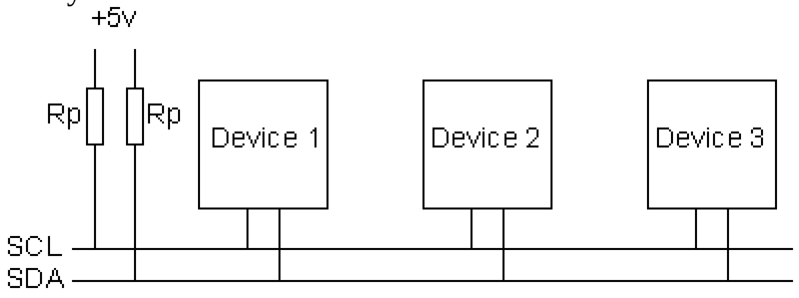
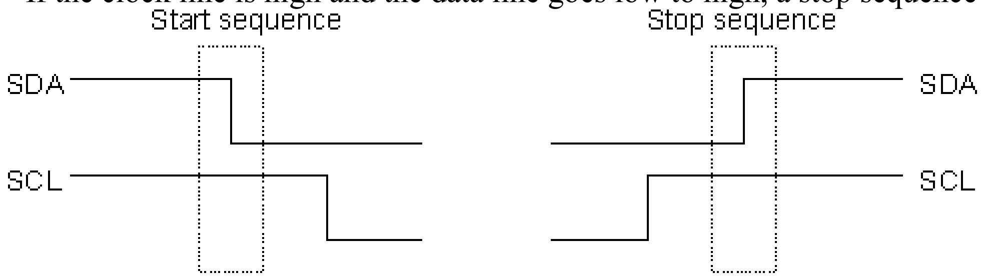
I^2C Bus

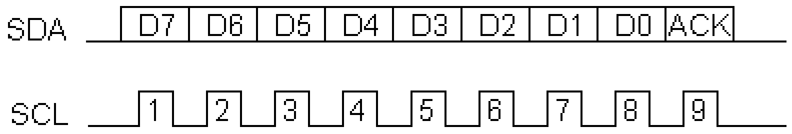
- The I^2C bus is a two-wire bus(plus ground) where the two wire are called
  SCL – Clock line
  SDA – Data line
  Gnd – Ground line
- This is a synchronous bus. SCL is the synchronizing signal.
- SCL and SDA are open drain – they can be driven low but they must have a passive pull up resistor.
- Multiple modules make use of the same lines. Only one set of pull up resistors is needed per system.



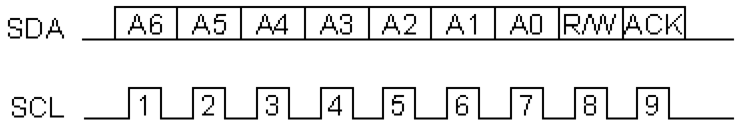
- Typically a system has one master module and multiple slaves. The master module is the one that drives the clock line.
- If the clock line is high and the data line goes high to low, a start sequence is initiated.
- If the clock line is high and the data line goes low to high, a stop sequence is initiated.



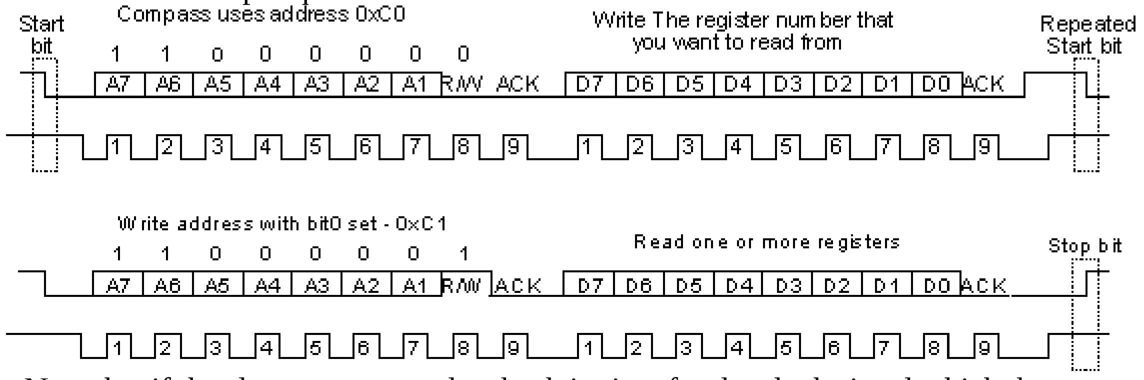
- Data is transferred in sequences of 8 bits. The bits are placed on the SDA line starting with the MSB (Most Significant Bit). The SCL line is then pulsed high, then low.
- For every 8 bits transferred, the device receiving the data sends back an acknowledge bit, so there are actually 9 SCL clock pulses to transfer each 8 bit byte of data.
- If the receiving device sends back a low ACK bit, then it has received the data and is ready to accept another byte. If it sends back a high then it is indicating it cannot accept any further data and the master should terminate the transfer by sending a stop sequence.



