VORAGO TECHNOLOGIES

VORAGO VA108x0 I^{2}C programming application note

MARCH 14, 2017 Version 1.1

VA10800/VA10820

Abstract

There are hundreds of peripheral devices utilizing the I^{2}C protocol. Most of these require more than single byte transfers. The VORAGO I^{2}C controller provides two 16-byte FIFOs to reduce the CPU overhead when transferring packets of information. This application note provides guidance on using the I^{2}C control block. Software drivers accompany the document for easy implementation into an application.

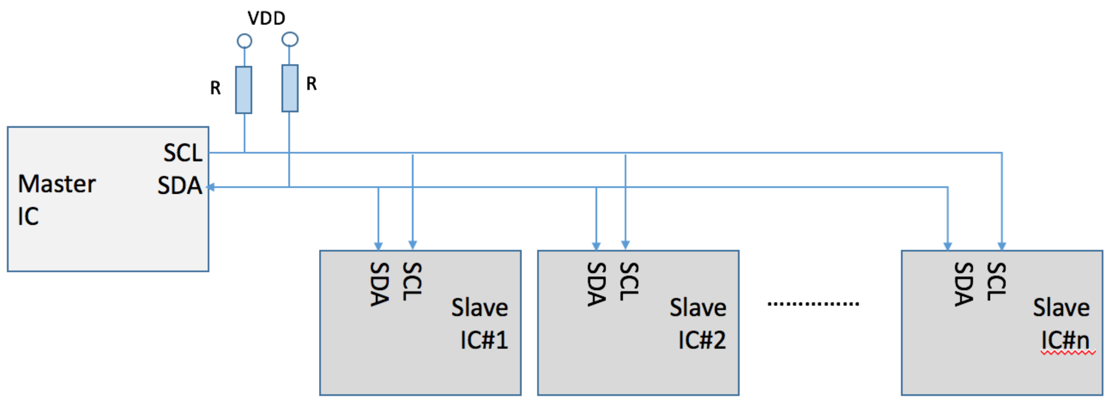
Table of Contents

1 Overview of I^{2}C

Philips Semiconductors (now NXP Semiconductors) introduced the 2-wire communication protocol in 1982. The latest revision of the specification can be found at: http://www.nxp.com/documents/user_manual/UM10204.pdf. Some of the more pertinent characteristics and features are listed here:

- Two wires are used, one is the clock (SCL) and the other is data (SDA).
- Discrete resistors on the printed circuit board pull-up both SCL and SDA to a logic high. Master and slave devices can only pull the lines low.

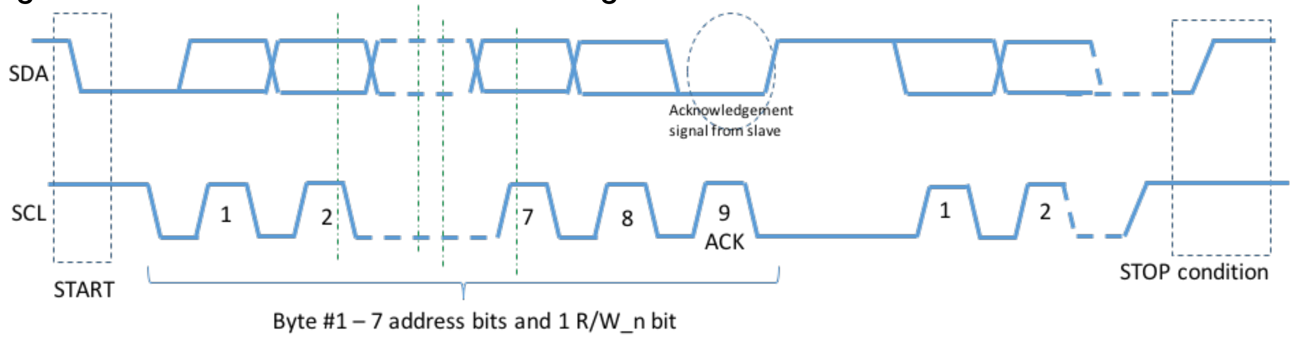
Figure 1 - Example of single master I²C bus



- Nodes are either master or slave devices. Multiple masters can reside on the same bus but every node is responsible for ensuring the bus is idle before commencing a packet transmission. When a slave node is addressed, it must acknowledge each byte of information sent to it. Failure to do so will cause that transaction to be aborted.
- A master node provides the clock for entire transaction. The slave node responds to clock edges and does not need to have a bit rate set.
- A master initiates a transaction and specifies which slave nodes should respond by supplying a device address (7 or 10 bits) at the beginning of each transaction. The initial byte of information sent contains a read / write bit in the least significant position that determines if the packet is reading from or writing to the slave IC.
- The standard data rate is 100 kbits/second. A fast mode with 400 kbits/sec is available. Higher data rates are sometimes implemented on single master systems with very low bus capacitance.
- For multi-master systems, bus arbitration is performed by both masters transmitting their intended destination address at the same time. The master with the lower destination address will win (active pull-down device will keep SDA low). When that happens, the losing master will drop off the bus for that transaction and broadcast later when the bus becomes free.
- I²C conducts transactions in a byte by byte fashion with the slave node being addressed acknowledging each byte before any additional activity.
  - A HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition.

- A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.

Figure 2 - Data transfer on I²C bus showing START and STOP conditions


SDA
SCL
Acknowledgement signal from slave
1
2
7
8
9
ACK
1
2
START
STOP condition
Byte #1 – 7 address bits and 1 R/W_n bit

2 VORAGO I²C block overview

The full description of the block can be found in the VA10800/820 Programmers Guide. This section provides supplemental information on how use the block in a final application. The controller block supports both slave and master operation. It also supports fast and normal mode along with 7 and 10 bit addressing modes. For sake of simplicity, this document focuses on master mode with 7-bit addressing and normal bit rate.

There are two identical I²C blocks on both the VA10800 or VA10820. There is a pair of dedicated open collector pins for each interface and no separate port pin initialization is required. Before accessing any I²C registers, the peripheral clock for that block must be enabled in the SYSCONFIG block's PERIPHERAL_CLK_ENABLE register.

2.1 Master Mode Register summary

Name	Description	Overview and Use
CTRL	Control register	Sets parameters for block operation including the enable bit
CLKSCALE	Clock Scale register	Divides system clock down to set bit rate
WORDS	Word Count register	Sets number of Words in a transaction
ADDRESS	Address register	This is the slave device's address. It includes the R/W bit in location b0.
DATA	Data register	This is entry point for Rx and Tx FIFO
CMD	Command register	Used to commence a transaction
STATUS	I^{2}C Controller Status register	Provides information on the status of block and FIFOs.
STATE	Controller State register	Used for test. Contains FIFO count information
TXCOUNT	TX Count register	Transaction data word count. Good to use for longer transactions. -
RXCOUNT	Rx Count register	Transaction data word count. Good to use for longer transactions.
IRQ_ENB	Interrupt Enable	Determines which of 14 interrupt sources are enabled
IRQ_RAW	Raw Interrupt Status	Provides status of all 14 interrupt sources regardless of enable bit
IRQ_END	Enabled Interrupt Status	Provides status of only enabled interrupt sources
IRQ_CLR	Clear Interrupt Status	Clears latched interrupt status bits. [1]
RXFIFOIRQTRG	Rx FIFO Trigger level register	Sets watermark for triggering interrupt
TXFIFOIRQTRG	Tx FIFO Trigger level register	Sets watermark for triggering interrupt
FIFO_CLR	FIFO Clear register	Clears both Rx and Tx FIFO counters. Used prior to a transaction
TMCONFIG	Timing Config register	Allows customized bit timing. By default, this is not enabled.
CLKTOLIMIT	Clock timeout register	Used to generate interrupt if the clock line stays low too long.

2.2 I^{2}C block functional partitioning:

Each of the functional sections of the I^{2}C module are briefly described in this section.

[1] Seven of the 14 interrupt sources come directly from the STATUS register including the most commonly used IDLE and I2CIDLE interrupts. These seven become active when the corresponding STATUS bit goes from a 0 to 1 and stay latched. To clear the active interrupt latch, a 1 must be written to the corresponding bit in IRQ_CLR.

