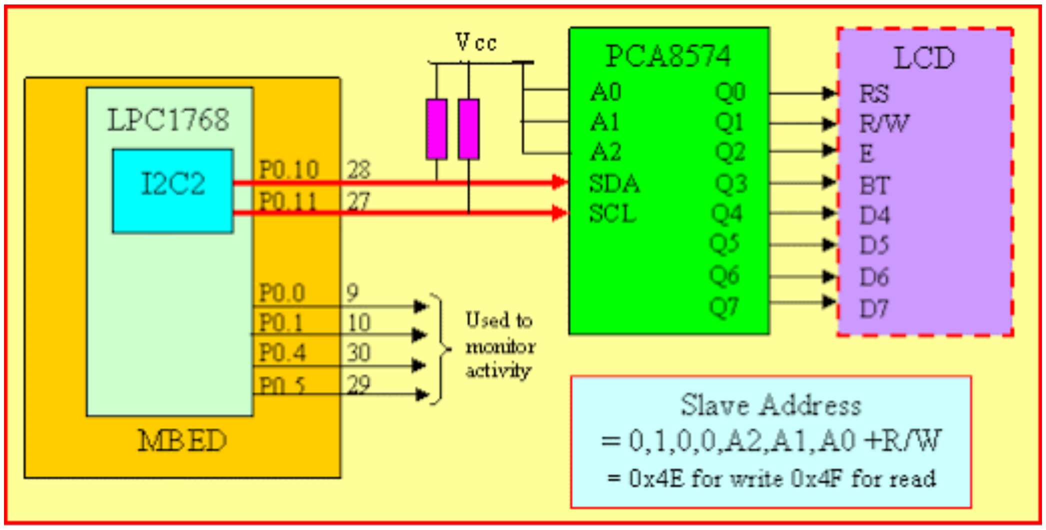
18b.ARM I2C >

I2C_LPC1768

I2C with the NXP LPC1768

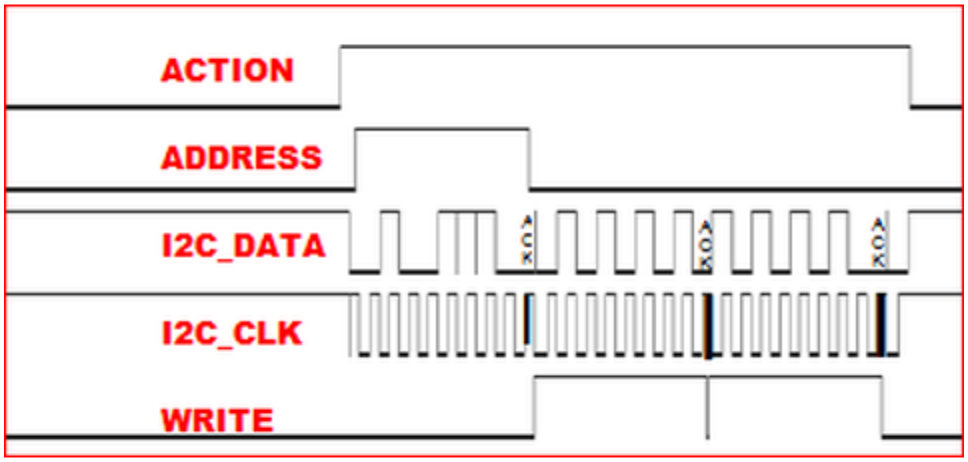
1. I2C and the NXP LPC1768

This page is concerned with the ARM I2C peripheral in the LPC1768 micro-controller. The page will interface the micro-controller to an I2C to parallel port interface as illustrated in the following circuit. The parallel port is used to drive an LCD as illustrated.



ARM LPC1768 I2C interface to an I2C to parallel port.

Sample waveforms are shown below



I2C Write 2 bytes waveforms.

A brief explanation is as follows:

1. The ARM I2C peripheral sends a START condition. Data going low while the clock is active high)
2. The slave address 0x4E is sent indicating thefollowing operations will be a write to the slave address 0x4E
3. The addressed slave acknowledges its address on the 9th clock pulse (asserts SDA low).
4. The ARM I2C peripheral writes one or more bytes to the slave. The slave will treat this information as data. In the case of the PCA8574 the data will appear on its output port.
5. After each byte of data on the 9th clock pulse the slave will acknowledge the receipt of the data.
6. The master sends as much data as it wishes (in this example 2 bytes) and terminates the transfer with a STOP sequence.

2. NXP LPC1766 Pins relevant to I2C.

Pin	**Function**	**Function 2**	**Function 3**	**Function 4**	**MBED**	**Application Board**

No	**1**	**Alternate Function 1**	**Alternate Function 2**	**Alternate Function 3**		
24	P0.28	SCL0	USB_SCL			
25	P0.27	SDA0	USB_SDA			
46	P0.0	CAN_RX1	TXD3	SDA1	Pin 9	XBee Tx
47	P0.1	CAN_TX1	RXD3	SCL1	Pin 10	XBee Rx
48	P0.10	TXD2	SDA2	MAT3.0	Pin 28	SDA Temperature Sensor Accelerometer
49	P0.11	RXD2	SCL2	MAT3.1	Pin 27	SCL Temperature Sensor Accelerometer

NXP LPC 1768 pins and functions relevant to I2C.

Since I2C_2 comes to the output pins of the MBED and will be used in the examples that follow. That is pins P010 and P0.11 must be programmed as alternate function 2.

