Sensor Interfacing and Preprocessing

Reference Design

FPGA-RD-02048 Version 1.3

September 2018

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
ACK	Acknowledge bit sent from RX side to TX side to indicate the received data parity check is OK
RX	Receiver
TX	Transmitter
I^2C	Inter-Integrated Circuit

1. Introduction

The iCE40 Sensor Hub Solution is a low power sensor hub solution for mobile devices using iCE40LM, iCE40 Ultra™, and iCE40™ UltraPlus FPGAs. It is designed to monitor sensors and periodically send the sensor data to the application processor. The Sensor Hub reference design acts as a buffer between the sensors and the application processor. When the Sensor Hub is used in a design, it allows the application processor to sleep for longer periods of time. The Sensor Hub reduces unnecessary communication between the sensors and the processor, thus saving power consumption by allowing the processor not to be in an *always on* state. The iCE40 Sensor Hub Solution is configurable, available as either standalone off the shelf or fully customizable, making it a sensor agnostic solution.

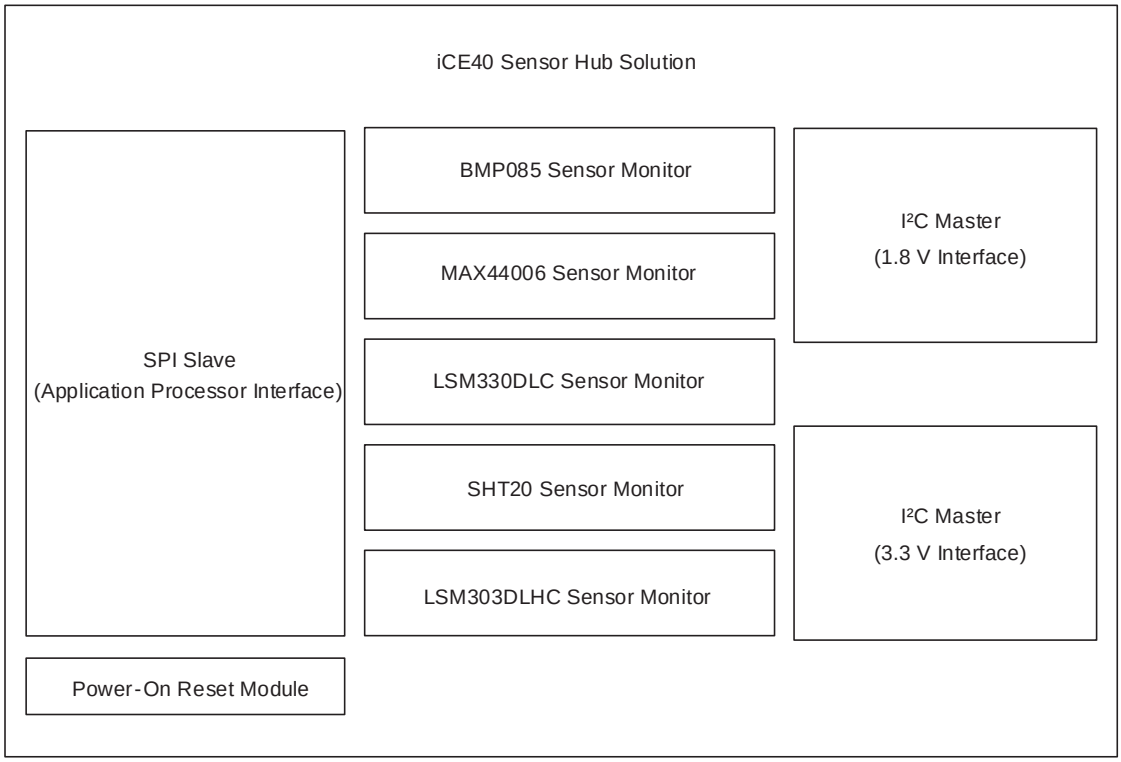


Figure 1.1. System Block Diagram

As a standalone solution, the iCE40 Sensor Hub connects to the application processor's Serial Peripheral Interface Bus (SPI) with clock frequency set to 10.8 MHz. This enables a fast communication speed to/from the processor. The iCE40 Sensor Hub standalone solution acts as I2C master to the supported sensors. There are two sets of I²C connections configured with multiple sensors at each connection in the iCE40 Sensor Hub standalone solution.

The default sensors supported in the standalone iCE40 Sensor Hub Solution are: Bosch BMP085 Digital Pressure Sensor; Maxim Integrated MAX44006 RGB Color, Infrared, and Temperature Sensors; Sensirion SHT20 Humidity and Temperature Sensor IC; STMicro LSM330DLC 3D accelerometer and 3D gyroscope; and STMicro LSM303DLHC 3D accelerometer and 3D magnetometer module.

The iCE40 Sensor Hub standalone solution has a system operating frequency of 27 MHz, SPI bus frequency to application processor of 10.8 MHz, and I2C clock frequency of 400 kHz. The SPI bus is configured to have a volt-age of 1.8 V, and the two I2C buses are configured to have voltages of 1.8 V and 3.3 V, enabling the solution to connect to sensors with different I/O voltages and bridging sensors of different I/O voltages to the application processor.

The fully customizable solution capability of this solution is due to its FPGA based architecture. This capability is ideal, but not limited to users who would like to include/remove supported sensors, change the data acquisition FIFO depth, create an I/O bridge between sensors and processor, or customize sensor monitoring time period.

1.1. Features List

- Configurable Sensor Hub for Mobile Devices
  - Sensor Agnostic Solution
  - Default System frequency of 27 MHz
  - Power-On Reset capability
- Serial Peripheral Interface (SPI) Bus connection to Application Processor with the following Default settings:
  - Interface frequency of 10.8 MHz
  - Interface voltage of 1.8 V
  - Solution is a *slave* of the Application Processor
  - SPI slave mode CPOL = 1 and CPHA = 1 (mode *3*)
  - SPI slave features MSB first
- One I²C Bus with default interface voltage of 1.8 V at 400 kHz for:
  - Bosch BMP085 Digital Pressure Sensor
  - Maxim Integrated MAX44006 RGB Color, Infrared, and Temperature Sensors
  - STMicro LSM330DLC 3D accelerometer and 3D gyroscope
- One I²C Bus with default interface voltage of 3.3 V at 400 kHz for:
  - Sensirion SHT20 Humidity and Temperature Sensor IC
  - STMicro LSM303DLHC 3D accelerometer and 3D magnetometer module

1.2. Applications

- Notebook PCs
- Smart Phones
- Tablets
- Handheld Gaming Units
- GPS Units
- Digital Cameras

1.3. Block Diagram

Figure 1.2 shows a block level diagram of the iCE40 Sensor Hub Solution.

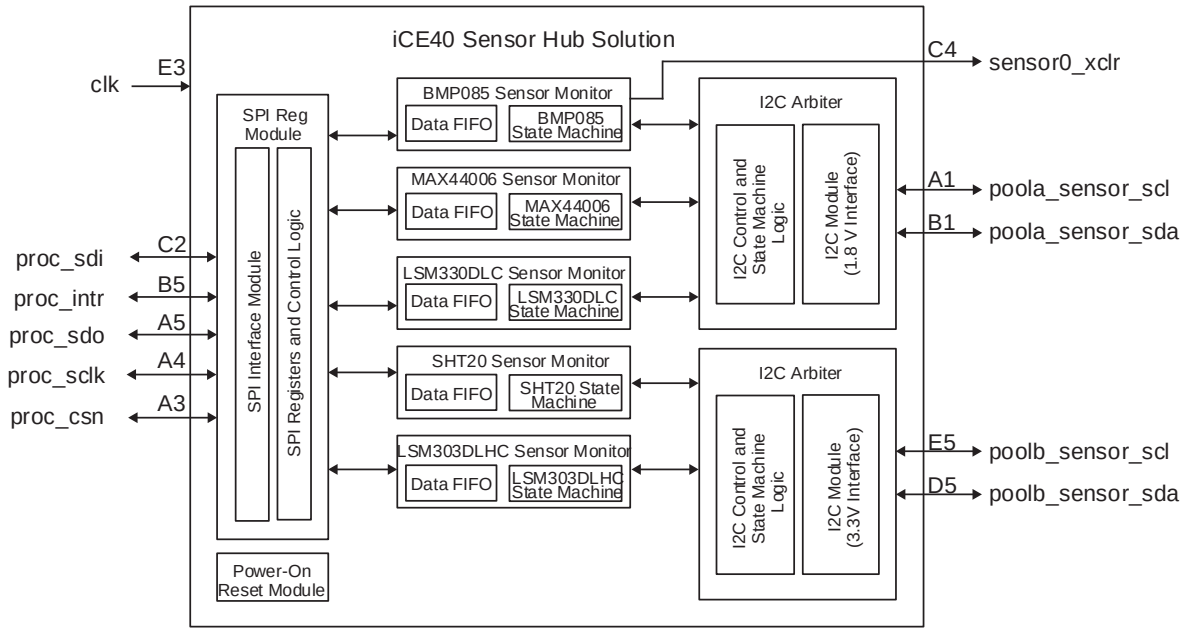


Figure 1.2. Functional Block Diagram

2. Related Documentation

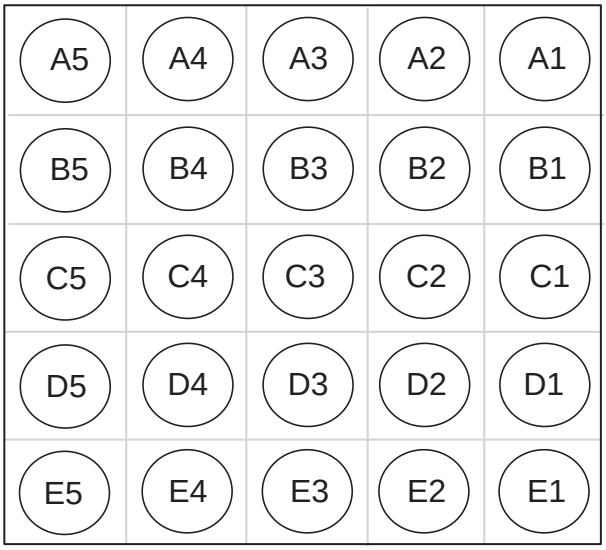
In addition to using this guide to help you get started developing iCE40 Low Power Sensor Hub Solution on your device, you can refer to other applicable documents that may contain more detailed information that is beyond the scope of this guide.