- ICLK generator: The I^{2}C bus clock is created as a divisor of the system bus clock. Standard speed, 100 kbps, requires the system bus be 20 times the I^{2}C rate. For high speed, 400 kbps, the bus must be 25 times the I^{2}C bit rate.
- Status and control registers: A set of registers is provided to configure the block and to monitor the status.
- Slave / Master controllers: There are separate blocks for master and slave control. Both master and slave modes provide two 16-word FIFOs to simplify the transmit and receive operations. These controllers handle all the bus activity including START / STOP events, acknowledgement, clock stretching and arbitration.
- Interrupt logic: Interrupt requests to the CPU can be generated when they are enabled and the specified event occurs such as a FIFO being full or empty. The full list of 14 interrupt sources is shown here.
    - i. I^{2}C Bus Idle
    - ii. I^{2}C Controller Idle
    - iii. Waiting
    - iv. Stalled
    - v. Arbitration lost
    - vi. NACK received on address
    - vii. NACK received on data
    - viii. Clock low time out
    - ix. Tx FIFO overflow
    - x. Rx FIFO overflow
    - xi. Tx FIFO ready for data
    - xii. Rx FIFO has data ready
    - xiii. Tx FIFO is empty
    - xiv. Rx FIFO is full
- IO interface and filters: Digital and analog glitch filters can be optionally enabled. When running the system clock at 50 MHz, the digital filter is recommended. When running below 20 MHz or using I^{2}C fast mode, the analog filter is recommended.

3 Examples

The following sections provide example software programs to setup and operate the I2C block.

3.1 I^{2}C peripheral initialization

Out of RESET, the block is disabled and all registers are set to their default value. Prior to writing any I^{2}C registers, the clock must be enabled in the System Configuration Peripheral -> Peripheral Clock Enable CTRL register. The pins used for the I^{2}C functions are dedicated and require no setup.

There are filters available on the clock and data pins. Analog or Digital sampling can be configured in the CTRL register.

The bit rate must be set by the CLKSCALE register. The programmer's manual has a convenient table to reference. For all examples in this AN, the bus frequency is 50 MHz, so the lower 8 bits of CLKSCALE must have 0x18.

The master or slave mode must be selected in the CTRL register.

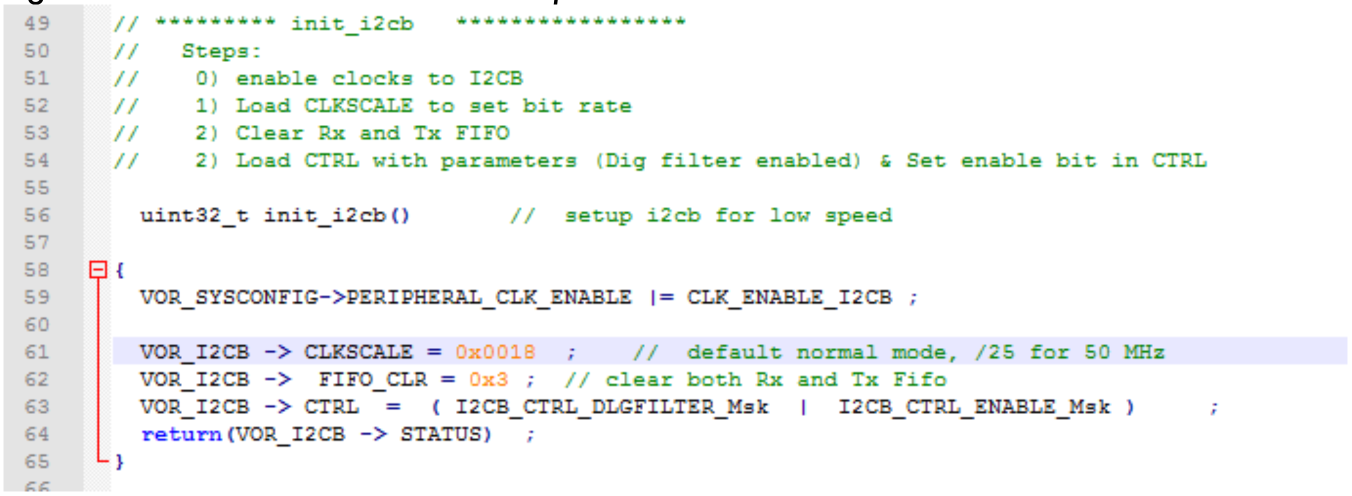
Summary of steps to setup the I^2C module

1. Enable peripheral clocks in SYSCONFIG->PERIPHERAL_CLK_ENABLE
2. Configure the I^2C block
    a. Set I^2C clock generator (ICLK) in CLKSCALE register
    b. Clear both Rx and Tx FIFO in the FIFO_CLR register
    c. Set operating parameters and enable the block in the CTRL register

At this point, the I^2C module is ready to transmit or receive information.

Example code to show block initialization is shown here.

Figure 3 - I^2C initialization code example



```
// ********* init_i2cb   *****************
//   Steps:
//    0) enable clocks to I2CB
//    1) Load CLKSCALE to set bit rate
//    2) Clear Rx and Tx FIFO
//    2) Load CTRL with parameters (Dig filter enabled) & Set enable bit in CTRL

  uint32_t init_i2cb()       //  setup i2cb for low speed

{
  VOR_SYSCONFIG->PERIPHERAL_CLK_ENABLE |= CLK_ENABLE_I2CB ;

  VOR_I2CB -> CLKSCALE = 0x0018  ;    //  default normal mode, /25 for 50 MHz
  VOR_I2CB ->  FIFO_CLR = 0x3 ;  // clear both Rx and Tx Fifo
  VOR_I2CB -> CTRL   =  ( I2CB_CTRL_DLGFILTER_Msk  |  I2CB_CTRL_ENABLE_Msk )     ;
  return(VOR_I2CB -> STATUS)  ;
}
```

3.2 I^2C master write operations

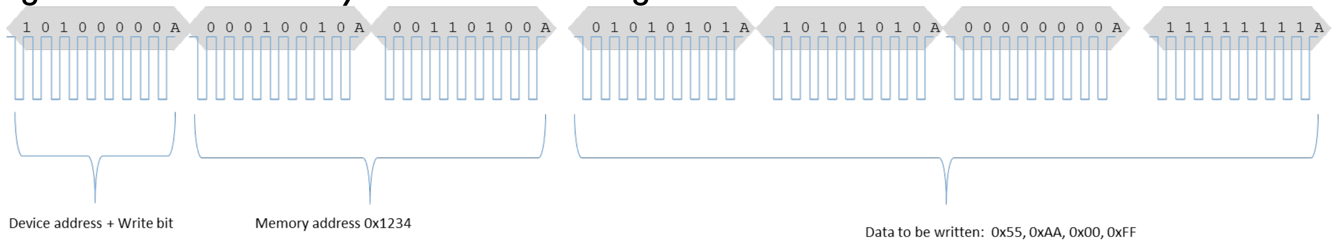
Write operations will have three pieces of information:

a) device address
b) address in the device's memory map
c) data to be written.

Depending on the slave device, the memory map may require 1, 2 or 3 bytes for an address. The data being written can be a single byte or multiple bytes. The master will retain ownership of the bus until it asserts a stop condition (releases both SDA and SCL).

For this example, consider the case of a 16k byte memory slave device with a device address of 0x50. Four bytes (0x55, 0xAA, 0x00, 0xFF) will be written to an address range starting at 0x1234. The below diagram shows the time line of bus activity. The 0, 1 and A characters in the grey boxes show the SDA information.

Figure 4 - I²C bus activity time line for writing data = 0x55AA00FF to address 0x1234



The controller has a register, WORDS, to determine the transaction length and clock the proper number of bytes. The WORDS register must be loaded before a transaction starts. For our example the number written to WORDS is 0x06.

The intended device address must be written to the ADDRESS register. This can be either the standard 7-bit address or the longer 10-bit address. For this example, the device address is 0x50. The value written to ADDRESS is the device address shifted left 1 position with a 0 in the least significant bit to designate a write operation. Therefore, 0x00A0 would be written to ADDRESS.

> *Note that not all slave device addresses are documented identically, some include the R/W bit and some do not. Pay close attention to how the slave device calls out the address as being either 7 or 8 bits.*

The DATA register is the entry point to the Rx and Tx FIFO. Data must be loaded into the FIFO before the transaction starts. For our example, there are 6 writes to the FIFO: 0x12, 0x34, 0x55, 0xAA, 0x00, 0xFF.