- The standard bit rate for the clock is 100 KHz. Philips defines faster rates up to a Mbit but most I^2C systems run at 100 KHz.
- We will use only a 7-bit address. Philips also defines a 10-bit address. The 7-bit address gives us up to 128 devices.



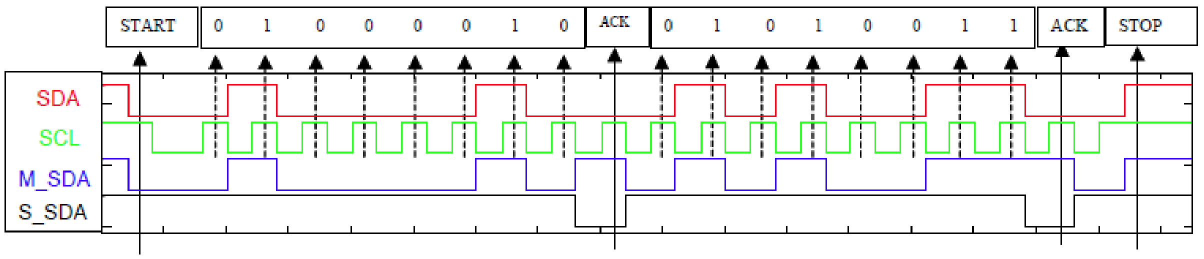
- The 8th bit tells whether we are reading or writing to the address. One means reading from the slave and zero means writing to the slave. Note that the 7-bit address is in the upper 7 bits.
- To write data to a slave device:
  1. Send a start sequence
  2. Send the I2C address of the slave with the R/W bit low (even address)
  3. Send the internal register number you want to write to. So if a slave module has 16 internal register you could send out the number 2 to write to register 2.
  4. Send the data byte
  5. [Optionally, send any further data bytes]
  6. Send the stop sequence.
- To read data from a slave device:
  1. Send a start sequence
  2. Send the I2C address of the slave with the R/W bit low (even address)
  3. Send the internal address of the register you want to read.
  4. Send a start sequence again (repeated start)
  2. Send the I2C address of the slave with the R/W bit high (odd address)
  6. Read data byte.
  7. Send the stop sequence.


Start bit
Compass uses address 0xC0
1 1 0 0 0 0 0 0
A7 A6 A5 A4 A3 A2 A1 R/W ACK
Write The register number that you want to read from
D7 D6 D5 D4 D3 D2 D1 D0 ACK
Repeated Start bit
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
Write address with bit0 set - 0xC1
1 1 0 0 0 0 0 1
A7 A6 A5 A4 A3 A2 A1 R/W ACK
Read one or more registers
D7 D6 D5 D4 D3 D2 D1 D0 ACK
Stop bit
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9

- Note that if the slave cannot get data back in time for the clock signal which the master controls, it can hold the clock line low until the data is ready. The master will recognize this and wait for the clock line to go high to get the data.

I^{2}C on the HMC6352 Compass

- The 6352 acts as a slave I^{2}C port at 100 KHz
- The write address is 42H and the read address is 43H
- The 6532 will operate from 2.7 to 5.2 volts on Vcc. The ARM is a 3.3 volt system so we will use this value. We need two 10K pull up resistors on the SCL and SDA I^{2}C bus lines
- The following example from the 6532 data sheet shows the compass receiving a "Sleep" command – ascii S



42(hex)
Write to This I²C Address

"S"
Command

- Commands to the 6532 are 1 to 3 bytes lone. The first byte is the command byte (an ascii character) and the next two bytes are a binary argument with the most significant byte first.
- The command are listed below.

Table 1 – HMC6352 Interface Commands/Responses

Command Byte ASCII (hex)	Argument 1 Byte (Binary)	Argument 2 Byte (Binary)	Response 1 Byte (Binary)	Response 2 Byte (Binary)	Description
w (77)	EEPROM Address	Data			Write to EEPROM
r (72)	EEPROM Address		Data		Read from EEPROM
G (47)	RAM Address	Data			Write to RAM Register
g (67)	RAM Address		Data		Read from RAM Register
S (53)					Enter Sleep Mode (Sleep)
W (57)					Exit Sleep Mode (Wakeup)
O (4F)					Update Bridge Offsets (S/R Now)
C (43)					Enter User Calibration Mode
E (45)					Exit User Calibration Mode
L (4C)					Save Op Mode to EEPROM
A (41)			MSB Data	LSB Data	Get Data. Compensate and Calculate New Heading

- The 6532 has internal EEPROM which determines its operational parameters. These are:

Table 2 – HMC6352 EEPROM Contents

EE Address (hex)	Byte Description	Factory Default
00	I²C Slave Address	42(hex)
01	Magnetometer X Offset MSB	factory test value
02	Magnetometer X Offset LSB	factory test value
03	Magnetometer Y Offset MSB	factory test value
04	Magnetometer Y Offset LSB	factory test value
05	Time Delay (0 – 255 ms)	01(hex)
06	Number of Summed measurements(1-16)	04(hex)
07	Software Version Number	> 01(hex)
08	Operation Mode Byte	50(hex)

The EEPROM is shadowed in RAM. On start up the EEPROM values are copied into RAM. If you write to EEPROM your data is also copied to RAM.