The following documents can be obtained on the Lattice website:

- iCE40LM Family Data Sheet (FPGA-DS-02043)
- iCE40 Ultra Family Data Sheet (FPGA-DS-02028)
- iCE40 UltraPlus Family Data Sheet (FPGA-DS-02008)
- iCEcube2 User Guide - The iCE40LM and iCE40 Ultra versions of this reference design are designed in iCEcube2 software and this user guide explains everything you need to know about iCEcube2.
- Lattice Radiant Software User Guide – The iCE40 UltraPlus version of this reference design is designed in Lattice Radiant software and this user guide explains everything you need to know about Lattice Radiant.

3. Pin Configuration and Function Descriptions

Figure 3.1 shows the bottom view of the iCE40LM4K-SWG25TR.



Note: Assumes operating under off-the-shelf standalone solution. This may vary depending on the device family and package type used.

Figure 3.1. Bottom View of iCE40LM4K-SWG25TR (Balls Up)

Table 3.1. Pin Function Description – iCE40LM*

Pad name	Port Name	Port Direction	Description
A1	poola_sensor_scl	Inout	I²C Interface SCL for sensors with 1.8 V interface
A2	VCCIOVB1	Input	I/O Power Supply
A3	proc_csn	Input	SPI bus slave select (Active Low)
A4	proc_sclk	Input	SPI bus serial clock
A5	proc_sdo	Input	SPI bus serial data in to slave
B1	poola_sensor_sda	Inout	I²C Interface SDA for sensors with 1.8 V interface
B2	GND	Input	Ground
B3	CRESET	Input	Configuration Reset (Active Low). See data sheets.
B4	VCC	Input	Core Power Supply
B5	proc_intr	Output	Processor Interrupt to Application Processor
C1	ice_SI	Output	Configuration Output to external SPI Memory
C2	proc_sdi	Output	SPI bus serial data out from slave
C3	CDONE	Output	Configuration Done. See data sheets.
C4	sensor0_xclr	Output	Master Clear for Pressure Sensor (Bosch BMP085)
C5	General Purpose I/O	Input/Output	3.3 V I/O for user interface
D1	flsh_sclk	Input	Configuration Clock
D2	ice_SO	Input	Configuration Input from external SPI Memory
D3	General Purpose I/O	Input/Output	3.3 V I/O for user interface
D4	GND	Input	Ground
D5	poolb_sensor_sda	Inout	I²C Interface SDA for sensors with 3.3 V interface
E1	flsh_cs	Input	Configuration Chip Select (Active Low)

Pad name	Port Name	Port Direction	Description
E2	VCCIOVB2	Input	I/O Power Supply
E3	clk	Input	System Clock
E4	General Purpose I/O	Input/Output	3.3 V I/O for user interface
E5	poolb_sensor_scl	Inout	I^2C Interface SCL for sensors with 3.3 V interface

***Note:** Assumes operating under *off-the-shelf standalone* solution. Pad Names may vary depending on the device family and package type used in this customizable solution.

4. Theory of Operations

The iCE40 Sensor Hub Solution monitors the sensors attached to its I²C buses. It monitors each sensor periodically for new values. Each sensor monitor can be configured separately. Each sensor monitor block has a dedicated FIFO to store the read values of corresponding sensors. After sensor data has been read, the sensor monitor sends an interrupt to the application request to indicate that sensor data is present. The application processor is then expected to read the FIFO contents through the SPI bus. Once the sensor data has been read, the application processor needs to clear both the FIFO and the interrupt.

The default supported sensors in the iCE40 Sensor Hub Solution are divided into two groups: those with 1.8 V interface and those with 3.3 V interface. The default supported sensors are as follow:

- 1.8 V interface voltage:
  - Bosch BMP085 Digital Pressure Sensor
  - Maxim Integrated MAX44006 RGB Color, Infrared, and Temperature Sensors
  - STMicro LSM330DLC 3D accelerometer and 3D gyroscope
- 3.3 V interface voltage:
  - Sensirion SHT20 Humidity and Temperature Sensor IC
  - STMicro LSM303DLHC 3D accelerometer and 3D magnetometer

4.1. Functional Descriptions

This sub-section describes the function of each sub-block in inside the iCE40 Sensor Hub Solution. Many of these blocks have HDL module associated with them.

4.1.1. Sensor Monitor Top Level

The Sensor Monitor Top Level is found in sensor_hub. This module contains submodules for all Sensor Monitors, the two I2C interfaces, and SPI interface to/from application processor. It also contains a Power-On Reset (POR) module. The POR module initiates a system reset upon power up for Tpor number of cycles. The iCE40 Sensor Hub Solution operates after system reset has been completed.

4.1.2. BMP085 Sensor Monitor

The BMP085 Sensor Monitor monitors Bosch BMP085 Digital Pressure Sensor, and it is found in BMP085_prsr. This module begins by calibrating the BMP085 sensor by reading the pre-defined values in the calibration matrix of the BMP085 sensor. Once calibration is completed, the BMP085 sensor monitor periodically reads the temperature and pressure values from the BMP085 sensor. As a standalone off the shelf solution the number of cycles with respect to the system clock at which this module reads the BMP085 sensor value is Tsampbmp085. For the fully customizable solution, you can change the number of cycles by changing the INTR_THRESHOLD parameter value. All values read from BMP085 are stored in a FIFO that's accessible by the application processor through the SPI Interface to Application Processor module. Once sensor data has been read, this module issues an interrupt for the Application Processor to indicate that data is present. When the Application Processor gets the interrupt, it first reads the BMP085 status register bit to determine whether the data available in the FIFO is calibration data or sensor data, which is used appropriately for calculations. After the Application Processor receives the data, it is expected that the interrupt and the FIFO is cleared by the Application Processor.

At the end of each acquisition this module issues XCLR signal to reset the sensor.

Figure 4.1 shows the BMP085 pressure sensor.

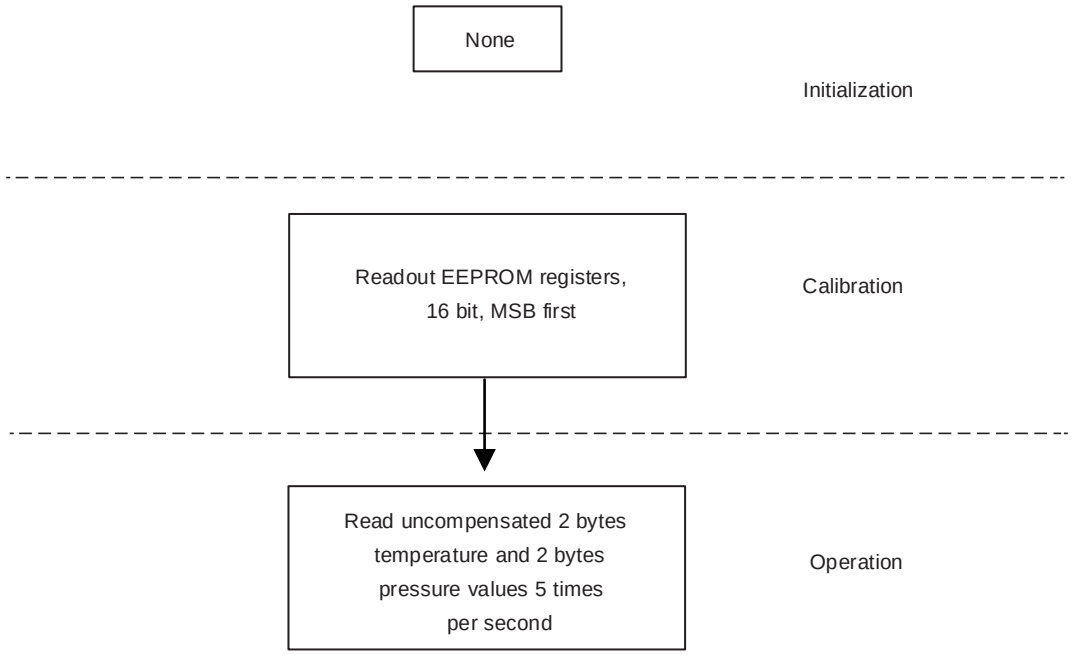


Figure 4.1. BMP085 Pressure Sensor

4.1.3. LSM303DLHC Sensor Monitor

The LSM303DLHC Sensor Monitor monitors STMicro LSM303DLHC's 3D magnetometer module, and it is found in LSM303_magneto. This module starts by initializing the LSM303DLHC by writing to CRA, CRB, and MR registers of the LSM303DLHC device to set the operation mode of the magnetometer. Once initialization is completed, the LSM303DLHC sensor monitor will periodically read the magnetic value from LSM303DLHC. As a standalone off the shelf solution the number of cycles with respect to the system clock at which this module reads the LSM303DLHC sensor value is Tsamplsm303. For the fully customizable solution, you can change the number of cycles by changing the INTR_THRESHOLD parameter value. All magnetic values read are stored in a FIFO that's accessible by the application processor through the SPI Interface to Application Processor module. Once sensor data has been read, this module issues an interrupt for the Application Processor to indicate that data is present. After the Application Processor has received the data, the interrupt and the FIFO is cleared by the Application Processor.

Figure 4.2 shows the LSM303 compass.

