Bienvenido: Ingresar
location: WebHome / Hardware / CortexNVIC

Manejo de Interrupciones

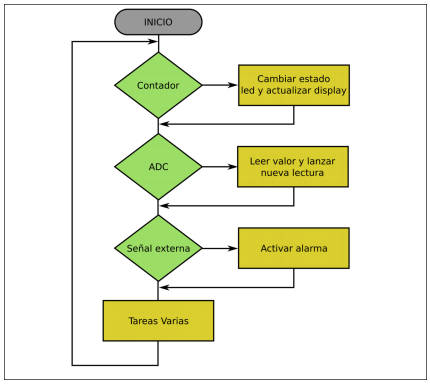
En un sistema con microprocesador, un evento de hardware ocurre en general de manera asicronica con respecto a el software que se está corriendo, en la mayoría de los casos es inviable determinar en que lugar del programa ocurrirá tal evento, a pesar de ello ocurren y deben ser atendido muchas veces con una baja latencia (tiempo transcurrido entre el evento y la ejecución del software que lo atiende).

Surgen dos opciones para tratar estos problemas.

Manejo de eventos por "polling" o "sondeo"

Se realiza constantemente consultas al dispositivo para determinar si ocurrió un evento, este acceso al hardware degrada el sistema, debido a las latencias que puede haber en las respuestas o simplemente por la propio código de consulta que debe ser repetido constantemente.

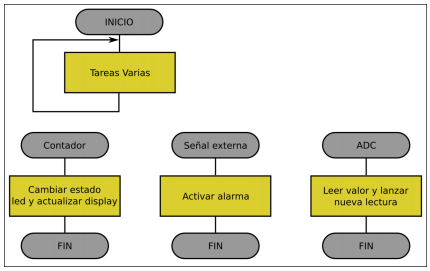
Fue la primera solución a eventos asíncronos que podían ocurrir en el sistema.



Manejo de eventos por interrupción

En el contexto de un sistema con microprocesador, la interrupción es un proceso por el cual ante un evento de un determinado periférico, se suspende momentáneamente la ejecución del programa principal del sistema, para ejecutar un código especifico que atienda dicho evento.

La interrupción se realiza mediante una señal que emite el periférico y que es recibida por un controlador de interrupción, el cual realiza la interrupción del proceso en ejecución dentro del micro y su posterior cambio a el código especifico.



Interrupción en el Cortex M4

Nested Vectored Interrupt Controller (NVIC) o Controlador de interrupciones vectorizadas anidadas, es el controlador encargado del manejo de interrupciones del Cortex-M4 Este controlador tiene la particularidad de permitir asociar cada interrupciones del microcontrolador con una dirección de

memoria (vector) donde se encuentra la subrutina que atenderá dicha interrupción, permitiendo una respuesta rápida (baja latencia) ante un evento.

Registros de la NVIC

Dirección	Nombre	Tipo	Desccripción
0xE000E100- 0xE000E11C	NVIC_ISER0- NVIC_ISER7	RW	Interrupt Set-enable Registers
0XE000E180- 0xE000E19C	NVIC_ICER0- NVIC_ICER7	RW	Interrupt Clear-enable Registers
0XE000E200- 0xE000E21C	NVIC_ISPR0- NVIC_ISPR7	RW	Interrupt Set-pending Registers
0XE000E280- 0xE000E29C	NVIC_ICPR0- NVIC_ICPR7	RW	Interrupt Clear-pending Registers
0xE000E300- 0xE000E31C	NVIC_IABR0- NVIC_IABR7	RW	Interrupt Active Bit Registers
0xE000E400- 0xE000E4EF	NVIC_IPR0- NVIC_IPR59	RW	Interrupt Priority Registers
0xE000EF00	STIR	WO	Software Trigger Interrupt Register

Vectores de Interrupción

Es un vector que contiene la dirección de memoria de la rutina para atender cada uno de las posibles interrupciones, depende específicamente del microcontrolador, debido a que los periféricos que posea serán los listados en este vector.

Ejemplo de los primeros 17 vectores en los micros de las familia LPC43xx

ID	Numero de excepción	Offset	Función
0	16	0x40	DAC
1	17	0x44	M0APP Cortex-M0APP
2	18	0x48	DMA
3	19	0x4C	Reserved
4	20	0x50	FLASHEEPROM
5	21	0x54	ETHERNET Ethernet interrupt
6	22	0x58	SDIO SD/MMC interrupt
7	23	0x5C	LCD
8	24	0x60	USB0 OTG interrupt
9	25	0x64	USB1
10	26	0x68	SCT SCT combined interrupt
11	27	0x6C	RITIMER
12	28	0x70	TIMER0
13	29	0x74	TIMER1
14	30	0x78	TIMER2
15	31	0x7C	TIMER3
16	32	0x80	MCPWM Motor control PW

La configuración de la interrupción requiere de los siguientes pasos

Configuración del Periférico

Además de configurar el periférico para su funcionamiento de acuerdo a los requerimientos de la aplicación, se debe definir cuales y cuantos eventos del mismo generarán interrupción, debido a que en general estos periféricos generan mas de un evento. Podemos citar por ejemplo la UART, este periférico generalmente se atiende por interrupción, por tener eventos asíncrono y relativamente lento (configurado

a 9600 baudios significa que se recibe o se transmite 1 byte cada aprox. 1 ms o 2x10^5 instrucciones en un micro de 200MHz).

Este periférico puede generar interrupciones cuando:

- llegó un byte.
- llegó el byte nro n (para un n predefinido).
- Se terminó de enviar un byte.
- La cola de envíos esta vacía.
- Se detectó un error de trama.
- etc.

Se debe evaluar entonces al momento de configurar el dispositivo cuales eventos generará interrupciones y cuales no.

Configuración de la NVIC

Una vez definido el periférico, debe configurarse el controlador de interrupciones para que efectivamente atienda a la interrupción, si es requerido en este paso se define además la prioridad, la NVIC permite hasta 255 niveles de prioridad siendo el 0 la mas alta, bajando la prioridad cuando subimos el valor.

Vector de Interrupción

Una vez configurado el periférico y luego la NVIC, solo resta programar la función que atenderá la interrupción, esta función se escribe de manera análoga a una función estándar con la diferencia que debe cargarse la dirección de la misma en el vector de interrupciones.

Ejemplo

A continuación se realiza un ejemplo de interrupción, el mismo implementa un retardo de "N" ms para ser utilizado como una retardo mas preciso que un simple bucle. El funcionamiento del mismo será el siguiente:

Se utiliza el periférico Repetitive Interrupt Timer (RIT), es un contador muy sencillo que permite realizar cuentas o interrupciones a intervalos definidos sin la necesidad de utilizar los contadores generales que dispone el micro. Se configura este timer para que cuente hasta un valor determinado que implicará un retardo de 1 ms generando una interrupción, dentro de la misma se incrementa una variable global, luego se realizará una función retardo que simplemente colocará a 0 la variable global y un bucle esperará que esa variable se incremente al valor deseado, generando "N" x 1ms = "N" ms de retardo

Configuración del periférico y NVIC

Habilitar el periférico

```
Chip_RIT_Init(LPC_RITIMER);
```

Función de la LPCOpen que configura el periférico para que cuente 1 ms, esta función realiza el cálculo de acuerdo a la frecuencia en que funciona el micro para que la cuenta dé aproximadamente los milisegundos enviados en el 2do parámetro

```
Chip_RIT_SetTimerInterval(LPC_RITIMER,1);
```

Por último deberíamos configurar el periférico para que el mismo genere interrupción por un evento dado, en el caso particular de RIT, la única interrupción que puede generar es por llegar a la cuenta, por ello no posee configuración de interrupción, la sola habilitación del periférico dentro de la NVIC, ya lo habilita a generar interrupciones.