3. Powering the ARM I2C peripheral

To conserve power the power to each peripheral may be enabled/disabled as illustrated in the following table. Note at reset the power to some peripherals is enabled while to others it is disabled. For the I2C the power is enabled at reset.

Bit	**Symbol**	**Description**	**Reset value**
0	-	Reserved.	NA
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	-	Reserved.	NA
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I2C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP 1 interface power/clock control bit.	1
11	-	Reserved.	NA
12	PCADC	A/D converter (ADC) power/clock control bit.	0
13	PCCAN1	CAN Controller 1 power/clock control bit.	0

14	PCCAN2	CAN Controller 2 power/clock control bit.	0
15	PCGPIO	Power/clock control bit for IOCON, GPIO, and GPIO interrupts.	1
16	PCRIT	Repetitive Interrupt Timer power/clock control bit.	0
17	PCMCPWM	Motor Control PWM	0
18	PCQEI	Quadrature Encoder Interface power/clock control bit.	0
19	PCI2C1	The I2C1 interface power/clock control bit.	1
20	-	Reserved.	NA
21	PCSSP0	The SSP0 interface power/clock control bit.	1
22	PCTIM2	Timer 2 power/clock control bit.	0
23	PCTIM3	Timer 3 power/clock control bit.	0
24	PCUART2	UART 2 power/clock control bit.	0
25	PCUART3	UART 3 power/clock control bit.	0
26	PCI2C2	I2C interface 2 power/clock control bit.	1
27	PCI2S	I2S interface power/clock control bit.	0
28	-	Reserved.	NA
29	PCGPDMA	GPDMA function power/clock control bit.	0
30	PCENET	Ethernet block power/clock control bit.	0
31	PCUSB	USB interface power/clock control bit.	0

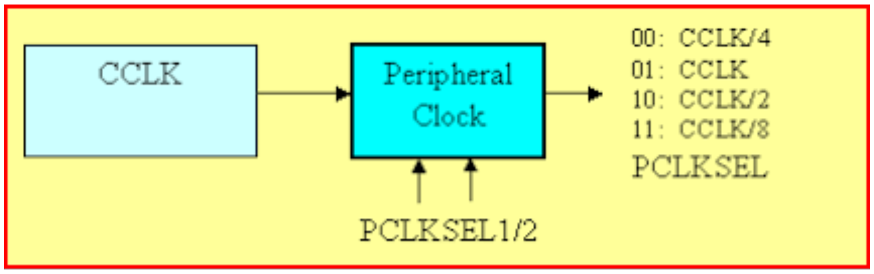
Power Control for Peripherals register (PCONP - address 0x400F C0C4) bit description

Example: Enable power to peripheral I2C2

LPC_SC->PCONP |= 1 <<26; //re-enable POWER to I2C_2 if required

4. The ARM I2C peripheral Clock

As illustrated each LPC1768 peripheral has a clock derived from the main clock.



Peripheral Clock Divider

As shown in the figure the frequency of the peripheral clock is determined by two bits in the PCLKSEL registers. Following reset PCLKSEL0/1 are both cleared which sets the peripheral clock frequency to CCLK/4.

The following table specifies the relationship between the peripherals and the bits in the PCLKSEL0/1 registers.

Bit	Symbol	Description	Reset value
1:0	PCLK_WDT	Peripheral clock selection for WDT.	00
3:2	PCLK_TIMER0	Peripheral clock selection for TIMER0.	00
5:4	PCLK_TIMER1	Peripheral clock selection for TIMER1.	00
7:6	PCLK_UART0	Peripheral clock selection for UART0.	00
9:8	PCLK_UART1	Peripheral clock selection for UART1.	00
11:10	-	Reserved.	NA
13:12	PCLK_PWM1	Peripheral clock selection for PWM1.	00
15:14	PCLK_I2C0	Peripheral clock selection for I2C0.	00
17:16	PCLK_SPI	Peripheral clock selection for SPI.	00
19:18	-	Reserved.	NA
21:20	PCLK_SSP1	Peripheral clock selection for SSP1.	00
23:22	PCLK_DAC	Peripheral clock selection for DAC.	00
25:24	PCLK_ADC	Peripheral clock selection for ADC.	00
27:26	PCLK_CAN1	Peripheral clock selection for CAN1.[1]	00
29:28	PCLK_CAN2	Peripheral clock selection for CAN2.[1]	00
31:30	PCLK_ACF	Peripheral clock selection for CAN acceptance filtering.[1]	00

Peripheral Clock Selection Register 0 (PCLKSEL0)

Bit	Symbol	Description	Reset value
1:0	PCLK_QEI	Peripheral clock selection for the Quadrature Encoder Interface.	00
3:2	PCLK_GPIOINT	Peripheral clock selection for GPIO interrupts.	00
5:4	PCLK_PCB	Peripheral clock selection for the Pin Connect block.	00
7:6	PCLK_I2C1	Peripheral clock selection for I2C1.	00
9:8	-	Reserved.	NA
11:10	PCLK_SSP0	Peripheral clock selection for SSP0.	00
13:12	PCLK_TIMER2	Peripheral clock selection for TIMER2.	00
15:14	PCLK_TIMER3	Peripheral clock selection for TIMER3.	00
17:16	PCLK_UART2	Peripheral clock selection for UART2.	00
19:18	PCLK_UART3	Peripheral clock selection for UART3.	00
21:20	PCLK_I2C2	Peripheral clock selection for I2C2.	00
23:22	PCLK_I2S	Peripheral clock selection for I2S.	00
25:24	-	Reserved.	NA
27:26	PCLK_RIT	Peripheral clock selection for Repetitive Interrupt Timer.	00
29:28	PCLK_SYSCON	Peripheral clock selection for the System Control block.	00