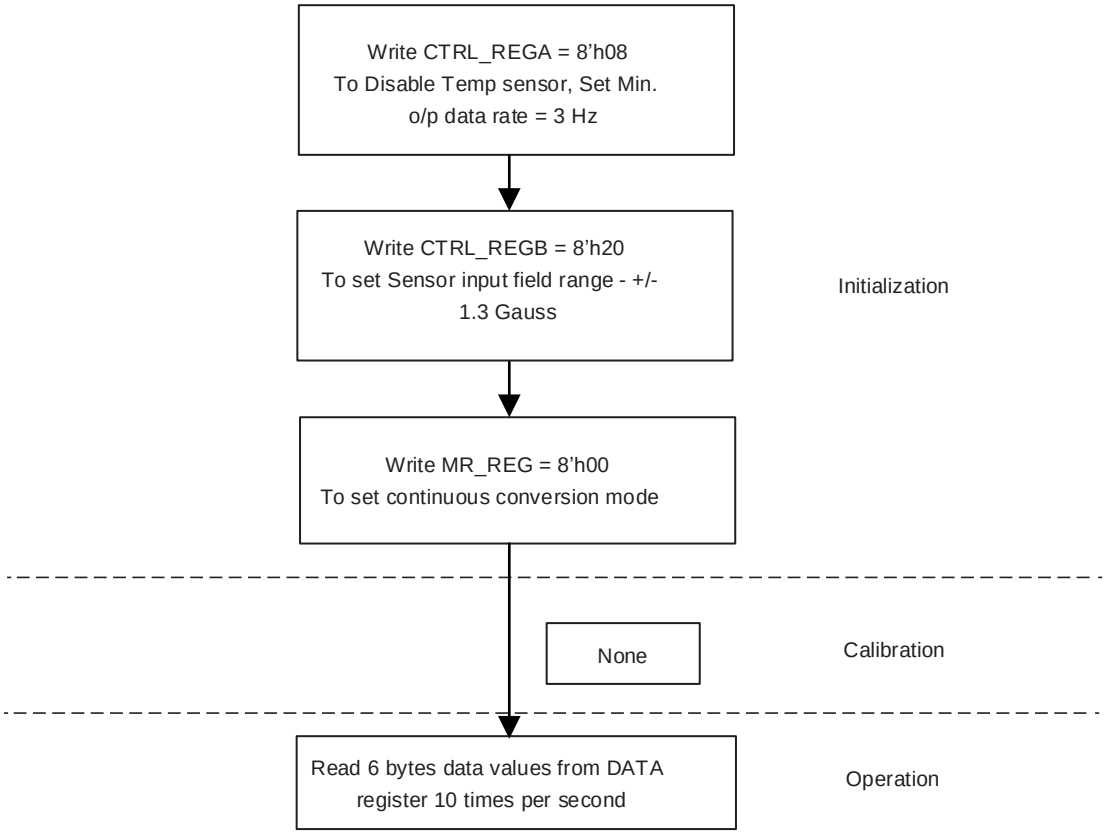


Figure 4.2. LSM303 Compass

4.1.4. LSM330DLC Accelerator Sensor Monitor

The LSM330DLC Accelerator Sensor Monitor monitors only the accelerometer portion of STMicro LSM330DLC 3D accelerometer and 3D gyroscope module, and it is found in LSM330DLC_accel. This module starts by initializing the LSM330DLC by writing to Control REG1A and Control REG4A registers of the LSM330DLC device to set the operation mode of the accelerometer. Once initialization is completed, the LSM330DLC sensor monitor periodically reads the accelerometer value from LSM330DLC. As a standalone off the shelf solution the number of cycles with respect to the system clock at which this module reads the LSM330DLC device is Tsamplsm330a. For the fully customizable solution, you can change the number of cycles by changing the INTR_THRESHOLD parameter value. All acceleration values read are stored in a FIFO that's accessible by the application processor through the SPI Interface to Application Processor module. Once sensor data has been read, this module issues an interrupt for the Application Processor to indicate that data is present. After the Application Processor has received the data, the interrupt and the FIFO is cleared by the Application Processor.

Figure 4.3 shows the LSM330DLC accelerometer.

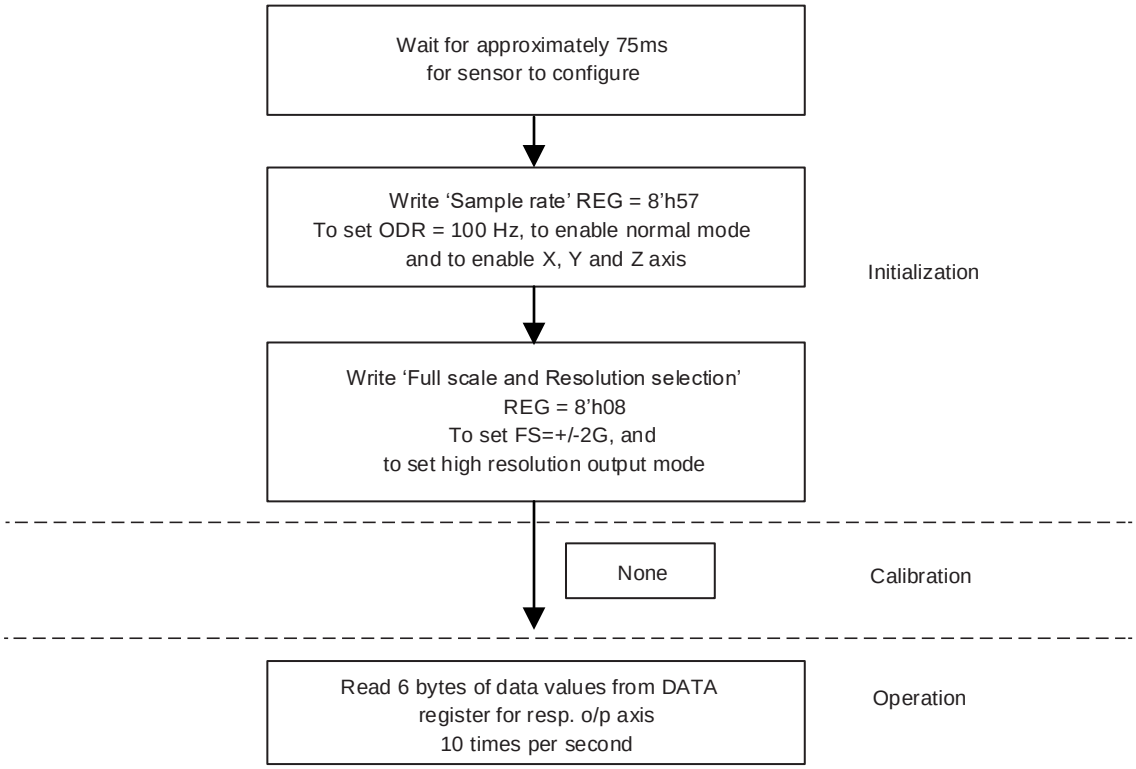


Figure 4.3. LSM330DLC Accelerometer

4.1.5. MAX44006 Sensor Monitor

The MAX44006 Sensor monitors the Maxim Integrated MAX44006 RGB Color, Infrared, and Temperature Sensors, and it is found in Max44006_als. This module starts by initializing the MAX44006 by confirming the Power-On State of the device; and setting up the UPR_THRM, UPR_THRL, LWR_THRM, LWR_THRL, PERSIST_TIMER, AMBIENT configuration, and sensor mode of the device. These sets the operational modes of the sensor, including the RGB interrupt thresholds, RGB gain, and ADC conversion timing. Once initialization is completed, the MAX44006 sensor monitor periodically reads the color, infrared, and temperature values from MAX44006. As a standalone off the shelf solution the number of cycles with respect to the system clock at which this module reads the MAX44006 device is Tsampmax44006. For the fully customizable solution, you can change the number of cycles by changing the INTR_THRESHOLD parameter value. All values read are stored in a FIFO that's accessible by the application processor through the SPI Interface to Application Processor module. Once sensor data has been read, this module issues an interrupt for the Application Processor to indicate that data is present. After the Application Processor has received the data, the interrupt and the FIFO is cleared by the Application Processor.

Figure 4.4 shows the MAX44006 ambient light sensor.