- There are three operational modes determined by the Operation Mode Byte at 08H in EEPROM shadowed at 74H in RAM. The three modes are *Standby*, *Query*, and *Continuous*. Standby is the factory default and we will use only this mode in class. Here is the Operation Mode byte:

Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
0	M. Rate_H	M. Rate_L	Per. S/R	0	0	Op Mode_H	Op Mode_L

Bits 1 and 0 are 00 – Standby, 01 – Query, 10 – Continuous. 11 is illegal. The Per.S/R is the periodic set/reset bit which does a recalibration every few minutes. This is probably not needed. Bits 5 and 6 determine the update rate in continuous mode.

• In Standby mode the 6532 waits for an 'A' command (get data). When it gets this command it does a reading of the sensor. At the next 'A' command it transmits the data. This data is always two-bytes of binary and may indicate either a *heading* or *magnetometer* data determined by the output data control byte in RAM at location 4EH.

Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
0	0	0	0	0	Mode	Mode	Mode

Bit 2	Bit 1	Bit 0	Description
0	0	0	Heading Mode
0	0	1	Raw Magnetometer X Mode
0	1	0	Raw Magnetometer Y Mode
0	1	1	Magnetometer X Mode
1	0	0	Magnetometer Y Mode

Note that the heading mode comes up by default on power up and this is what we will use in class. The two bytes in heading mode are in tenths of a degree from 0 to 3599.

• Writing to EEPROM is slow. Here are the delays required by each command.

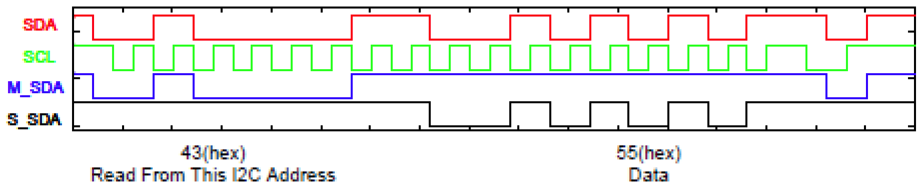
Table 3 – Interface Command Delays

Command Byte ASCII (hex)	Description	Time Delay (μsec)
w (77)	Write to EEPROM	70
r (72)	Read from EEPROM	70
G (47)	Write to RAM Register	70
g (67)	Read from RAM Register	70
S (53)	Enter Sleep Mode (Sleep)	10
W (57)	Exit Sleep Mode (Wakeup)	100
O (4F)	Update Bridge Offsets (S/R Now)	6000
C (43)	Enter User Calibration Mode	10
E (45)	Exit User Calibration Mode	14000
L (4C)	Save Op Mode to EEPROM	125
A (41)	Get Data. Compensate and Calculate New Heading	6000

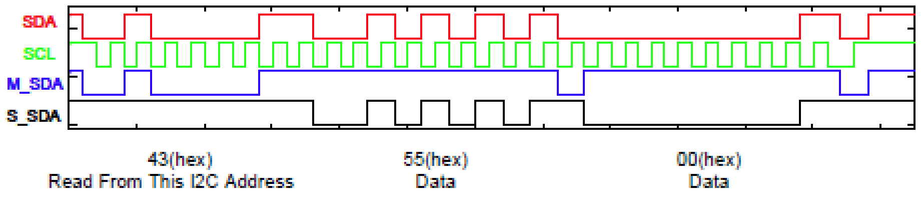
• The following page gives examples of waveforms for reading and writing to the 6532.

Waveform Examples

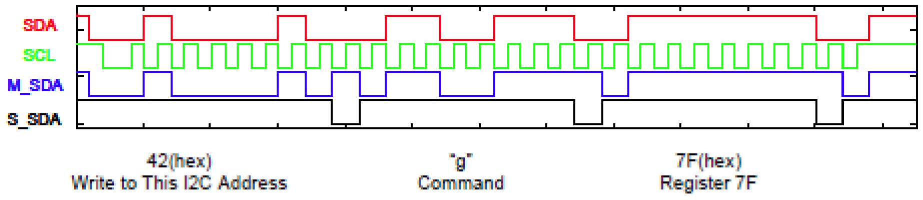
Example 1: This example shows how to read a single byte from the HMC6352. The Slave (HMC6352) continues to hold the SDA line low after the acknowledge (ACK) bit because the first bit of the data byte is a zero. Remember that the data read is last command sensitive.



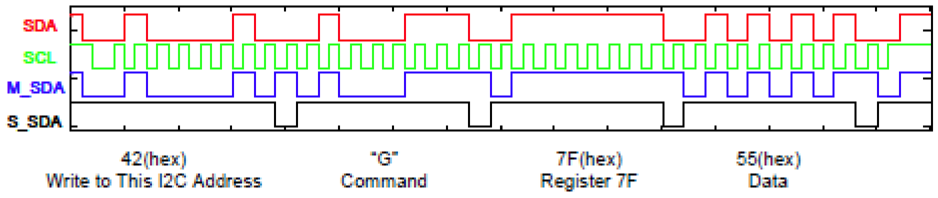
Example 2: This example shows how to read two bytes from the HMC6352 (slave). The slave continues to hold the SDA line low after the acknowledge bit because the first bit of the data bytes is zero.