31:30	PCLK_MC	Peripheral clock selection for the Motor Control PWM.	00

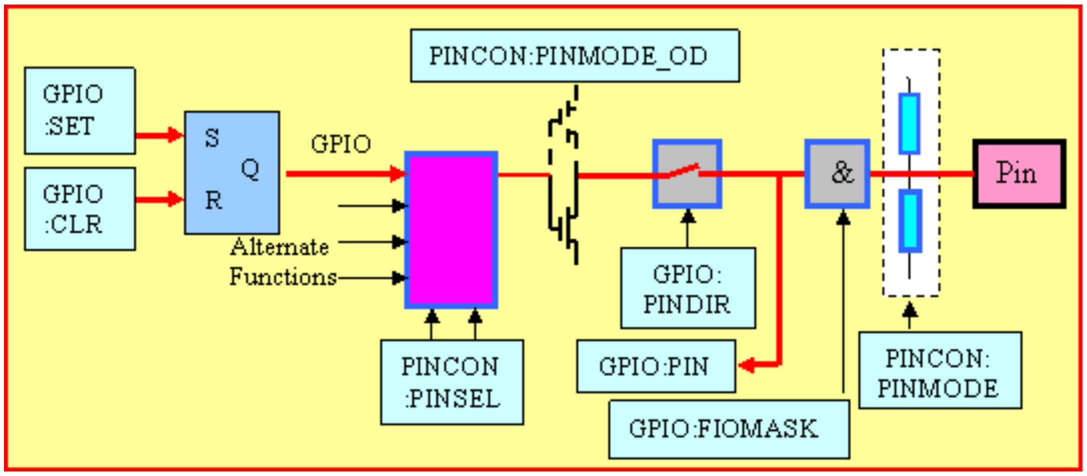
Peripheral Clock Selection Register 1 (PCLKSEL1)

Example Return the peripheral clock for I2C2 to the system clock frequency (CCLK).

LPC_SC-&GT;PCLKSEL1 |= 1<<20; //pclk = cclk

5. Selecting the Alternate I2C Function

The following sections will work through the design of the SPI interface. At power up the pins are general purpose inputs so they must first be programmed for their alternate function.



Simplified block diagram of NXP LPC1768 input-output pin.

Each pin is controlled by 2 bits in the PINSEL register. PINSEL0[1:0] controls PIN P0.0, PINSEL[3:2] controls PIN P0.1. PINSEL[31:30] controls PIN P0.15, PINSEL1[1:0] controls PIN P0.16, PINSEL2[1:0] controls PIN P1.0 etc.

Register	Description	Logic	Default
PINSEL	Selects function to drive output	00: GPIO 01: AF1 selected 10: AF2 selected 11: AF3 selected	GPIO selected

I2C Example The I2C interface to a NXP LPC1768 microcontroller uses pins P0.10 and P0.11.

The following code will select alternate function 2 (I2C) for pins P0.10 and P0.11:

LPC_PINCON->PINSEL0 |= 0x02<<20; //Pin P0.10 allocated to alternate function 2
LPC_PINCON->PINSEL0 |= 0x02<<22; //Pin P0.11 allocated to alternate function 2

6. Configuring the LPC1768 output pins.

I2C2 on the MBED uses pins P0.10 through P0.11. These pins must be configured as outputs, open drain, no pull up or pull down.

The relevant registers are:

Register	**Description**	**Logic**	**Default**
PINDIR	Sets port as input/output	0:input. 1: output	Pin is input
PINMODE	Enables pullup pulldown resistor.	00: Pull up 01:repeater mode 10: No pull up or pull down 11: Pull down	Pull up active
PINMODE_OD	Open drain on driver transistors.	0: Push pull 1: Open drain	Push pull output

The registers are distributed across the GPIO (DIR) and PCON (Pin Control Block).

With 5 general purpose input-output ports numbered GPIO0 through GPIO4 there will be corresponding DIR and PINMODE_OD register numbered 0 through 4. However for each port there will be two PINMODE (and PINSEL) registers numbered 0 through 9. PINMODE0 and PINMODE1 will configure the pull up / pull down resistors for GPIO0 with PINMODE0 configuring GPIO0 bits 0 through 15 while PINMODE1 configures GPIO0 bits 16 through 31. PINMODE2 and PINMODE3 will refer to GPIO1 bits 0-15 and bits 16-31 etc.

Example: Program the I2C2 pins as output, open drain and no pull up or pull down resistors.

```
LPC_GPIO0->FIODIR |= 1<<10;              //Bit P0.10 an output
LPC_GPIO0->FIODIR |= 1<<11;              //Bit P0.11 an output
LPC_PINCON->PINMODE0 &= ~(3<<20);
LPC_PINCON->PINMODE0 |= (2<<20);         //P0.10 has no pull up/down resistor
LPC_PINCON->PINMODE0 &= ~(3<<22);
LPC_PINCON->PINMODE0 |= (2<<22);         //P0.10 has no pull up/down resistor
LPC_PINCON->PINMODE_OD0 |= 1<<10; //Bit P0.10 is open drain
LPC_PINCON->PINMODE_OD0 |= 1<<11; //Bit P0.11 is open drain
```

