ARM Programming on Blueboard-LPC214x

GPIO Register

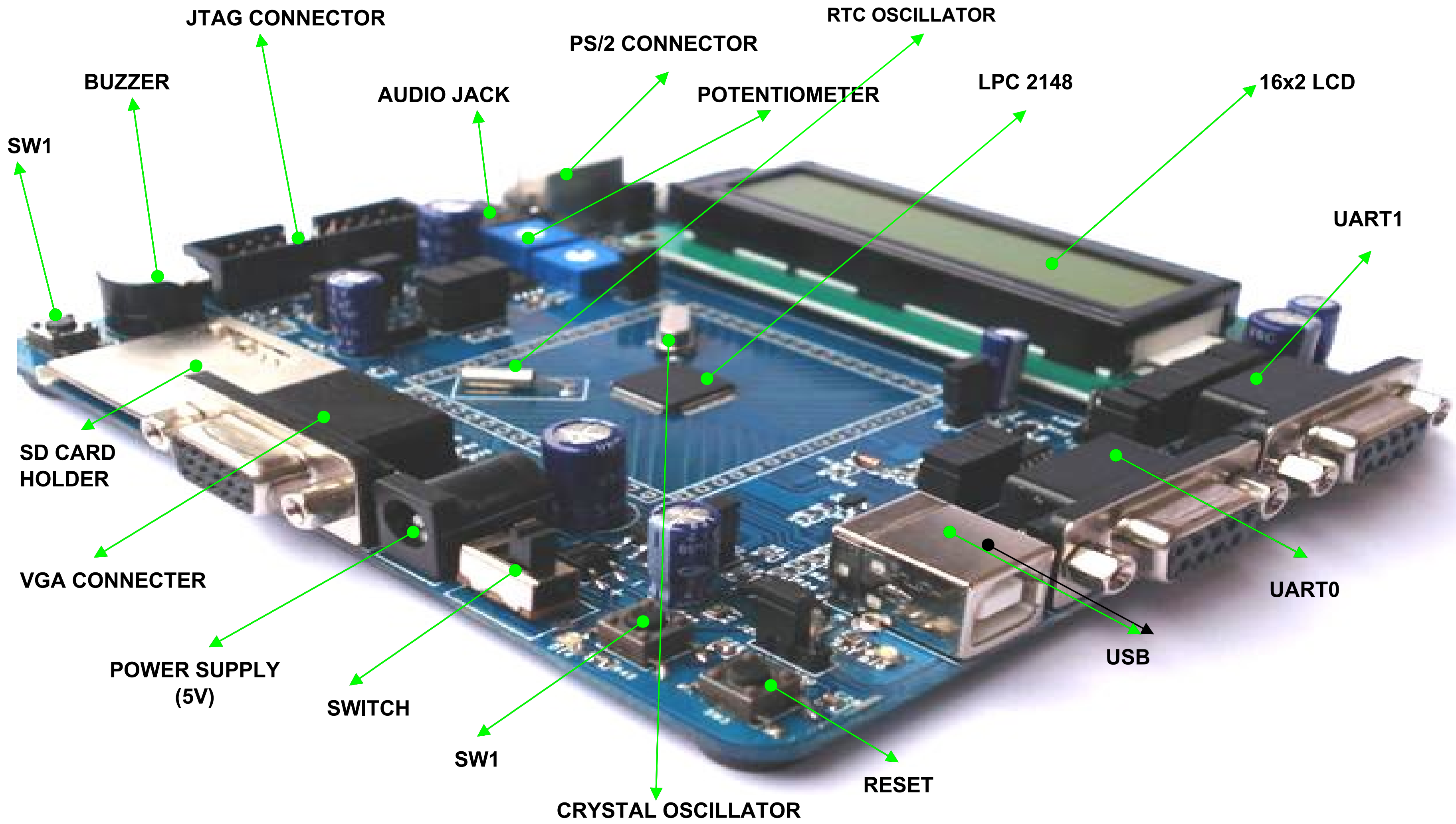


Fig 21.

LPC2148 GPIO

GPIO Block

GPIO Register

LPC2148 GPIO Ports:

PORT0 - 32 (P0.0 – P0.31) I/O Pins
PORT1 -16 (P1.16 – P1.31) I/O Pins

LPC2148 GPIO Registers:

IOPIN (R/W): To read the current status of IO port, regardless of direction

IOSET (R/W): GPIO Port Output Set register. Writing ones produces highs at the corresponding port pins

IODIR (R/W): GPIO Port Direction control register. This register individually controls the direction of each port pin

IOCLR (W): GPIO Port Output Clear register. Writing ones produces lows at the corresponding port pins and clears the corresponding bits

GPIO Programming

Programming Algo:

Step-1: Make sure the Pins are configured as GPIO's (Pin Connect Block)

Step-2: Set the direction of the PIN (IODIR Register)

Step-3: To make pin High set the corresponding bit in IOSET Register

Step-4: To make pin High set the corresponding bit in IOCLR Register

Step-5: To read the status of pin check the same bit value in IOPIN Register.

GPIO Programming

```
#include <LPC214x.H>          /* LPC214x definitions */

#define BUZZER (1 << 25)      /* Buzzer Connected to Port-1, Pin-25 */
#define BUZZER_DIR IO1DIR
#define BUZZER_SET IO1SET
#define BUZZER_CLR IO1CLR

void turn_on_buzzer(void)
{
  BUZZER_DIR |= BUZZER;
  BUZZER_CLR |= BUZZER;  /* Connected through Transister base( 0 -> ON ) */
}

void turn_off_buzzer(void)
{
   BUZZER_DIR |= BUZZER;
   BUZZER_SET |= BUZZER; /* Connected through Transister base( 1 -> OFF ) */
}
```

LPC2148 Interrupts

LPC214x Interrupts:

- Total 22-interrupts source are there in LPC2148.
- Can be served as FIQ or IRQ (Vectored IRQ / Non Vectored IR)
- FIQs – Highest priority
- Vectored IRQs – Middle priority
  - Only 16 of 32 can be assigned.
  - Any of the 32 requests can be assigned to any of the 16 vectored
  - IRQ slots. 0 – highest priority and 15 - lowest priority
  - For a Vectored IRQ, VIC provides a hardware lookup table for the address of each ISR.
- Non Vectored IRQ have the least priority

IRQ Block Registers

Vectored IRQ Registers

Registers associated with IRQs and programming :

PINSEL0 :

P0.15 P0.14	P0.13 P0.12	P0.11 P0.10	P0.9 P0.8	P0.7 P0.6	P0.5 P0.4	P0.3 P0.2	P0.1 P0.0
00 GPIO Port 0.7 0 01 SSEL0 (SPI0) 10 PWM2 11 EINT2	00 GPIO Port 0.6 0 01 MOSI0 (SPI0) 10 Capture 0.2 11 Reserved	00 GPIO Port 0.5 0 01 MISO0 (SPI0) 10 Match 0.1 11 AD0.7	00 GPIO Port 0.4 0 01 SCK0 (SPI0) 10 Capture 0.1 11 AD0.6	00 GPIO Port 0.3 0 01 SDA0 (I2C0) 10 Match 0.0 11 EINT1	00 GPIO Port 0.2 0 01 SCL0 (I2C0) 10 Capture 0.0 11 Reserved	00 GPIO Port 0.1 0 01 RxD (UART0) 10 PWM3 11 EINT0	00 GPIO Port 0.0 0 01 TXD (UART0) 10 PWM1 11 Reserved

P0.15

P0.31 P0.30	P0.29 P0.28	P0.27 P0.26	P0.25 P0.24	P0.23 P0.22	P0.21 P0.20	P0.19 P0.18	P0.17 P0.16
00 GPIO Port 0.15 0 01 Reserved 10 EINT2 11 Reserved	00 GPIO Port 0.14 0 01 Reserved 10 EINT1 11 SDA1 (I2C1)	00 GPIO Port 0.13 0 01 Reserved 10 Match 1.1-T1 11 Reserved	00 GPIO Port 0.12 0 01 Reserved 10 Match 1.0-T1 11 Reserved	00 GPIO Port 0.11 0 01 Reserved 10 Capture 1.1-T1 11 SCL1 (I2C1)	00 GPIO Port 0.10 0 01 Reserved 10 Capture 1.0-T1 11 Reserved	00 GPIO Port 0.9 0 01 RxD (UART1) 10 PWM6 11 EINT3	00 GPIO Port 0.8 0 01 TXD UART1 10 PWM4 11 Reserved

VICVectCntl0-15:

4:0 -- int_request / sw_int_assig (reff : page 59 of 2148 user manual)
5 -- When 1, this vectored IRQ slot is enabled
and rest are reserved.

Vectored IRQ Registers

VICVectAddr0-15: Hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots

VICIntSelect:

EINT1	EINT0	RTC	PLL	SPI1/SSP	SPI0	I2C0	PWM0	UART1	UART0	TIMER1	TIMER0	ARM Core1	ARM Core0		WDT
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

									USB	AD1	BOD	I2C1	AD0	EINT3	EINT2
31									22	21	20	19	18	17	16

VICIntEnable: To Enable or disable the Interrupt source. Bits are same as VICIntSelect Register.