Finally, to commence the transaction, the CMD register is written to. Both the START and STOP bits in this register need to be set for an atomic transaction. For our example, CMD =

0x3 would be written. At this point the module would begin the transaction. If the slave device is acknowledging, the above waveform should be seen. If the slave device is not active, only the first byte is transmitted.

Summary of steps:

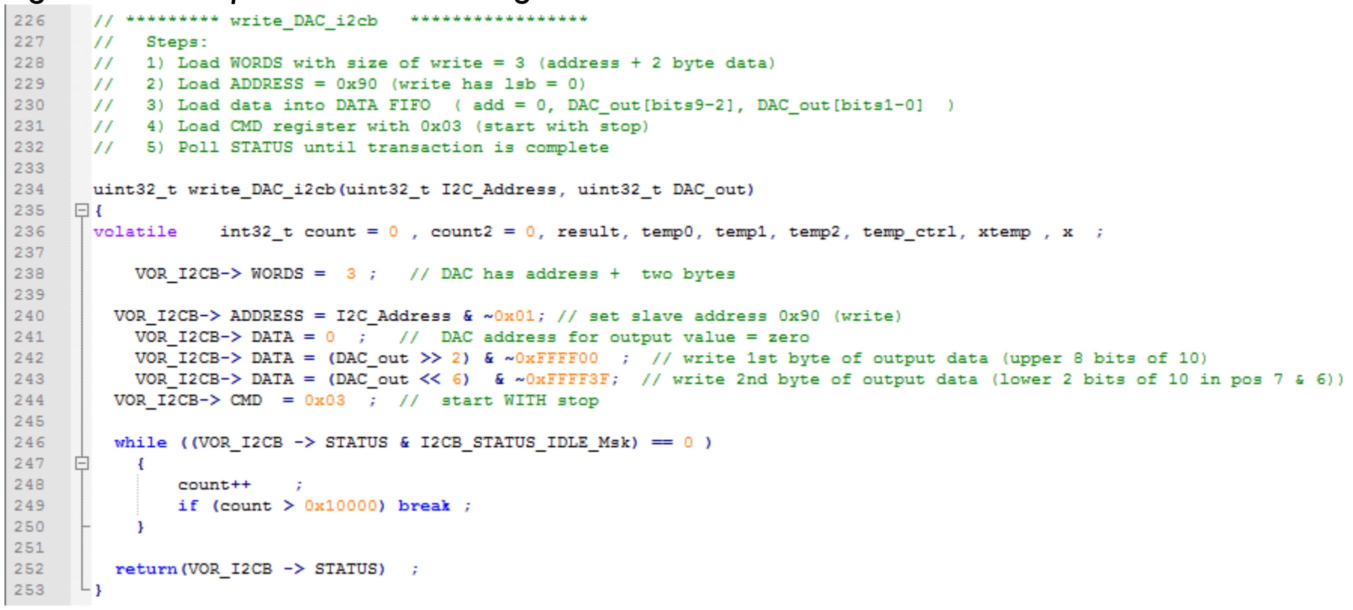
1. Set number of data bytes, WORDS = 0x06
2. Load Tx FIFO with data, DATA = 0x12, 0x34, 0x55, 0xAA, 0x00, 0xFF. (6 separate write operations)
3. Set slave address, ADDRESS = 0x0A
4. Start transaction, CMD = 0x3

3.2.1 Example code and waveform for a 10-bit DAC

The following code snippet sends data to a 10-bit Digital to Analog converter. The converter has an input format of the 8 most-significant-bits in first byte and the 2 least-significant-bits in second byte.

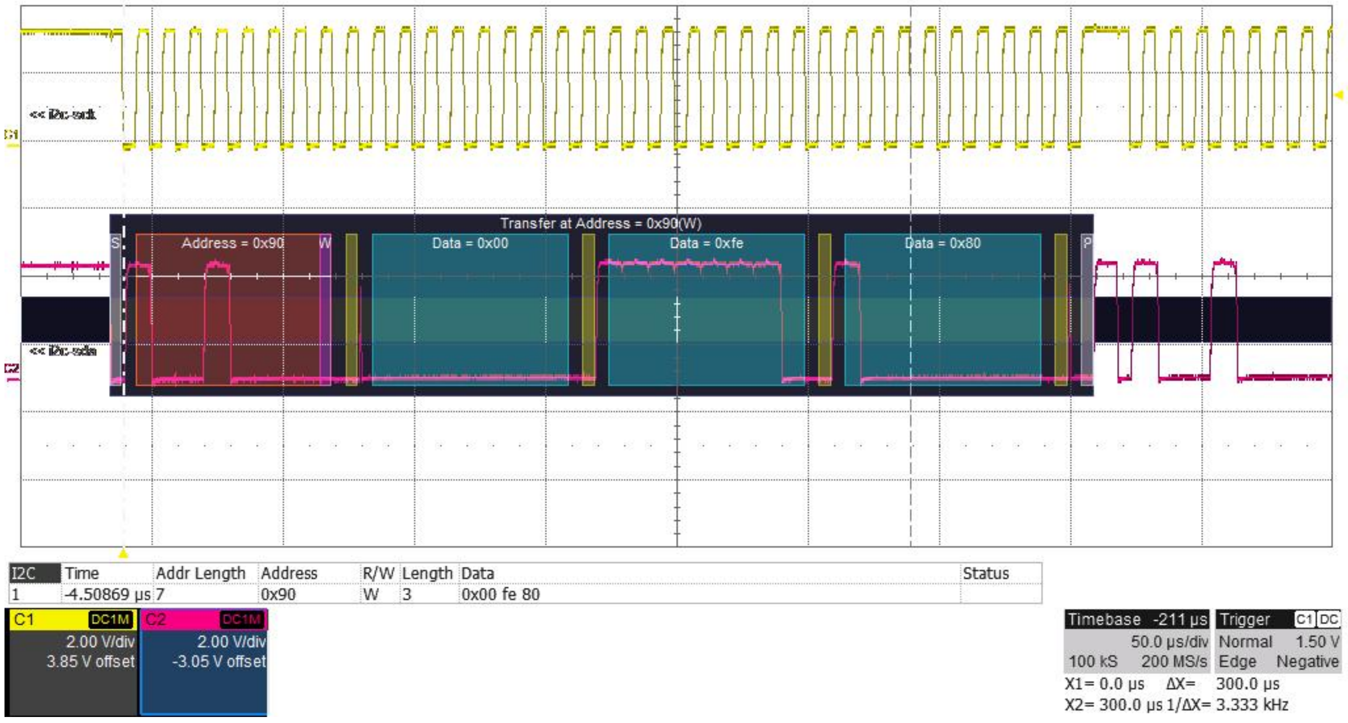
Byte 1 – Bits 9 to 2	Byte 0 – Bits 1 & 0

Figure 5 - Example code for sending data to a 10-bit DAC.



```
// ********* write_DAC_i2cb   *****************
//   Steps:
//   1) Load WORDS with size of write = 3 (address + 2 byte data)
//   2) Load ADDRESS = 0x90 (write has lsb = 0)
//   3) Load data into DATA FIFO  ( add = 0, DAC_out[bits9-2], DAC_out[bits1-0]  )
//   4) Load CMD register with 0x03 (start with stop)
//   5) Poll STATUS until transaction is complete

uint32_t write_DAC_i2cb(uint32_t I2C_Address, uint32_t DAC_out)
{
volatile   int32_t count = 0 , count2 = 0, result, temp0, temp1, temp2, temp_ctrl, xtemp , x  ;

    VOR_I2CB-> WORDS =  3 ;   // DAC has address +  two bytes

  VOR_I2CB-> ADDRESS = I2C_Address & ~0x01; // set slave address 0x90 (write)
    VOR_I2CB-> DATA = 0  ;   //  DAC address for output value = zero
    VOR_I2CB-> DATA = (DAC_out >> 2) & ~0xFFFF00  ;  // write 1st byte of output data (upper 8 bits of 10)
    VOR_I2CB-> DATA = (DAC_out << 6)  & ~0xFFFF3F;  // write 2nd byte of output data (lower 2 bits of 10 in pos 7 & 6))
  VOR_I2CB-> CMD  = 0x03  ;  //  start WITH stop

  while ((VOR_I2CB -> STATUS & I2CB_STATUS_IDLE_Msk) == 0 )
    {
        count++    ;
        if (count > 0x10000) break ;
    }

  return(VOR_I2CB -> STATUS)  ;
}
```