7. I2C SCL HIGH and LOW Duty Cycle register.

Bit	Symbol	Description	Reset value
15:0	SCLH	Count for SCL HIGH time period selection.	0x0004
31:16	-	Reserved. The value read from a reserved bit is not defined.	NA

I2C SCL HIGH Duty Cycle register bit description (I2C0SCLH - address 0x4001 C010; I2C1SCLH - address 0x4005 C010; I2C2SCLH - 0x400A 0010)

Bit	Symbol	Description	Reset value

15:0	SCLL	Count for SCL low time period selection.	0x0004
31:16	-	Reserved. The value read from a reserved bit is not defined.	NA

I2C SCL Low duty cycle register bit description (I2C0 - I2C0SCLL: 0x4001 C014; I2C1SCLL: 0x4005 C014; I2C2SCLL: 0x400A 0014)

The I2C_bit_frequency is given by the formula:

I2C_bit_frequency = PCLK_I2C / (I2CSCLH + I2CSCLL)

Example: Determine possible values of I2CSCLH and I2CSCLL to give standard (100kHz) I2C rate given:

PCLK_I2C = 12Mhz following earlier example (using the internal clock).

Require: (I2CSCLH + I2CSCLL) = PCLK_I2C / I2C_bit_frequency = 12 /0.1 = 120

Select: I2CSCLH = I2CSCLL = 60

```
LPC_I2C2->I2SCLH = 60; //100kHz from 12MHz
LPC_I2C2->I2SCLL = 60; //100kHz from 12MHz
```

8. I2C Slave Address registers .

These registers apply to I2C slave devices. Each slave can be programmed with 4 different device addresses.

Bit	Symbol	Description	Reset value
0	GC	General Call enable bit. When this bit is set, the General Call address (0x00) is recognized.	0
7:1	Address	The I2C device address for slave mode.	0x00
31:8	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

I2C Slave Address registers bit description
I2C0ADR[0, 1, 2, 3]- address 0x4001 C0[0C, 20, 24, 28];
I2C1ADR[0, 1, 2, 3] - address 0x4005 C0[0C, 20, 24, 28];
I2C2ADR[0, 1, 2, 3] - address 0x400A 00[0C, 20, 24, 28]

9. I2C Mask registers

Four mask registers associated with each slave I2C.

Bit	Symbol	Description	Reset

			value
0	-	Reserved. User software should not write ones to reserved bits. This bit reads always back as 0.	0
7:1	MASK	Mask bits. Any bit which is set to ‘1’ will cause an automatic compare on the corresponding bit of the received address ie. bits in an I2ADRn register which are masked are not taken into account in determining an address match. The mask register has no effect on comparison to the General Call address (0000000). When an address-match interrupt occurs, the processor will have to read the data register (I2DAT) to determine which received address actually caused the match.	0x00
31:8	-	Reserved. User software should not write ones to reserved bits. These bits read always back as zeroes.	0

I2C Mask registers bit description
I2C0MASK[0, 1, 2, 3] - 0x4001 C0[30, 34, 38, 3C];
I2C1MASK[0, 1, 2, 3] - address 0x4005 C0[30, 34, 38, 3C];
I2C2MASK[0, 1, 2, 3] - address 0x400A 00[30, 34, 38, 3C]

10. I2C Monitor mode control register.

Used for monitoring the I2C bus without associating any control or participating in the handshake.

Bit	Symbol	Description	Reset value
0	MM_ENA	**Monitor mode enable.** 0 Monitor mode disabled. 1 In this mode the SDA output will be put in high impedance mode. This prevents the I2C module from outputting data of any kind (including ACK) onto the I2C data bus.	0
1	ENA_SCL	**SCL output enable**. 0 the SCL output will be forced high when the module is in monitor mode. This will prevent the module from having any control over the I2C clock line. 1 acting as a slave peripheral, the I2C module can “stretch” the clock line (hold it low) until it has had time to respond to an I2C interrupt.	0
2	MATCH_ALL	**Select interrupt register match**. 0 an interrupt will only be generated when a match occurs to one of the (up-to) four address registers, I2ADR0 through I2ADR3. That is, the module will respond as a normal slave as far as address-recognition is concerned.	0

		1 When this bit is set to ‘1’ and the I2C is in monitor mode, an interrupt will be generated on ANY address received. This will enable the part to monitor all traffic on the bus.	
31:3	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

I2C Monitor mode control register bit description (I2C0MMCTRL - 0x4001 C01C; I2C1MMCTRL- 0x4005 C01C; I2C2MMCTRL- 0x400A 001C)

11. I2C Control Set and Clear Registers.

Bit	Symbol	Description	Reset value
1:0	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	AA	Assert acknowledge flag. When set to 1 in master receive mode, an acknowledge (low level to SDA) will be returned when a data byte has been received. The AA bit can be cleared by writing 1 to the AAC bit in the I2CONCLR register. When AA is 0, in master receive mode a not acknowledge will be returned during the acknowledge clock pulse See handbook for other conditions.	0
3	SI	I2C interrupt flag. This bit is set when the I2C state changes. SI must be reset by software, by writing a 1 to the SIC bit in I2CONCLR register only after the required bit(s) has (have) been set and the value in I2DAT has been loaded or read.	0
4	STO	STOP flag. Setting this bit causes the I2C interface to transmit a STOP condition in master mode. When the bus detects the STOP condition, STO is cleared automatically.	0
5	STA	START flag. Causes the I2C interface to enter master mode and transmit a START condition or repeated START condition if already in master mode. If the bus is not free, it waits for a STOP condition (which will free the bus) then generates a START condition.	0
6	I2EN	I2C interface enable. When I2EN is 1, the I2C interface is enabled. Cleared by writing 1 to the I2ENC bit in the I2CONCLR register.	0
31:7	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