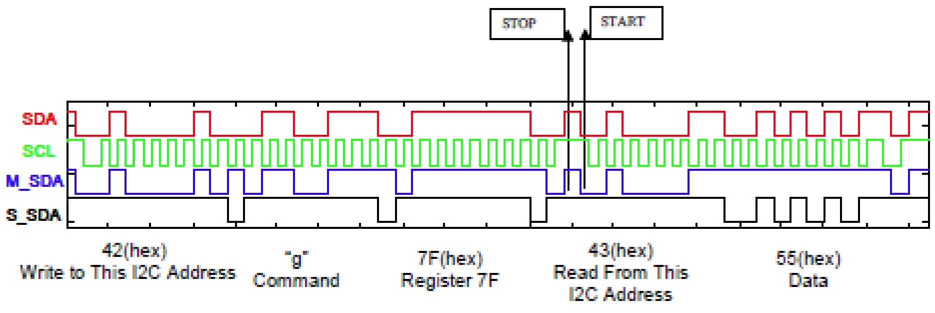
Example 3: This example shows how to command HMC6352 to read a RAM register by sending the "g" command and the register address 7F(hex). Note that this example does not show the process of reading the answer. See example 1 for reading a byte.



Example 4: This example shows how to write to a RAM register in the HMC6352 by sending the "G" command, the register address 7F(hex), and the data byte 55(hex) to the HMC6352 slave.



Example 5: The final example shows how to read RAM register 7F(hex). First perform a write operation to command the HMC6352 to read a RAM register and define which register to read (Example 3). The sensor puts the answer in the data buffer. Then perform a read operation to clock out the answer (Example 1). There is a Stop/Start event in between the write operation and the read operation. This example is just a combination of Examples 3 and 1, but it is provided to show that reading a register involves both a write and a read operation.



I^{2}C on the ARM 2138

1. The ARM 2138 has two I^{2}C ports
   SCL0 – p22 and SDA0 – p26
   SCL1 – p37 and SDA1 – p41
   We will use I^{2}C0. From the Pin Function assignment table we set
```
pinsel0 = 0x50;
```

2. To set up the interrupt we need to do the following three items:
   A) From the interrupt sources table determine the interrupt channel number for I^{2}C. This is channel 9. For the interrupt control register we need to make bit 5 a 1 and set the first four bits to the channel number.
```
VICVectCntl1 = 0x00000029;
```
   B) We need to set the address of the interrupt service routine into the interrupts address register. If we call this routine I2CISR we can do this by the following:
```
VICVectAddr1 = (unsigned)I2CISR;
```
   C) We need to enable the appropriate interrupt. We are using channel 9 so we need a 1 in bit 9 of the interrupt enable register.
```
VICIntEnable = 0x00000200;
```

3. Next we set up the bit rate for the I^{2}C bus. This is determined as
   I^{2}C clock rate PCLK/(SCLH+SCLL) = 100 KHz
   For our case CCLK is 4 x 14.7456 = 58.9824 MHz and if we set VPBDIV = 2 we get PCLK = 29.4912 MHz.
   29.4912MHz/(SCLH + SCLL) = 100 KHz
   SCLH + SCLL = 294.12. SCLH is the number of cycles of high time in the clock and SCLL is the number of cycles of low time in the clock. We can make the high time 150 and the low time at 294 – 150 = 144.
```
I2SCLH = 150;
ISSCLL = 144;
```

4. At this point we are ready to enable the I^{2}C transmission and send a command. To do this we will use the default standby mode and send an 'A' command. This causes the compass to read the sensors. A second 'A' command will cause the compass to transmit the data back as two bytes. There are seven registers that control the I^{2}C bus. The five that we need to write are
I2CONSET – I^{2}C Control Set

	7	6	5	4	3	2	1	0
I2CONSET	-	I2EN	STA	STO	SI	AA	-	-

   I2EN is the enable bit, STA is the start bit, STO is the stop bit, SI is the interrupt flag, and AA is the assert acknowledge bit.
I2CONCLR – I^{2}C Control Clear. Write all ones to this register to clear the system
I2DAT – I^{2}C Data Register.   8-bits of data
I2ADR – I^{2}C Slave Address Resister.   Bits 1 to 7 have a 7 bit address. Bit 0 is zero to write and 1 to read.
I2STAT – The most significant 5 bits in this register are the status.

Pin Function Select Register 0 (PINSEL0 - 0xE002C000)

The PINSEL0 register controls the functions of the pins as per the settings listed in Table 49. The direction control bit in the IO0DIR register is effective only when the GPIO function is selected for a pin. For other functions, direction is controlled automatically.