VICVectAddr: Get loaded with any of VICVectAddr[0 -15] value for which interrupt has been generated and ready to run the ISR.

EASY ARM

Vectored IRQ Programming

Programming Algo:

Step-1: Configure the device (Interrupt Source) to generate Interrupt.

Step-2: Select the IRQ mode to server the Interrupt (VICIntSelect Reg).

Step-3: Select the slot <N> from 0 to 15 according to the priority required.

Step-4: Set the value of Vector Control[N] Register.

Step-5: Set the ISR address to Vector Address[N] Register.

Step-6: Enable the Interrupt (VICIntEnable Reg).

Inside ISR while Leaving:

1. Clear the device Interrupt status bit.
2. Write dummy value to VICVectAddr register indicating Interrupt Controller to server next interrupt.

Vectored IRQ Programming

```
#define VIC_EXTI2_CH     (0x01 << 16)
#define VIC_INT_CH_EN    (0x01 << 5)

void ExtInt2_ISR(void)  __irq
{
    if(BUZZER_STATE){
        BUZZER_OFF;
    }
    else {
        BUZZER_ON;
    }

    EXTINT |= 0x04;          /* ACK to peripheral block */
    VICVectAddr = 0x00;      /* ACK to VIC Block */
}

void configExtInt2()
{
    PINSEL0 = 0x80000000;                    /* Pin select to External INT2  */

    VICVectCntl13 =  (VIC_INT_CH_EN | 16 );       /* Select IRQ Channel and Enable it */
    VICVectAddr13 = (unsigned int)ExtInt2_ISR;    /* Assign ISR Handler address */

    VICIntSelect &= (~VIC_EXTI2_CH);         /* This is optional to ensure the IRQ Mode */
    VICIntEnable = VIC_EXTI2_CH;             /* Enabel the Interrupt for listening request */
}
```

Non-Vectored IRQ Programming

Programming Algo:

Step-1: Configure the device (Interrupt Source) to generate Interrupt.

Step-2: Select the IRQ mode to server the Interrupt (VICIntSelect Reg).

Step-3: Set the ISR address to VICDefVectAddr Register.

Step-4: Enable the Interrupt (VICIntEnable Reg).

Note:

Multiple Interrupt Sources can be configured for Non-Vectored Mode.

Status Register (VICIRQStatus) can used to identify the source inside ISR.

Inside ISR while Leaving:

1. Clear the device Interrupt status bit.
2. Write dummy value to VICVectAddr register indicating Interrupt Controller to server next interrupt.

Non-Vectored IRQ Programming

```
#define VIC_EXTI2_CH     (0x01 << 16)
#define VIC_EXTI1_CH     (0x01 << 15)

void ExtInt_ISR(void)    __irq
{
    if(VICIRQStatus & VIC_EXTI1_CH ) {
        BUZZER_ON;
        EXTINT |= 0x04;          /* ACK to peripheral block */
    }
    else if( VICIRQStatus & VIC_EXTI2_CH ) {
        BUZZER_OFF;
        EXTINT |= 0x04;          /* ACK to peripheral block */
    }

    VICVectAddr = 0x00;      /* ACK to VIC Block */
}

void configExtInt_NVI(void)
{
    PINSEL0 = 0x80000000 | 0x20000000;                /* Pin select to External INT2  */

    VICDefVectAddr = (unsigned int)ExtInt_ISR;        /* Assign ISR Handler address */
    VICIntSelect &= (~(VIC_EXTI2_CH | VIC_EXTI1_CH));   /* This is optional to ensure the
    IRQ Mode */
    VICIntEnable = (VIC_EXTI2_CH | VIC_EXTI1_CH);        /* Enabel the Interrupt for
    listening request */
}
```

FIQ Programming

Programming Algo:

Step-1: Configure the device (Interrupt Source) to generate Interrupt.

Step-2: Select the FIQ mode to server the Interrupt (VICIntSelect Reg).

Step-3: Check the FIQ ISR function assiged to FIQ_Addr in startup code.

Setp-4: Implement the FIQ ISR in application code.

Step-5: Enable the Interrupt (VICIntEnable Reg).

Inside ISR while Leaving:

1. Clear the device Interrupt status bit.
2. Write dummy value to VICVectAddr register indicating Interrupt Controller to server next interrupt.

FIQ Programming

Startup.s

```
// Exception Vectors
// Mapped to Address 0.
// Absolute addressing mode must be used.

Vectors:        LDR     PC,Reset_Addr
                LDR     PC,Undef_Addr
                LDR     PC,SWI_Addr
                LDR     PC,PAbt_Addr
                LDR     PC,DAbt_Addr
                NOP                             /* Reserved Vector */
;               LDR     PC,IRQ_Addr
                LDR     PC,[PC, #-0x0FF0]       /* Vector from VicVectAddr */
                LDR     PC,FIQ_Addr

Reset_Addr:     DD      Reset_Handler
Undef_Addr:     DD      Undef_Handler?A
SWI_Addr:       DD      SWI_Handler?A
PAbt_Addr:      DD      PAbt_Handler?A
DAbt_Addr:      DD      DAbt_Handler?A
                DD      0                       /* Reserved Address */
IRQ_Addr:       DD      IRQ_Handler?A
FIQ_Addr:       DD      FIQ_Handler?A

// Reset Handler

Reset_Handler:
```

FIQ Handler

FIQ Programming

```
void FIQ_Handler (void) __fiq;      //declare FIQ ISR
void initFiq (void);


void FIQ_Handler (void) __fiq
{
    IOSET1          = 0x00FF0000;       //Set the LED pins
    EXTINT          = 0x00000002;       //Clear the peripheral interrupt flag

}

void initFiq(void)
{
    IODIR1          = 0x00FF0000;       //Set the LED pins as outputs
    PINSEL0         = 0x20000000;       //Enable the EXTINT1 interrupt
    VICIntSelect    = 0x00008000;       //Enable a Vic Channel as FIQ
    VICIntEnable    = 0x00008000;
}
```

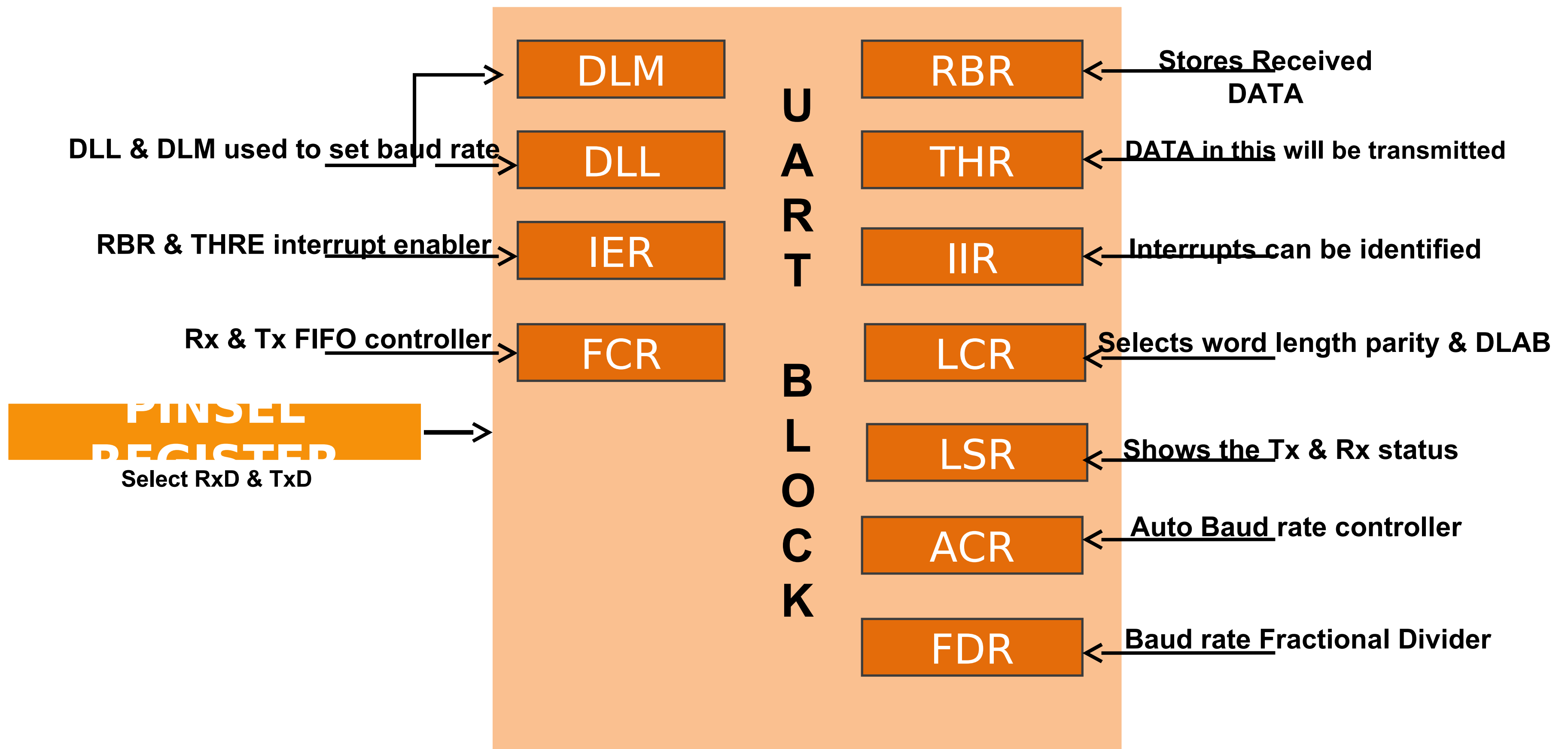
LPC2148 UART

LPC2148 UART

Features:

- LPC2148 have 2 UARTs,
- UART0 is Null modem (Rx & Tx pin only) where as UART1 is modem capable.
- 16 byte Receive and Transmit FIFOs
- Register locations conform to ‘550 industry standard.
- Receiver FIFO trigger points at 1, 4, 8, and 14 bytes.
- Built-in fractional baud rate generator with autobauding capabilities.
- Mechanism that enables software and hardware flow control implementation.

UART BLOCK



UART0 Registers

UART0 register map

Name	Description	Bit functions and addresses								Access	Reset value[1]	Address
		MSB							LSB			
		BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0			
U0RBR	Receiver Buffer Register	8-bit Read Data								RO	NA	0xE000 C000 (DLAB=0)
U0THR	Transmit Holding Register	8-bit Write Data								WO	NA	0xE000 C000 (DLAB=0)
U0DLL	Divisor Latch LSB	8-bit Data								R/W	0x01	0xE000 C000 (DLAB=1)
U0DLM	Divisor Latch MSB	8-bit Data								R/W	0x00	0xE000 C004 (DLAB=1)
U0IER	Interrupt Enable Register	-	-	-	-	-	-	En.ABTO	En.ABEO	R/W	0x00	0xE000 C004 (DLAB=0)
		-	-	-	-	-	En.RX Lin.St.Int	Enable THRE Int	En.RX Dat.Av.Int			
U0IIR	Interrupt ID Reg.	-	-	-	-	-	-	ABTO Int	ABEO Int	RO	0x01	0xE000 C008
		FIFOs Enabled		-	-	IIR3	IIR2	IIR1	IIR0			
U0FCR	FIFO Control Register	RX Trigger		-	-	-	TX FIFO Reset	RX FIFO Reset	FIFO Enable	WO	0x00	0xE000 C008
U0LCR	Line Control Register	DLAB	Set Break	Stick Parity	Even Par.Selct.	Parity Enable	No. of Stop Bits	Word Length Select		R/W	0x00	0xE000 C00C
U0LSR	Line Status Register	RX FIFO Error	TEMT	THRE	BI	FE	PE	OE	DR	RO	0x60	0xE000 C014
U0SCR	Scratch Pad Reg.	8-bit Data								R/W	0x00	0xE000 C01C
U0ACR	Auto-baud Control Register	-	-	-	-	-	-	ABTO Int.Clr	ABEO Int.Clr	R/W	0x00	0xE000 C020
		-	-	-	-	-	Aut.Rstrt.	Mode	Start			
U0FDR	Fractional Divider Register	Reserved[31:8]									0x10	0xE000 C028
		MulVal				DivAddVal						
U0TER	TX. Enable Reg.	TXEN	-	-	-	-	-	-	-	R/W	0x80	0xE000 C030

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

UART0 Registers

UxLCR (UART Line Control Register)

7	6	5	4	3	2	1	0
0 Disable access to Divisor Latches 1 Enable access to Divisor Latches.	0 Disable break transmission 1 Enable break transmission	00 Odd parity. 01 Even Parity 10 Forced "1" 11 Forced "0"		0 Disable parity 1 Enable parity.	0 1 stop bit 1 2 stop bits	00 5 bit character 01 6 bit character 10 7 bit character 11 8 bit character	

Baud Rate Calculation:

Divisor = Pclk / (16 x BAUD) [Where Divisor is 16bit value]

UxDLL = Divisor[7-0]

UxDLM = Divisor[15-8]

UART PROGRAMMING

Programming Algo:

Step-1: Configure the multiplexed I/O Pins for UART Tx & Rx mode.

Step-2: Calculate the divisor latch value for a given baud rate (Previous slide PCLK, DLL, DLM).

Step-3: Set the UART options like Data length, parity, HW Flow using UxLCR with latch access bit enable so that we can update the DLL & DLM values.

Step-4: Update the DLL & DLM, Disable the divisor latch access bit in UxLCR.

Step-5: Data can be read from THR when THRE bit of UxLSR is zero.

Step-6: To transmit data write on RBR & wait till RDR is zero.

UART PROGRAMMING

```
#define RDR      0x01
#define THRE     0x20

typedef unsigned char uc;

void uart0_init()
{
   /* initialize the serial interface   */
  PINSEL0 = 0x00000005;           /* Enable RxD0 and TxD0                    */
  U0LCR = 0x83;                   /* 8 bits, no Parity, 1 Stop bit           */
  U0DLL = 97;                     /* 9600 Baud Rate @ 15MHz VPB Clock
                                     U0DLL = 194 for 30MHz and
                                    U0DLM=  0x01; U0DLL = 0x84;  at VBP + 60MHz      */
  U0LCR = 0x03;                   /* DLAB = 0     */
}

uc get_char()
{
    while (!(U0LSR & RDR));
  return (U0RBR);
}

void put_char(uc dt)
{
    while(!(U0LSR & THRE)); // Wait until UART0 ready to send character
    U0THR = dt; // Send character
}
```

Programming Serial Interrupt Mode :

Step-1: Configure the multiplexed I/O Pins for UART Tx & Rx mode.

Step-2: Calculate the divisor latch value for a given baud rate (Previous slide PCLK, DLL, DLM).

Step-3: Set the UART options like Data length, parity, HW Flow using UxLCR with latch access bit enable so that we can update the DLL & DLM values.

Step-4: Update the DLL & DLM, Disable the divisor latch access bit in UxLCR.

Step-5: Set the FIFO Size (U0FCR)

Step-6: Do all the Vector Interrupt related stuff

Step-7: Set the UART Interrupt Enable Register U0IER

UART Interrupt

```
#define UART0_CH_NUM     6
#define VIC_UART0_CH     (0x01 << UART0_CH_NUM)
#define VIC_INT_CH_EN    (0x01 << 5)
#define MAX_BUFF_SZ 10

uc RX_BUFF[MAX_BUFF_SZ];
uc byteCount=0;

void UART0_ISR(void) __irq
{
    if( U0IIR & RDA) {                  /* Check if Recive data Interrupt event */
        if(byteCount < MAX_BUFF_SZ) {    /* Validation for buffer overflow */
                RX_BUFF[byteCount++] = U0RBR; } }
    VICVectAddr = 0x00;                   /* ACK the VIC */
}

void uart0_INT_init()
{
  PINSEL0 = 0x00000005;             /* Enable RxD0 and TxD0                    */
  U0LCR = 0x83;                     /* 8 bits, no Parity, 1 Stop bit           */
  U0DLL = 97;                       /* 9600 Baud Rate @ 15MHz VPB Clock
                                       U0DLL = 194 for 30MHz and
                                      U0DLM=  0x01; U0DLL = 0x84;  at VBP + 60MHz     */
  U0LCR = 0x03;                     /* DLAB = 0    */
  U0FCR |= 0x01;                    /* Enable Uart FIFO */
    /* Configure the VIC for Interrupt Handling */
    VICVectCntl8 |= (VIC_INT_CH_EN | UART0_CH_NUM);
    VICVectAddr8 = (unsigned int) UART0_ISR;
    VICIntSelect &= (~VIC_UART0_CH);    /* This is optional to ensure the IRQ Mode */
    VICIntEnable |= VIC_UART0_CH;       /* Enabel the Interrupt for listening request */
   /* Enable UART Interrupt */
  U0IER |= 0x01;
}
```