Antes de concluir la inicialización, se borra la bandera de interrupción

```
Chip_RIT_ClearInt(LPC_RITIMER);
```

Finalmente se habilita el periférico en la NVIC, Cada periférico tiene asignado un nro dentro de la tabla de vectores de interrupción, en el caso del RIT le corresponde la posición nro 11.

```
NVIC_EnableIRQ(11);
```

Vector de Interrupción

En primer lugar se debe realizar la función que atienda a la interrupción, en el caso del ejemplo podemos llamar a la función que atiende a RIT como `Timer_IRQ` y se codificará de la siguiente manera

```
volatile int var_global=0;      // variable global
```

```
void Timer_IRQ(void) {
    var_global++;      // incrementamos la variable global
    Chip_RIT_ClearInt(LPC_RITIMER); // borramos la bandera de interrupción
}
```

Esta función puede ser escrita en una archivo separado del vector.c, para este caso, se deberá escribir el prototipo de la funcion en el .h correspondiente y luego ese .h ser incluido en el archivo que posee el vector de interrupciones (**vector.c**), este archivo que acompaña a cada proyecto, en el mismo se encuentra el vector de interrupciones ya inicializado con las funciones para atender las interrupciones principales (15 en total) para luego continuar con los vectores de los periféricos o interrupciones de usuario apuntando todos a la función genérica `ISR_NoHandler()` Localizada el nro de interrupción (11 en el ejemplo), se debe modificar del nombre genérico `ISR_NoHandler` al nombre de la función asignada en el ejemplo `Timer_IRQ` Quedando de la siguiente forma

```
   ISR_NoHandler,      /* 0x17 0x0000005C - No Handler set for ISR LCD (IRQ 7) */
   ISR_NoHandler,      /* 0x18 0x00000060 - No Handler set for ISR USB0 (IRQ 8) */
   ISR_NoHandler,      /* 0x19 0x00000064 - No Handler set for ISR USB1 (IRQ 9) */
   ISR_NoHandler,      /* 0x1a 0x00000068 - No Handler set for ISR SCT (IRQ 10) */
   Timer_IRQ,      /* 0x1b 0x0000006C - No Handler set for ISR RIT (IRQ 11) */
   ISR_NoHandler,      /* 0x1c 0x00000070 - No Handler set for ISR TIMER0 (IRQ 12)
*/
   ISR_NoHandler,      /* 0x1d 0x00000074 - No Handler set for ISR TIMER1 (IRQ 13)
*/
   ISR_NoHandler,      /* 0x1e 0x00000078 - No Handler set for ISR TIMER2 (IRQ 14)
*/
```

Finalmente se puede definir la función retardo como sigue:

```
void retardo(int ms)
{
   var_global=0;
   while(var_global<=ms) {
   }
}
```

UntitledWiki: WebHome/Hardware/CortexNVIC (última edición 2017-09-26 22:07:57 efectuada por GuillermoSteiner)

Bienvenido: Ingresar
location: WebHome / Hardware / Teclas

Manejo de Teclado en EDU-CIAA

Para ejemplificar la habilitación de un pin como entrada para un pulsador, se habilitará el bit corresponde al Switch 1 de la placa EDU-CIAA-NXP

Pasos:

1. Selección del pin como GPIO

Esto se realiza en la SCU, además se establece el buffer de entrada y se activa el pullup.

```
Chip_SCU_PinMux(1,0,MD_PUP|MD_EZI|MD_ZI,FUNC0); /* P1_0 en GPIO 0 bit 4 */
```

2. Configuración de la dirección del pin

Ahora ya en el periférico GPIO se configura el bit del puerto como salida (0)

```
Chip_GPIO_SetDir(LPC_GPIO_PORT, 0,(1<<4),0);
```

3. Anular rebotes

El rebote mecánico del pulsador en los cambios de estado del mismo (el momento de pulsarlo o soltarlo), se traduce como una oscilación ente 1 y 0 en la entrada digital del micro. Para evitar que estas fluctuaciones se tomen como una serie de pulsaciones consecutivas, se deben tomar algunas precauciones.

```
int ValorTecla1()
{
   int valor,valor_anterior;
   int contador;
   while(contador<1000) {
      valor = !Chip_GPIO_ReadPortBit(LPC_GPIO_PORT,0,4); // leer tecla (las teclas
se activa por 0)
      if(valor!=valor_anterior) {
         valor_anterior = valor;
         contador=0;
      } else {
         contador++;
      }
   }
   return valor;
}
```

4. Anular múltiples repeticiones

```
main() {
   int tecla1,tecla1_anterior;
   while(1) {
```

```
      tecla1 = ValorTecla1()  ;
      if(tecla1!=tecla1_anterior && tecla1==1) {
         Led_Color_Toggle(LED1);
      }
      tecla1_anterior = tecla1;
   }
}
```

UntitledWiki: WebHome/Hardware/Teclas (última edición 2016-09-06 21:15:39 efectuada por GuillermoSteiner)

Bienvenido: Ingresar
location: WebHome / Hardware / UART

Puertos Series (USART) en el microcontrolador LPC43XX con la biblioteca LPCOpen

Los periféricos USART (por sus siglas en ingles de Universal Synchronous/Asynchronous Receiver/Transmitter) son dispositivos de comunicación serie síncrona o asíncrona, que permite conectar el microcontrolador con otros equipos, obteniendo de esta forma un método simple para comunicar o recibir datos del exterior.

El LPC43xx posee 3 USART y 1 UART (solo permite el modo asíncrono) esta última con los pines necesarios para implementar una comunicación con un MODEM.

Lineas de comunicación

USART 0,2,3

Señal	Descripción
RxD	Entrada
TxD	Salida
DIR	Utilizado en RS-485 y EIA-485
UCLK	Clock para modo síncrono
BAUD	Solo disponible en USART3 para interfaz IrDA

UART 1

Señal	Descripción
RxD	Entrada
TxD	Salida
CTS	Clear To Send
DCD	Data Carrier Detect
DSR	Data Set Ready
DTR	Data Terminal Ready
RI	Ring Indicator
RTS	Request To Send

UART (Universal Asynchronous Receiver/Transmitter)

EL modo de transmisión asíncrono (UART) corresponde a uno de los protocolos de comunicación serie mas difundidos, en este modo de transición los datos son enviados sin una señal de reloj externa.

Todas las UART del LPC43xx (las 3 USART funcionando en modo asíncrono y la UART) son compatibles con el estándar 550, esto significa que son compatibles con la UART diseñada por National Semiconductors 16550 y salida al mercado en 1987, esta UART reemplazaba a la 8250 primera interfaz serie de la PC XT.

En una transmisión asíncrona, los datos son divididos en palabras de una longitud fija, generalmente octetos, a los cuales se le agregan al comienzo códigos de sincronismo y luego al final del word se pueden agregar espacios, con lo cual permite al receptor identificar un nuevo código de sincronismo.

Como se comentó arriba, las palabras suelen ser octetos, este tipo de comunicación asíncrona se denomina orientada a caracter y uno de los estándar mas comunes es el RS-232 (Recommended Standard 232).

RS-232

El estándar RS-232 es una norma para el intercambio de datos entre dos dispositivos denominados DTE (Equipo terminal de datos) con un DCE (Equipo de comunicación de datos), A pesar de que este estándar soporta comunicación síncrona, su uso mas habitual es en comunicaciones asíncronas por ejemplo el implementado en una PC.

En el caso habitual de una PC llamaremos a la misma DTE, donde el DCE generalmente era el modem, en el caso de comunicarse dos PC entre si (DTE con DTE) no habrá modem, con lo cual la conexión se llamará Null Modem, en estos casos el cable que une las dos computadoras, debe unir las señales de manera cruzada, es decir por ejemplo que la señal denominada TxD en una de las PC debe unirse con la denominada RxD de la otra PC.

Registros

RBR	RO	0x000	Receiver Buffer Register. contiene el proximo byte a ser leido (DLAB = 0).
THR	WO	0x000	Transmit Holding Register se escribe el proximo byte a enviar (DLAB = 0).
DLL	R/W	0x000	Divisor Latch LSB. parte menos significativa del divisor (DLAB = 1).
DLM	R/W	0x004	Divisor Latch MSB. parte mas significativa del divisor (DLAB = 1).
IER	R/W	0x004	Interrupt Enable Register. contiene la habilitación de las 7 posibles fuentes de interrupción (DLAB = 0)
IIR	RO	0x008	Interrupt ID Register. Interrupciones pendientes
FCR	WO	0x008	FIFO Control Register. Controla la FIFO
LCR	R/W	0x00C	Line Control Register. Contiene el formato de del frame
-		0x010	Reservado
LSR	RO	0x014	Line Status Register. Contiene las banderas del estado de la transmisión y recepción
-		0x018	Reservado
SCR	R/W	0x01C	Scratch Pad Register. Registro de datos temporales
ACR	R/W	0x020	Auto-baud Control Register.
ICR	R/W	0x024	IrDA control register (solo USART3)
FDR	R/W	0x028	Fractional Divider Register.
OSR	R/W	0x02C	Oversampling Register.
-		0x030-0x3C	Reservado
HDEN	R/W	0x040	Half-duplex enable Register
-		0x044	Reservado
SCICTRL	R/W	0x048	Smart card interface control register
RS485CTRL	R/W	0x04C	RS-485/EIA-485 Control.
RS485ADRMATCH	R/W	0x050	RS-485/EIA-485 address match
RS485DLY	R/W	0x054	RS-485/EIA-485 direction control delay.
SYNCCTRL	R/W	0x058	Synchronous mode control register.
TER	R/W	0x05C	Transmit Enable Register.