Table 46: Pin Function Select Register 0 for LPC2132/2138 (PINSEL0 - 0xE002C000)

PINSEL0	Pin Name	Function when 00	Function when 01	Function when 10	Function when 11	Reset Value
1:0	P0.0	GPIO Port 0.0	TxD (UART0)	PWM1	Reserved	00
3:2	P0.1	GPIO Port 0.1	RxD (UART0)	PWM3	EINT0	00
5:4	P0.2	GPIO Port 0.2	SCL0 (I^2C)	Capture 0.0 (TIMER0)	Reserved	00
7:6	P0.3	GPIO Port 0.3	SDA0 (I^2C)	Match 0.0 (TIMER0)	EINT1	00
9:8	P0.4	GPIO Port 0.4	SCK (SPI0)	Capture 0.1 (TIMER0)	AD0.6	00
11:10	P0.5	GPIO Port 0.5	MISO (SPI0)	Match 0.1 (TIMER0)	AD0.7	00
13:12	P0.6	GPIO Port 0.6	MOSI (SPI0)	Capture 0.2 (TIMER0)	AD1.0 (LPC2138)	00
15:14	P0.7	GPIO Port 0.7	SSEL (SPI0)	PWM2	EINT2	00
17:16	P0.8	GPIO Port 0.8	TxD UART1	PWM4	AD1.1 (LPC2138)	00
19:18	P0.9	GPIO Port 0.9	RxD (UART1)	PWM6	EINT3	00
21:20	P0.10	GPIO Port 0.10	RTS (UART1)	Capture 1.0 (TIMER1)	AD1.2 (LPC2138)	00
23:22	P0.11	GPIO Port 0.11	CTS (UART1)	Capture 1.1 (TIMER1)	SCL1 (I^2C1)	00
25:24	P0.12	GPIO Port 0.12	DSR (UART1)	Match 1.0 (TIMER1)	AD1.3 (LPC2138)	00
27:26	P0.13	GPIO Port 0.13	DTR (UART1)	Match 1.1 (TIMER1)	AD1.4 (LPC2138)	00
29:28	P0.14	GPIO Port 0.14	CD (UART1)	EINT1	SDA1 (I^2C1)	00
31:30	P0.15	GPIO Port 0.15	RI (UART1)	EINT2	AD1.5 (LPC2138)	00

Vector Control Registers 0-15 (VICVectCntl0-15 - 0xFFFFF200-23C, Read/Write)

Each of these registers controls one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest. Note that disabling a vectored IRQ slot in one of the VICVectCntl registers does not disable the interrupt itself, the interrupt is simply changed to the non-vectored form.

Table 38: Vector Control Registers (VICVectCntl0-15 - 0xFFFFF200-23C, Read/Write)

VICVectCntl0-15	Function	Reset Value
5	1: this vectored IRQ slot is enabled, and can produce a unique ISR address when its assigned interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0
4:0	The number of the interrupt request or software interrupt assigned to this vectored IRQ slot. As a matter of good programming practice, software should not assign the same interrupt number to more than one enabled vectored IRQ slot. But if this does occur, the lower-numbered slot will be used when the interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0

Vector Address Registers 0-15 (VICVectAddr0-15 - 0xFFFFF100-13C, Read/Write)

These registers hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.

Table 39: Vector Address Registers (VICVectAddr0-15 - 0xFFFFF100-13C, Read/Write)

VICVectAddr0-15	Function	Reset Value
31:0	When one or more interrupt request or software interrupt is (are) enabled, classified as IRQ, asserted, and assigned to an enabled vectored IRQ slot, the value from this register for the highest-priority such slot will be provided when the IRQ service routine reads the Vector Address register (VICVectAddr).	0

Interrupt Enable Register (VICIntEnable - 0xFFFFF010, Read/Write)

This register controls which of the 32 interrupt requests and software interrupts contribute to FIQ or IRQ.

Table 33: Interrupt Enable Register (VICINtEnable - 0xFFFFF010, Read/Write)

VICIntEnable	Function	Reset Value
31:0	When this register is read, 1s indicate interrupt requests or software interrupts that are enabled to contribute to FIQ or IRQ. When this register is written, ones enable interrupt requests or software interrupts to contribute to FIQ or IRQ, zeroes have no effect. See the VICIntEnClear register (Table 34 below), for how to disable interrupts.	0

Vector Address Registers 0-15 (VICVectAddr0-15 - 0xFFFFF100-13C, Read/Write)

These registers hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.

Table 39: Vector Address Registers (VICVectAddr0-15 - 0xFFFFF100-13C, Read/Write)

VICVectAddr0-15	Function	Reset Value
31:0	When one or more interrupt request or software interrupt is (are) enabled, classified as IRQ, asserted, and assigned to an enabled vectored IRQ slot, the value from this register for the highest-priority such slot will be provided when the IRQ service routine reads the Vector Address register (VICVectAddr).	0

INTERRUPT SOURCES

Table 43 lists the interrupt sources for each peripheral function. Each peripheral device has one interrupt line co
Vectored Interrupt Controller, but may have several internal interrupt flags. Individual interrupt flags may also rep
than one interrupt source.