LPC2148
Timers & Counters

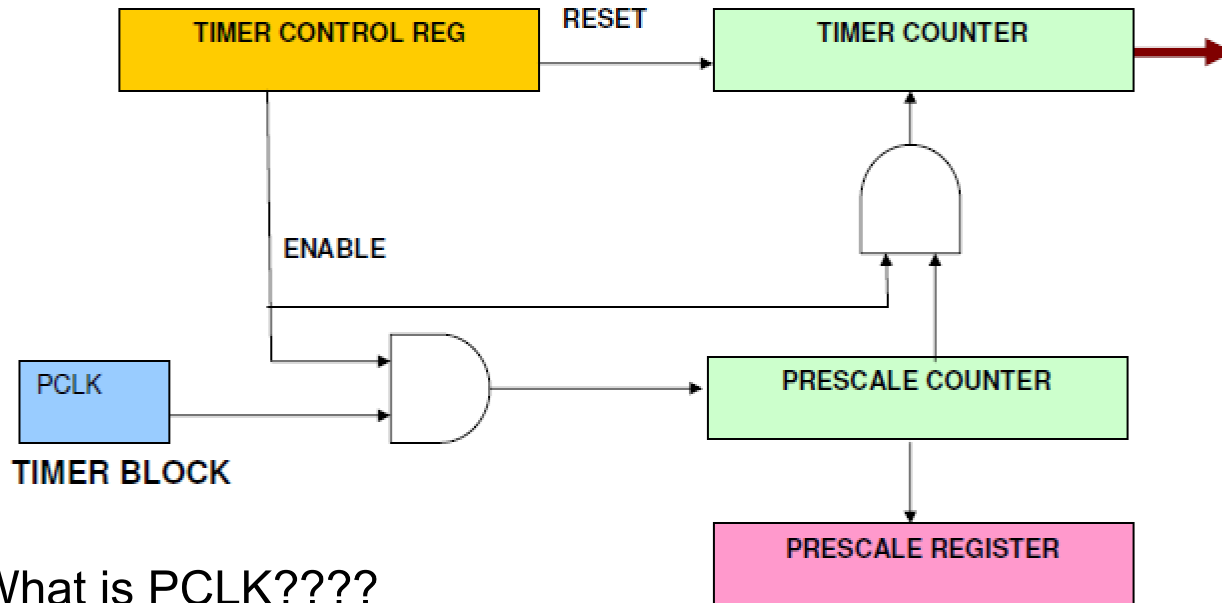
LPC214x Timers & Counters

Features :

- A 32-bit Timer/Counter with a programmable 32-bit Prescaler.
  - Counter or Timer operation
  - Up to four 32-bit capture channels per timer, that can take a snapshot of the timer value when an input signal transitions. A capture event may also optionally generate an interrupt.
- Four 32-bit match registers that allow:
  - Continuous operation with optional interrupt generation on match.
  - Stop timer on match with optional interrupt generation.
  - Reset timer on match with optional interrupt generation.
- Up to four external outputs corresponding to match registers, with the following capabilities:
  - Set low on match.
  - Set high on match.
  - Do nothing on match.

LPC214x Timer Block

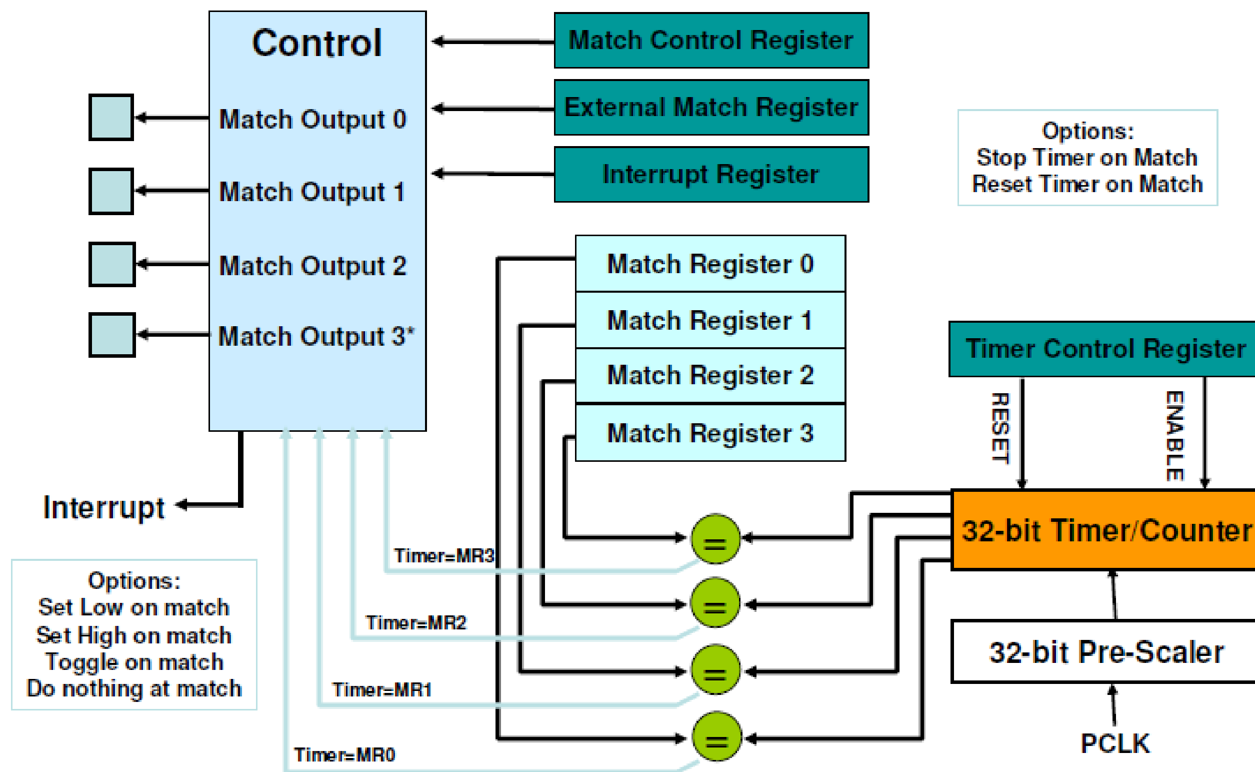
LPC214x Timer Block



What is PCLK????

- The Prescale Counter is incremented on every PCLK.
- When Prescale Counter reaches the value stored in the Prescale Register, the Timer Counter is incremented and the Prescale Counter is reset on the next PCLK

Timer Match Block



The Match register values are continuously compared to the Timer Counter value, when the two values are equal, actions (Match pin / Interrupt) can be triggered automatically.

Timer Registers

Registers associated with Timer/Counter programming:

T1PR - Timer Prescale register

T1MR0 - The Match register values are continuously compared to the Timer Counter value

T1MCR – Ref Page no. 248 of LPC2148 user manual

T1TCR – To start the Timer

For Timer Interrupt:

VICVectCntl -- To select the channel and Enable it. (What is channel??)

VICVectAddr – To assign the interrupt vector address

VICIntEnable – To Enable the interrupt (Timer interrupt)

Timer Programming

Programming Algo:

Step-1. Switch off the timer using TCR

Step-2. Select the Timer or Counter

Step-3. Set the counter registers [Optional]

Step-4. Set the Prescale register

Step-5. Configure Match / Capture Register

Step-6. Configure the interrupt if generated.

Step-7. Switch ON the timer.

Timer Programming-1

```
#define TIMER_TICK_1_MS (59999) /* VPB at 60 MHz clock */

void TimerInit()
{
    T1TCR = 0x00;        /* ensure timer 1 is off */
    /* set timer counter and prescale counter */
    T1TC = 0x0;
    T1PC = 0x0;

    /* set prescale and match values to give 1 ms "tick" */
    T1PR = 0x0;
    T1MR0 = TIMER_TICK_1_MS;

    /* set control to interrupt on MR0 match, and reset to 0 (i.e. "tick") */
    T1MCR = 0x3;

    /* vector TIMER 1 as mid-level priority and enabled */
    VICVectCntl11 = 0x25;

    /* vector TIMER 1 interrupt address */
    VICVectAddr11 = (unsigned int) Timer_ISR;

    /* enable TIMER 1 in VIC */
    VICIntEnable = 0x0020;

    /* go start timer 1... */
    T1TCR = 0x1;
}
```