RBR/THR Registro de Recepción y Transmisión

La misma posición de memoria es utilizada por dos registros, depende del tipo de operación realizada sobre esa memoria a cual de ello se acceda.

- Operación de lectura, se accede al registro de recepción el cual posee el byte recibido mas antiguo contenido en la FIFO, en caso de configurar una trama menor a 8 bits de datos, los bits mas altos de este registro permanecerán en 0.
- Operación de escritura, el registro será colocado en la FIFO para ser transmitido.

En ambos casos el DLAB (Divisor Latch Access Bit) debe ser cero

DLR (Divisor Latch Register)

Esta compuesto por dos registros de 8 bits cada uno el LSB y MSB, donde el LSB comparte la dirección de memoria con el registro de recepción y transmisión y el MSB lo hace con el registro de habilitación de interrupción.

El valor total formado por los dos registros, tendrá una longitud de 16bits y es el divisor del VPB que se deberá configurar para determinar el baud rate de la transmisión y/o recepción.

El baud rate será calculado como:

```
               VPB
baud rate = ----------
            DLR x 16
```

Conociendo la velocidad con la cual deseamos transmitir y la frecuencia del VPB, podemos despejar el DLR, el cual nos dará un valor de 16 bits, guardando en DLL el byte menos significativo y en DLM el mas significativo.

IER Registro de habilitación de interrupción

Este registro es el usado para habilitar las diferentes interrupciones que puede realizar la UART según distintos eventos

- bit 0 RBRIE habilita la generación de una interrupción cuando hay un dato disponible para leer
- bit 1 THREIE habilita la generación de la interrupción cuando la FIFO de transmisión se ha vaciado, el estado de esta interrupción puede ser leída en el bit 5 del LSR
- bit 2 RXIE habilita la generación de una interrupción por Rx, el estado de esta interrupción puede ser leída de LSR[4:1].
- bit 8 ABEOINTEN habilita la generación de una interrupción por auto-baud.
- bit 9 ABTOINTEN habilita la generación de una interrupción por auto-baud time out .

IIR Registro de identificación de Interrupción

Muestra el estado de las interrupciones, pudiendo leer el origen y la prioridad de una interrupción pendiente, las interrupciones son congeladas en un acceso a la U0IIR, si ocurriera alguna interrupción en el momento en que se accede, esta se reflejara recién la próxima vez

- bit 0 interrupción pendiente 0 = al menos una interrupción está pendiente, las interrupciones pendientes pueden ser leídas en U0IIR[3:1]
- bits 1 a 3 identificación de interrupción:

bits	Prioridad	Denominación	Descripción	Reset
0x3	1	Receive Line Status (RLS)	Es generada por cuatro condiciones de error, (OE) overrun error, (PE) Parity error, (FE) Framing error o error de trama y (BI) break interrupt	leyendo el U0LSR
0x2	2a	Receive Data Available (RDA)	Se activa cuando la FIFO de lectura llegue a la cantidad establecida en U0FCR[7:6].	leyendo U0RBR o modificando

				la FIFO a un valor mas alto
0x6	2b	Character Time-out Indicator (CTI)	Ocurre cuando hay al menos un carácter en la FIFO de recepción y no ocurrió ninguna actividad en la misma por 3,5 a 4,5 tiempos de caracter, esta interrup., permite leer uno o varios datos que quedaron en la FIFO pero que por su cantidad no llega a generar una RDA	leyendo el URBR
0x1	3	THRE Interrupt	Es activada cuando la FIFO de transmisión se vacía	leyendo el U0IIR o escribiendo el U0THR

- bit 6 a 7 FIFO equivalente al los bits correspondiente al FCR[6:7]
- bit 8 ABEOINT el auto-band finalizó con éxito
- bit 9 ABTOINT auto-band generó un time out

.

FCR Registro de Control de FIFO

- bit 0 FIFOEN habilitar FIFO, activo por alto, habilita las dos FIFO (recepción y transmisión), este bit además habilita el resto de la configuración de este registro
- bit 1 RXFIFORES borrar FIFO de Rx, un 1 es este bit borra toda la FIFO de recepción, luego de esto, es puesto a 0 por la misma UART
- bit 2 TXFIFORES borrar FIFO de Tx, un 1 es este bit borra toda la FIFO de transmisión, luego de esto, es puesto a 0 por la misma UART
- bit 3 DMAMODE Activar modo DMA
- bit 6 a 7 Selector del nivel de trigger de Rx, selecciona a cuantos caracteres recibidos se activará la interrupción RDA
    - 00: trigger nivel 0 (1 caracter)
    - 01: trigger nivel 1 (4 caracteres)
    - 10: trigger nivel 2 (8 caracteres)
    - 11: trigger nivel 3 (14 caracteres)

LCR Registro de Control de Linea

Determina el formato de los datos a transmitir o recibir

- bits 0 a 1: WLS Selección del largo del word
    - 00: 5 bit de largo
    - 01: 6 bit de largo
    - 10: 7 bit de largo
    - 11: 8 bit de largo
- bit 2: SBS Largo del bit de stop
    - 0: 1 bit de stop
    - 1: 2 bit de stop (1.5 en caso de 5 bit de largo)
- bit 3: PE Habilitar Paridad esto significa que con 1, estos agregando un bit extra a los datos, el valor de este bit depende de la selección de paridad (bit 4:5)
- bits 4 a 5: PS
    - 00: paridad impar
    - 01: paridad par
    - 10: fuerza a 1 el bit de paridad
    - 11: fuerza a 0 el bit de paridad
- bit 6: BC Control de Corte, cuando se configura a 1, fuerza al pin de transmisión a un valor lógico 0
- bit 7: DLAB bit de acceso al divisor de latch (DLAB) este bit en 1 permite el acceso a los registro del divisor de latch (DLL y DLM)

LSR Registro de Estado de la Linea

es un registro de solo lectura que provee acceso a estado del bloque de transmisión y recepción

- bit 0: Receiver Data Ready (RDR), un 1 en este bit indica que hay un dato dentro de la FIFO sin leer.
- bit 1: Overrun error (OE), se activa en el caso de que un nuevo caracter es recibido y la FIFO se encuentra llena, el caracter se pierde y se activa el OE
- bit 2: Parity Error (PE), se activa cuando la verificación de la paridad en el caracter recibido falla
- bit 3: Framing Error (FE), ocurre cuando en e bit de stop se lee un 0, esto puede ocurrir cuando no está correctamente sincronizado el receptor con respecto al transmisor queda a la espera de recibir todos 1
- bit 4: Break Interrupt (BI): se activa cuando recibe 0 durante el periodo de un dato (bit de start + datos + bit de stop) en este caso el receptor
- bit 5: Transmitter Holding Empty (THRE), un 0 indica que el registro de transmisión contiene datos 1 indica que este registro está vacío
- bit 6: Transmitter Empty (TEMT) un 1 indica que se transmitieron todos los datos
- bit 7: Error in Rx FIFO (RXFE) ocurre cuando el dato leído posee algún error (FE,PE o BI)
- bit 8: Error in transmitted character (TXERR)

Utilización de la USART del LPC43xx

Para implementar una interfaz serie RS-232, utilizaremos una USART en particular, la número 2, en la placa EDU-CIAA esta USART fue cableada al integrado FTDI2232, el mismo es el encargado de realizar la interfaz JTAG del microcontrolador y el USB, además de esta tarea, incorpora un puerto extra configurado en este caso como puerto serie asíncrono. Es en ese puerto donde se conecta los pines de la USART2 permitiendo realizar una muy sencilla conexión entre la PC y la EDU-CIAA.

Para la configuración de el puerto, se utilizará la librería LPCOpen

Configurar el SCU

Se deben configurar los pines para ser usados por la USART, en su modo asincrónico solo utiliza dos señales RX y TX, estas serán las que se deberan configurar en la SCU.