Figure 6 - Scope capture of transaction to 10-bit DAC



I2C	Time	Addr Length	Address	R/W	Length	Data	Status
1	-4.50869 µs	7	0x90	W	3	0x00 fe 80	

```
C1  DC1M          C2  DC1M
    2.00 V/div        2.00 V/div
    3.85 V offset     -3.05 V offset

Timebase  -211 µs   Trigger  C1 DC
          50.0 µs/div  Normal   1.50 V
100 kS    200 MS/s     Edge     Negative
X1= 0.0 µs     ΔX=    300.0 µs
X2= 300.0 µs 1/ΔX= 3.333 kHz
```

3.3 I²C block master read operations

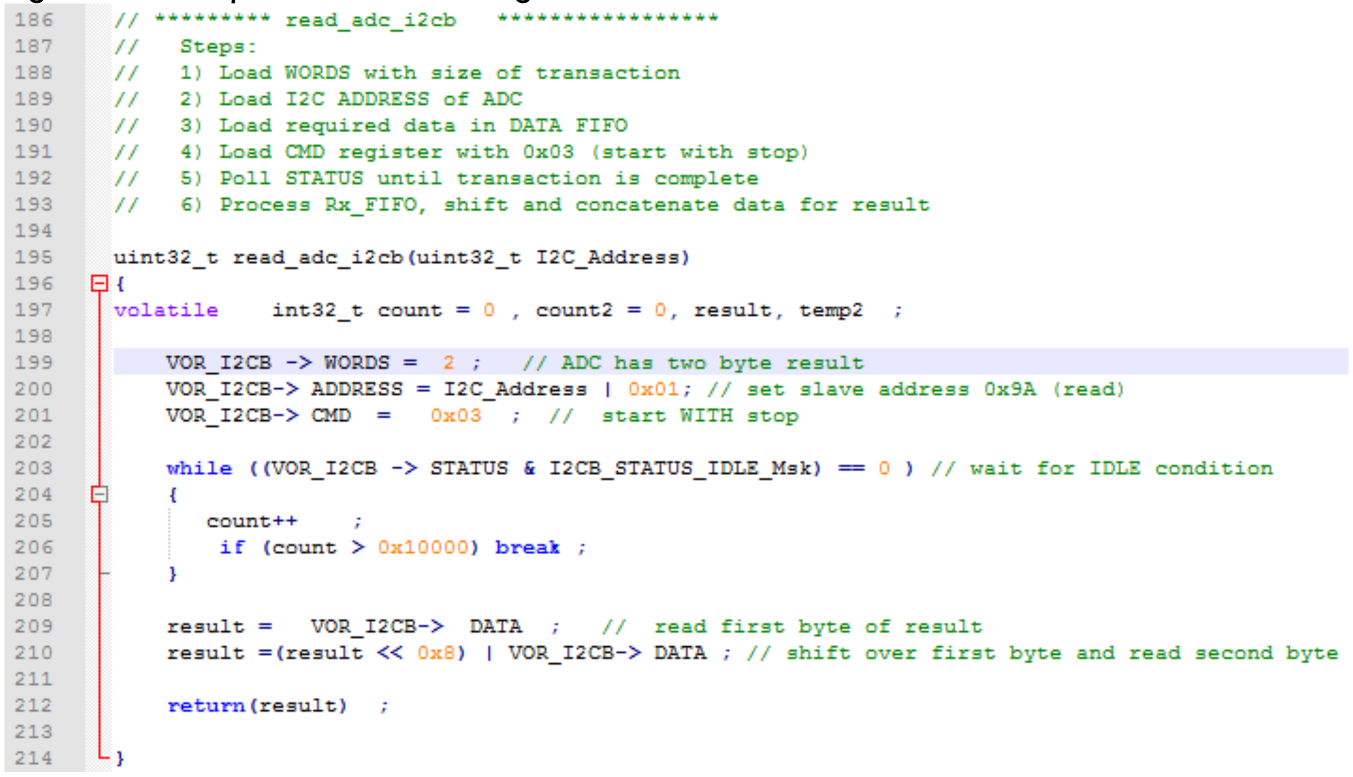
Read operations are like the write operation except instead of writing to the Tx FIFO before the transaction, the read data is pulled from the Rx FIFO after the transaction is complete. A 10-bit ADC is used for this example. It has an address of 0x9A and requires no initial setup commands.

Steps to setup the module

1. Initialize I²C block as shown in section 3.1
2. Set WORD = 2
3. Set ADDRESS = 0x9A with b0 = 1 for read.
4. Set CMD = 3, start and stop transaction
5. Poll for IDLE = 1
6. Read Rx FIFO and shift and combine data for result

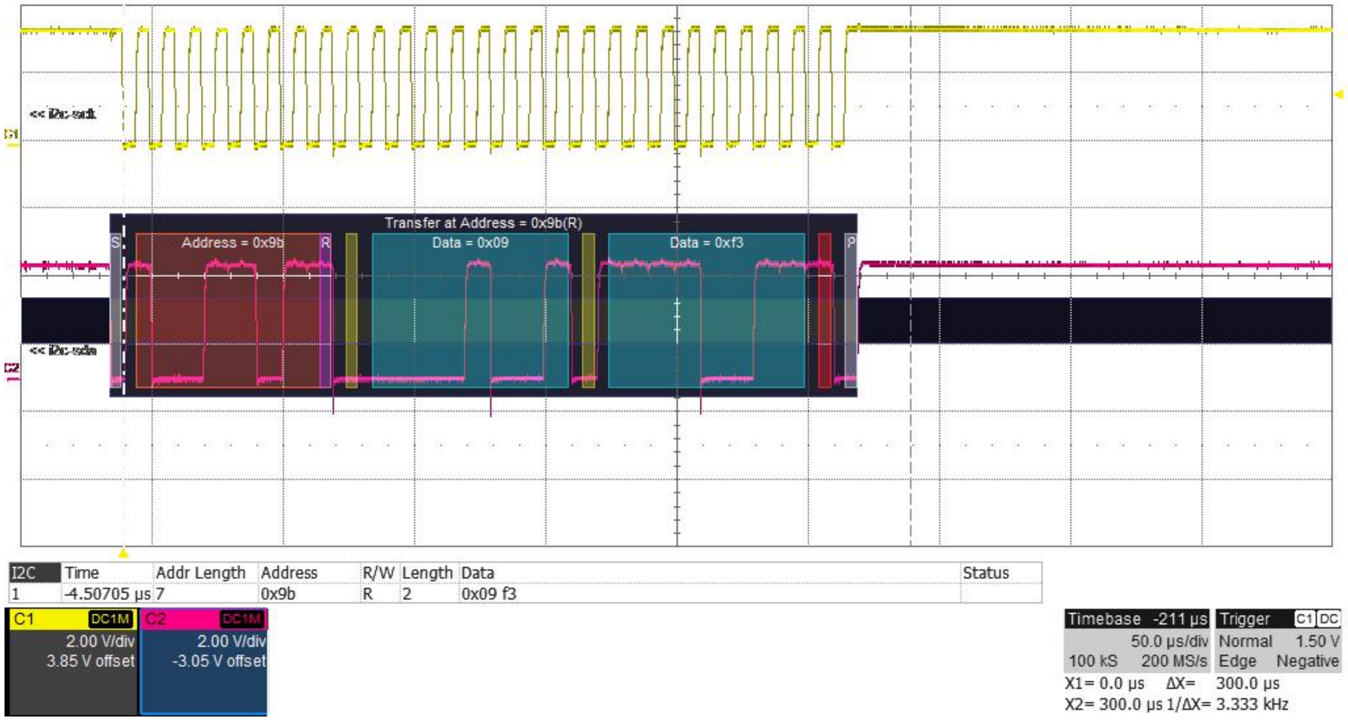
A short code example to perform these steps is shown here.