I2C Control Set bit descriptions. (I2C0CONSET - address 0x4001 C000, I2C1CONSET - address 0x4005

C000, I2C2CONSET - address 0x400A 0000)

Bit	Symbol	Description
1:0	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.
2	AAC	Assert acknowledge Clear bit. Writing 1 clears AA bit in the I2CONSET register.
3	SIC	I2C interrupt Clear bit. Writing 1 clears SI bit in the I2CONSET register.
4	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.
5	STAC	START flag Clear bit. Writing 1 clears STA bit in the I2CONSET register.
6	I2ENC	I2C interface Disable bit. Writing 1 clears the I2EN bit in the I2CONSET register.
31:7	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.

I2C Control Clear register bit description (I2C0CONCLR - 0x4001 C018; I2C1CONCLR - 0x4005 C018; I2C2CONCLR - 0x400A 0018)

Examples

For many I2C operations the firmware must wait until some action is complete. This will be defined by the I2C hardware setting the SI bit in I2CONSET. For convience a wait statement has been defined:

```
#define    WAIT_SI    while (!(LPC_I2C2->I2CONSET & (1<<3)))
```

After the next action is taken the SI bit should be cleared. For example once the START operation is complete the next operation will be to transmit the slave address. Once this is loaded into the data register at that point the SI bit should be cleared. This uses the statement

```
#define    CLEAR_SI    LPC_I2C2->I2CONCLR = 1<< 3
```

In addition to assist highlight the different activities in the mixed signal oscilloscope (MSO) display several I/O pins have been used. These are defined below. The statement ACTION will be executed just prior to an I2C START and ACTION_E once the I2C STOP is complete.

```
#define    ACTION     LPC_GPIO0->FIOSET0 = 1
#define    ACTION_E    LPC_GPIO0->FIOCLR0 = 1
#define    ADDRESS     LPC_GPIO0->FIOSET0 = 2
#define    ADDRESS_E    LPC_GPIO0->FIOCLR0 = 2
```

Example 1: Enable ARM I2C peripheral

```
void        I2C2_enable ( ) {
            LPC_I2C2->I2CONSET |= 1<< 6;            //enable I2C2
            }
```

Example 2: Set Start Bit — ie. Generate I2C Start

```
void        I2C_Start( ) {
            ACTION;                                 //for MSO display
            LPC_I2C2->I2CONSET |= 1<< 5;            //START I2C2
            WAIT_SI;                                //wait until done
            //need to load slave address
            //then clear start bit in I2CLR register
            }
```

Example 3: Set Stop Bit — ie. Generate I2C Stop

```
void        I2C2_Stop( ) {
            LPC_I2C2->I2CONSET |= 1<<4;             //STOP I2C
            CLEAR_SI;                               //clear SI
            while (LPC_I2C2->I2CONSET & (1<<4));    //wait until H/w stops I2c
            ACTION_E;                               //turn off MSO signa
            }
```

12. I2C Data Register

Bit	Symbol	Description	Reset value
7:0	Data	This register holds data values that have been received or are to be transmitted. The CPU can read and write to this register only while it is not in the process of shifting a byte, ie when the SI bit is set. Data in I2DAT remains stable as long as the SI bit is set. Data in I2DAT is always shifted msb first.	0
31:8	-	Reserved. User software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

I2C Data register bit description (I2C0DAT - 0x4001 C008; I2C1DAT - 0x4005 C008; I2C2DAT - 0x400A 0008)

Examples.

The following examples illustrate the data register and the interaction with the status register. To aid the display additional GPIO pins will be toggled defined by the following statements.

```
#define    WRITE      LPC_GPIO0->FIOSET0 = 1<<4
#define    WRITE_E    LPC_GPIO0->FIOCLR0 = 1<<4
#define    READ       LPC_GPIO0->FIOSET0 = 1<<5
#define    READ_E     LPC_GPIO0->FIOCLR0 = 1<<5
```

Example 1 Send Slave Address

Always follows a start condition

```
unsigned char  I2C2_Address(unsigned char add) {
               ADDRESS;                          //signal for MSO
               LPC_I2C2->I2DAT = add;            //the address
               LPC_I2C2->I2CONCLR = 1<<5;        //clear start
               LPC_I2C2->I2CONCLR = 1<< 3;       //clear SI
               WAIT_SI;                          //wait until change in status
               ADDRESS_E;
               return (LPC_I2C2->I2STAT);
               }
```

Example 2 Write data to slave

```
void           I2C2_Write(char dat) {
               WRITE;                            //for display only
               LPC_I2C2->I2DAT = dat ;           //new data
               CLEAR_SI;
               WAIT_SI;                          //wait until change in status
               WRITE_E;
               }
```

Example 3 Read data from slave

Set AA for more bytes - clear AA for last byte

```
unsigned char  I2C2_Read(char ack ) {
               READ;                             //for display only
               if (ack) LPC_I2C2->I2CONSET =1<<2; //assert AA -ACK more bytes to come
                   else LPC_I2C2->I2CONCLR = 1<<2; //No ack - last byte
               CLEAR_SI;
               WAIT_SI;                          //wait until change in status
               READ_E;
               return (LPC_I2C2->I2DAT);         //the data
               }
```

13. I2C Status register

The read-only I2C Status register reflects the condition of the corresponding I2C interface.