Para evitar anular un dispositivo por ser requerido los pines para otra función, este microcontrolador mapea en varios pines cada linea de la USART, en el caso de la USART2 tenemos las siguientes opciones

U2_TXD	en los pines P1_15, P2_10, P7_1, PA_1
U2_RXD	en los pines P1_16, P2_11, P7_2, PA_2

En el momento de rutear la placa se eligió la dupla P7_1 y P7_2 como los pines para ser unidos al FTDI2232, eso obliga a seleccionar esos pines como la salida para la UART

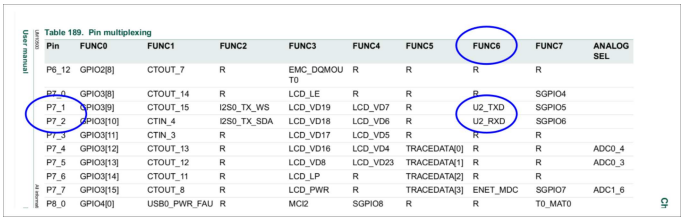


Table 189. Pin multiplexing

Pin	FUNC0	FUNC1	FUNC2	FUNC3	FUNC4	FUNC5	FUNC6	FUNC7	ANALOG SEL
P6_12	GPIO2[8]	CTOUT_7	R	EMC_DQMOUT0	R	R	R	R	
P7_0	GPIO3[8]	CTOUT_14	R	LCD_LE	R	R	R	SGPIO4	
P7_1	GPIO3[9]	CTOUT_15	I2S0_TX_WS	LCD_VD19	LCD_VD7	R	U2_TXD	SGPIO5	
P7_2	GPIO3[10]	CTIN_4	I2S0_TX_SDA	LCD_VD18	LCD_VD6	R	U2_RXD	SGPIO6	
P7_3	GPIO3[11]	CTIN_3	R	LCD_VD17	LCD_VD5	R	R	R	
P7_4	GPIO3[12]	CTOUT_13	R	LCD_VD16	LCD_VD4	TRACEDATA[0]	R	R	ADC0_4
P7_5	GPIO3[13]	CTOUT_12	R	LCD_VD8	LCD_VD23	TRACEDATA[1]	R	R	ADC0_3
P7_6	GPIO3[14]	CTOUT_11	R	LCD_LP	R	TRACEDATA[2]	R	R	
P7_7	GPIO3[15]	CTOUT_8	R	LCD_PWR	R	TRACEDATA[3]	ENET_MDC	SGPIO7	ADC1_6
P8_0	GPIO4[0]	USB0_PWR_FAU	R	MCI2	SGPIO8	R	R	T0_MAT0	

```
Chip_SCU_PinMux(7,1,0,FUNC6); /* P7_1: UART2 TXD */
Chip_SCU_PinMux(7,2,0,FUNC6); /* P7_2: UART2 RXD */
```

Configurar el Periférico

Como segunda fase, se encuentra la configuración del propio dispositivo

en este caso se deberá

- Inicializar la UART

```
void Chip_UART_Init(LPC_USART_T *pUART);
```

- Configurar velocidad de transmisión

```
Status Chip_UART_SetBaud(LPC_USART_T *pUART,uint32_t baudrate);
```

- Configurar la FIFO

```
STATIC INLINE void Chip_UART_SetupFIFOS(LPC_USART_T *pUART,uint32_t fcr);
```

Donde fcr se deberá configurar de la siguiente forma UART_FCR_FIFO_EN para habilitar la FIFO y UART_FCR_TRG_LEV0 para generar evento por cada byte recibido

Todas estas acciones se realizan sobre el registro FCR, este registro es utilizado para configurar la FIFO, se muestra a continuación las distintas opciones que posee,(extraído del manual del usuario)

40.6.6 USART FIFO Control Register

The FCR controls the operation of the USART RX and TX FIFOs.

Table 930. USART FIFO Control Register Write Only (FCR, addresses 0x4008 1008 (USART0), 0x400C 1008 (USART2), 0x400C 2008 (USART3)) bit description

Bit	Symbol	Value	Description	Reset value
0	FIFOEN		FIFO Enable.	0
		0	Disabled. USART FIFOs are disabled. Must not be used in the application.	
		1	Enabled. Active high enable for both USART Rx and TX FIFOs and FCR[7:1] access. This bit must be set for proper USART operation. Any transition on this bit will automatically clear the USART FIFOs.	
1	RXFIFORES		RX FIFO Reset.	0
		0	No effect. No impact on either of USART FIFOs.	
		1	Clear. Writing a logic 1 to FCR[1] will clear all bytes in USART Rx FIFO, reset the pointer logic. This bit is self-clearing.	
2	TXFIFORES		TX FIFO Reset.	0
		0	No effect. No impact on either of USART FIFOs.	
		1	Clear. Writing a logic 1 to FCR[2] will clear all bytes in USART TX FIFO, reset the pointer logic. This bit is self-clearing.	
3	DMAMODE		DMA Mode Select. When the FIFO enable bit (bit 0 of this register) is set, this bit selects the DMA mode.	0
5:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7:6	RXTRIGLVL		RX Trigger Level. These two bits determine how many receiver USART FIFO characters must be written before an interrupt is activated.	0
		0x0	Level 0. Trigger level 0 (1 character or 0x01).	
		0x1	Level 1. Trigger level 1 (4 characters or 0x04).	
		0x2	Level 2. Trigger level 2 (8 characters or 0x08).	
		0x3	Level 3. Trigger level 3 (14 characters or 0x0E).	
31:8	-	-	Reserved	-

- Habilitar la Transmisión

```
STATIC INLINE void Chip_UART_TXEnable(LPC_USART_T *pUART);
```

Uso de la UART

Una vez habilitada y en funcionamiento, se podrá enviar o leer los bytes recibido por medio de dos funciones de la LPCOpen

- Enviar un byte

```
STATIC INLINE void Chip_UART_SendByte (LPC_USART_T *pUART, uint8_t data);
```

- Recibir un byte

```
STATIC INLINE uint8_t Chip_UART_ReadByte (LPC_USART_T *pUART);
```

UntitledWiki: WebHome/Hardware/UART (última edición 2017-07-31 21:56:14 efectuada por GuillermoSteiner)

Bienvenido: Ingresar
location: WebHome / Hardware / ModuloADC

ADC (Analog Digital Converter)

Este periférico permite convertir señales analógicas en una cuenta o valor digital.

Características y Registros ADC

El LPC43xx posee 2 ADC idénticos, cada uno de ellos posee las siguientes características:

- Conversor de 10 bits
- 8 entradas multiplexadas
- Rango de entrada de 0 a 3,3 V
- Tiempo de conversión 2,45us
- Modo Ráfaga (Burst) para una entrada o múltiples entradas (multiplexado).
- Un registro de resultado para cada canal AD

Entradas

- ADC0_0 a ADC0_7 Entrada del ADC 0.
- ADC1_0 a ADC1_7 Entrada del ADC 1.
- ADCTRIG0 Entrada Trigger ADC0/1.
- ADCTRIG1 Entrada Trigger ADC0/1.

Estos nombres deben buscarse dentro del mapa del SCU y configurar adecuadamente el pin para su uso como ADC.

Alimentación

El ADC posee entradas separadas de alimentación denominadas V_{DDA} y V_{SSA}, usadas para el bloque analógico, las mismas deben estar correctamente aisladas de las demás señales para no generar errores en la conversión producto de las fluctuaciones en estas lineas.

V_{DDA} es usada además como V_{REF}

Registro de control (CR)

bits 7-0	SEL	Estos bits determina en cual o cuales de los 8 canales se realizará la conversión, correspondiendo cada bits a cada una de las entradas, este registro funciona de dos maneras diferentes: * Controlado por software, en este modo solo un bits puede ser puesto en 1, el cual será el canal donde se realizará la conversión. * Controlado por hardware, en este modo se selecciona con 1 a todos los canales donde se deben realizar una conversión, una vez activo el ADC en este modo, se producirá un barrido de todos estos canales.
bits 15-8	CLKDIV	El clock interno del ADC se extrae del VPB (PCLK) dividido el valor guardado en CLKDIV, el resultado no deberá ser superior a 4,5 Mhz, generalmente este divisor de configura para producir la máxima frecuencia posible del ADC (4,5Mhz) pero en ciertos casos conviene una menor frecuencia, por ejemplo en casos de que la señal a medir provenga de fuentes de alta impedancia.
bit 16	BURS	Si el bit es 0, la conversión es controlada por software requiriendo 11 ciclos de reloj, si en cambio es 1, se activa el control por hardware, ahora el tiempo de conversión y el rango de la misma será designado por CLKS, produciendo un barrido en cada canal que se encuentre activo en el registro SEL, comenzando por el bit menos significativo

		en 1. La conversión continuará hasta que se borre el el bits del BURST, con lo cual se detendrá el proceso una vez que la convención que se esté efectuando termine.
bits 19-17	CLKS	Este campo es usando en el modo de control por hardware y determina el número de ciclos usados para la conversión y el número de bits de precisión, pudiendo elegir entre 11 clock (10bits) = 000, 10 clock (9bits) = 001 hasta 4 clock (3bits) = 111.
bit 20		Reservado.
bit 21	PDN	1 = el ADC está operacional 0 = el ADC está en modo power down.
bits 23-22		Reservado.
bits 26-24	START	Configura de que manera el ADC comenzará a convertir, en caso de estar configurado en modo control por software (BURST = 0). 0x0: no comienza, este es el valor a usar cuando PDN = 0. 0x1: comenzar la conversión ahora. 0x2: flanco del tipo establecido en EDGE en el pin CTOUT_15. 0x3: flanco del tipo establecido en EDGE en el pin CTOUT_8. 0x4: flanco del tipo establecido en EDGE en el pin ADCTRIG0. 0x5: flanco del tipo establecido en EDGE en el pin ADCTRIG1. 0x6: flanco del tipo establecido en EDGE en el pin Motocon PWM output MCOA2. 0x7: reservado.
bit 27	EDGE	Este bits es utilizado en caso de configurar a START en algún modo que requiera flanco (modo 0x2 a 0x6). EDGE = 0 la conversión inicia en un flanco de bajada. EDGE = 1 la conversión inicia en un flanco de subida.
bits 31-28		Reservado.