Timer Programming-2

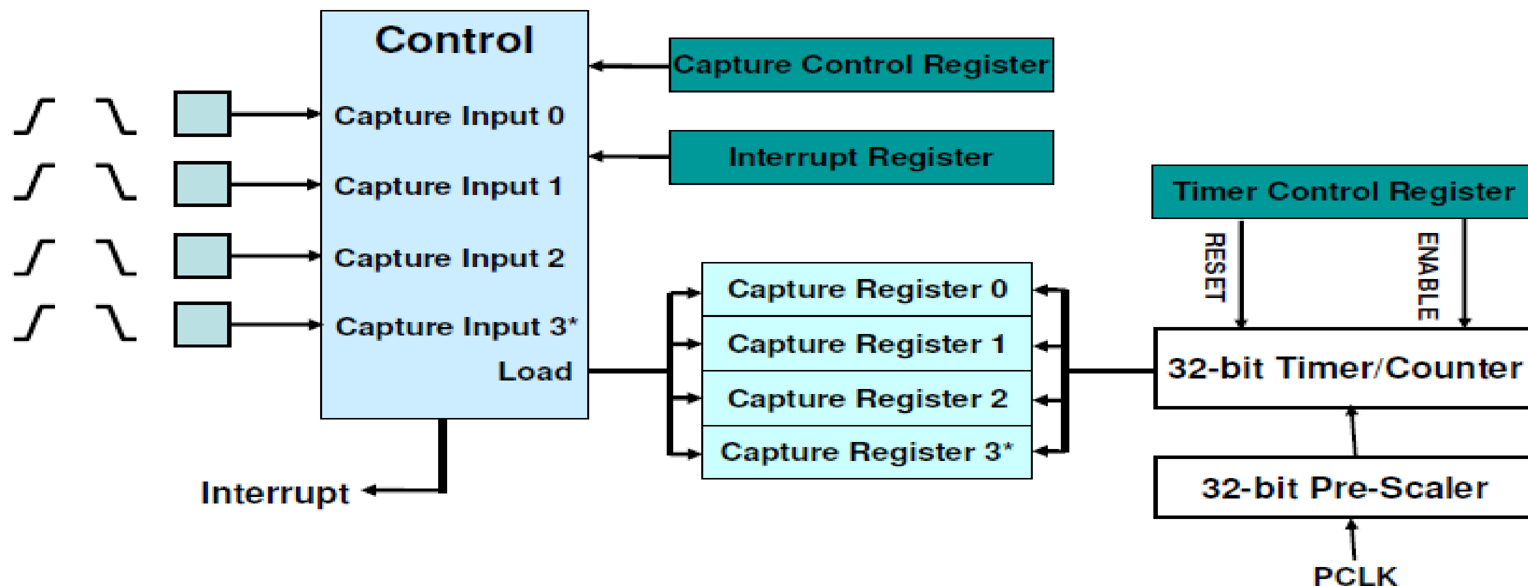
```
static unsigned int ms_count;

void Timer_ISR(void) __irq
{
    ms_count++;    /* Count increments for every 1MSec */

    if(ms_count == 1000) {
        if(IO1PIN & BUZZER)
        {
            BUZZER_ON;
        }
        else
        {
            BUZZER_OFF;
        }
        ms_count = 0;
    }

    T1IR = 0x01;
    VICVectAddr = 0x00;
}
```

Timer 0/1 Block Diagram (Capture)



When Capture Input on pin occurs Timer Counter Reg. value is latched to Capture Register.

Capture Mode Programming

```
void T0isr(void)     __irq;

void TimerCaptureInit(void)
{
    VPBDIV       = 0x00000002;               //Set pclk to 30 Mhz
    PINSEL0      = 0x00000020;               //Enable pin 0.2 as capture channel0
    T0PR         = 0x0000001E;               //Load prescaler for 1 Msec tick

    T0TCR        = 0x00000002;               //Reset counter and prescaler
    T0CCR        = 0x00000005;               //Capture on rising edge of channel0
    T0TCR        = 0x00000001;               //enable timer

    VICVectAddr4 = (unsigned)T0isr;          //Set the timer ISR vector address
    VICVectCntl4 = 0x00000024;               //Set channel
    VICIntEnable = 0x00000010;               //Enable the interrupt
}

void T0isr (void)    __irq
{
    static int value;
    value         = T0CR0;                   // read the capture value
    T0IR         |= 0x00000001;              //Clear match 0 interrupt
    VICVectAddr   = 0x00000000;              //Dummy write to signal end of interrupt

}
```

LPC214X RTC

LPC214X RTC Features

Features:

- RTC is a onchip module to measure time passage using counter register.

  And can be used to maintain calendar and clock.
- It provides all Seconds, Minutes, Hours, Day of Month, Month, Year,

  Day of Week, and Day of Year.
- RTC runs on 32KHz clock and can have dedicated crystal or driven by VPB clock
- It also provides comparison registers to generate ALARM.
- Provides Consolidated to perform single read to a set of registers.

RTC Block

RTC Registers

Name	Size	Description	Access	Reset value[1]	Address
ILR	2	Interrupt Location Register	R/W	*	0xE002 4000
CTC	15	Clock Tick Counter	RO	*	0xE002 4004
CCR	4	Clock Control Register	R/W	*	0xE002 4008
CIIR	8	Counter Increment Interrupt Register	R/W	*	0xE002 400C
AMR	8	Alarm Mask Register	R/W	*	0xE002 4010
CTIME0	32	Consolidated Time Register 0	RO	*	0xE002 4014
CTIME1	32	Consolidated Time Register 1	RO	*	0xE002 4018
CTIME2	32	Consolidated Time Register 2	RO	*	0xE002 401C
SEC	6	Seconds Counter	R/W	*	0xE002 4020
MIN	6	Minutes Register	R/W	*	0xE002 4024
HOUR	5	Hours Register	R/W	*	0xE002 4028
DOM	5	Day of Month Register	R/W	*	0xE002 402C
DOW	3	Day of Week Register	R/W	*	0xE002 4030
DOY	9	Day of Year Register	R/W	*	0xE002 4034
MONTH	4	Months Register	R/W	*	0xE002 4038
YEAR	12	Years Register	R/W	*	0xE002 403C
ALSEC	6	Alarm value for Seconds	R/W	*	0xE002 4060
ALMIN	6	Alarm value for Minutes	R/W	*	0xE002 4064
ALHOUR	5	Alarm value for Seconds	R/W	*	0xE002 4068
ALDOM	5	Alarm value for Day of Month	R/W	*	0xE002 406C
ALDOW	3	Alarm value for Day of Week	R/W	*	0xE002 4070
ALDOY	9	Alarm value for Day of Year	R/W	*	0xE002 4074
ALMON	4	Alarm value for Months	R/W	*	0xE002 4078
ALYEAR	12	Alarm value for Year	R/W	*	0xE002 407C
PREINT	13	Prescaler value, integer portion	R/W	0	0xE002 4080
PREFRAC	15	Prescaler value, integer portion	R/W	0	0xE002 4084

RTC ALARM Programming

```
void RTC_init(void)
{
    PREINT          = 0x00000392;        //Set RTC prescaler for 12.000Mhz Xtal
    PREFRAC         = 0x00004380;
    SEC             = 0x00000000;
    ALSEC           = 0x00000009;        //Set alarm register for 3 seconds
    AMR             = 0x000000FE;        //Enable seconds Alarm

    CCR             = 0x00000001;        //Start the RTC

    VICVectAddr13   = (unsigned)RTC_isr; //Set the timer ISR vector address
    VICVectCntl13   = 0x0000002D;        //Set channel
    VICIntEnable    = 0x00002000;        //Enable the interrupt
}

void RTC_isr(void)  __irq
{
    if(ILR & 0x00000002)    {     // RTC Alarm Interrupt
       if(BUZZER_STATE) {
           BUZZER_OFF;
        }
       else {
           BUZZER_ON;
        }
        ILR         = 0x00000002;          //Clear the interrupt register
    }
  VICVectAddr   = 0x00000000;       //Dummy write to signal end of interrupt for VIC
}
```

ADC Registers

Features:

- RTC is a onchip module to measure time passage using counter register.

And can be used to maintain calendar and clock.

- It provides all Seconds, Minutes, Hours, Day of Month, Month, Year,

  Day of Week, and Day of Year.

- RTC runs on 32KHz clock and can have dedicated crystal or driven by VPB clock
- It also provides comparison registers to generate ALARM.
- Provides Consolidated to perform single read to a set of registers.