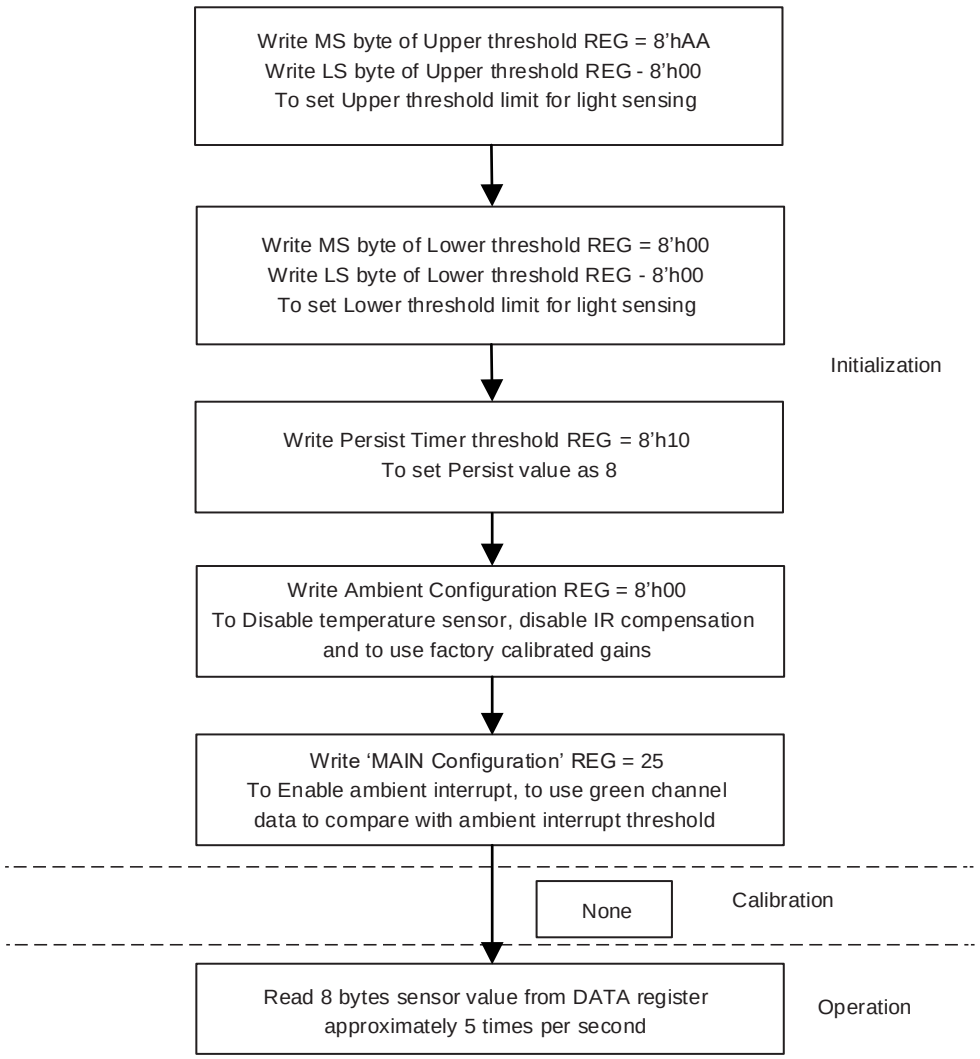


Figure 4.4. MAX44006 Ambient Light Sensor

4.1.6. SHT20 Humidity Sensor

The SHT20 Sensor Monitor monitors only the humidity sensor of the Sensirion SHT20 Humidity and Temperature Sensor IC, and it is found in SHT20_humidity. This module starts by initializing the SHT20 by resetting the sensor, and reading and writing the user register of the SHT20 device. These set the operation mode of the SHT20 device. Once initialization is completed, the SHT20 sensor monitor will periodically read the humidity value from SHT20. As a standalone off the shelf solution the number of cycles with respect to the system clock at which this module reads the SHT20 device is Tsampsht20. For the fully customizable solution, you can change the number of cycles by changing the INTR_THRESHOLD parameter value. All humidity values read are stored in a FIFO that's accessible by the application processor through the SPI Interface to Application Processor module. Once sensor data has been read, this module issues an interrupt for the Application Processor to indicate that data is present. After the Application Processor has received the data, the interrupt and the FIFO is cleared by the Application Processor.

Figure 4.5 shows the SHT20 humidity sensor.

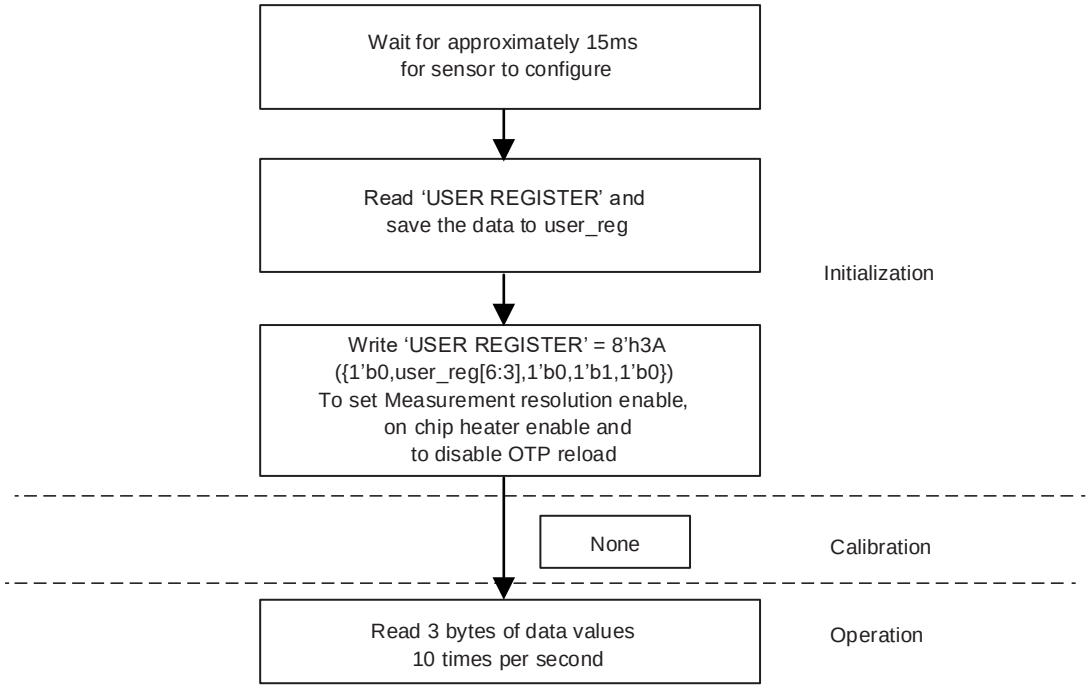


Figure 4.5. SHT20 Humidity Sensor

4.1.7. LSM330DLC Gyro Sensor Monitor

The LSM330DLC Sensor Monitor monitors only the gyroscope portion of STMicro LSM330DLC 3D accelerometer and 3D gyroscope module, and it is found in LSM330DLC_gyro. This module starts by initializing the LSM330DLC by writing to Control REG1G register of the LSM330DLC device to enable the output data rate and to enable normal power mode for gyroscope operation. Once initialization is completed, the LSM330DLC gyro sensor monitor periodically reads the gyroscope value from LSM330DLC. As a standalone off the shelf solution the number of cycles with respect to the system clock at which this module reads the LSM330DLC gyro sensor value is Tsamplsm330g. For the fully customizable solution, you can change the number of cycles by changing the INTR_THRESHOLD parameter value. All gyroscope values read are stored in a FIFO that's accessible by the application processor through the SPI Interface to Application Processor module. Once sensor data has been read, this module issues an interrupt for the Application Processor to indicate that data is present. After the Application Processor has received the data, the interrupt and the FIFO is cleared by the Application Processor.

Figure 4.6 shows the LSM330DLC gyroscope.

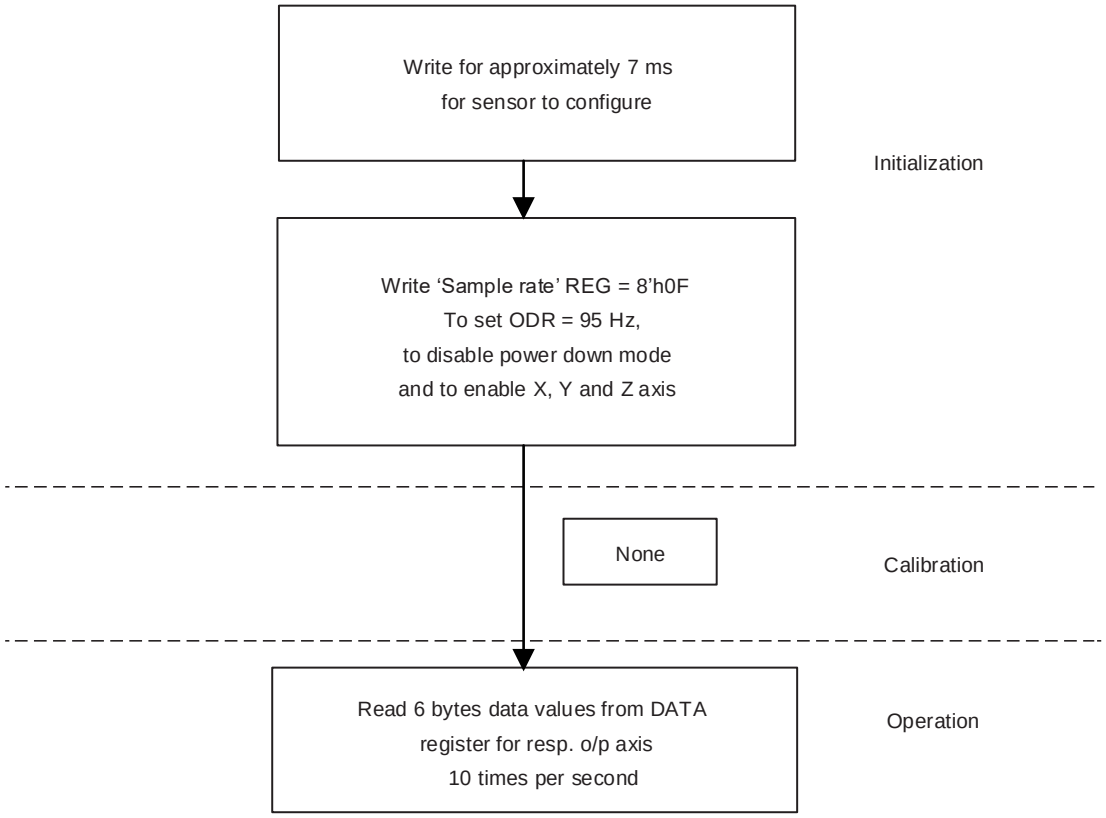


Figure 4.6. LSM330DLC Gyroscope

4.1.8. I²C Arbiter

The I²C Arbiter module muxes/demuxes the control signals to/from the I2C master to and from all sensor monitors. It goes through all sensor monitors one-by-one in a round-robin fashion to see whether a sensor data read request is present. When a sensor monitor is not available, it goes to the next sensor and so on. This module can be found in i2c_arbiter, and there are two instances of this module: one for the 1.8 V interface, and another for the 3.3 V interface. This module sends and receives command from each sensor monitor so that data can be sent/received through the I²C interface. The I²C arbiter also contains the control for I²C interface (found in i2c_reg_ctrl).

4.1.9. SPI Interface to Application Processor

This module is used to interface between the iCE40 Sensor Hub Solution and the application processor. It is found in spi_reg module. This module contains the SPI registers VERSION, ISR, CNTRL, and STATUS.

This module starts by initializing the SPI Control Register 1 and waits until an interrupt or read from a sensor command is received from the processor. When a read command is received, this module goes to read from RXDR followed by byte write. It then determines if the MISO byte is valid followed by write to TXDR command. Once these are completed, it goes back to an idle state. The state machine is in spi_slave module.