Registro global de datos (GDR)

bits 15-6	V/V3A	indica el resultado de la conversión.
bits 26-24	CHN	Indica el canal de la ultima conversión.
bit 30	OVERRUN	En modo BURST este bits está en 1 si una o varias conversiones fueron perdidas por no leer a tiempo el registro antes de que una nueva la pisara.
bit 31	DONE	Este registro indica con 1 que la conversión a finalizado, es borrado cuando este registro es leído y cuando el CR es escrito, si el CR es escrito mientras que se estaba realizando una conversión, este bit es puesto a 1 y una nueva conversión arranca

Registro de habilitación de interrupción (INTEN)

bits 7-0	ADINTEN	Indica que canal/es generará/n interrupción, un 1 en el bit 0 indica que el canal 0 generará interrupción al finalizar la conversión, un 1 en el bit 1 corresponderá una interrupción para el canal 1 y asi con los demás canales.
bits 8	ADGINTEN	Cuando su valor es 1 habilita la interrupción cuando el bit DONE se ponga en 1, en caso de 0 la interrupción la genera los canales individuales indicados en ADINTEN.

Registro de Datos (DR0-DR7)

Es una copia del registro GDR pero particular para cada canal, disponiendo del último dato convertido para un canal en particular, el bit DONE para verificar si hay un nuevo dato a leer y OVERRUN para detectar perdida de datos convertidos.

Registro de Status (STAT)

Permite acceder a los bit DONE y OVERRUN de todos los canales

bits 7-0	DONE	Bits DONE de los 8 canales.
bits 15-8	OVERRUN	Bits OVERRUN de los 8 canales.
bit 16	ADINT	Este bit se pone en 1 cuando algún canal termina de convertir y el mismo está habilitado para generar interrupción via el registro ADINTEN

Configuración y Lectura de un ADC

Para la configuración y lectura del ADC se utilizará las librerías de las LPCOpen

Configuración de la SCU

Una diferencia importante que comparten el ADC junto con el DAC con respecto a los demás periférico, es la configuración de los pines de entrada (ADC) o salida (DAC), como se detalló en otros casos, el primer paso es configurar los pines para que estos puedan ser utilizados por el periférico, en la mayoría de los casos esto se realiza con los registros SFSP utilizando la funcion de la LPCOpen `Chip_SCU_PinMux()`.

La situación es diferente para los dos periféricos analógicos (ADC y DAC), en estos casos, es necesario configurar el registro ENAIO (ENAIO0 y ENAIO1 para las dos ADC y ENAIO2 para el caso del DAC), el mismo permite cambiar la configuración de los pines de digital a analógico quedando de esta forma el pin listo para su uso en el periférico correspondiente. De esta manera se configura entonces mediante la siguiente función del LPCOpen.

```
STATIC INLINE void Chip_SCU_ADC_Channel_Config(uint32_t ADC_ID, uint8_t channel);
```

Como ejemplo si se quiere configurar al canal 1 del del ADC0 se escribirá lo siguiente.

```
Chip_SCU_ADC_Channel_Config(0,ADC_CH1);
```

Inicialización del Periférico

La inicialización del ADC se realiza con la función Chip_ADC_Init(), esta función inicializa el periférico y establece una configuración para su funcionamiento por defecto, esta configuración por defecto será la siguiente:

- Velocidad de conversión = 400Ksamples
- Cantidad de bits = 10
- Modo burst (ráfaga) desactivado.

El formato de la función es el siguiente

```
void Chip_ADC_Init(LPC_ADC_T *pADC, ADC_Clock_Setup_T *ADCSetup);
```

Donde ADCSetup es una estructura que devuelve la configuración por defecto.

Habilitación del Canal

Con el comando

```
Chip_ADC_EnableChannel(LPC_ADC_T *pADC, ADC_CHANNEL_T channel, FunctionalState NewState)
```

Se habilita o se deshabilita uno de los canales del ADC

- **channel** es el número de canal (ADC_CH0, ADC_CH1 ... ADC_CH7)
- **NewState** es la acción que se desea aplicar sobre el canal (ENABLE para activar el canal DISABLE para desactivarlo)

Modo de Arranque

El paso final es configurar el modo de arranque del ADC, es decir que evento generará un **Start Conversion**

```
Chip_ADC_SetStartMode(LPC_ADC_T *pADC, ADC_START_MODE_T mode, ADC_EDGE_CFG_T EdgeOption)
```

mode puede tener los siguientes valores:

- ADC_NO_START : Para el caso de funcionamiento modo Burst
- ADC_START_NOW : Arranca la conversión ahora
- DDC_START_ON_CTOUT15 : flanco del tipo establecido en EdgeOption en el pin CTOUT_15
- ADC_START_ON_CTOUT8 : flanco del tipo establecido en EdgeOption en el pin CTOUT_8
- ADC_START_ON_ADCTRIG0 : flanco del tipo establecido en EdgeOption en el pin ADCTRIG0
- ADC_START_ON_ADCTRIG1 : flanco del tipo establecido en EdgeOption en el pin ADCTRIG1
- ADC_START_ON_MCOA2 : flanco del tipo establecido en EdgeOption en el pin Motocon PWM output MCOA2

Mientras que EdgeOption

- ADC_TRIGGERMODE_RISING : el evento se produce en el flanco de subida
- ADC_TRIGGERMODE_FALLING : el evento se produce en el flanco de bajada

Lectura del ADC

Una vez realizada la configuración del periférico ADC y generado el **Start Conversion**, solo queda efectuar la lectura del valor.

La lectura se realizará en dos pasos

Verificación de Fin de conversión

Antes de realizar la lectura del calor convertido es necesario verificar si el proceso de conversión ya terminó, esto se realiza cun la instrucción

```
Chip_ADC_ReadStatus(LPC_ADC_T *pADC, uint8_t channel, uint32_t StatusType)
```

donde StatusType indica la bandera que se busca leer

- ADC_DR_DONE_STAT: bandera que indica fin de conversión.
- ADC_DR_OVERRUN_STAT: bandera para verificar si la última conversión sobrescribió la anterior sin ser antes.
- ADC_DR_ADINT_STAT: si algún canal terminó de convertir y el mismo está habilitado para generar interrupciones.

Esta función devuelve **SET** para el caso afirmativo o **RESET** para el caso contrario.

Lectura del valor

La lectura del valor convertido se realiza mediante la función

```
Chip_ADC_ReadValue(LPC_ADC_T *pADC, uint8_t channel, uint16_t *data)
```

Donde data es el puntero a la variable donde se guardará el valor.

La propia instrucción verifica antes de leer el dato si un valor nuevo espera ser leído (bit DONE) devolviendo **ERROR** si no hay dato nuevo o **SUCCESS** en caso de una lectura exitosa.

Manejo del ADC por Interrupción

Aquellos periféricos que poseen un tiempo de respuesta relativamente lento respecto a la velocidad de procesamiento del microcontrolador, los coloca como candidatos a ser administrados mediante interrupciones. El ADC entra como periférico de lenta respuesta, debido a que desde el inicio de conversión hasta su finalización transcurre 2,5 uS, un tiempo donde un micro con un clock a 200MHz puede ejecutar 500 instrucciones. Para la configuración del ADC por interrupciones, se deberá configurar lo siguiente

Habilitar el pedido de interrupción del ADC

```
void Chip_ADC_Int_SetChannelCmd(LPC_ADC_T *pADC, uint8_t ichannel,FunctionalState NewState)
```

Donde ichannel indica el canal (ADC_CH0, ADC_CH1 .. ADC_CH7) y NewState indica la habilitación (ENABLE) o la deshabituación (DISABLE) del canal seleccionado para realizar interrupciones.

Finalmente con la instrucción de la NVIC

```
NVIC_EnableIRQ(17)
```

Habilita la NVIC para capturar los pedidos de interrupción del ADC

Función para atender la interrupción

La función deberá leer el valor convertido y eventualmente lanzar una nueva conversión.