Bit	Symbol	Description	Reset value
2:0	-	These bits are unused and are always 0.	0
7:3	Status	The status information about the I2C interface. (26 possible codes) When status code is 0xF8, there is no relevant information available. (SI bit is not set). Other 25 status codes correspond to defined I2C states. When any of these states entered, the SI bit (I2C Interrupt) will be set.	0x1F
31:8	-	Reserved. The value read from a reserved bit is not defined.	NA

I2C Status register bit description

Sample Status Codes

See manual for complete listing.

Master Transmitter mode.

Status	Explanation	Next Action
0x08	A START condition has been transmitted.	Load (transmit) SLA+W; clear STA
0x10	A repeated START condition has been transmitted.	Load/(transmit) SLA+W As above.
		Load (transmit) SLA+R; Clear STA The I2C block then switched to MST/REC mode.
0x18	SLA+W has been transmitted; ACK has been received.	Load/transmit data byte; After transmission ACK bit will be received. Also Repeated START or STOP or STOP condition followed by a START condition will be transmitted; With STOP reset STO flag.
0x20	SLA+W has been transmitted; No ACK has been received.	Conclude transmission. ie send STOP.
0x28	Data byte in I2DAT has been transmitted; ACK has been received.	Load (transmit) further data bytes; ACK bit will be received after each. Also Repeated START or STOP or STOP condition followed by a START condition will be transmitted; With STOP reset STO flag.

Example

End I2C activity if no acknowledgement from slave address

```
st = I2C2_Address(0x4E);    //See data register examples
if (st == 0x18)             //ACK received to slave address
    { //OTHER ACTIONS};
if (st == 0x20) I2C2_Stop( ); //Slave address but NO ACK
```

Master Receiver mode

Status	Explanation	Next Action
0x08	A START condition has been transmitted.	Load (transmit) SLA+R; clear STA
0x10	A repeated START condition has been transmitted.	Load/(transmit) SLA+R As above.
		Load (transmit) SLA+W; The I2C block then switched to MST/TRX mode.
0x40	SLA+R has been transmitted; ACK has been received.	Data byte will be received; No more bytes expected NOT ACK bit will be returned. Additional bytes expected ACK bit will be returned.
0x50	Data byte has been received; ACK has been returned.	Read data byte; No more bytes expected NOT ACK bit will be returned. Additional bytes expected ACK bit will be returned.

14. Example 1: Transmitting when a slave is not present.

If a slave is not available for testing just transmitting the start, an abittary slave address and a stop signal will allow some understanding of the I2C signals. The full initialisation code is given below. By removing the highlighted code the internal pull up resistors were activated eliminating the need for external pull ups. (NXP recomend that external pull up resistors be used.) The first line enables a number of pins as basic output devices. The outputs on these pins will only be used to assist displaying the I2C waveforms -they have no relevance to the I2C operation.

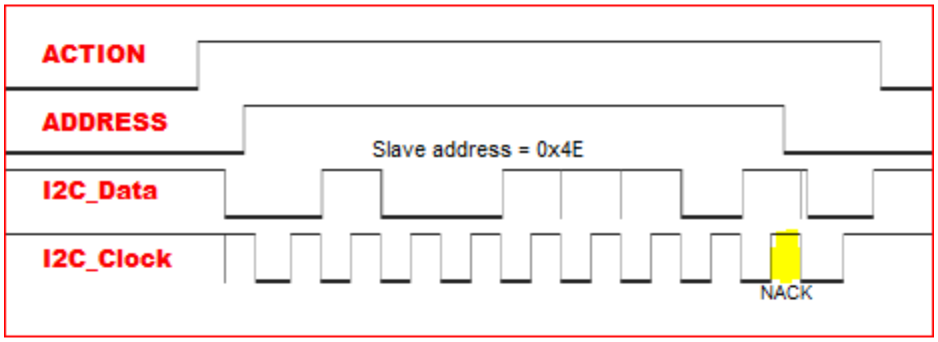
```
void SystemInit() {
        LPC_GPIO0->FIODIR0 |= 0x33;                 //P0.5 P0.4 P0.1 P0.0 ext pins 29,30,10,9
        LPC_SC->PCONP |= 1 << 26;                   //re-enable POWER to I2C_2 if required
        LPC_SC->PCLKSEL1 |= 1<<20;                  //pclk = cclk
        LPC_PINCON->PINSEL0 |= 0x02<<20;            //Pin P0.10 allocated to alternate function 2
        LPC_PINCON->PINSEL0 |= 0x02<<22;            //Pin P0.11 allocated to alternate function 2
        LPC_GPIO0->FIODIR |= 1<<10;                 //Bit P0.10 an output
        LPC_GPIO0->FIODIR |= 1<<11;                 //Bit P0.11 an output
        LPC_PINCON->PINMODE0 &= ~(3<<20);           //P0.10 has pull up/down resistor
        LPC_PINCON->PINMODE0 |= (2<<20);            //omit to use internal pull up
        LPC_PINCON->PINMODE0 &= ~(3<<22);           //P0.11 has pull up/down resistor
        LPC_PINCON->PINMODE0 |= (2<<22);            //omit to use internal pull up
        LPC_PINCON->PINMODE_OD0 |= 1<<10;           //Bit P0.10 is open drain
        LPC_PINCON->PINMODE_OD0 |= 1<<11;           //Bit P0.11 is open drain
        LPC_I2C2->I2SCLH = 60;                      //100kHz from 12MHz
        LPC_I2C2->I2SCLL = 60;                      //100kHz from 12MHz
}
```

The main code is given below. The code executes the I2C syntax

1. Start
2. Transmit slave address
3. If slave does not respond complete operation with a stop

```
int main ( ) {
unsigned    char st;
            I2C2_enable( );              //Refer to I2CONSET register
            I2C2_Start();                //Refer to I2CONSET register
            st = I2C2_Address(0x4E);     //Refer to I2DAT register
            if (st == 0x18)              //Refer to I2STATUS register
                 { //OTHER ACTIONS };
            if (st == 0x20) I2C2_Stop( ); //Slave address but NO ACK
            while (1);                   //Loop forever
            }
```

The resulting traces are shown below:



Transmitting when slave not present.