When a read command is received from the application processor, this module sends commands to the desired sensor so that the stored sensor values in the FIFOs are read out. This module then receives the stored sensor values and sends them to the application processor. Once the data has been read, the application processor will need to clear the FIFO and the interrupt register for the corresponding sensor.

When a write command is received from the application processor, this module decodes the commands and sends the appropriate data to the desired register locations.

Finally, this module also contains an interrupt arbiter (intr_arb) module that polls interrupt (that is to indicate that sensor data is present) from the sensor monitors. This module ensures that the processor is interrupted by only one of the sensor monitors at a time and the interrupts by other sensor monitor during that time are latched and presented one after the other. Once the interrupt is recognized and served by the processor, the interrupt arbiter module will acknowledge the sensor monitor that the interrupt has been received.

4.2. Block Descriptions

The purpose of this section is to provide detailed descriptions of each block of the iCE40 Sensor Hub to assist users who want to use this solution using alternative sensors.

4.2.1. Top Level Module (sensor_hub)

This module contains all of the sensor monitors, SPI interface to the application processor, and two sets of I²C interface to the sensors.

The HDL code begins with the POR logic, which is set to Tpor cycles.

All the sensor monitors are then instantiated with their respective parameters set. In addition to system clock and system reset, each sensor monitor has connections to/from one of the I²C interfaces and the SPI interface. Notice that each wire connection has the same name prefix dev#, where # is an integer, and name suffixes that corresponds to the module port name. Each wire connection is connected to SPI ports with corresponding dev# and name suffix. On the other hand, the I²C wire connections may not have the same sensor monitor dev# (although, each sensor monitor's dev# set must be connected to the same I²C interface port's dev# set), but will have the same name suffix. There's only one I²C and one SPI connection for each sensor.

The code then instantiates two i2c_arbiter modules, which contain the I²C interface. Each module has connection to the sensor monitors, and I²C pins of the solution. Not all ports of the i2c_arbiter may be connected to a sensor monitor, thus giving room for sensor expansion. The connection rule between the I_2C interface and the sensor monitor as described above applies.

Finally, the code instantiates the spi_reg module, which contains the SPI interface to the application processor. This module has connection to the sensor monitors. All sensor monitor need to be connected to this module. The connection rule between the SPI interface and the sensor monitor as described above applies.

4.2.2. Sensor Monitors (BMP085, LSM303, MAX44006, SHT20, LSM330DLC Accelerometer and Gyroscope)

Each sensor monitor has the same general functions and may include operations specific to the target sensor. Each module initializes the sensor, reads the sensor data periodically via 12C interface, stores the sensor read data, and sends interrupt to the application processor via SPI interface to indicate that sensor data is present.

The following table summarizes the ports to/from the sensor monitors:

Table 4.1. Ports To/From the Sensor Monitor

Signal		Direction	Description
For BMP085, LSM303, MAX44006, SHT20	For LSM330DLC accelerometer and gyroscope		
fifo_data[7:0]	o_fifo_data[7:0]	Output	Contains stored sensor read data (from the data FIFO). This is the data to be sent to the Application Processor through the SPI interface.
fifo_clr_ack	o_fifo_clr_ack	Output	Signal for the SPI interface logic to indicate that the data FIFO has been cleared successfully (Active HIGH).
intr	o_intr	Output	Signal for the SPI interface logic to indicate that sensor read data is present for Application Processor to access (Active HIGH).
calibration	o_calibration	Output	Signal for the SPI interface logic to indicate that the sensor monitor is performing calibration operation on the sensor (Active HIGH).
on_ack	o_on_ack	Output	Signal for the SPI interface logic to indicate that DEV_ON has been successfully processed (Not implemented; it is tied to logic HIGH).
off_ack	o_off_ack	Output	Signal for the SPI interface logic to indicate that DEV_OFF has been successfully processed (Not implemented; it is tied to logic HIGH).
underflow	o_underflow	Output	Signal for the SPI interface logic to indicate that FIFO under- flow has occurred (Not implemented; it is tied to logic LOW).
overflow	o_overflow	Output	Signal for the SPI interface logic to indicate that FIFO overflow has occurred (Not implemented; it is tied to logic LOW).
active	o_active	Output	Signal for the SPI interface logic to indicate that the sensor monitor is performing a function (e.g. reading sensor data) (Active HIGH).
full	o_full	Output	Signal for the SPI interface logic to indicate that the data FIFO is full (Active HIGH).
empty	o_empty	Output	Signal for the SPI interface logic to indicate that the data FIFO is empty (Active HIGH).
xclr	—	Output	XCLR signal specifically for BMP085 device (Active HIGH)
clk	i_sys_clk	Input	System Clock
rst	i_sys_rst	Input	System Reset
fifo_rden	i_fifo_rden	Input	Signal from SPI interface logic to read the data FIFO for the sensor data (Active HIGH).
fifo_clr	i_fifo_clr	Input	Signal from SPI interface logic to clear the data FIFO after the data has been read by SPI interface logic (Active HIGH).
inte	i_inte	Input	Signal from SPI interface logic to enable interrupt for the sensor - NOT IMPLEMENTED IN SENSOR MONITOR.
intr_ack	i_intr_ack	Input	Signal from SPI interface logic to indicate that *intr* has been received and to deassert the *intr* in the sensor monitor (Active HIGH).
on	i_on	Input	Signal from SPI interface logic to control DEV_ON - NOT IMPLEMENTED IN SENSOR MONITOR.
off	i_off	Input	Signal from SPI interface logic to control DEV_OFF - NOT IMPLEMENTED IN SENSOR MONITOR.
i2c_start	o_i2c_start	Output	Signal to the I^2C interface logic to indicate to access the sensor (Active HIGH).
read_write_n	o_read_write_n	Output	Signal to the I^2C interface logic to indicate whether the accessed command is a read or a write (Read = HIGH, Write = LOW).
slave_address[7:0]	o_slave_address[7:0]	Output	Signal to the I^2C interface logic to indicate the slave device (sensor) address.
read_byte_count[7:0]	o_read_byte_count[7:0]	Output	Signal to the I^2C interface logic to indicate the number of bytes to read.
reg_address[7:0]	o_reg_address[7:0]	Output	Signal to the I^2C interface logic to indicate the slave device's

Signal		Direction	Description
For BMP085, LSM303, MAX44006, SHT20	For LSM330DLC accelerometer and gyroscope		
			(sensor) register's address.
fwrite_data[7:0]	o_write_data[7:0]	Output	Contains the data to write to the slave device (sensor) through the I^2C interface logic.
read_data[7:0]	i_read_data[7:0]	Input	Contains the data read from the slave device (sensor) through the I^2C interface logic.
read_data_valid	i_read_data_valid	Input	Indicates whether the read_data is valid (Active HIGH).
i2c_done	i_i2c_done	Input	Indicates whether the I^2C transaction has been completed (Active HIGH).

Note: Although the port names for LSM330DLC Accelerometer and Gyroscope are different from the other sensors, functionally the two set of port names are the same.

The following is a walkthrough of a sensor monitor code, specifically for SHT20 sensor. Codes in this section are taken directly from the HDL file. Note that in most cases, the topics in each paragraph below are presented in the order in which they appear in the HDL code.

The first action in the code is to define two parameters: INTR_THRESHOLD and INIT_THRESHOLD. The former parameter defines the number of clock cycles (with respect to the system clock) at which the sensor monitor will periodically read the data from the target sensor. The latter parameter defines the number of clock cycles (with respect to the system clock) at which the sensor monitor need to wait before issuing any operation so that the sensor operates properly (that is by initialization time). To calculate the time value, simply multiply the values of each parameter with the clock period.

The counters for the above thresholds are written under the following comments:

- //init_time
- *This interrupt is generated for every
- *Counter

Note: The code under the second comment above generates the interrupt to trigger read data from sensor (sensor_int).

The second action is to define the slave address of the sensor to be monitored. This is performed by the following code:

```
/*
* Slave address of the sensor device
*/
assign slave_address = 8'h40; // Example Slave Address Setting Code
```

It is important to use the wire name *slave_address* to define the slave address. Note that the slave address value depends on the desired sensor, and it is an 8-bit value.

A state machine is then defined in the HDL. The state machine contains sensor initialization procedures. Once initialization has been completed, the state machine moves to loopback between waiting and reading sensor states. The state machine is written under the following comments:

```
/*
* State machine to control sensor configuration and data acquisition
*/
```

The states of the state machine are then used to control various logic used to control the following wires:

- i2c_start: This net signals the I_2C interface to access the sensor.
- read_write_n: This net signals tells the I_2C interface whether the I_2C operation is a read (1) or write (0)
- read_byte_count: This bus determines the number of byte to be read by the I_2C operation
- reg_address: This bus determines the slave device's (sensor) register to be accessed by the I_2C operation
- write_data: This bus contains data to be written in to the slave device (sensor) through the I_2C operation.

The code location of the each of the above logic can be found under the comment *//<WIRE_NAME>*. Note that additional sensor specific logic for I2C transactions are implemented around this area. For example, SHT20 has *user_reg_data* and *word_count* logic.

A code for interrupt to the application processor is then written. This interrupt signals the processor that sensor read data is present for reading by the processor; it is found in *intr* and it is active HIGH. The code is located under the comment *//intr*.

Calibration for sensor logic is implemented. Not all sensors require calibration, and in the case of SHT20, it is not present. The code is under the comment *//calibration*.

Logic to signal the SPI interface logic that the sensor is busy (such as performing data acquisition) is implemented under the comment *//active*. The signal name is *active*, and it is active HIGH.

The remaining set of codes involves data FIFO and its control.