Esta función deberá reemplazar a la función por defecto que se encuentra en el vector de interrupciones dentro de **vector.c** correspondiente al vector 0x21 o IRQ 17.

```
   ISR_NoHandler,      /* 0x1e 0x00000078 - No Handler set for ISR TIMER2 (IRQ 14)
*/
   ISR_NoHandler,      /* 0x1f 0x0000007C - No Handler set for ISR TIMER3 (IRQ 15)
*/
   ISR_NoHandler,      /* 0x20 0x00000080 - No Handler set for ISR MCPWM (IRQ 16)
*/
   ADC0_IRQ,           /* 0x21 0x00000084 - No Handler set for ISR ADC0 (IRQ 17) */
   ISR_NoHandler,      /* 0x22 0x00000088 - No Handler set for ISR I2C0 (IRQ 18) */
   ISR_NoHandler,      /* 0x23 0x0000008C - No Handler set for ISR I2C1 (IRQ 19) */
   ISR_NoHandler,      /* 0x24 0x00000090 - No Handler set for ISR SPI (IRQ 20) */
   ISR_NoHandler,      /* 0x25 0x00000094 - No Handler set for ISR ADC1 (IRQ 21) */
```

(Ejemplo reemplazando la función generica ISR_NoHandler por la función ADC0_IRQ)

UntitledWiki: WebHome/Hardware/ModuloADC (última edición 2016-10-18 21:40:36 efectuada por GuillermoSteiner)

Bienvenido: Ingresar
location: WebHome / Hardware / LPC2114 / ModuloADC

ADC (Analog Digital Converter)

El ADC es un conversor analógico digital, este periférico que posee el LPC2114/2124, permite convertir señales analógicas en una cuenta o valor digital.

Si bien este periférico es único dentro del micro controlador, posee un multiplexor analógico que permite conmutar entre varias señales de entrada.

Las características del mismo son las siguientes

cantidad de bits	10
cantidad de entradas	hasta 4
modo de bajo consumo o power down	Si
rango de entrada	0 a 3 V
tiempo de conversión	minino 2.44 us

Entradas

El ADC utiliza hasta 4 pines de entrada multiplexados

- pin 11 AIN0
- pin 13 AIN1
- pin 14 AIN2
- pin 15 AIN3

Cada pin tiene diversos usos, los mismo deben se configurados para ser usados por el ADC.

Alimentación

El ADC posee entradas separadas de alimentación denominadas V_{3A} y V_{SSA}, usadas para el bloque analógico, las mismas deben estar correctamente aisladas de las demás señales, para no generar errores en la conversión producto de las fluctuaciones en estas lineas

Registro de control (ADCR = 0xE0034000)

bits 7-0	SEL	Estos bits determina en cual o cuales de los 4 canales se realizará la conversión, solo son utilizados los 4 bits menos significativos correspondiendo cada bits a cada una de las entradas, este registro funciona de dos maneras diferentes: Controlado por software, en este modo solo un bits puede ser puesto en 1, el cual será el canal donde se realizará la conversión. Controlado por hardware, en este modo se selecciona con 1 a todos los canales donde se quieren hacer una conversión, una vez activo el ADC en este modo se producirá un barrido de todos estos canales.
bits 15-8	CLKDIV	El clock interno del ADC se extrae del VPB (PCLK) dividido el valor guardado en CLKDIV, el resultado no deberá ser superior a 4,5 Mhz, generalmente este divisor de configura para producir la máxima frecuencia posible del ADC (4,5Mhz) pero en ciertos casos conviene una menor frecuencia, por ejemplo en casos de que la señal a medir provenga de fuentes de alta impedancia.
bit 16	BURS	Si el bit es 0, la conversión es controlada por software requiriendo 11 ciclos de reloj, si en cambio es 1, se activa el control por hardware, ahora el tiempo de conversión y el rango de la misma será designado por CLKS, produciendo un barrido en cada canal que se encuentre activo en el registro SEL, comenzando por el bits menos significativo en 1.La conversión continuará hasta que se borre el el bits del BURST, con lo cual se detendrá el proceso una vez que la convención que se esté efectuando termine.

bits 19-17	CLKS	Este campo es usando en el modo de control por hardware y determina el numero de ciclos usados para la conversión y el numero de bits de precisión, pudiendo elegir entre 11 clock (10bits) = 000, 10 clock (9bits) = 001 hasta 4 clock (3bits) = 111
bit 21	PDN	1 = el ADC está operacional 0 = el ADC está en modo power down
bits 23-22	TEST	estos bits se utilizan para la verificación del dispositivo 00 = operación normal, 01 = test digital, 10 = test del DAC y 11 = modo de simple conversión
bits 26-24	START	Configura de que manera el ADC comenzará a convertir, en caso de estar configurado en modo control por software (BURST = 0) 000: no comienza este es el valor a usar cuando PDN = 0 001: comenzar la conversión ahora 010: flanco del tipo establecido en EDGE en el pin P0.16 011: flanco del tipo establecido en EDGE en el pin P0.22 100: flanco del tipo establecido en EDGE en el pin P0.27/MAT0.1 101: flanco del tipo establecido en EDGE en el pin P0.29/MAT0.3 110: flanco del tipo establecido en EDGE en el pin P0.12/MAT1.0 111: flanco del tipo establecido en EDGE en el pin P0.13/MAT1.1
bit 27	EDGE	Este bits es utilizado en caso de configurar a START entre 010 a 111 EDGE = 0 la conversión inicia en un flanco de bajada EDGE = 1 la conversión inicia en un flanco de subida

Registro de datos (ADDR = 0xE0034004)

bit 31	DONE	Este registro indica con 1 que la conversión a finalizado, es borrado cuando este registro es leído y cuando el ADCR es escrito, si el ADCR es escrito mientras que se estaba realizando una conversión, este bit es puesto a 1 y una nueva conversión arranca
bit 30	OVERRUN	En modo BURST este bits está en 1 si una o varias conversiones fueron perdidas por no leer a tiempo el registro antes de que una nueva la pisara.
bits 26-24	CHN	indica el canal de la ultima conversión
bits 15-6	V/V3A	indica el resultado de la conversión

Configuración y Lectura de un ADC

Este es un ejemplo muy simple del uso del ADC Para configurar el ADC comenzamos directamente a través del registro ADCR, falta en este proceso configurar los pines en el modo ADC, esto no es necesario, debido a que este es el modo por defecto.

La configuración la vamos realizando previamente en una variable "configura", esto permite ir observando cada bit de configuración que modificamos, una vez realizada la carga de esta variable, la copiamos al registro ADCR Solo dejamos pendiente el arranque de la conversión el cual es conveniente realizarlo en una escritura separada, cuando ya la configuración esta lista

al final un do while esperará hasta que el bit 31 del ADDR esté en 1 indicando fin de conversión, usamos luego la variable leida "salida" para directamente extraerle el resultado de la conversión

```
int main (void)
{

  unsigned int configura,salida;
  unsigned char canal;

  canal = 1;
```

```
  configura = (16 <<8 );   /* CLKDIV = 16 configuro divisor externo  */
  configura |= (1 << 21 ); /* PDN = 1 ADC en modo operacional*/
  configura |= canal;      /* canal a realizar la conversión */

  AD_ADCR = configura;

  while(1) {
    AD_ADCR |= (1 << 24);  /* START = 001 comenzar la conversión inmediatamente  */

    do {
      salida = AD_ADDR;
    } while (( salida & (1<<31) ) == 0); /* DONE = 0 termino de convertir */

    salida = (salida >>6) & 0x03FF; /* tomar el resultado del bit 6 a 15*/
  }
  return 0;
}
```

UntitledWiki: WebHome/Hardware/LPC2114/ModuloADC (última edición 2013-10-27 16:09:52 efectuada por GuillermoSteiner)

Bienvenido: Ingresar
location: WebHome / Hardware / LPC2114 / ModuloGPIO

Entradas y Salidas Digitales GPIO

material para la clase:

- Manuales del usuario del lpc21xx version nueva y vieja capitulo 9 en ambos.