An explanation of the signals is as follows

1. The top signal is activated in the start routine and de-activated at the end of the stop routine. It illustrates the duration of the I2C activity.

2. The second signal is activated at the start of the address routine and de-activated at the conclusion. It highlight the duration of the address routine. In this example the ARM I2C peripheral starts in state 0x08 and conclude in state 0x20.

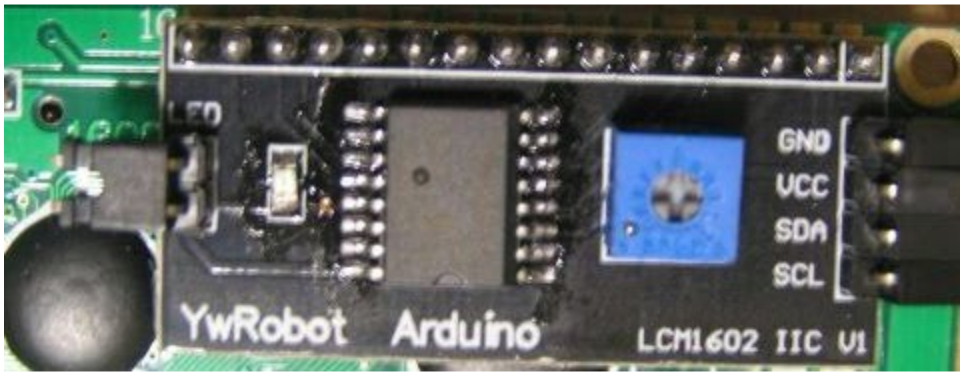
3. The data line. Initially the data rests high. The data line then
   1. Moves low while the clock is still high to indicate a start bit.
   2. Following the start signal the I2C bus expects the 7 bit slave address. In this example it is 0x27.
   3. The 8th bit will be the read write signal. In this example it is low (giving an 8 bit address of 0x4E)

indicating the I2C master expects to write further data to the slave.

4. On bit 9 it is expected that the addressed slave will assert the bus to acknowledge receipt of the address. Since there is no slave the data bus will be pulled high by the pull up resistor.
5. With no acknowledgement (ie the peripheral is in state 0x20) a stop signal is generated. That is the data goes high while the clock is high.

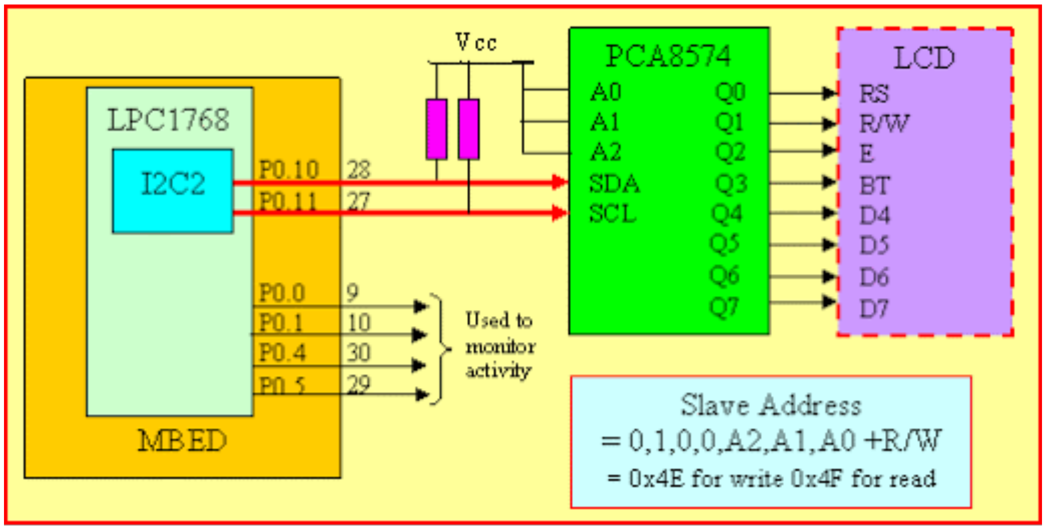
4. The 9 clock pulses for each byte. These clock pulses are automatically generated by the write to the data register.

15. Example 2: I2C interface to a I2C to parallel port device.



The I2C to parallel interface. Used to drive an LCD.

This example is concerned with interfacing the LPC1768 micro-controller to an I2C to parallel port interface as illustrated in the following circuit. For convenience this circuit is repeated from the introduction.