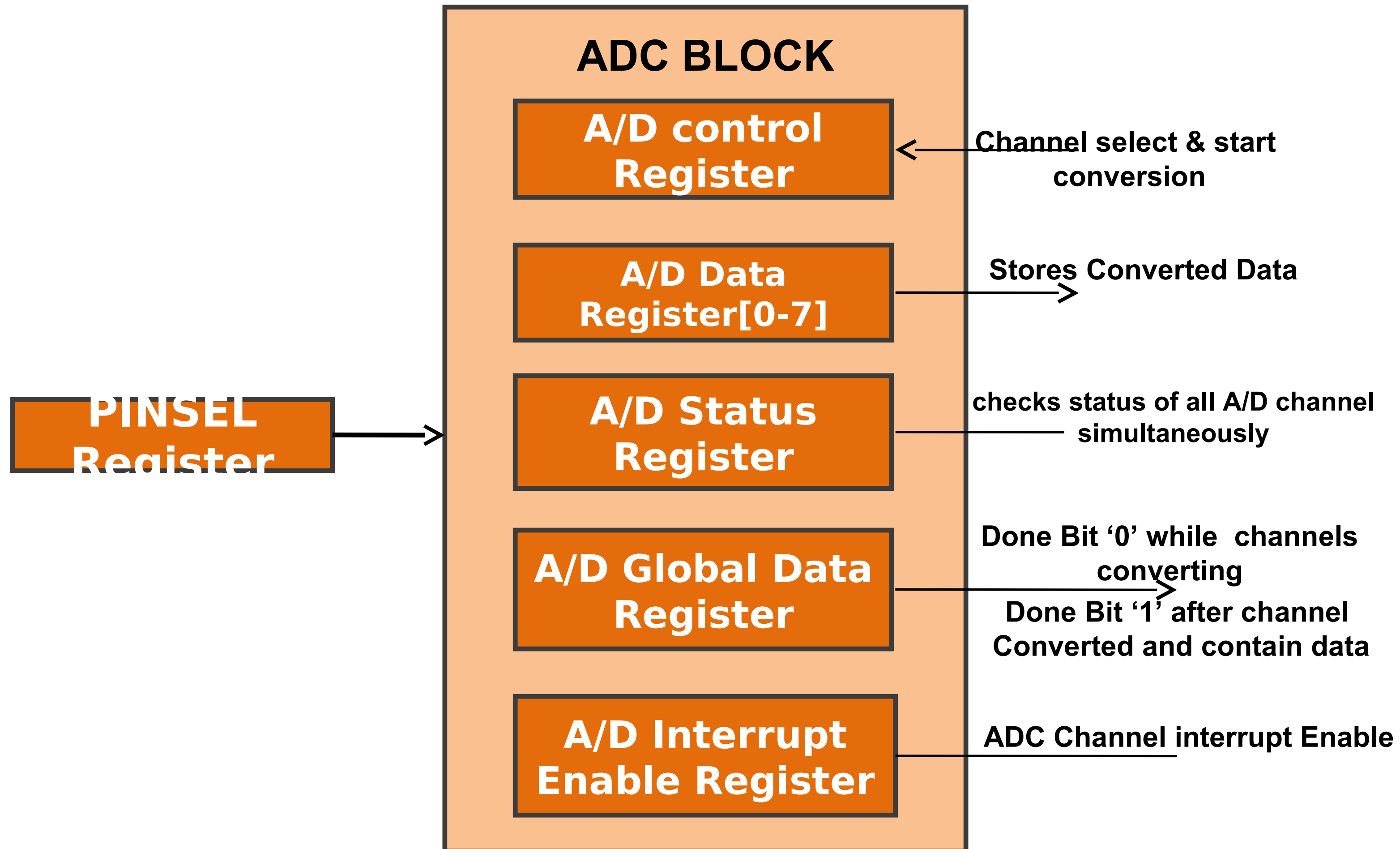
LPC214X ADC

LPC214X ADC Features

Features:

- Two ADC with 8 channels each(ADx0 to ADx7).
- 10-bit successive approximation Analog-to-Digital Converter (ADC).
- Power-down mode.
- Measurement range 0 to 3.6 V. Do not exceed the VDD voltage level.
- 10-bit conversion time ≥ 2.44 μs.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition on input pin or Timer Match signal.
- Global Start command for both converters (LPC2144/6/8 only).

LPC214X ADC BLOCK



ADC Registers

Features:

- Two ADC with 8 channels each(ADx0 to ADx7).
- 10-bit successive approximation Analog-to-Digital Converter (ADC).
- Power-down mode.
- Measurement range 0 to 3.6 V. Do not exceed the VDD voltage level.
- 10-bit conversion time ≥ 2.44 µs.
- Burst conversion mode for single or multiple inputs.
- Optional conversion on transition on input pin or Timer Match signal.
- Global Start command for both converters (LPC2144/6/8 only).

ADC Control Register:

ADC Data Register

Table 260: A/D Data Registers (ADDR0 to ADDR7, ADC0: AD0DR0 to AD0DR7 - 0xE003 4010 to 0xE003 402C and ADC1: AD1DR0 to AD1DR7- 0xE006 0010 to 0xE006 402C) bit description

Bit	Symbol	Description	Reset value
5:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AIN pin, divided by the voltage on the V_{REF} pin (V/V_{REF}). Zero in the field indicates that the voltage on the AIN pin was less than, equal to, or close to that on V_{SSA}, while 0x3FF indicates that the voltage on AIN was close to, equal to, or greater than that on V_{REF}.	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

ADC Programming

Programming Algo:

Step-1. Configure the multiplexed I/O Pins for ADC mode [PINSEL]

Step-2. Calculate the CLKDIV value such that ADC should operate at 4.5MHz or less.

Step-3. Select the ADC Channel.

Step-4. Start ADC conversion by using the ADC CR register

Step-5. Wait until conversion is done by using the Data Register.

ADC Programming

```
#include <LPC214x.h>

#define DONE    (0x80000000)

void init_adc0(void)
{
  PINSEL1 = (PINSEL1 & ~(3 << 28)) | (1 << 28);
}


unsigned short adc0_read(unsigned char ch)
{
    AD0CR = 0x00200D00 | (1<<ch);        // CLKDIV, PDN and Channel is set
    AD0CR |= 0x01000000;                 // Start A/D Conversion

    while(AD0GDR & (unsigned int)DONE); // Wait till converstion done.

    return ((AD0GDR >> 6) & 0x03FF); // bit 6:15 is 10 bit AD value
}
```

LPC214x DAC

LPC214X DAC Features

Features:

- It’s a 10-bit DAC
- Easy to program only one register.
- Power-down mode supported.
- Resistor string Architecture.

DAC Register

DAC Register (DACR - 0xE006 C000)

This read/write register includes the digital value to be converted to analog, and a bit that trades off performance vs. power. Bits 5:0 are reserved for future, higher-resolution D/A converters.

Table 262: DAC Register (DACR - address 0xE006 C000) bit description

Bit	Symbol	Value	Description	Reset value
5:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the A_{OUT} pin (with respect to V_{SSA}) is VALUE/1024 * V_{REF}.	0
16	BIAS	0	The settling time of the DAC is 1 µs max, and the maximum current is 700 υA.	0
		1	The settling time of the DAC is 2.5 µs and the maximum current is 350 µA.	
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

DAC Programming

```
/* DAC Initilization */
void dacInit (void)
{
    PINSEL1 = (PINSEL1 & ~PINSEL1_P025_MASK) | PINSEL1_P025_AOUT;
    DAC_CR = 0;
}

/* DAC Operation */
unsigned int dacSet (unsigned int newValue)
{
  unsigned int dacCR;
  unsigned int dacCurrentValue;

  dacCR = DAC_CR;
  dacCurrentValue = (dacCR & DAC_CR_VALUEMASK) >> DAC_CR_VALUESHIFT;
  dacCR = (dacCR & ~DAC_CR_VALUEMASK) | ((newValue << DAC_CR_VALUESHIFT) & DAC_CR_VALUEMASK);
  DAC_CR = dacCR;

  return dacCurrentValue;
}
```

LPC214x I2C

I2C Features

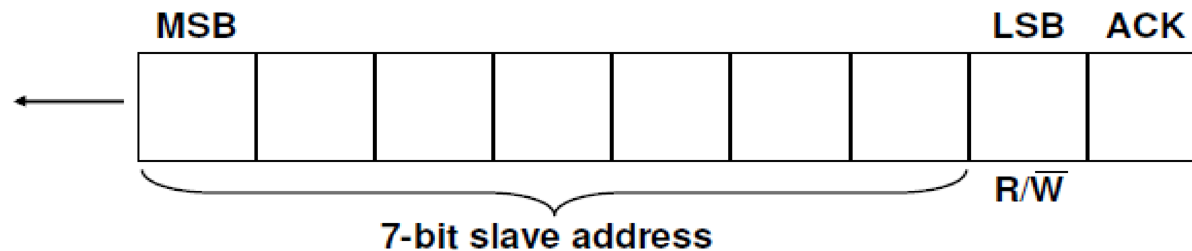
Features:

- Standard Fast-I2C compliant bus interface (7-bit addressing)
- Easy to configure as Master, Slave, or Master/Slave
- Programmable clocks allow versatile rate control
- Bi-directional data transfer between masters and slaves
- Multi-master bus (no central master)

- Standard mode speed: 100KHz
- Fast mode speed : 400KHz
- High speed mode: 3.4MHz

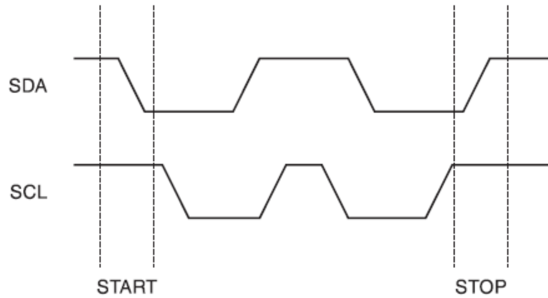
I2C Protocol

First Byte Transmitted on the I²C Bus

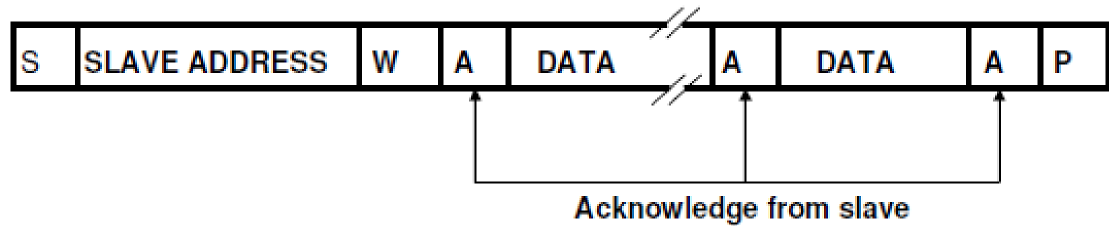


R/$\overline{W}$:
0 - Slave will be written by master.
1 - Slave will be read by master.