El GPIO son lineas que pueden actuar como entrada o salida digital, pudiendo modificar su modo individualmente, estas lineas están agrupadas en puertos, en el caso del microcontrolador LPC2114, son dos

Nombre de los Pines	**Tipos**	**Denominación del puerto**
P0.0 a P0.31	I/O	GPIO0
P1.16 a P1.31	I/O	GPIO1

Como vemos entonces cada linea de entrada salida posee un numero y un puerto, el puerto 0 o GPIO0 agrupa 32 lineas y el puerto 1 o GPIO1 agrupa 16 enumeradas de la 16 a la 31 Cada puerto posee una seria de registros para manejar los mismos, permitiendo configurar cada linea como entrada o salida, leer una entrada en particular o escribir sobre una salida Tendremos entonces registros para distintas acciones por cada puerto, donde cada bits de esos registros representa a cada linea de su puerto

Modo entrada, una linea configurada como entrada, permite leer una señal conectada a la misma, el valor leído es 0 o 1 dependiendo del estado de la linea en el momento de la lectura. Modo Salida, una linea configurada como salida, permite sacar un valor lógico alto o bajo.

Registros de GPIO

Nombre Genérico	Descripción	Acceso	Valor en Reset	Dirección y Nombre	
				Puerto 0	Puerto 1
IOPIN	En modo entrada permite leer el valor instantáneo de cada pin, en el modo salida permite determinar la salida lógica	R/W	NA	0xE0028000 IO0PIN	0xE0028010 IO1PIN
IOSET	Utilizado solo en modo salida, permite pasar a alto una salida escribiendo un 1 en el bit correspondiente, la salida de los bit en 0 no son tocadas	R/W	0x0000000	0xE0028004 IO0SET	0xE0028014 IO1SET
IODIR	Configura individualmente a cada pin como entrada o salida, 1 indica salida 0 entrada	R/W	0x0000000	0xE0028008 IO0DIR	0xE0028018 IO1DIR
IOCLR	Utilizado solo en modo salida, permite pasar a bajo una salida escribiendo un 1 en el bit correspondiente, la salida de los bit en 0 no son tocadas	W	0x0000000	0xE002800C IO0CLR	0xE002801C IO1CLR

La denominación dada a cada registro es estándar y puede encontrarse en el archivo de cabecera lp2114.h, en los siguientes ejemplos se usa este nombre en lugar de la dirección física.

Uso de la GPIO en modo salida

El primer paso es configurar cada linea como entrada o salida, esto se realiza con los registro IOxDIR, si queremos configurar por ejemplo las primeras 8 lineas del puerto 1 como salida y dejar al resto como entrada, realizamos lo siguiente

```
IO1DIR = 0x00FF0000;
```

Aquí vemos que escribimos 1 en los bits 16 a 23 del registro IO1DIR, registro que configura la dirección del puerto, además como el puerto 1 comienza en el bit 16, efectivamente estamos configurando los 8 bits mas bajos del puerto como salida dejando los 8 mas alto como entrada.

Ahora escribimos en el puerto configurado como salida el valor 0x55, esto lo podemos hacer de dos formas:

Método 1

```
unsigned int a;
a = 0x55;
IO1PIN = (a << 16);
```

Método 2

```
unsigned int a;
a = 0x55;
IO1CLR = (~a & 0xFF) <<16;
IO1SET = a <<16;
```

- **Método 1** es bastante sencillo, asignamos el valor a un entero (32 bits) lo corremos 16 bits para ponerlo en la posición adecuada dentro del word y establecemos la salida grabando ese valor en el registro IO1PIN.
- **Método 2** es un poco mas complicado, aquí usamos primero el registro IOCLR, para esto negamos el entero a sacar, hacemos un and con una mascara para dejar solo el byte menos significativo y ahora si corremos los 16 bits y actualizamos el IOCLR, luego corremos el entero sin negar 16 bits y actualizamos el IO1SET.Estos dos pasos tienen un sentido muy importante, de este modo forzamos a 0 escribiendo unos en el IOCLR y forzamos a 1 a los bit que queremos que sean 1 con el IOSET, pero en ningún momento modificamos el estado de cualquier bits fuera de los 8 correspondiente al byte, como vemos en IOCLR cualquier bit fuera de los 8 de nuestra salida nunca tendrán un valor 1, lo mismo que en IOSET, esto permite encapsular estos 8 bits, permitiendo dividir el puerto en sub grupos que no interactúen entre si.El método 1 solo se utilizará cuando no es necesario esta división, permitiendo modificar el estado de los bits con una sola operación.

Uso de la GPIO en modo entrada

En el modo entrada el único registro a usar es el IOxPIN, mediante este registro se puede leer el estado instantáneo de una entrada, generalmente para luego determinar el estado de una señal en particular se enmascara las demás lineas y se compara con 0 el estado final.

Ejemplo Para determinar el estado de un botón que se encuentra en la linea P0.15 se procede de la siguiente forma.

```
int i;
while(true) {
   if(IO1PIN & (1<<15)) {
      for(i=0;i<1000;i++);
      if(IO1PIN & (1<<15)) {
         /* botón efectivamente apretado */
         while(IO1PIN & (1<<15));
      }
   }
}
```

En el ejemplo realizamos un if(IO1PIN & (1<<15)) esto nos da verdadero si y solo si el bit 15 del IO1PIN está en 1, cualquier elemento mecánico (botón, llave, etc) posee un tiempo de establecimiento o rebote durante el cual el microcontrolador puede leer 0 y 1 alternativamente, para evitar esto y para asegurarse

de que efectivamente se presionó una tecla y no fue un ruido, se realiza un retardo de un par de milisegundos, luego se lee nuevamente la entrada con otro if(, asegurándose la correcta lectura.

Luego de atender la presión del botón y antes de salir a esperar otro evento, se puede esperar en un bucle hasta que suelte el botón, evitando que entre inmediatamente por continuar el pin P0.15 en 1.

UntitledWiki: WebHome/Hardware/LPC2114/ModuloGPIO (última edición 2013-10-27 16:09:51 efectuada por GuillermoSteiner)

Bienvenido: Ingresar
location: WebHome / Hardware / LPC2114 / ModuloPLL

PLL (PHASE LOCKED LOOP)

El PLL o Lazo Enganchado de Fase, es básicamente un sistema de lazo cerrado de frecuencia, se basa en un control de fase que mide la diferencia entre una frecuencia de entrada patrón y la frecuencia de un oscilador controlado, de esta forma el error de fase actuará sobre el oscilador de frecuencia variable modificando su salida acercándola a la frecuencia del oscilador patrón.

Intercalando divisores podemos generar frecuencias superiores a la frecuencia patrón, este uso es el dado por el microcontrolador, permitiendo de este modo elevar la frecuencia desde un cristal de 10 a 20 Mhz a 10 a 60 Mhz de frecuencia interna.

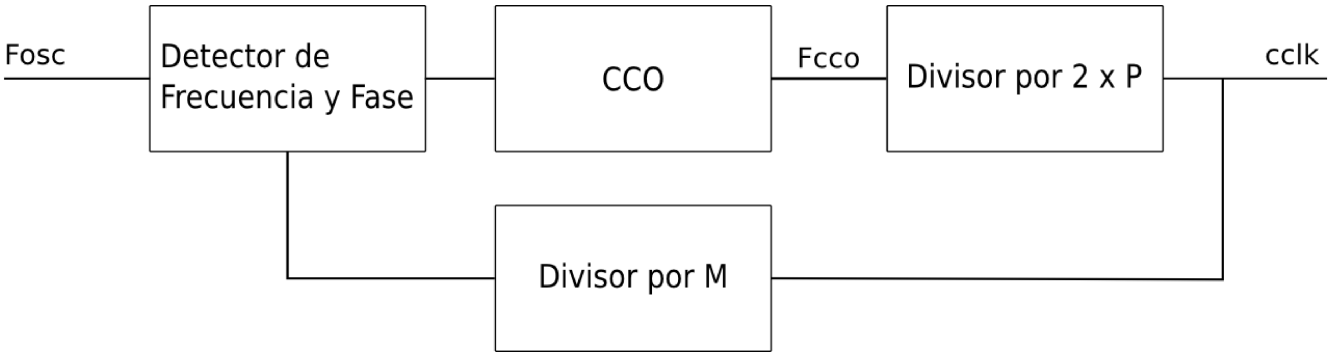


Tabla de las distintas frecuencias que intervienen en el PLL

Nombre	Descripción	Rango de operación
Fosc	Frecuencia del oscilador externo	10 a 20 Mhz
Fcco	Frecuencia del oscilador controlado por corriente	156 Mhz a 320 Mhz
cclk	Frecuencia interna del microcontrolador	10 a 60 Mhz

En la figura se muestra el diagrama simplificado del PLL aplicado al microcontrolador, como vemos, tenemos dos divisores que permiten configurar al dispositivo a una frecuencia interna para una frecuencia dada del cristal externo.

Por un lado tenemos el multiplicador M, este modulo divide la frecuencia interna por un entero y este valor es comparado con la frecuencia externa, el echo de llamarlo multiplicador, se debe a que del punto de vista de la frecuencia interna, ésta será la frecuencia del oscilador externo multiplicada por M

Si por ejemplo tenemos un oscilador de 10 Mhz y este divisor lo configuramos a 6, entonces el oscilador se acomodará para dar una salida a la frecuencia interna de 60 Mhz.

