADC_dato_disponible;

void ADC_IRQHandler(void)
{
	uint32_t adc_data_reg = 0;
	static uint32_t acumulado = 0;
	static uint32_t cont = CANT_MUESTRAS_PROMEDIO;

	// leo el registro de resultado del canal 5
	adc_data_reg = AD0DR5;
	// chequeo bit DONE == 1 -> hay un resultado disponible
	if ((adc_data_reg >> 31) == 0x01) {
		// obtengo los 12 bits del valor medido y los acumulo para promediar
		acumulado += (adc_data_reg >> 4) & 0xFFF;
		cont--;
		// cuando tengo todas la muestras, calculo el promedio
		if (cont == 0) {
			ADC_valor = acumulado / CANT_MUESTRAS_PROMEDIO;
			acumulado = 0;
			cont = CANT_MUESTRAS_PROMEDIO;
			ADC_dato_disponible = 1;
		}
	}
}
```

Table 535: A/D Data Registers (AD0DR0 to AD0DR7 - 0x4003 4010 to 0x4003 402C) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin, as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}.	NA
29:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits.This bit is cleared by reading this register.	
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read.	NA

Máquina de Estados: Controlador()

Controla el funcionamiento del ADC (enable/disable modo BURST), se encarga de transmitir los datos disponibles y de mostrar el estado actual y los valores obtenidos en el LCD.



Implementación de la máquina de estados: Controlador()

```
void Controlador(void)
{
	static uint32_t estado_medidor = MEDIDOR_RESET;
	static char msg_LCD[2*LARGO_RENGLON];

	// MdE: medición del ADC y transmisión de datos
	switch (estado_medidor) {
		case MEDIDOR_RESET:
			LCD_DisplayMsg("ADC+UART+LCD    ", LCD_RENGLON_1, 0);
			LCD_DisplayMsg("@info2 CL2017  ", LCD_RENGLON_2, 0);
			estado_medidor = MEDIDOR_INACTIVO;
			break;
		case MEDIDOR_INACTIVO:
			// chequeo si llegó un comando para iniciar la medición
			if (flag_IniciarMedicion) {
				flag_IniciarMedicion = 0;
				ADC_BurstEnable();
				LCD_DisplayMsg("> MIDIENDO     ", LCD_RENGLON_1, 0);
				IO_LED_Set(IO_LED_STICK, ON);
				estado_medidor = MEDIDOR_ACTIVO;
			}
			break;
		case MEDIDOR_ACTIVO:
			// si hay un dato disponible, lo envío por el puerto serie
			if (ADC_dato_disponible) {
				ADC_dato_disponible = 0;
				// transmito la trama con un nuevo dato
				PushTx('#');
				PushTx((char)((ADC_valor>>8) & 0xFF));
				PushTx((char)(ADC_valor & 0xFF));
				PushTx('$');
			}
```

```
                        // chequeo si llegó un comando para detener la medición
                        if (flag_DetenerMedicion) {
                                flag_DetenerMedicion = 0;
                                ADC_BurstDisable();
                                LCD_DisplayMsg("> ADC INACTIVO  ", LCD_RENGLON_1, 0);
                                LCD_DisplayMsg("                ", LCD_RENGLON_2, 0);
                                IO_LED_Set(IO_LED_STICK, OFF);
                                estado_medidor = MEDIDOR_INACTIVO;
                        }
                        // actualizo el valor mostrado en el LCD
                        if (flag_updateValorLCD) {
                                flag_updateValorLCD  = 0;
                                sprintf(msg_LCD, "valorADC=%04d ", ADC_valor);
                                LCD_DisplayMsg(msg_LCD, LCD_RENGLON_2, 0);
                        }
                        break;
                default:
                        estado_medidor = MEDIDOR_INACTIVO;
                        break;
        }
}
```

Inicialización y main()

```
void Kit_Init(void)
{
        //Configuro el PLL: seteo el clock del micro en 100MHz
        PLL_Init();
        //Inicialización de UART0 en 9600-8-E-1
        UART0_Init();
        //Inicialización de ADC en modo burst por interrupción
        ADC_Init();
        //Configuro SysTick para interrupmpir cada 2,5ms (2500us)
        SysTick_Init(2500);
        // Inicializo Entradas/Salidas (IO)
        IO_Init();
        // Inicializo LCD
        LCD_Init();
}

int main(void)
{
        Kit_Init();

    while(1) {
        RX_Mensajes();
        Controlador();
    }

    return 0 ;
}
```

Aplicación en Qt