Table 43: Connection of Interrupt Sources to the Vectored Interrupt Controller

Block	Flag(s)	VIC Channel #
WDT	Watchdog Interrupt (WDINT)	0
-	Reserved for software interrupts only	1
ARM Core	Embedded ICE, DbgCommRx	2
ARM Core	Embedded ICE, DbgCommTx	3
TIMER0	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	4
TIMER1	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	5
UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)	6
UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI)	7
PWM0	Match 0 - 6 (MR0, MR1, MR2, MR3, MR4, MR5, MR6)	8
I^{2}C0	SI (state change)	9
SPI0	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	10
SPI1 (SSP)	Tx FIFO at least half empty (TXRIS) Rx FIFO at least half full (RXRIS) Receive Timeout condition (RTRIS) Receive overrun (RORRIS)	11
PLL	PLL Lock (PLOCK)	12
RTC	Counter Increment (RTCCIF) Alarm (RTCALF)	13
System Control	External Interrupt 0 (EINT0)	14
	External Interrupt 1 (EINT1)	15
	External Interrupt 2 (EINT2)	16
	External Interrupt 3 (EINT3)	17
A/D0	A/D Converter 0	18
I^{2}C1	SI (state change)	19

Table 3. Master Transmitter Mode

STATUS CODE (S1STA)	STATUS OF THE I²C BUS AND SIO1 HARDWARE	APPLICATION SOFTWARE RESPONSE					NEXT ACTION TAKEN BY SIO1 HARDWARE
		TO/FROM S1DAT	TO S1CON				
			STA	STO	SI	AA	
08H	A START condition has been transmitted	Load SLA+W	X	0	0	X	SLA+W will be transmitted; ACK bit will be received
10H	A repeated START condition has been transmitted	Load SLA+W or	X	0	0	X	As above
		Load SLA+R	X	0	0	X	SLA+W will be transmitted; SIO1 will be switched to MST/REC mode
18H	SLA+W has been transmitted; ACK has been received	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received
		no S1DAT action or	1	0	0	X	Repeated START will be transmitted;
		no S1DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset
		no S1DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset
20H	SLA+W has been transmitted; NOT ACK has been received	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received
		no S1DAT action or	1	0	0	X	Repeated START will be transmitted;
		no S1DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset
		no S1DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset
28H	Data byte in S1DAT has been transmitted; ACK has been received	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received
		no S1DAT action or	1	0	0	X	Repeated START will be transmitted;
		no S1DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset
		no S1DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset
30H	Data byte in S1DAT has been transmitted; NOT ACK has been received	Load data byte or	0	0	0	X	Data byte will be transmitted; ACK bit will be received
		no S1DAT action or	1	0	0	X	Repeated START will be transmitted;
		no S1DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset
		no S1DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset
38H	Arbitration lost in SLA+R/$\overline{W}$ or Data bytes	No S1DAT action or	0	0	0	X	I²C bus will be released; not addressed slave will be entered
		No S1DAT action	1	0	0	X	A START condition will be transmitted when the bus becomes free

Table 4. Master Receiver Mode

STATUS CODE (S1STA)	STATUS OF THE I²C BUS AND SIO1 HARDWARE	APPLICATION SOFTWARE RESPONSE					NEXT ACTION TAKEN BY SIO1 HARDWARE
		TO/FROM S1DAT	TO S1CON				
			STA	STO	SI	AA	
08H	A START condition has been transmitted	Load SLA+R	X	0	0	X	SLA+R will be transmitted; ACK bit will be received
10H	A repeated START condition has been transmitted	Load SLA+R or	X	0	0	X	As above
		Load SLA+W	X	0	0	X	SLA+W will be transmitted; SIO1 will be switched to MST/TRX mode
38H	Arbitration lost in NOT ACK bit	No S1DAT action or	0	0	0	X	I²C bus will be released; SIO1 will enter a slave mode
		No S1DAT action	1	0	0	X	A START condition will be transmitted when the bus becomes free
40H	SLA+R has been transmitted; ACK has been received	No S1DAT action or	0	0	0	0	Data byte will be received; NOT ACK bit will be returned
		no S1DAT action	0	0	0	1	Data byte will be received; ACK bit will be returned
48H	SLA+R has been transmitted; NOT ACK has been received	No S1DAT action or	1	0	0	X	Repeated START condition will be transmitted
		no S1DAT action or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset
		no S1DAT action	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset
50H	Data byte has been received; ACK has been returned	Read data byte or	0	0	0	0	Data byte will be received; NOT ACK bit will be returned
		read data byte	0	0	0	1	Data byte will be received; ACK bit will be returned
58H	Data byte has been received; NOT ACK has been returned	Read data byte or	1	0	0	X	Repeated START condition will be transmitted
		read data byte or	0	1	0	X	STOP condition will be transmitted; STO flag will be reset
		read data byte	1	1	0	X	STOP condition followed by a START condition will be transmitted; STO flag will be reset

The I^{2}C interface contains 7 registers as shown in Table 89. below.