The FIFO is 64 x 8-bit, and it is instantiated using the *afifo_8* module, which has the following ports:

Table 4.2. afifo_8 Module Ports

Signal	Direction	Description
rst	Input	Reset
wren	Input	Write Enable
wrclk	Input	Write Clock
wrdata[7:0]	Input	Write Data
rden	Input	Read Enable
rdclk	Input	Read Clock
rddata[7:0]	Output	Read Data
full	Output	Full
empty	Output	Empty

In addition to FIFO control, there is logic to process *fifo_clr*, which is a signal sent by the application processor via SPI interface to clear the data FIFO, and logic to acknowledge that FIFO has been cleared (*fifo_clr_ack*). Additional FIFO control is present, but currently, underflow and overflow logic have not been implemented.

Also available to this block is the signal *i2c_done*. This signal is sent from the I²C interface to indicate whether the I²C transaction has been completed.

The condition that triggers the sensor monitor to read data from the sensor is written in the following code:

```
assign trig_acquisition = sensor_intr && empty && ~intr;
```

This means that read is triggered when sensor monitor indicates the periodic reads (sensor_intr), the data FIFO is empty (*empty*), and the sensor monitor is not interrupting the application processor (*~intr*).

Note that logic to process DEV_ON and DEV_OFF, as well as the logic to create their acknowledges have not been implemented.

4.2.3. I²C Interface Module (i2c_arbiter)

The I²C Interface Module is found in the i2c_arbiter file. It is used to provide connection between the sensor monitors to the sensors via I²C interface. In addition to I²C interfacing and processing I²C commands to/from sensors, this module contains the logic that polls the sensor monitors one-by-one in a round-robin fashion to see whether a sensor data read request is present. When a sensor monitor is not available, it goes to the next sensor and so on.

The following table summarizes the ports to/from the I²C Interface Module:

Table 4.3. Ports To/From I²C Interface Module

Signal	Direction	Description
clk	Input	System Clock
rst	Input	System Reset
scl_in	Input	I²C clock output
sda_in	Input	I²C data input
scl_oe_n	Output	I²C output clock enable control
sda_oe_n	Output	I²C output data enable control
dev#_i2c_start	Input	Signal from the sensor monitor to indicate to access the sen- sor (Active HIGH) - Connect to logic LOW if unused.
dev#_read_write_n	Input	Signal from the sensor monitor to indicate whether the accessed command is a read or a write (Read = HIGH, Write = LOW) - Connect to logic LOW if unused.
dev#_slave_address[7:0]	Input	Signal from the sensor monitor to indicate the slave device (sensor) address - Set to 0 if unused.
dev#_read_byte_count[7:0]	Input	Signal from the sensor monitor to indicate the number of bytes to read - Set to 0 if unused.
dev#_reg_address[7:0]	Input	Signal from the sensor monitor to indicate the slave device's (sensor) register's address - Set to 0 if unused.
dev#_write_data[7:0]	Input	Contains the data from sensor monitor to write to the slave device (sensor) - Set to 0 if unused.
dev#_read_data[7:0]	Output	Contains the data read from the slave device (sensor). The data is sent to sensor monitor.
dev#_read_data_valid	Output	Indicates whether the read_data is valid. It is sent to sensor monitor. (Active HIGH).
dev#_i2c_done	Output	Indicates whether the I²C transaction has been completed. It is sent to sensor monitor. (Active HIGH).

Note: # is an integer from 0 to 3 (that is 4 sensors). With proper code modification, it can be expandable.

The following is a walkthrough of the i2c_arbiter code. Codes in this section are taken directly from the HDL file. Note that in most cases, the topics in each paragraph below are presented in the order in which they appear in the HDL code.

Notice that one of the first parameters defined is the device number (DEV0 to DEV3).

The first logic implemented is the state machine that polls the sensor monitor to see whether there's a request of I2C transaction using the *dev#_i2c_start* signal (*i2c_start*). The state machine leaves the processing state when *i2c_done* (provided by the submodule *i2_reg_ctrl*) is asserted. Note the round robin scheme of polling the sensor monitor.

The next set of logic simply latches the signals received from sensor monitor for the I2C interface to process. These signals are: i2c_start, read_write_n, slave_address, read_byte_count, reg_address, and write_data.

The next set of logic is used to process the data received from each sensor before they are sent to the target sensor monitor. These logics are for the following signals: read_data_valid, read_data, and i2c_done.

Finally, the i2c_arbiter calls *i2c_reg_ctrl* which contains the actual I²C controller. This module also contains the logic to control i2c_done, count number of bytes, process read or write, and receive read_data (and read_data_valid).

The following table summarizes the ports to/from the i2c_reg_ctrl:

Table 4.4. Ports To/From i2c_reg_ctrl

Signal	Direction	Description
read_data[7:0]	Output	Contains the data read from the slave device (sensor). The data is sent to sensor monitor.
read_data_valid	Output	Indicates whether the read_data is valid. It is sent to sensor monitor. (Active HIGH).
i2c_done	Output	Indicates whether the I^2C transaction has been completed. It is sent to sensor monitor. (Active HIGH).
scl_oe_n	Output	I^2C output clock enable control
sda_oen	Output	I^2C output data enable control
clk	Input	System Clock
rst	Input	System Reset
i2c_start	Input	Signal from the sensor monitor to indicate to access the sensor (Active HIGH)
read_write_n	Input	Signal from the sensor monitor to indicate whether the accessed command is a read or a write (Read = HIGH, Write = LOW).
slave_addr[7:0]	Input	Signal from the sensor monitor to indicate the slave device (sensor) address.
read_byte_count[7:0]	Input	Signal from the sensor monitor to indicate the number of bytes to read.
reg_address[7:0]	Input	Signal from the sensor monitor to indicate the slave device's (sensor) register's address.
write_data[7:0]	Input	Contains the data from sensor monitor to write to the slave device (sensor).
scl_in	Input	I^2C clock output
sda_in	Input	I^2C data input

Lattice Semiconductor does not recommend that the i2c_reg_ctrl be modified.

4.2.4. SPI Interface Module (spi_reg)

The SPI Interface Module is found in the spi_reg file. It is used to provide connection between the sensor monitors to the application processor via SPI interface (spi_slave). In addition to SPI interfacing and processing SPI commands to/from application processor, this module contains an interrupt arbiter (intr_arb) module that polls interrupt (that is to indicate that sensor data is present) from the sensor monitors. Finally, the SPI registers are contained in this module.

The following table summarizes the ports to/from the SPI Interface Module:

Table 4.5. Ports To/From the SPI Interface Module

Signal	Direction	Description
clk	Input	System clock
rst	Input	System reset
SPI_SCLK	Input	SPI interface (connected to proc_sclk pin)
SPI_SS_N	Input	SPI interface (connected to proc_csn pin)
SPI_MOSI	Input	SPI interface (connected to proc_sdo pin)
SPI_MISO	Output	SPI interface (connected to proc_sdi pin)
intr	Output	Interrupt to Application Processor. It is connected to proc_intr pin. This signals the processor that there's read data available for the processor and to begin the read operation. (Active HIGH).
soft_reset	Output	Reset the systems when interrupt has reached timeout limit - NOT USED IN SYSTEM (Active HIGH)
dev#_fifo_rden	Output	Reads sensor monitor FIFO (Active HIGH)
dev#_fifo_data[7:0]	Input	Sensor read data (fifo_data[7:0] port) from sensor monitor. This data will be sent to Application Processor via SPI interface logic.
dev#_fifo_clr	Output	This signal clears the sensor monitor FIFO data after data has been read by SPI interface logic (Active HIGH).
dev#_fifo_clr_ack	Input	Signal from the sensor monitor to indicate that the FIFO data has been cleared successfully (Active HIGH).
dev#_inte	Output	Enables interrupt of the sensor that corresponds to the sensor monitor - NOT IMPLEMENTED IN SENSOR MONITOR (Active HIGH)
dev#_intr	Input	Signal from the sensor monitor to indicate that sensor read data is present for Application Processor to access (Active HIGH).
dev#_calibration	Input	Signal from the sensor monitor to indicate that the sensor monitor is per- forming calibration operation on the sensor. Not all sensors have this. (Active HIGH).
dev#_intr_ack	Output	Signal for the sensor monitor to indicate that *intr* has been received and to deassert the *intr* in the sensor monitor (Active HIGH).
dev#_on	Output	Signal for the sensor monitor to control DEV_ON - NOT IMPLEMENTED IN SENSOR MONITOR.
dev#_on_ack	Input	Signal from the sensor monitor to indicate that DEV_ON has been successfully processed (Tied to logic HIGH at sensor monitor).
dev#_off	Output	Signal for the sensor monitor to control DEV_OFF - NOT IMPLEMENTED IN SENSOR MONITOR.
dev#_off_ack	Input	Signal from the sensor monitor to indicate that DEV_OFF has been successfully processed (Tied to logic HIGH at sensor monitor).
dev#_underflow	Input	Signal from the sensor monitor to indicate that FIFO underflow has occurred (Tied to logic LOW at sensor monitor).
dev#_overflow	Input	Signal from the sensor monitor to indicate that FIFO overflow has occurred (Tied to logic LOW at sensor monitor).
dev#_active	Input	Signal from the sensor monitor to indicate that the sensor monitor is performing a function (e.g. reading sensor data) (Active HIGH).
dev#_fifo_full	Input	Signal from the sensor monitor to indicate that the data

Signal	Direction	Description
		FIFO is full (Active HIGH).
dev#_fifo_empty	Input	Signal from the sensor monitor to indicate that the data FIFO is empty (Active HIGH).

Note: # is an integer from 0 to 6 (that is 7 sensors). With proper code modification, it can be expandable.

The following is a walkthrough of the spi_reg code. Codes in this section are taken directly from the HDL file. Note that in most cases, the topics in each paragraph below are presented in the order in which they appear in the HDL code.

The first parameters defined in the code are: VERSION (which is for VERSION register), and INTR_TIMEOUT_COUNTER (which is used for timeout in the interrupt arbiter).

The code then proceeds to process the information received from application processor (*mosi_byte*). Mosi_byte is then decoded into command information and device number.

The decoded information is then processed into either Register Write Interface, Register Read Interface, FIFO Data Read Interface, or Processor Interrupt Mechanism.