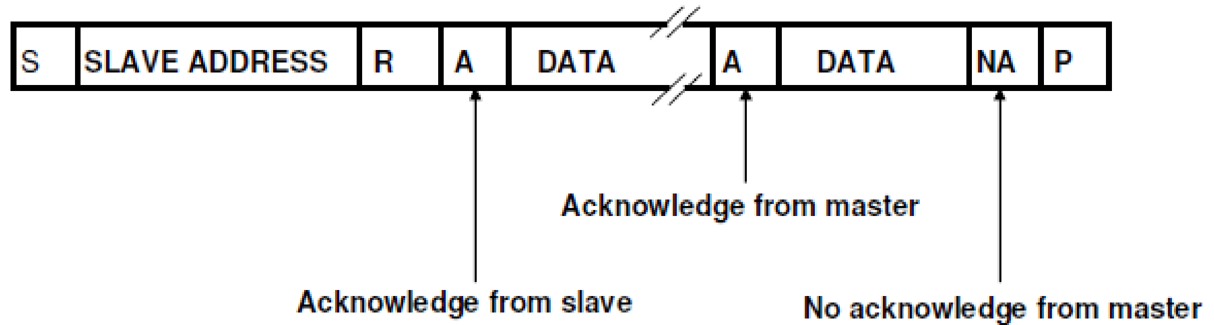
i2C Protocol



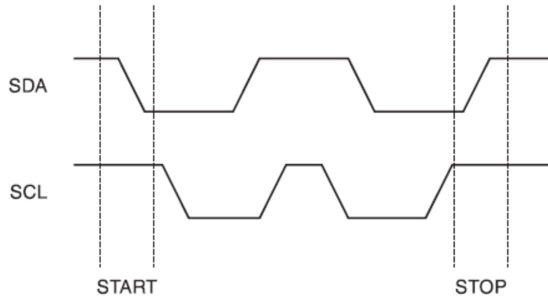
- **Master Write:**



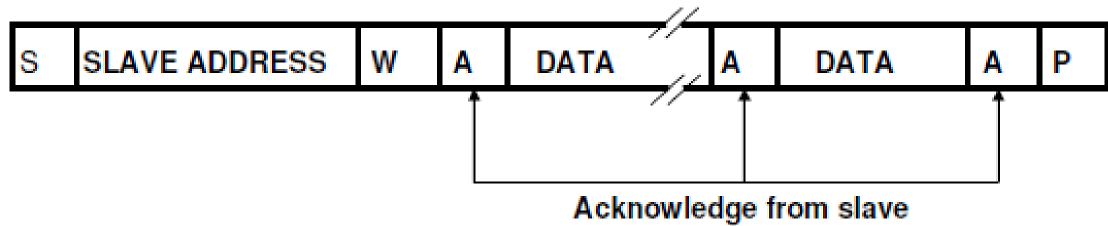
- **Master Read:**



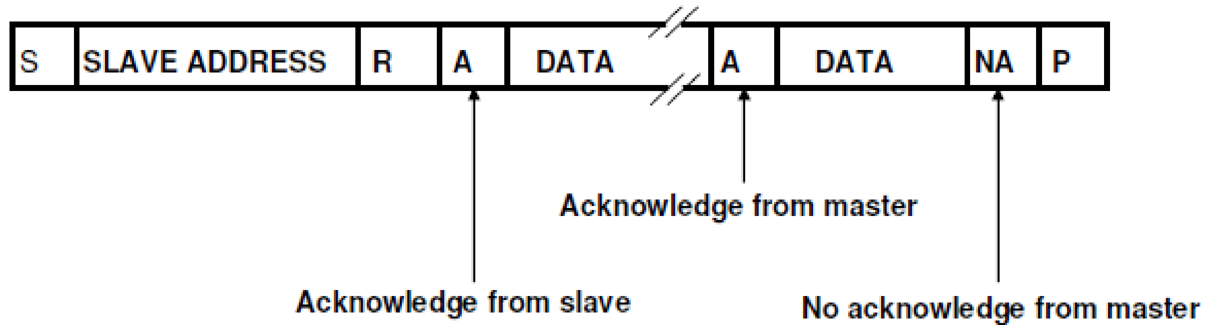
i2C Protocol



- **Master Write:**



- **Master Read:**



i2C Registers

Name	Description	Access	Reset value[1]	I^2C0 Address and Name	I^2C1 Address and Name
I2CONSET	**I^2C Control Set Register.** When a one is written to a bit of this register, the corresponding bit in the I^2C control register is set. Writing a zero has no effect on the corresponding bit in the I^2C control register.	R/W	0x00	0xE001 C000 I2C0CONSET	0xE005 C000 I2C1CONSET
I2STAT	**I^2C Status Register.** During I^2C operation, this register provides detailed status codes that allow software to determine the next action needed.	RO	0xF8	0xE001 C004 I2C0STAT	0xE005 C004 I2C1STAT
I2DAT	**I^2C Data Register.** During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register.	R/W	0x00	0xE001 C008 I2C0DAT	0xE005 C008 I2C1DAT
I2ADR	**I^2C Slave Address Register.** Contains the 7-bit slave address for operation of the I^2C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the general call address.	R/W	0x00	0xE001 C00C I2C0ADR	0xE005 C00C I2C1ADR
I2SCLH	**SCH Duty Cycle Register High Half Word.** Determines the high time of the I^2C clock.	R/W	0x04	0xE001 C010 I2C0SCLH	0xE005 C010 I2C1SCLH
I2SCLL	**SCL Duty Cycle Register Low Half Word.** Determines the low time of the I^2C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I^2C master and certain times used in slave mode.	R/W	0x04	0xE001 C014 I2C0SCLL	0xE005 C014 I2C1SCLL
I2CONCLR	**I^2C Control Clear Register.** When a one is written to a bit of this register, the corresponding bit in the I^2C control register is cleared. Writing a zero has no effect on the corresponding bit in the I^2C control register.	WO	NA	0xE001 C018 I2C0CONCLR	0xE005 C018 I2C1CONCLR

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

i2C Programming

```
#define  I2C_FLAG_AA     (1<<2)
#define  I2C_FLAG_SI     (1<<3)
#define  I2C_FLAG_STO    (1<<4)
#define  I2C_FLAG_STA    (1<<5)
#define  I2C_FLAG_I2EN   (1<<6)
#define SCHL  400    //75

void i2c_lpc_init(void)
{
  PINSEL0 = (PINSEL0 & ~(3 << 4)) | (1 << 4);
  PINSEL0 = (PINSEL0 & ~(3 << 6)) | (1 << 6);

   I2C0SCLH = SCHL;
   I2C0SCLL = SCHL;

   I2C0CONCLR = 0xFF;             //-- Clear all flags
   I2C0CONSET = 0x40;             //-- Set Master Mode
   I2C0CONSET |= I2C_FLAG_I2EN; //--- Enable I2C
}

//------------------------------------------------------------------
static void i2c_lpc_wr_byte(int byte)
{
   I2C0DAT = byte;
   I2C0CONCLR = I2C_FLAG_SI;                  //-- Clear SI
   while(!(I2C0CONSET & I2C_FLAG_SI));        //-- End wr POINT
}
```