Figure 7 - Example code for reading 10-bit ADC via I^2C



```
// *********  read_adc_i2cb   *****************
//    Steps:
//    1) Load WORDS with size of transaction
//    2) Load I2C ADDRESS of ADC
//    3) Load required data in DATA FIFO
//    4) Load CMD register with 0x03 (start with stop)
//    5) Poll STATUS until transaction is complete
//    6) Process Rx_FIFO, shift and concatenate data for result

uint32_t read_adc_i2cb(uint32_t I2C_Address)
{
volatile    int32_t count = 0 , count2 = 0, result, temp2   ;

    VOR_I2CB -> WORDS =  2 ;   // ADC has two byte result
    VOR_I2CB-> ADDRESS = I2C_Address | 0x01; // set slave address 0x9A (read)
    VOR_I2CB-> CMD   =   0x03  ;  //  start WITH stop

    while ((VOR_I2CB -> STATUS & I2CB_STATUS_IDLE_Msk) == 0 ) // wait for IDLE condition
    {
       count++    ;
        if (count > 0x10000) break ;
    }

    result =   VOR_I2CB->  DATA  ;   //  read first byte of result
    result =(result << 0x8) | VOR_I2CB-> DATA ; // shift over first byte and read second byte

    return(result)  ;

}
```

Figure 8 - Scope capture of I^2C transaction reading 10-bit ADC



I2C	Time	Addr Length	Address	R/W	Length	Data	Status
1	-4.50705 µs	7	0x9b	R	2	0x09 f3	

C1 DC1M
2.00 V/div
3.85 V offset

C2 DC1M
2.00 V/div
-3.05 V offset

Timebase -211 µs
50.0 µs/div
100 kS 200 MS/s

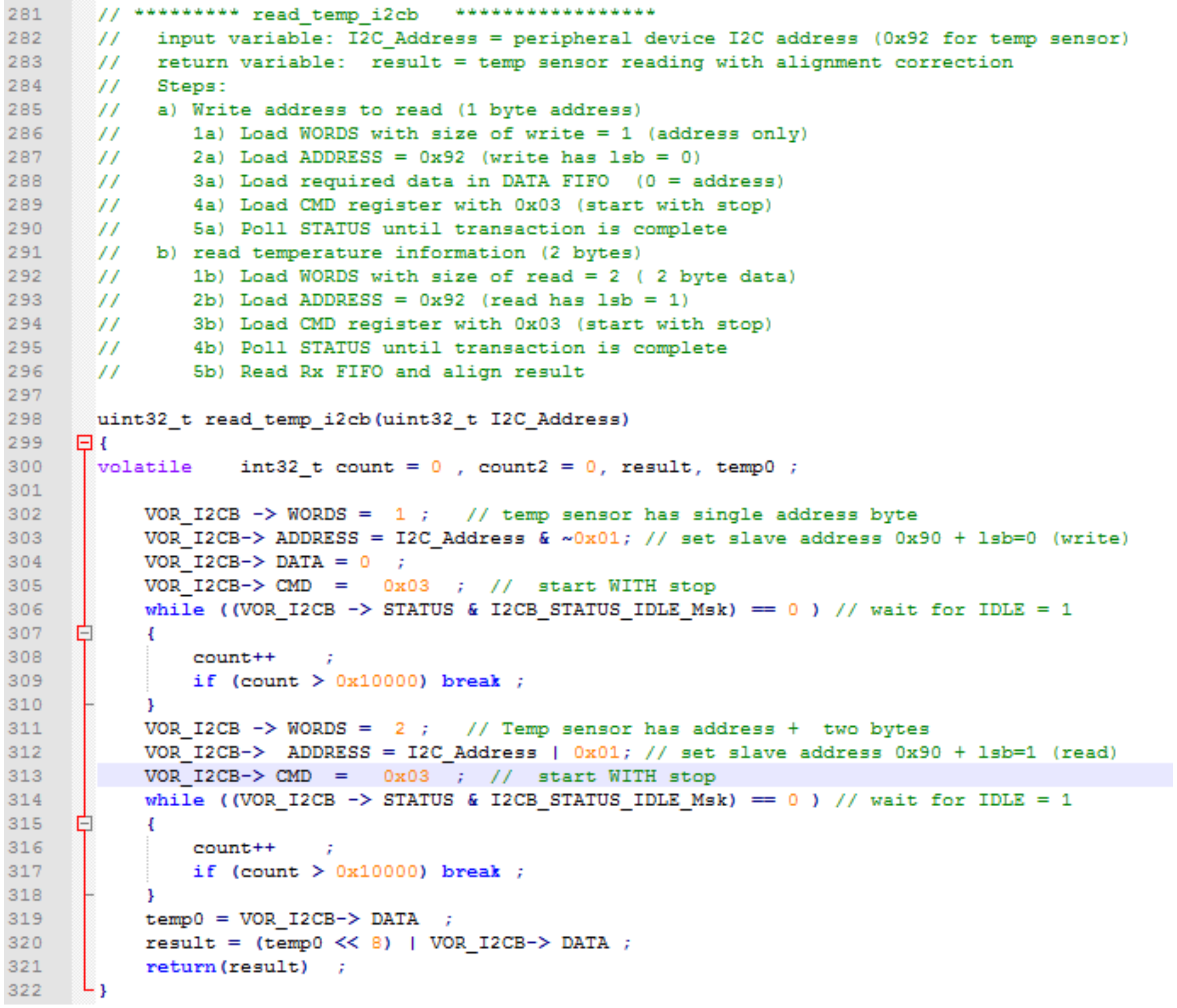
Trigger C1 DC
Normal 1.50 V
Edge Negative

X1= 0.0 µs ΔX= 300.0 µs
X2= 300.0 µs 1/ΔX= 3.333 kHz

3.4 I^{2}C block master write then read operation

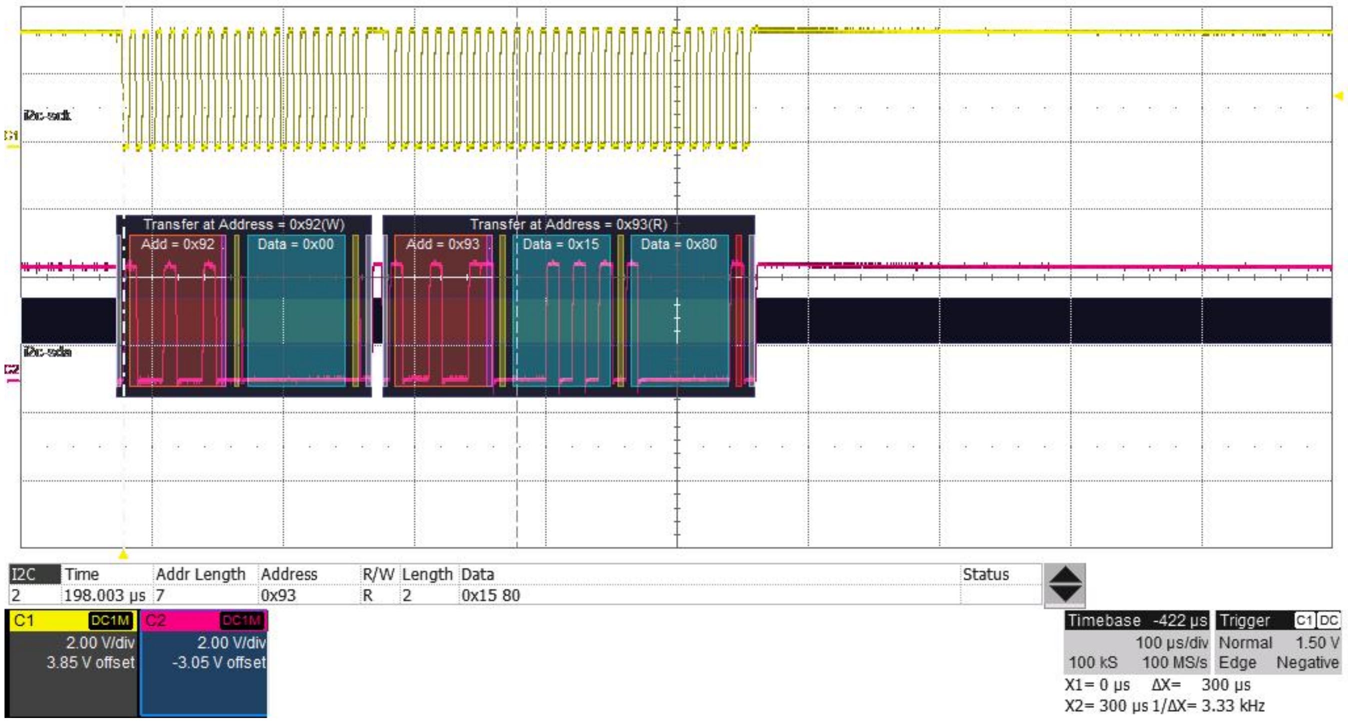
Some I^{2}C slave devices require an address pointer to be written prior to data being read. The below code example reads temperature sensor that requires an address to be sent prior to the read command.