Table 89: I^{2}C Register Map

Name	Description	Access	Reset Value*	I2C0 Address & Name	I2C1 Address & Name
I2CONSET	I^{2}C Control Set Register	Read/Set	0	0xE001C000 I2C0CONSET	0xE005C000 I2C1CONSET
I2STAT	I^{2}C Status Register	Read Only	0xF8	0xE001C004 I2C0STAT	0xE005C004 I2C1STAT
I2DAT	I^{2}C Data Register	Read/Write	0	0xE001C008 I2C0DAT	0xE005C008 I2C1DAT
I2ADR	I^{2}C Slave Address Register	Read/Write	0	0xE001C00C I2C0ADR	0xE005C00C I2C1ADR
I2SCLH	SCL Duty Cycle Register High Half Word	Read/Write	0x04	0xE001C010 I2C0SCLH	0xE005C010 I2C1SCLH
I2SCLL	SCL Duty Cycle Register Low Half Word	Read/Write	0x04	0xE001C014 I2C0SCLL	0xE005C014 I2C1SCLL
I2CONCLR	I^{2}C Control Clear Register	Clear Only	NA	0xE001C018 I2C0CONCLR	0xE005C018 I2C1CONCLR

Table 90: I^{2}C Control Set Register (I2C0CONSET - 0xE001C000, I2C1CONSET - 0xE005C000)

I2CONSET	Function	Description	Reset Value
0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
1	Reserved	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	0
3	SI	I^{2}C interrupt flag	0
4	STO	STOP flag	0
5	STA	START flag	0
6	I2EN	I^{2}C interface enable	0
7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

I^{2}C Control Clear Register (I2C0CONCLR - 0xE001C018, I2C1CONCLR - 0xE005C018)

Table 91: I^{2}C Control Clear Register (I2C0CONCLR - 0xE001C018, I2C1CONCLR - 0xE005C018)

I2CONCLR	Function	Description	Reset Value
0	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
1	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
2	AAC	Assert Acknowledge Clear bit. Writing a 1 to this bit clears the AA bit in the I2CONSET register. Writing 0 has no effect.	NA
3	SIC	I^{2}C Interrupt Clear Bit. Writing a 1 to this bit clears the SI bit in the I2CONSET register. Writing 0 has no effect.	NA
4	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
5	STAC	Start flag clear bit. Writing a 1 to this bit clears the STA bit in the I2CONSET register. Writing 0 has no effect.	NA
6	I2ENC	I^{2}C interface disable. Writing a 1 to this bit clears the I2EN bit in the I2CONSET register.Writing 0 has no effect.	NA
7	Reserved	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

I^{2}C Status Register (I2C0STAT - 0xE001C004, I2C1STAT - 0xE005C004)

This is a read-only register. It contains the status code of the I^{2}C interface. The least three bits are always 0. There are 26 possible status codes. When the code is F8H, there is no relevant information available and the SI bit is not set. All other 25 status codes correspond to defined I^{2}C states. When any of these states entered, SI bit will be set. Refer to Table 3 to Table 6 in "80C51 Family Derivatives 8XC552/562 Overview" datasheet available on-line at

http://www.semiconductors.philips.com/acrobat/various/8XC552_562OVERVIEW_2.pdf

for a complete list of status codes.

Table 92: I^{2}C Status Register (I2C0STAT - 0xE001C004, I2C1STAT - 0xE005C004)

I2STAT	Function	Description	Reset Value
2:0	Status	These bits are always 0	0
7:3	Status	Status bits	1

I^{2}C Data Register (I2C0DAT - 0xE001C008, I2C1DAT - 0xE005C008)

This register contains the data to be transmitted or the data just received. The CPU can read and write to this register while it is not in the process of shifting a byte. This register can be accessed only when SI bit is set. Data in I2DAT remains stable as long as the SI bit is set. Data in I2DAT is always shifted from right to left: the first bit to be transmitted is the MSB (bit 7), and after a byte has been received, the first bit of received data is located at the MSB of I2DAT.

Table 93: I^{2}C Data Register (I2DAT - 0xE001C008)

I2DAT	Function	Description	Reset Value
7:0	Data	Transmit/Receive data bits	0

I^{2}C Slave Address Register (I2C0ADR - 0xE001C00C, I2C1ADR - 0xE005C00C)

This register is readable and writable, and is only used when the I^{2}C is set to slave mode. In master mode, this register has no effect. The LSB of I2ADR is the general call bit. When this bit is set, the general call address (00h) is recognized.