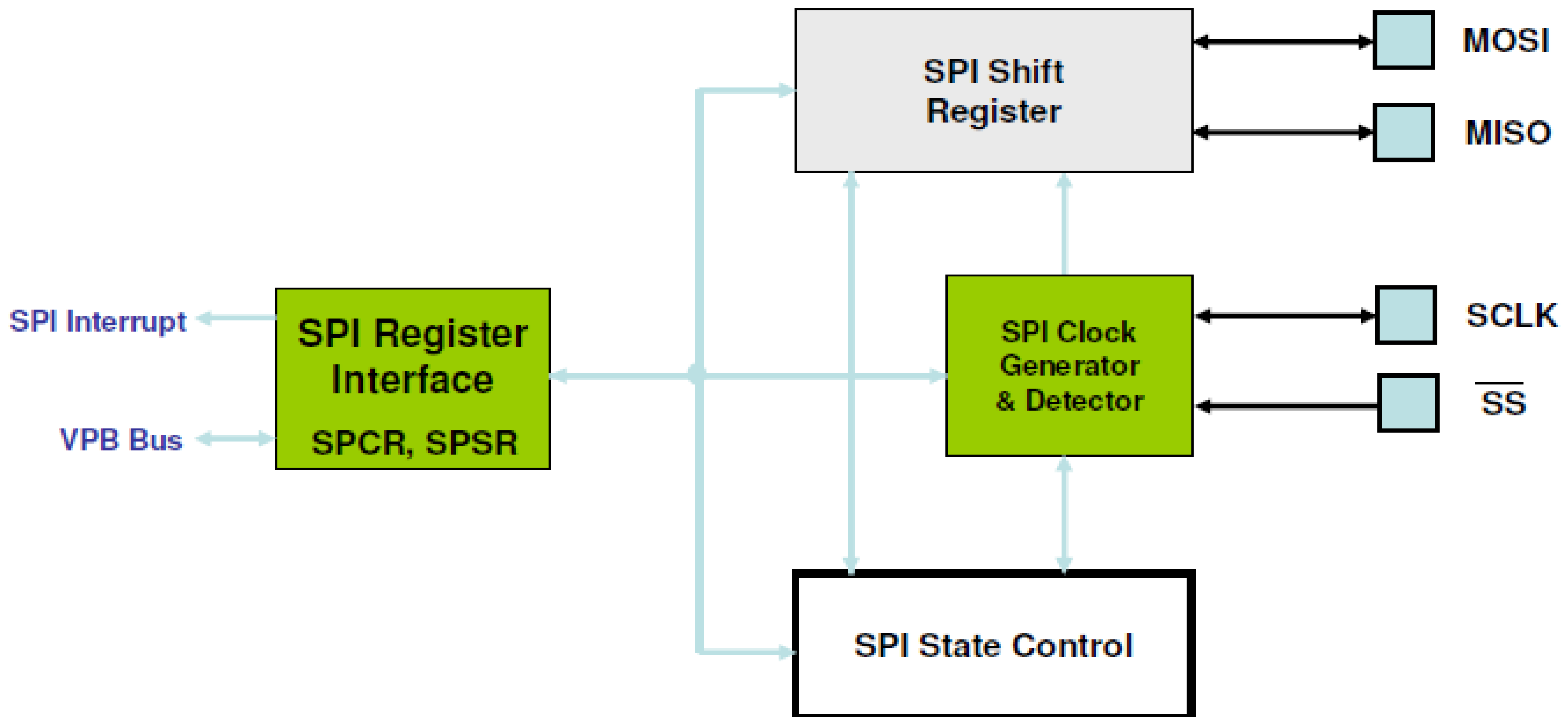
LPC214x SPI

SPI Features

Features:

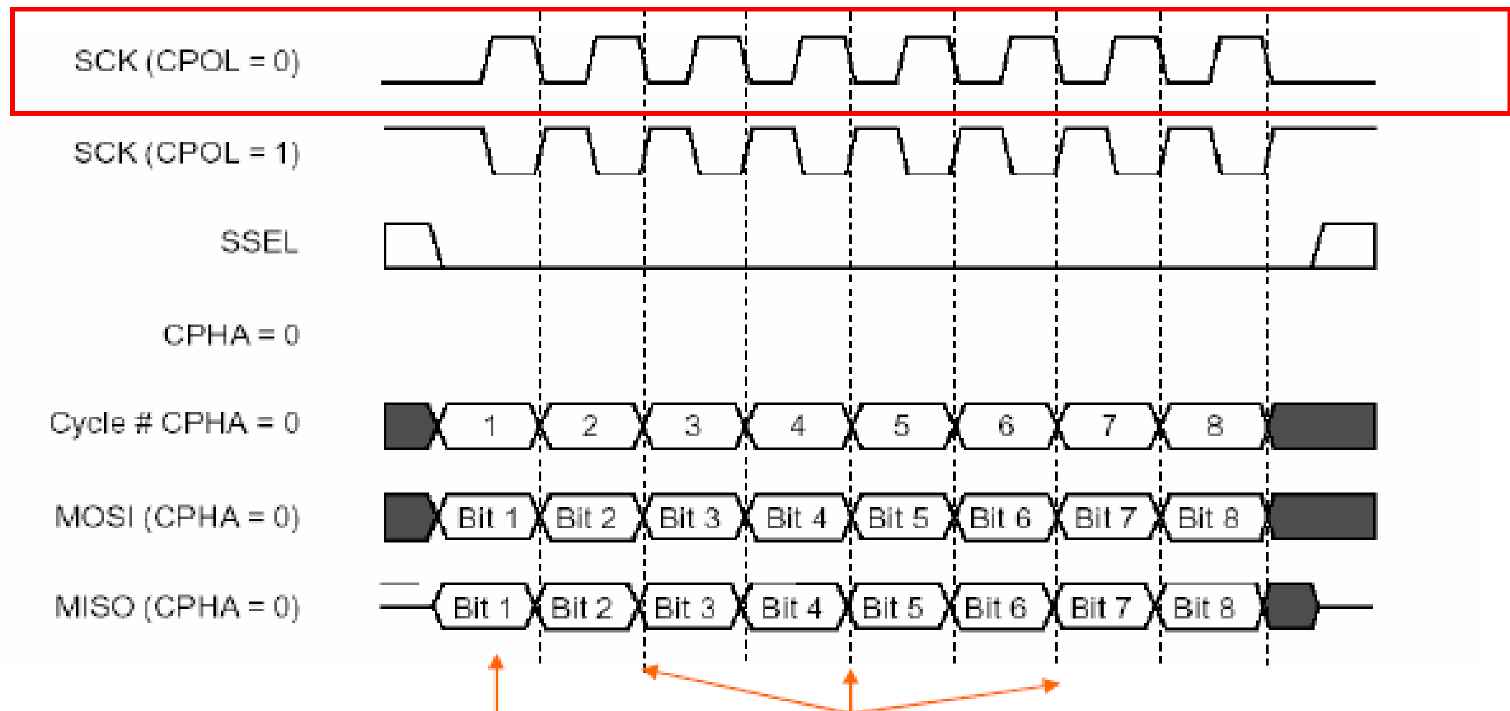
- Compliant with Serial Peripheral Interface (SPI) specification
- Combined SPI master and slave function
- Maximum data bit rate of 1/8 of the peripheral clock rate
- Programmable settings for data transmit/receive operations
  - Clock polarity and clock phase
  - MSB / LSB first

SPI Block



EASY ARM

i2C Protocol



CPOL And CPHA Settings	First Data Driven	Other Data Driven	Data Sampled
CPOL = 0, CPHA = 0	Prior to first SCK rising edge	SCK falling edge	SCK rising edge
CPOL = 0, CPHA = 1	First SCK rising edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 0	Prior to first SCK falling edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 1	First SCK falling edge	SCK falling edge	SCK rising edge

SPI Registers

Table 155: SPI register map

Name	Description	Access	Reset value[1]	Address
S0SPCR	SPI Control Register. This register controls the operation of the SPI.	R/W	0x00	0xE002 0000
S0SPSR	SPI Status Register. This register shows the status of the SPI.	RO	0x00	0xE002 0004
S0SPDR	SPI Data Register. This bi-directional register provides the transmit and receive data for the SPI. Transmit data is provided to the SPI0 by writing to this register. Data received by the SPI0 can be read from this register.	R/W	0x00	0xE002 0008
S0SPCCR	SPI Clock Counter Register. This register controls the frequency of a master's SCK0.	R/W	0x00	0xE002 000C
S0SPINT	SPI Interrupt Flag. This register contains the interrupt flag for the SPI interface.	R/W	0x00	0xE002 001C

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

SPI Programming

```
#define SPI_PORT_CHIP_READY       BIT_(SPI_RDY)

#define SPI_SPSR_SPIF            ( 0x01 << 7)

//SPI Configuration No Interrupt | MSB First | Master | Mode:3 | Reserved | Reserved |
Reserved
#define SPI_CONFIGURATION         ( 0 | 0 | BIT_(MSTR) | BIT_(CPOL) | BIT_(CPHA) | 0 | 0 | 0 )

#define SPI_CLOCK_DIVISOR         64  // n > 8 and even number  (64 = 234k)


void SPI_Init(void)
{
  unsigned char dt;
  PINSEL0 = (PINSEL0 & ~(3 << 12)) | (1 << 12);
  PINSEL0 = (PINSEL0 & ~(3 << 10)) | (1 << 10);
  PINSEL0 = (PINSEL0 & ~(3 << 8)) | (1 << 8);

    /* initialize SPI UART */
   SPI_PORT_DATA_SET_REG  = SPI_PORT_DATA_REG_INIT;   /* Initialize Chip Select High */

   S0SPCCR = SPI_CLOCK_DIVISOR;                      /* SPI Clock Divisor        */
   S0SPCR  = SPI_CONFIGURATION;                      /* SPI Config Register      */
   dt   = S0SPSR;                                /* Clear Pending Status     */
   dt   = S0SPDR;                                /* Clear Read Data Register */
   dt = dt;     // Just to remove compiler warning
}
```

SPI Programming

```
void SPI_ChipSelect(unsigned char Select)
{
   if (Select)  {
      SPI_PORT_DATA_CLR_REG = SPI_PORT_CHIP_SELECT;   //Clear bit

   }  else  {
      SPI_PORT_DATA_SET_REG = SPI_PORT_CHIP_SELECT;   //Set   bit

   }
}


unsigned char SPI_WriteByte(unsigned char valueIn)
{
  S0SPDR = valueIn;

  while (!(S0SPSR & SPI_SPSR_SPIF));

  return S0SPDR;
}
```