El divisor P, es utilizado para configurar la frecuencia del oscilador interno a un rango dado por el fabricante, en este caso 156 Mhz a 320 Mhz, una vez establecido la frecuencia que queremos mediante M, establecemos P para que el OCC funcione en el rango en el cual fue diseñado, siguiendo el ejemplo con P = 2 la frecuencia del OCC es 60 * 2 * 2 = 240Mhz el cual es una frecuencia valida.

Veamos ahora el calculo para un oscilador externo de 14,7456 Mhz

Establecemos M = 4

cclk = M * Fosc = 4 * 14,7456 = 58,9824

ahora P deberá ser 2

Fcco = cclk * 2 * 2 = 58,9824 * 2 * 2 = 235,9296 Mhz

Queda para este ejemplo entonces M = 4 y P = 2

Configuración del PLL en el ARM

La configuración del PLL se realiza mediante los siguientes registros

Dirección	Nombre	Descripción	Acceso
0xE01FC080	PLLCON	Registro de Control, permite la habilitación del PLL, las modificaciones realizadas no tienen efecto hasta que se realiza la secuencia de actualización.	R/W
0xE01FC084	PLLCFG	Registro de Configuración, establece los divisores P y M, las modificaciones realizadas no tienen efecto hasta que se realiza la secuencia de actualización	R/W
0xE01FC088	PLLSTAT	Registro de STATUS, informa sobre el estado del PLL y los divisores que se están usando en este momento	R
0xE01FC08C	PLLFEED	Registro FEED o de habilitación del PLL, escribiendo una secuencia de caracteres en este registro (secuencia de actualización) se habilitan las modificaciones echas en PLLCON y PLLCFG	W

Registro de Control

El PLLCON es un registro de 8 bits de los cuales solo dos bits tienen función.

- bit 0 (PLLE) es el que habilita el PLL para que se enganche a la fase, valor de reset = 0.
- bit 1 (PLLC) en el momento que este bit y PLLE están en 1 y luego de una secuencia de actualización, el PLL pasa a ser el oscilador del microcontrolador, valor de reset = 0.
- bit 2 - 7 son reservados y no deben ser escrito con 1.

La secuencia correcta es: establecer primero el PLL y que este se enganche a la fase (PLLE = 1) y luego activar el PLL (PLLE = 1 y PLLC = 1)

Las diferentes combinaciones de los bits PLLC y PLLE pueden mostrarse en la siguiente tabla

PLLC	PLLE	Estado del PLL
0	0	el PLL es desactivado y desconectado, el microcontrolador funciona con el reloj externo conectado en forma directa
0	1	el PLL es activado, el PLL comienza a funcionar, pudiendo ser conectado cuando la fase se enganche (PLOCK = 1)
1	0	igual que 0 0, esto previene que se conecte el PLL sin ser habilitado previamente
1	1	el PLL está activo y es el oscilador interno del microcontrolador

Registro de Configuración

Como se describió anteriormente este registro de 8 bits, posee los valores de P y M

- MSEL bit 0 a 4 valor del multiplicador M, donde para un valor M = 1 tenemos MSEL = 00000, para M = 2 MSEL = 00001 y así hasta M = 32, si bien este sería el valor mas alto, los valores posibles son de 1 a 6, dada la menor frecuencia del oscilador externo (10 Mhz) y el mayor valor del oscilador interno (60 Mhz)
- PSEL bit 5 a 6 el divisor P se configura de la siguiente forma:

PSEL bits	Valor de P
00	1
01	2
10	4
11	5

- bit 7 es un bit reservado y no se debe escribir 1

Registro de STATUS

Provee los valores actuales del PLL, estos valores pueden ser distintos de los guardados en los registros de control y configuración, solo la secuencia de actualización hace efectivo los cambios de esos registros al PLL y mientras que esta no se produzca, los valores internos del PLL no se modifican.

- bits 0 a 4, MSEL valores actuales del multiplicador M
- bits 5 a 6, PSEL valores actuales del divisor P
- bits 7, reservado
- bits 8, PLLE valores actuales del PLLE
- bits 9, PLLC valores actuales del PLLC
- bits 10,PLOCK, refleja el estado del lazo del PLL, es decir si el PLL ya se encuentra enganchado en fase, este bits es muy importante porque permite decidir cuando se está en condiciones de que el PLL pase a ser el oscilador del microcontrolador.
- bits 11 a 15 reservados.

Interrupción

El bit PLOCK puede ser conectado al controlador de interrupciones, lo que permite activar el PLL y continuar haciendo otra cosa, en el momento en que se enganche la fase (PLOCK = 1) se genera una interrupción, la cual coloca al PLL con entrada del oscilador interno y desactiva la interrupción.

PLL Feed Register o registro de actualización

El registro de actualización es utilizado para generar la secuencia de actualización, esta secuencia es para registrar los cambios que se realicen en el PLL, esto evita cualquier error de escrituras accidentales en los registros de configuración.

La secuencia son dos caracteres que se escriben en este registro en ciclos de bus consecutivos, estos caracteres son 0xAA y 0x55, esto significa que la interrupción debe ser desactivada en este período de configuración, si los caracteres no son los correctos, o alguna de las anteriores condiciones no se cumple, los cambios en PLLCON o PLLCFG no se realizan.

Programa de configuración

Este programa realizado en assembler se debe intercalar en el STARTUP y permite configurar un PLL en esta caso para que oscile a 58,9824 Mhz partiendo de un cristal que lo hace a 14.7456 Mhz

```
...........
...........
...........
/*
* Establecer PLL
* -------------------------------------------------------------------------
*/
        @ Se usa r0 para cargar la posición del primer registro del PLL
        @ para acceder luego a los demás registros se utilizará esta dirección como
base mas un offset
        ldr r0, PLLBASE
        @ PLLCFG = PLLCFG_VALUE se configura el divisor D y el multiplicador P según
una constante
        mov r3, #PLLCFG_VALUE
        str r3, [r0, #PLLCFG_OFFSET]

        @ PLLCON = PLLCON_PLLE  se configura el PLLE = 1 y PLLC = 0 esto es para
activar el PLL
        mov r3, #PLLCON_PLLE
        str r3, [r0, #PLLCON_OFFSET]

        @ PLLFEED = PLLFEED1, PLLFEED2
        @ se realiza la secuencia de actualización para hacer efectivos los cambios
```

```
        mov r1, #PLLFEED1
        mov r2, #PLLFEED2
        str r1, [r0, #PLLFEED_OFFSET]
        str r2, [r0, #PLLFEED_OFFSET]

        @ repetir mientras ((PLLSTAT & PLLSTAT_PLOCK) == 0)
        @ este bucle se realiza hasta que se enganche la fase
pll_loop:
        ldr r3, [r0, #PLLSTAT_OFFSET]
        tst r3, #PLLSTAT_PLOCK
        beq pll_loop

        @ PLLCON = PLLCON_PLLC||PLLCON_PLLE
        @ una vez enganchada la fase se procede a configurar el PLL como oscilador del
microcontrolador
        mov r3, #PLLCON_PLLC||PLLCON_PLLE
        str r3, [r0, #PLLCON_OFFSET]

        @ PLLFEED = PLLFEED1, PLLFEED2
        @ una nueva secuencia de actualización hace efectivo el cambio y ahora el PLL
es el oscilador del sistema
        str r1, [r0, #PLLFEED_OFFSET]
        str r2, [r0, #PLLFEED_OFFSET]
...........
...........
...........
PLLBASE:        .word   0xE01FC080
/*
* las constantes de 8 bits son usadas como valores inmediatos y offset
*/

@ PLL configuration

        .equ PLLCON_OFFSET,   0x0
        .equ PLLCFG_OFFSET,   0x4
        .equ PLLSTAT_OFFSET,  0x8
        .equ PLLFEED_OFFSET,  0xC

        .equ PLLCON_PLLE,    (1 << 0)
        .equ PLLCON_PLLC,    (1 << 1)
        .equ PLLSTAT_PLOCK,  (1 << 10)
        .equ PLLFEED1,        0xAA
        .equ PLLFEED2,        0x55

        @ el valor del CFG corresponde a un M = 4 MSEL = 00011 y P = 2 PSEL = 01
        .equ PLLCFG_VALUE,    0x23

@ configuracíon del PLL para cristal de 14,7456 Mhz
```

UntitledWiki: WebHome/Hardware/LPC2114/ModuloPLL (última edición 2013-10-27 16:09:51 efectuada por GuillermoSteiner)