Figure 9 - Example code to read temperature sensor



```
// ********* read_temp_i2cb   *****************
//   input variable: I2C_Address = peripheral device I2C address (0x92 for temp sensor)
//   return variable:  result = temp sensor reading with alignment correction
//   Steps:
//   a) Write address to read (1 byte address)
//      1a) Load WORDS with size of write = 1 (address only)
//      2a) Load ADDRESS = 0x92 (write has lsb = 0)
//      3a) Load required data in DATA FIFO  (0 = address)
//      4a) Load CMD register with 0x03 (start with stop)
//      5a) Poll STATUS until transaction is complete
//   b) read temperature information (2 bytes)
//      1b) Load WORDS with size of read = 2 ( 2 byte data)
//      2b) Load ADDRESS = 0x92 (read has lsb = 1)
//      3b) Load CMD register with 0x03 (start with stop)
//      4b) Poll STATUS until transaction is complete
//      5b) Read Rx FIFO and align result

uint32_t read_temp_i2cb(uint32_t I2C_Address)
{
volatile    int32_t count = 0 , count2 = 0, result, temp0 ;

    VOR_I2CB -> WORDS =  1 ;   // temp sensor has single address byte
    VOR_I2CB-> ADDRESS = I2C_Address & ~0x01; // set slave address 0x90 + lsb=0 (write)
    VOR_I2CB-> DATA = 0  ;
    VOR_I2CB-> CMD  =   0x03  ;  //  start WITH stop
    while ((VOR_I2CB -> STATUS & I2CB_STATUS_IDLE_Msk) == 0 ) // wait for IDLE = 1
    {
        count++    ;
        if (count > 0x10000) break ;
    }
    VOR_I2CB -> WORDS =  2 ;   // Temp sensor has address +  two bytes
    VOR_I2CB->  ADDRESS = I2C_Address | 0x01; // set slave address 0x90 + lsb=1 (read)
    VOR_I2CB-> CMD  =   0x03  ;  //  start WITH stop
    while ((VOR_I2CB -> STATUS & I2CB_STATUS_IDLE_Msk) == 0 ) // wait for IDLE = 1
    {
        count++    ;
        if (count > 0x10000) break ;
    }
    temp0 = VOR_I2CB-> DATA  ;
    result = (temp0 << 8) | VOR_I2CB-> DATA ;
    return(result)  ;
}
```

Figure 10 - Scope capture of I^{2}C transaction to read a temperature sensor



I2C	Time	Addr Length	Address	R/W	Length	Data	Status
2	198.003 µs	7	0x93	R	2	0x15 80	

C1 DC1M 2.00 V/div 3.85 V offset
C2 DC1M 2.00 V/div -3.05 V offset
Timebase -422 µs 100 µs/div 100 kS 100 MS/s
Trigger C1 DC Normal 1.50 V Edge Negative
X1= 0 µs ΔX= 300 µs
X2= 300 µs 1/ΔX= 3.33 kHz

3.5 Interrupt operation mode

Continually polling the STATUS register for a transaction to complete is the simplest way to control activity on the I^{2}C bus. However, this can consume valuable CPU cycles that could be used for other tasks. A single byte I^{2}C transaction at normal speed is approximately 90 microseconds (9 bits x (1/100kHz)). For longer transactions, the use of polling can become prohibitive and using interrupts is preferred.

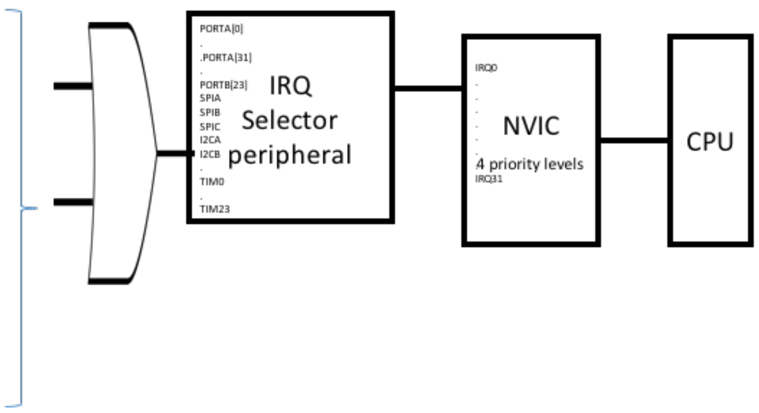
Using interrupts can allow the I^{2}C to be performing transactions while the CPU is handling other tasks in parallel. It does add to the complexity of configuring the module and a separate interrupt subroutine is required.

The VA108xx family allows 14 sources of interrupts from the I^{2}C block that all get OR'ed together as shown in Figure 11. Seven of the sources come from the STATUS register and are latched when the STATUS bit goes from a 0 to 1. All interrupt sources can be monitored in IRQ_RAW. To clear latched interrupts, a 1 must be written to the corresponding bit position in IRQ_CLR. Only interrupt sources that have the corresponding bit set in IRQ_ENB can trigger an interrupt.

The IRQ selector (IRQSEL) peripheral routes the pin, timer and peripheral interrupt signals to 32 IRQ inputs of the NVIC. The M0 NVIC has 4 levels of priorities. The NVIC interrupt pending flag will be cleared when the interrupt is serviced. Software does not need to manually clear this bit. See Figure 11 for the path an interrupt source traverses from the I^{2}C block to the CPU.

Figure 11 - Interrupt source table and path to CPU

Source	DESCRIPTION	LATCHED ON 0 TO 1 TRANSITION	IRQ_RAW(R ONLY)	IRQ_CLR (W ONLY)	IRQ_ENB
STATUS	I2C bus IDLE	Y	•	RST/HLD	Y/N
STATUS	IDLE	Y	•	RST/HLD	Y/N
STATUS	WAITING	Y	•	RST/HLD	Y/N
STATUS	STALLED	Y	•	RST/HLD	Y/N
STATUS	ARB_LOST	Y	•	RST/HLD	Y/N
STATUS	NACKADDR	Y	•	RST/HLD	Y/N
STATUS	NACKDATA	Y	•	RST/HLD	Y/N
OTHER	CLKLOTO	N	•	RST/HLD	Y/N
OTHER	TXOV	N	•	RST/HLD	Y/N
OTHER	RXOV	N	•	RST/HLD	Y/N
OTHER	TXREADY	N	•	RST/HLD	Y/N
OTHER	RXREADY	N	•	RST/HLD	Y/N
OTHER	RXEMPTY	N	•	RST/HLD	Y/N
OTHER	RXFULL	N	•	RST/HLD	Y/N