Table 94: I^{2}C Slave Address Register (I2C0ADR - 0xE001C00C, I2C1ADR - 0xE005C00C)

I2ADR	Function	Description	Reset Value
0	GC	General Call bit	0
7:1	Address	Slave mode address	0

```
/* Interrupt driven I2C example for LPC2134
For use with the HMC6532 Compass module

Writes the 'A' command to the I2C0 port twice
and read two bytes of data.  After that it
toggles the upper 8-bits of port 1 indefinitely.
*/

#include <LPC213x.H>
#include <stdarg.h>

void I2CISR (void) __irq ;
//I2C interrupt routine
//Starts Master and writes one byte of data
void I2CWriteByte(unsigned int I2CAddr,unsigned char data);
//Starts Master and reads two bytes of data
void I2CReadBytes(unsigned int I2CAddr);

unsigned char     count,       //To keep track of first and second byte in interrupt
                  I2CAddress,  //Address for reading and writing to compass
                  memData,     //Data sent to Compass (commands)
                  d1In, d2In,  //Binary heading from compass
                  lock;        //Used to continue interrupt
int main(void)
  {VPBDIV = 2;                          //PClk = CClk/2
   IODIR1 = 0xFFFF0000;                 //Upper 16 bits of port 1 to output
   lock = 0;                            //Initilise the lock flag
   VICVectCntl1 = 0x00000029;  //select a priority slot for a given interrupt
   VICVectAddr1 = (unsigned)I2CISR;     //pass the address of the IRQ into the VIC slot
   VICIntEnable = 0x00000200;           //enable interrupt

   PINSEL0 = 0x50;                      //Switch GPIO to I2C pins

   I2C0SCLH = 150;   //Set bit rate 14.7456Mhz/VPBDIV+SCLH+SCLL = 14.7456/4+8+8 = 57.6Khz
   I2C0SCLL = 144;

   I2CWriteByte(0x42,'A');    //write 'A' command to the compass via I2C port
   I2CWriteByte(0x42,'A');    //write another 'A'
   I2CReadBytes(0x43);        //read back two bytes of data
   //At this point there are two bytes of data in d2In,d1In
   I2C0CONCLR     = 0x000000FF;         //Clear all I2C settings
   I2C0CONSET = 0;                      //Disable I2C
   VICIntEnable = 0x00000000;           //disable interrupt
   while(1)
     {IOPIN1 = IOPIN1 | 0xFF000000;     //Toggle upper 8 bits of P1 to
      IOPIN1 = IOPIN1 & 0x00FFFFFF;     //  show that we got here
     }
  }

//Write a byte to the I2C port
void I2CWriteByte(unsigned int I2CAddr,unsigned char data)
  {while(lock == 1);        //Wait for interrupt to signal end of I2C activity
   lock    = 1;             //Set I2C bus as active.  Interrupt releases lock

   I2CAddress = I2CAddr; //Place address and data in Globals to be used by the interrupt
   memData = data;
   I2C0CONCLR     = 0x000000FF;                      //Clear all I2C settings
   I2C0CONSET     = 0x00000040;                      //Enable the I2C interface
   I2C0CONSET     = 0x00000020;                      //Start I2C
  }
//Read two bytes to global d1In and d2In
void I2CReadBytes(unsigned int I2CAddr)
  {count = 1;               // Read two bytes
```

```
   while(lock == 1);        //Wait for interrupt to signal end of I2C activity
   lock    = 1;                 //Set I2C bus as active

   I2CAddress     = I2CAddr;    //Place address and data in Globals to be used by the
interrupt
   I2C0CONCLR     = 0x000000FF;                   //Clear all I2C settings
   I2C0CONSET     = 0x00000040;                   //Enable the I2C interface
   I2C0CONSET     = 0x00000020;                   //Start condition
  }
//I2C Interrupt Service Routine
void I2CISR (void)    __irq             //I2C interrupt routine
  {switch (I2C0STAT)                    //Read result code and switch to next action
    {// Start and Send byte conditions
     case ( 0x08):                      //Start bit
       I2C0CONCLR     = 0x20;           //Clear start bit
       I2C0DAT       = I2CAddress;      //Send address and write bit
     break;
     case (0x18):                        //Slave address+W, ACK
       I2C0DAT       = memData;          //Write Memory start address to tx register
     break;
     case (0x20):                        //Salve address +W, Not ACK
       I2C0DAT       = I2CAddress;       //Resend address and write bi
     break;
     case (0x28):
         {I2C0CONSET     = 0x10;         //Stop condition
          lock = 0;                      //Signal end of I2C activity
         }
     break;
     case (0x30)    :                    //Data sent, NOT Ack
       I2C0DAT      = memData;           //Write data to tx register
     break;
    //Receive byte conditions
     case (0x40) :                        //Slave Address +R, ACK
       I2C0CONSET     = 0x04;             //Enable ACK for data byte
     break;
     case (0x48) :                        //Slave Address +R, Not Ack
       I2C0CONSET     = 0x20;             //Resend Start condition
     break;
     case (0x50) :                        //Data Received, ACK
       if(count>0)
         {d2In    = I2C0DAT;
          count--;
         }
       else
         {d1In = I2C0DAT;
          I2C0CONSET     = 0x10;           //Stop condition
          lock         = 0;                //Signal end of I2C activity
         }
     break;
     case (0x58):                        //Data Received, Not Ack
       I2C0CONSET     = 0x20;            // Resend Start condition
     break;
     default :
     break;
    }
   I2C0CONCLR     = 0x08;                //Clear I2C interrupt flag
   VICVectAddr = 0x00000000;             //Clear interrupt in
  }
```