At Register Write Interface:

- If reg_address = 3, then it is a reset from the processor. Reset from processor is currently not implemented, but there's a soft_reset mechanism when timeout in the interrupt arbiter occurs.
- If reg_address = 4 then the CNTRL register are updated for FIFO clear (which is required after a sensor read), DEV_ON, and DEV_OFF. Note that DEV_ON and DEV_OFF logic are not implemented at the sensor monitor.
- If reg_address = 2 then the interrupt enable registers are updated. Note that the interrupt enable is not implemented in the sensor monitor.
- If reg_address = 5 then the STATUS registers are updated for underflow and overflow. Note that the overflow and underflow logic are not implemented in the sensor monitor.

The Register Read Interface, grabs commands from the SPI interface through *miso_byte_req* to update registers (*update_register*). When *update_register* signal is true, values of CNTRL and STATUS are updated. One clock cycle later, the requested register (that is by *reg_address*) is sent to the application processor via *miso_byte* signal.

The FIFO Data Read Interface contains the logic to perform the data read from the sensor monitor.

The Processor Interrupt Mechanism is used to read or write the ISR register (which is located in the interrupt arbiter).

At the end of the SPI Interface Module code, the spi_slave and intr_arb are instantiated.

The following table summarizes the ports to/from the interrupt arbiter module:

Table 4.6. Ports To/From Interrupt Arbiter Module

Signal	Direction	Description
clk	Input	System Clock
rst	Input	System Reset
intr	Output	Interrupt to Application Processor. It is connected to proc_intr pin. This signals the processor that there's read data available for the processor and to begin the read operation. (Active HIGH).
dev#_intr	Input	Signal from the sensor monitor to indicate that sensor read data is present for Application Processor to access (Active HIGH).
dev#_intr_ack	Output	Signal for the sensor monitor to indicate that *intr* has been received and to deassert the *intr* in the sensor monitor (Active HIGH).
isr[7:0]	Output	Content of ISR register (for spi_reg).
timeout_reset	Output	Determines if the interrupt state has reached timeout limit. It is used to create a soft_reset at SPI interface logic - NOT USED IN SYS- TEM (Active HIGH).
isr_read	Input	Signal from SPI interface logic to determine if Application Processor wants to read ISR register (Active HIGH).
isr_write	Input	Signal from SPI interface logic to determine if Application Processor wants to write ISR register (Active HIGH).

Note: # is an integer from 0 to 6 (that is 7 sensors). With proper code modification, it can be expandable

Notice that the INTR_TIMEOUT_COUNTER parameter is used in this module for interrupt output.

The round robin sensor monitor polling state machine is implemented first. When an interrupt from sensor monitor is received, the state machine processes the interrupt (ASSERT_INTR state). In the event that ASSERT_INTR has reached timeout limit, RESET_INTR state is reached. Counters for timeout are implemented directly after the state machine.

The interrupt arbiter also contains logic to decode which sensor issued the interrupt. It also asserts the corresponding ISR location for each sensor. Notice that only 1-bit of the ISR register is active at any time. Finally, there's logic to send interrupt to the processor (*intr*) to indicate that sensor data is present and ready for read.

During the WAIT_FOR_ACK state, the sensor that issues interrupt will get an acknowledge signal (dev#_intr_ack). This acknowledge will be sent to sensor monitor when the application processor has read the sensor data and cleared the ISR register (via *isr_write* signal). Finally, there's reset logic for *timeout_reset* when WAIT_FOR_ACK is too long. It is currently not used in the solution. The following table summarizes the ports to/from the spi_slave module:

Table 4.7. Ports To/From Interrupt spi_slave Module

Signal	Direction	Description
i_sys_clk	Input	System Clock
i_sys_rst	Input	System Reset
miso_byte[7:0]	Input	Data to send to Application Processor
miso_byte_valid	Input	Determines if data to send is valid
miso_byte_req	Output	Determines whether the received command is write (Active HIGH) or read (Active LOW)
mosi_byte[7:0]	Output	Data received from Application Processor
mosi_byte_valid	Output	Determines if received data is valid (Active HIGH)
cmd_byte	Output	Determines if received data is a command byte (Active HIGH)
o_miso	Output	SPI interface (connected to proc_sdi pin)
i_mosi	Input	SPI interface (connected to proc_sdo pin)
i_csn	Input	SPI interface (connected to proc_csn pin)
i_sclk	Input	SPI interface (connected to proc_sclk pin)

The spi_slave module contains the hard SPI module called *SB_SPI*. It contains logic that determines whether the command is write or read, and state machine to process the SPI master commands so as to prepare data for the backend interface.

4.2.5. I²C Clock and Data I/O Control

The following code is used to create the bi-directional I/O as required in I²C standard.

```
assign poola_sensor_sda = (!poola_sda_oe_n)? 1'b0: 1'bZ;
assign poola_sensor_scl = (!poola_scl_oe_n)? 1'b0: 1'bZ;

assign poolb_sensor_sda = (!poolb_sda_oe_n)? 1'b0: 1'bZ;
assign poolb_sensor_scl = (!poolb_scl_oe_n)? 1'b0: 1'bZ;
```

Note that the assigned wire above need to be declared as *inout* at the top level module. The above code is to drive the output line. As input line, simply connect the assigned wire to the desired destination.

5. Design Considerations

5.1. SPI Interface

This section describes the SPI interface between iCE40 Sensor Hub and the Application Processor.

The Application Processor obtains sensor data over SPI lines through the spi_reg module. This module expects SPI in mode *3* format, that is CPHA = 1 and CPOL = 1, and MSB first while transmitting a byte of data over the bus.

The first byte after chip select assertion is treated as command byte, which would give the address of the register to-be-accessed. A dummy byte is sent in case of processor read operation to allow the read logic to decode the command and provide appropriate data in successive bytes. In case of write, the SPI Master would place the data bytes on the bus immediately after the command byte.

The following timing diagrams show various read/write access patterns. Multi byte transaction is supported only for read operation.

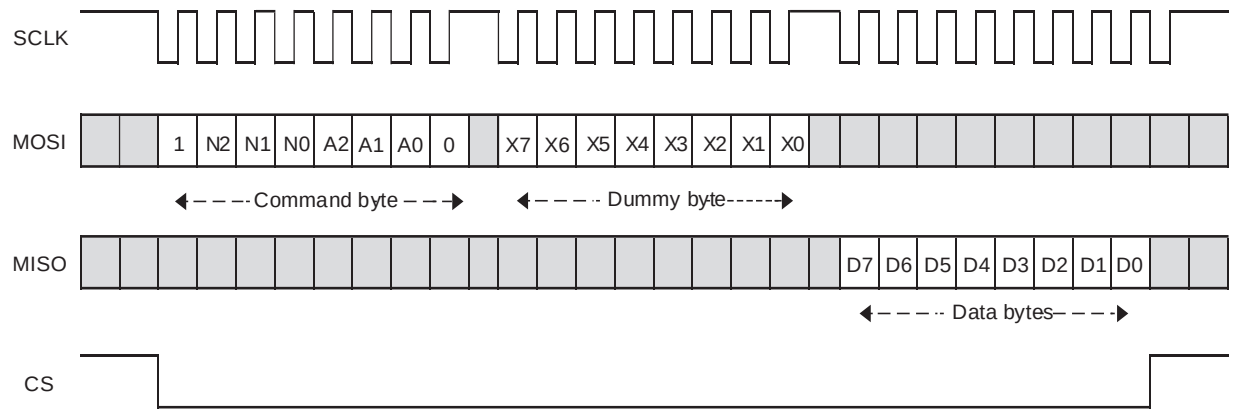


Figure 5.1. Singe Byte Read Operation

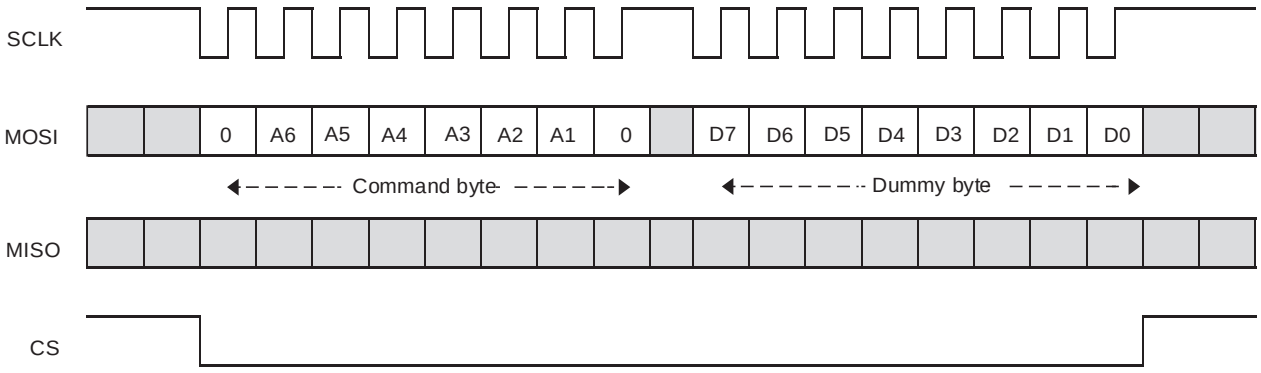


Figure 5.2. Singe Byte Write Operation

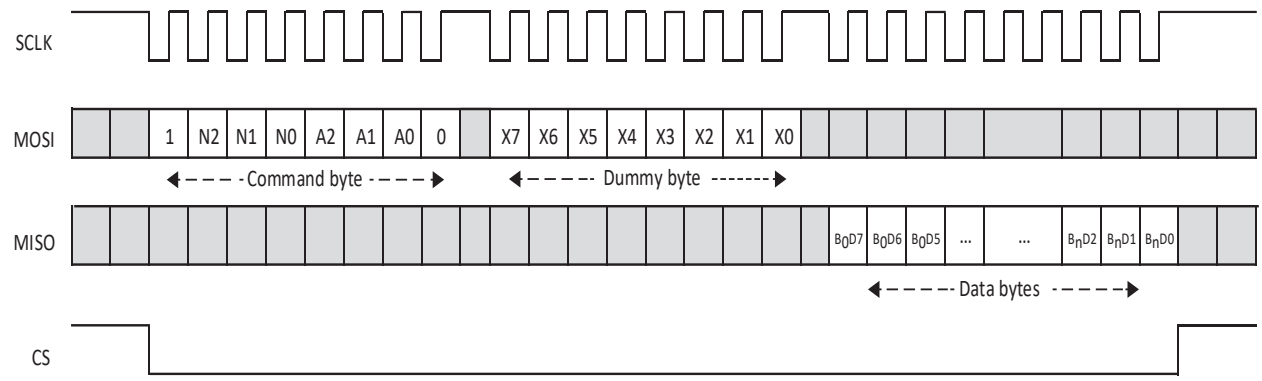


Figure 5.3. Multi-Byte Read Operation

Notes:

- In the above timing diagrams, N2, N1, and N0 under the Command byte indicate the device (sensor) number to which the current register read/write is applicable. Valid range is *000* to *101*.
- A2, A1, and A0 in the Command byte indicate address of the register. See SPI Registers Description for more details.
- When A2 = 0, N2, N1, and N0 are don't care.
- For a read operation from processor, MSB of command byte is always 1.
- For a write operation from processor, MSB of command byte is always 0.
- LSB of command byte is always 0 for both read and write from processor.
- CS must not be asserted until all the bytes are read in case of multiple bytes read.
- Multiple byte write operation is not defined.