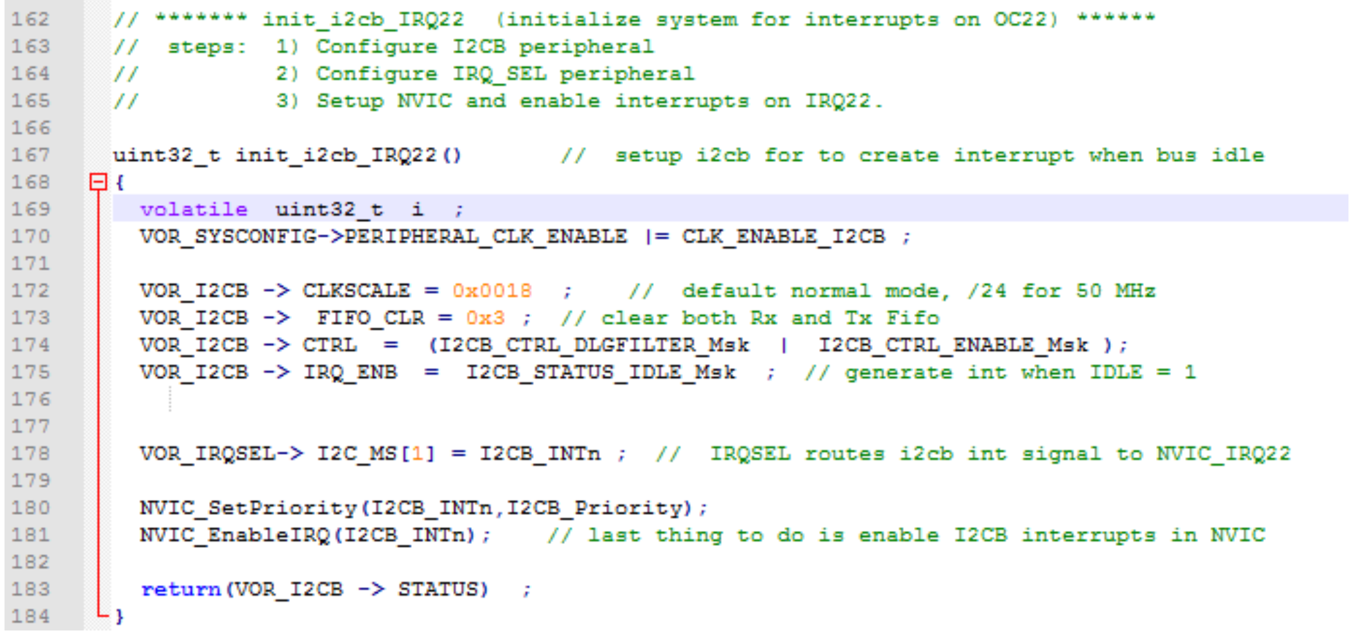


Steps to setup the module for generating interrupts on an IDLE condition

1. Initialize the I^{2}C block as described in section 3.1.
2. Set the IDLE bit in the IRQ_ENB register
3. Assign the I2CB interrupt to NVIC input 22 in the IRQSEL block.
4. In the NVIC, set priority level of IRQ22 and enable interrupts on the NVIC OC22.

Example code is shown here.

Figure 12 - Example code to prepare for interrupts from I2CB IDLE condition



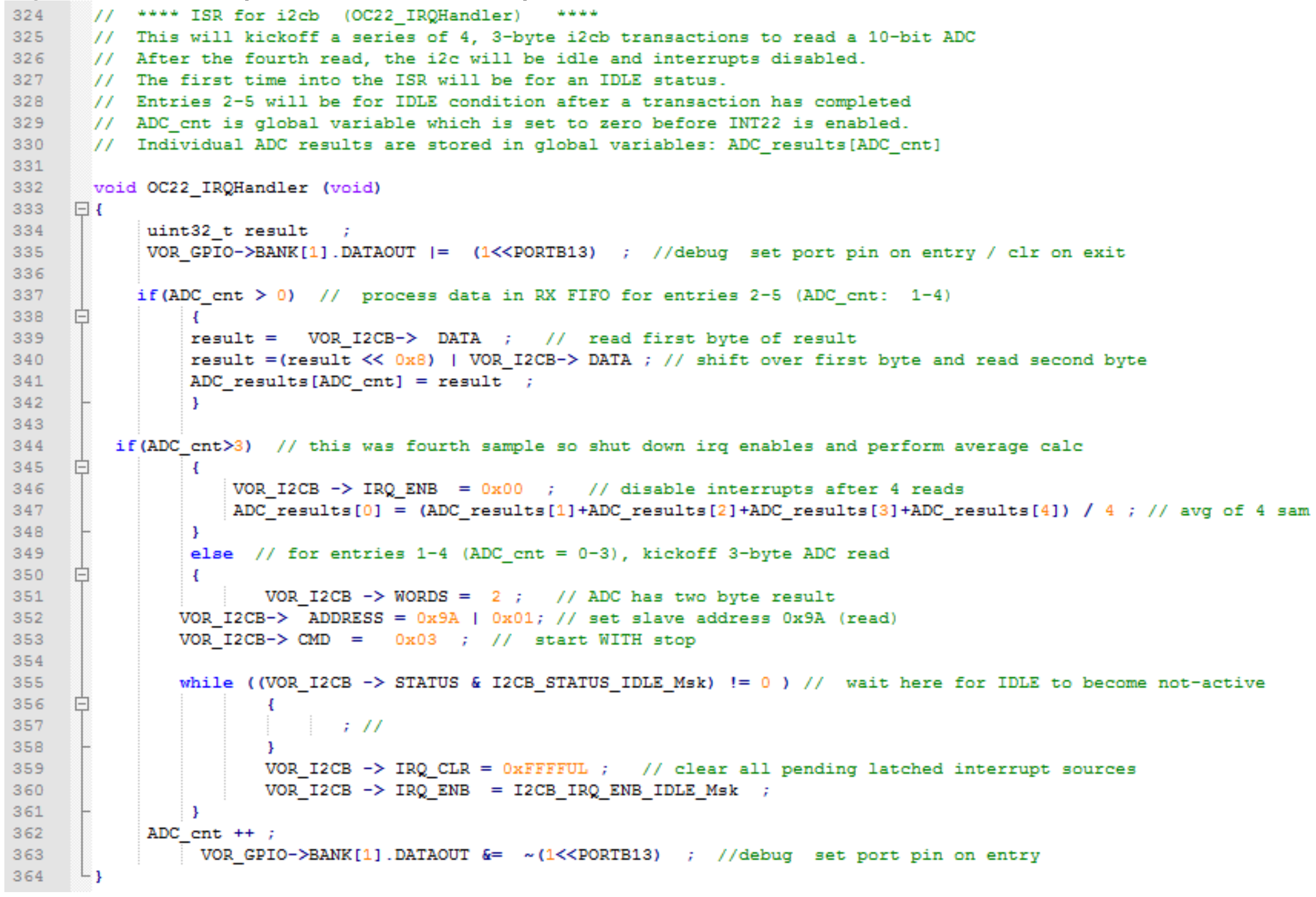
```
// ******* init_i2cb_IRQ22  (initialize system for interrupts on OC22) ******
//  steps:  1) Configure I2CB peripheral
//          2) Configure IRQ_SEL peripheral
//          3) Setup NVIC and enable interrupts on IRQ22.

uint32_t init_i2cb_IRQ22()       //  setup i2cb for to create interrupt when bus idle
{
  volatile  uint32_t  i  ;
  VOR_SYSCONFIG->PERIPHERAL_CLK_ENABLE |= CLK_ENABLE_I2CB ;

  VOR_I2CB -> CLKSCALE = 0x0018  ;    //  default normal mode, /24 for 50 MHz
  VOR_I2CB ->  FIFO_CLR = 0x3 ;  // clear both Rx and Tx Fifo
  VOR_I2CB -> CTRL  =  (I2CB_CTRL_DLGFILTER_Msk  |  I2CB_CTRL_ENABLE_Msk );
  VOR_I2CB -> IRQ_ENB  =  I2CB_STATUS_IDLE_Msk  ;  // generate int when IDLE = 1


  VOR_IRQSEL-> I2C_MS[1] = I2CB_INTn ;  //  IRQSEL routes i2cb int signal to NVIC_IRQ22

  NVIC_SetPriority(I2CB_INTn,I2CB_Priority);
  NVIC_EnableIRQ(I2CB_INTn);    // last thing to do is enable I2CB interrupts in NVIC

  return(VOR_I2CB -> STATUS)  ;
}
```

Steps to service the I^{2}C block in an ISR for a read operation

1. Read the contents of the DATA register to empty the FIFO
2. If another transaction is to follow:
    a. Write to WORDS, ADDRESS, DATA and CMD as required
    b. Clear the pending IDLE interrupt bit by writing a 1 to the corresponding bit in IRQ_CLR
3. If no more transactions are to follow:
    a. Optionally, disable all interrupts by writing a 0x0 to IRQ_ENB
    b. Optionally, the NVIC OC22 can be disabled.