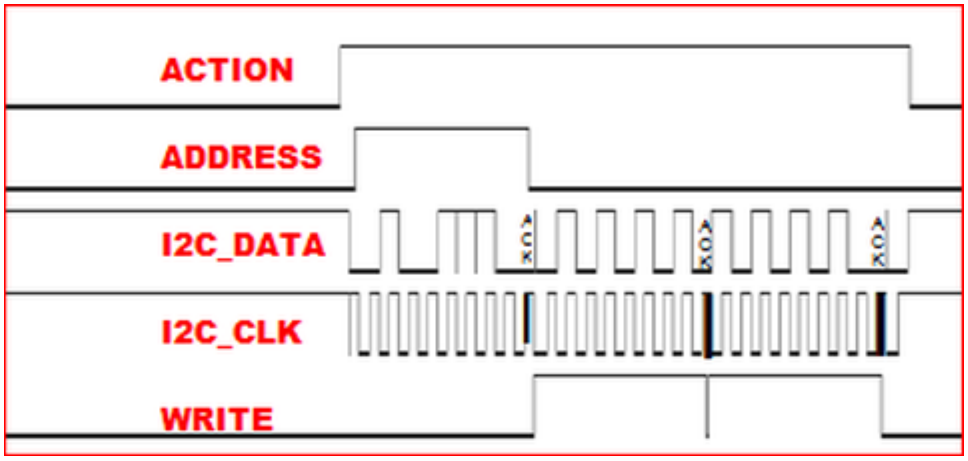


ARM LPC1768 I2C interface to an I2C to parallel port.

In the previous example the I2C_Address() routine returned 0x20 indicating there was no acknowledgement of the a slave address. Including a slave device with a valid address generates an acknowledgement and the ARM I2C peripheral returns the status 0x18. In this example the I2C is programmed to send two bytes of data and conclude with a stop sequence.

```
int main ( ) {
unsigned     char st;
             I2C2_enable( );              //Refer to I2CONSET register
             I2C2_Start();                //Refer to I2CONSET register
             st = I2C2_Address(0x4E);     //Refer to I2DAT register
             if (st == 0x18)              //Refer to I2STATUS register
                 { I2C2_Write(0x55);      //Refer to I2DAT register
                   I2C2_Write(0xAA);
                   I2C2_Stop( ); };
             if (st == 0x20) I2C2_Stop( ); //Slave address but NO ACK
             while (1);                   //Loop forever
             }
```

The resulting waveforms are shown below



I2C Write 2 bytes waveforms.

A brief explanation is as follows:

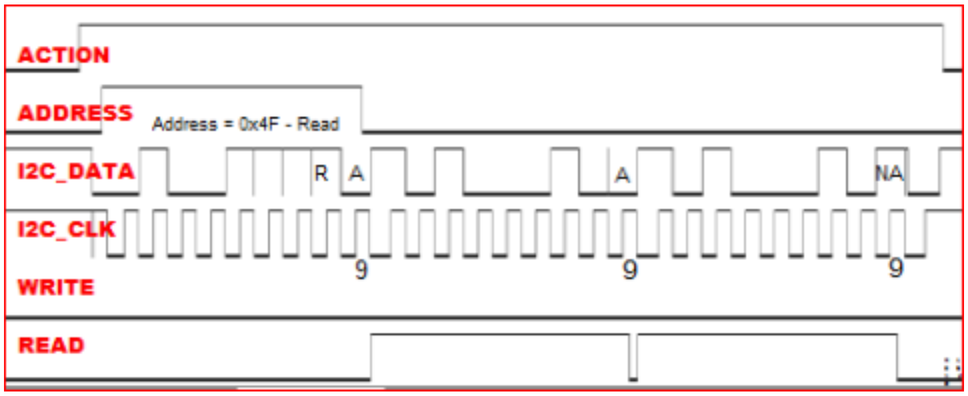
1. The slave address 0x4E is sent as in the previous example but since the slave is present it acknowledges its address on the 9th clock pulse (SDA low).
2. Since there has been an acknowledge the ARM I2C peripheral is in the state 0x18. For the state 0x18 the ARM I2C peripheral writes one or more bytes to the slave. The slave will treat this information as data. In the case of the PCA8574 the data will appear on its output port.
3. After each byte of data on the 9th clock pulse the slave will acknowledge the receipt of the data.
4. The master sends as much data as it wishes (in this example 2 bytes) and terminates the transfer with a STOP sequence.

16. Example 3: Reading data from the slave.

The PAC8574 can also be used to demonstrate the I2C read operations. The main code is given below. Note for a read the slave address will be 0x4E + 1 = 0x4F:

```
int main ( ) {
      unsigned char st;
                      I2C2_enable( );              //Refer to I2CONSET register
                      I2C2_Start();                //Refer to I2CONSET register
                      st = I2C2_Address(0x4F); //Refer to I2DAT register
                      if (st == 0x40)              //Refer to I2STATUS register
                            { st=I2C2_Read(1);     //Refer to I2DAT register
                              st= I2C2_Read(0);
                               I2C2_Stop( ); };
                      while (1);                   //Loop forever
                      }
```

The resultatnt waveforms are shown below:



Reading two bytes from the slave.

Notes on waveforms

1. The read address is 0x4F with bit 8 set for a read.
2. The slave will acknowledge receipt of its address.
3. The ARM I2C peripheral generates 8+1 clock pulses
4. On each of the 8 clock pulse the slave places data onto the data bus
5. On the 9th clock pulse the ARM I2C peripheral asserts the data line to acknowledge receipt of the data.
6. On the final byte the ARM I2C peripheral does not acknowledged so the slave knows no more data is expected.
7. The transfer is completed with a stop signal.

Developed by John Kneen RMIT University Monday, December 30, 2013