Example code for reading a 10-bit ADC four times is shown here

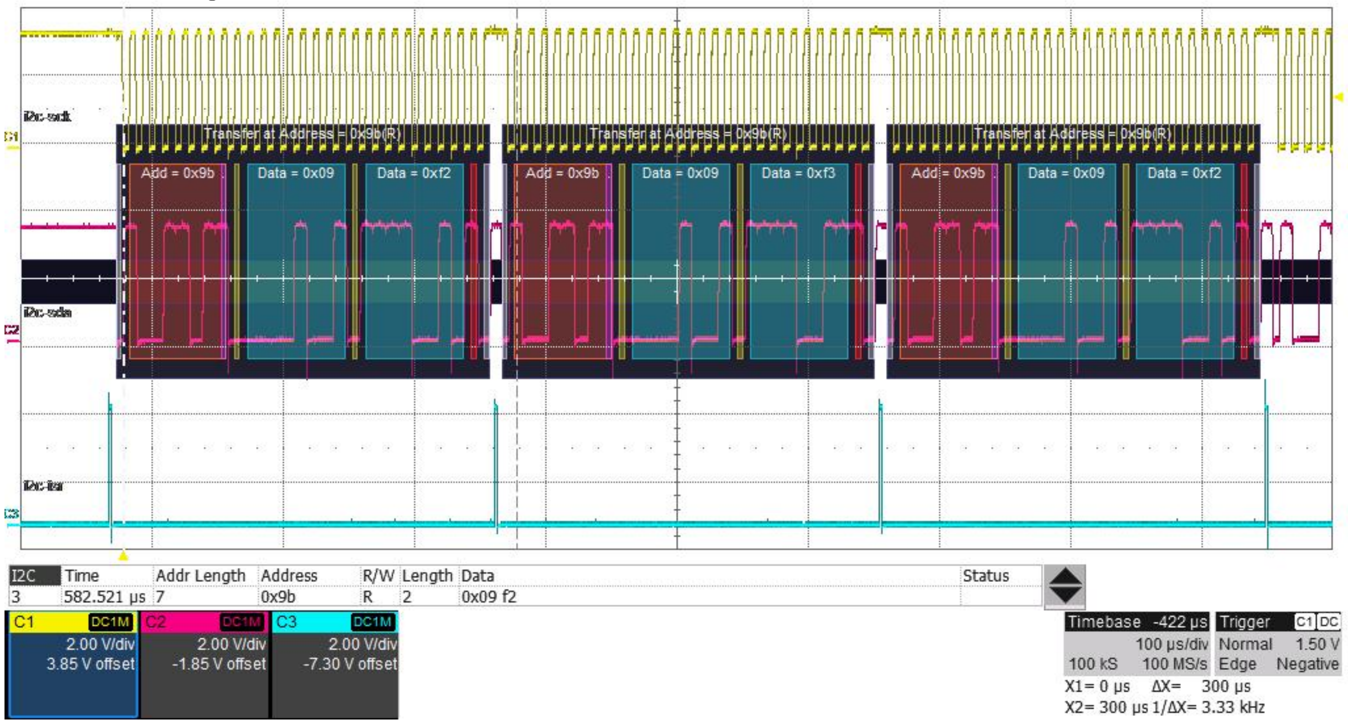
Figure 13 - Example code for Interrupt Service routine for an ADC on I^{2}C bus



```
//   **** ISR for i2cb  (OC22_IRQHandler)   ****
//   This will kickoff a series of 4, 3-byte i2cb transactions to read a 10-bit ADC
//   After the fourth read, the i2c will be idle and interrupts disabled.
//   The first time into the ISR will be for an IDLE status.
//   Entries 2-5 will be for IDLE condition after a transaction has completed
//   ADC_cnt is global variable which is set to zero before INT22 is enabled.
//   Individual ADC results are stored in global variables: ADC_results[ADC_cnt]

void OC22_IRQHandler (void)
{
       uint32_t result   ;
       VOR_GPIO->BANK[1].DATAOUT |=  (1<<PORTB13)  ;  //debug  set port pin on entry / clr on exit

      if(ADC_cnt > 0)  //  process data in RX FIFO for entries 2-5 (ADC_cnt:  1-4)
          {
          result =   VOR_I2CB->  DATA  ;   //  read first byte of result
          result =(result << 0x8) | VOR_I2CB-> DATA ; // shift over first byte and read second byte
          ADC_results[ADC_cnt] = result  ;
          }

   if(ADC_cnt>3)  // this was fourth sample so shut down irq enables and perform average calc
          {
                VOR_I2CB -> IRQ_ENB  = 0x00  ;   // disable interrupts after 4 reads
                ADC_results[0] = (ADC_results[1]+ADC_results[2]+ADC_results[3]+ADC_results[4]) / 4 ; // avg of 4 sam
          }
          else  // for entries 1-4 (ADC_cnt = 0-3), kickoff 3-byte ADC read
          {
                   VOR_I2CB -> WORDS =  2 ;   // ADC has two byte result
         VOR_I2CB->  ADDRESS = 0x9A | 0x01; // set slave address 0x9A (read)
         VOR_I2CB-> CMD  =   0x03  ;  //  start WITH stop

         while ((VOR_I2CB -> STATUS & I2CB_STATUS_IDLE_Msk) != 0 ) //  wait here for IDLE to become not-active
                   {
                          ; //
                   }
                   VOR_I2CB -> IRQ_CLR = 0xFFFFUL ;   // clear all pending latched interrupt sources
                   VOR_I2CB -> IRQ_ENB  = I2CB_IRQ_ENB_IDLE_Msk  ;
          }
       ADC_cnt ++ ;
            VOR_GPIO->BANK[1].DATAOUT &=  ~(1<<PORTB13)  ;  //debug  set port pin on entry
}
```

The waveform shown in Figure 14 shows 3 of the 4 ADC read transactions. The bottom trace shows the time the CPU is executing code in the ISR.

Figure 14 - Scope capture of 3 of 4 ADC read transactions. Bottom trace shows time spent in ISR servicing the I^{2}C.



I2C	Time	Addr Length	Address	R/W	Length	Data	Status
3	582.521 µs	7	0x9b	R	2	0x09 f2	

C1 DC1M	C2 DC1M	C3 DC1M
2.00 V/div	2.00 V/div	2.00 V/div
3.85 V offset	-1.85 V offset	-7.30 V offset

```
Timebase  -422 µs   Trigger   C1 DC
          100 µs/div Normal    1.50 V
100 kS    100 MS/s  Edge      Negative
X1= 0 µs    ΔX=    300 µs
X2= 300 µs 1/ΔX= 3.33 kHz
```

4 Conclusions

The I^{2}C block has many options and a 16-word FIFO that makes it very flexible and capable of very efficient transaction management for a wide variety of I^{2}C peripherals. For simple I^{2}C devices like a port expander, using a polled method may be perfectly fine. However, for I^{2}C memory devices that require lots of data to be transferred, making use of the FIFO and interrupt features is highly recommended.

This application note has provided several example operations for reading and writing to different I^{2}C devices using both polling and interrupt driven methods. You should be able to quickly adapt one of these examples to interface to the peripheral you are using.

5 Common questions and issues

1. The MCU successfully sends a device address byte but the slave device does not acknowledge. What can be going on?

   a. Most likely the wrong device address is being sent. Check the slave device's address which can sometimes be configured via pins on the device.
   b. Other possibilities include the SCL and SDA pins being swapped or the slave device not being powered.
2. The slave device responds with an acknowledge but not all transactions are completed successfully.
   a. Check the integrity of the data line (SDA). Sometimes the pull-up resistor is not properly sized or there is too much board capacitance.
   b. Other master nodes may be active when a transaction starts. Before transmitting, check that the bus is idle.
3. The ISR for an IDLE condition immediately calls itself time after time even though the IDLE flag in the STATUS register is not set. What is going on?
   a. The IDLE bit in IRQ_RAW is a latched value. The last time the STATUS:IDLE bit changed from a 0 to 1, the IRQ_RAW:IDLE bit was set and stays set until the IRQ_CLR:IDLE bit has a 1 written to it. Before leaving the ISR, write a 1 to the IRQ_CLR:IDLE bit.

6 Other Resources

VORAGO VA108x0 programmers guide:
http://www.voragotech.com/sites/default/files/VA10800_VA10820_PG_July2016revision1.16%5B4%5D.pdf

VORAGO MCU products: http://www.voragotech.com/VORAGO-products

VORAGO Application notes: http://www.voragotech.com/resources

VORAGO VA108xx REB1board user guide: Part of Board Support Package (BSP)
http://www.voragotech.com/products/reb1

I^{2}C Specification: http://www.nxp.com/documents/user_manual/UM10204.pdf

Revision log:

March 27, 2017 – Revision 1.1
- Print font set to Futura
- Corrected spelling of Philips
- Added Table of Contents