5.2. SPI Registers Description

Each sensor has the following set of registers to configure the Sensor Hub to acquire sensor data. These registers are accessed by A2, A1, and A0 bits of the Command byte. The following table describes registers accessed by the A2, A1, and A0 bits.

Table 5.1. Register Map for A2, A1, and A0 bits

Address (A2, A1, A2 as 3-bit hex)	Register Name	Access Type	Description
0x00	VERSION	R	Indicates the firmware version (independent of sensor selected).
0x01	ISR	R/W	Interrupt status register (independent of sensor selected).
0x04	CNTRL	R/W	Control register (sensor specific)
0x05	STATUS	R/W	Status register (sensor specific)
0x06	DATA	R	Acquired data register (sensor specific)

Note: Sensor specific registers require N2, N1, and N0 bits of the Command byte to access.

Table 5.2. VERSION Register Bit Description

7	6	5	4	3	2	1	0
Firmware Version							

Table 5.3. ISR Register Bit Description

7	6	5	4	3	2	1	0
0	0	INT5	INT4	INT3	INT2	INT1	INT0

INT0 – interrupt by sensor 0. This will be set by the sensor and cleared by the processor

INT1 – interrupt by sensor 1. This will be set by the sensor and cleared by the processor

INT2 – interrupt by sensor 2. This will be set by the sensor and cleared by the processor

INT3 – interrupt by sensor 3. This will be set by the sensor and cleared by the processor

INT4 – interrupt by sensor 4. This will be set by the sensor and cleared by the processor

INT5 – interrupt by sensor 5. This will be set by the sensor and cleared by the processor

Interrupt to the processor will be OR of INT0, INT1, INT2, INT3, INT4 and INT5.

Note: Interrupts are always enabled.

Table 5.4. CNTRL Register Bit Description

7	6	5	4	3	2	1	0
0	0	0	0	0	FIFO_CLR	DEV_ON	DEV_OFF

FIFO_CLR – Writing '1' to this will clear the FIFO contents.

DEV_ON – Writing '1' to this will enable the device if it is turned off.

DEV_OFF – Writing '1' to this will force the device into power down mode if it is active.

Note:

- All the bits of the CNTRL are set by the processor and will be reset by the sensor hub.
- DEV_ON and DEV_OFF logic is currently not implemented.

Table 5.5. STATUS Register Bit Description

7	6	5	4	3	2	1	0
0	0	EMPTY	FULL	CALIB	UNDERFLOW	OVERFLOW	ACTIVE

ACTIVE (R) – *1* indicates the device is operational.

OVERFLOW – *1* indicates the processor failed to read acquired data before being overwritten by the next sample (this is a sticky bit can be cleared by the processor).

UNDERFLOW – *1* indicates the processor has issued read command before the data acquisition is complete (this is a sticky bit can be cleared by the processor).

CALIB (R) – *1* indicates calibration data is present in the FIFO FULL(R) – '1' indicates the data FIFO is full.

EMPTY(R) – *1* indicates the data FIFO is empty.

Note: UNDERFLOW and OVERFLOW logic is currently not implemented

Table 5.6. DATA Register Bit Description

7	6	5	4	3	2	1	0
Acquired Sensor Data [7:0]							

Data reading is multi-byte read operation (with single address). While reading the acquired sensor data it is expected that the application processor is aware of the numbers bytes to be read for a particular device.

5.3. Complete SPI Registers Location

The table below lists the first byte to be transmitted from SPI master (AP) to iCE on MOSI Line. This is combination of register address listed in SPI Registers Description section, and also the control signal values listed after the SPI Timing figures (see Notes in SPI Interface section).

Table 5.7. First Byte Transmitted – SPI Master to iCE on MOSI Line

First Byte for SPI Read (1,N2,N1,N0,A2,A1,A0,0 as 8-bit hex)	First Byte for SPI Write (0,N2,N1,N0,A2,A1,A0,0 as 8-bit hex)	Register Description
0x80	—	Version (device independent)
0x82	0x02	ISR (device independent)
0x88	0x08	BMP085 pressure sensor control register
0x98	0x18	LSM303DLHC magnetometer control register
0xA8	0x28	LSM330DLC accelerometer control register
0xB8	0x38	MAX 44006 Ambient Light Sensor control register
0xC8	0x48	SHT20 Humidity sensor control register
0xD8	0x58	LSM330DLC gyroscope control register
0x8A	0x0A	BMP085 pressure sensor status register
0x9A	0x1A	LSM303DLHC magnetometer status register
0xAA	0x2A	LSM330DLC accelerometer status register
0xBA	0x3A	MAX 44006 Ambient Light Sensor status register
0xCA	0x4A	SHT20 Humidity sensor status register
0xDA	0x5A	LSM330DLC gyroscope status register
0x8C	—	BMP085 pressure sensor data register
0x9C	—	LSM303DLHC magnetometer data register
0xAC	—	LSM330DLC accelerometer data register
0xBC	—	MAX 44006 Ambient Light Sensor data register
0xCC	—	SHT20 Humidity sensor data register
0xDC	—	LSM330DLC gyroscope data register

Example:

Sensor Hub generates interrupt when LSM330DLC accelerometer data is available in that sensor's FIFO. When sensor data is available, Sensor Hub interrupts the application processor by generating a high on "proc_intr" pin. Processor must follow the below mentioned steps to read accelerometer data from iCE.

- When *proc_intr* goes high, read the ISR register (0x82) by writing 0x82 as first byte on proc_sdo (MOSI) line.
- Based on the ISR register value (3rd byte on MISO line), read the sensor data from corresponding device. For LSM330DLC accelerometer, the ISR value is 0x04.
- To read the sensor data, write the corresponding data register address as first byte on proc_sdo (MOSI) line. To read accelerometer data, the data register address is 0xAC.
- After reading 6 bytes of sensor data, write the value *0x00* to ISR register (0x02).

5.4. Pseudo-Code Example for Application Processor

The following code illustrates how an Application Processor could process the interrupt received from the sensor hub to obtain the sensor data.

```
while(interrupt received) {
      Read ISR (0x01)
      Read FIFOx; FIFOx decoded based on ISR bits and predefined data length for
that FIFO. Example: For Accelerometer, ISR bit 2 is checked and number of bytes to
be read is 6.
      Reset FIFOx content.
      Reset ISR to indicate end of processing.
```

5.5. Design Customization Considerations

Since this is an FPGA based solution, you can customize this solution by changing the source code of the Sensor Hub Solution or add additional functions to this solution.

5.6. Adding Sensors

The Block Description section describes the Sensor Monitor, I2C Interface Module, and SPI Interface module in detail. When adding sensors, you must implement the operations described in those sections. The following is a list of items to do and consider when adding a sensor.

5.6.1. Sensor Monitor

- Follow the port naming convention of the sensor monitor module.
- Set the INIT_THRESHOLD
- Set the INTR_THRESHOLD.
- Create counters for the threshold values.
- Define slave_address.
- Create a state machine for sensor initialization, periodic sensor read, and wait between reads.
- Create logic for the following I2C interface signals:
  - I2c_start
  - Read_write_n
  - Read_byte_count
  - Reg_address
  - Write_data
- Create logic for *intr* signal for the SPI interface.
- Create calibration logic if required by sensor.
- Create logic of *active* signal for the SPI interface.
- Add a data FIFO with appropriate width and depth.
- Create logic to process *fifo_clr* and to create *fifo_clr_ack*.
- Create sensor read trigger logic.
- Create additional sensor specific logic as needed.
- Use *i2c_done* signal to let sensor monitor know when I²C transaction is complete.

5.6.2. I²C Interface Module

- Add sensor to the polling state machine.
- Create registers for the following signals from the sensor monitor:
  - I2c_start – Read_write_n
  - Read_byte_count
  - Reg_address
  - Write_data
- Create connection to the sensor monitor for the following signals:
  - Read_data_valid
  - Read_data
  - I2c_done
- Make sure *i2c_reg_ctrl* module still has its connection.

5.6.3. SPI Interface Module

- Assign a unique 3-bit value that can be used by *device_no* bus for the sensor.
- Change VERSION value, if needed.
- Change INTR_TIMEOUT counter, if needed.
- Add sensor to the following logic section:
  - Register Write Interface
  - Register Read Interfact
  - FIFO Data Read
- Add device to the “intr_arb” module.
  - Add device to the interrupt monitor poll.
  - Add device to the ISR assertion logic.

5.7. Removing Sensors

The following is a list of items to do and consider when removing a sensor.

5.7.1. Sensor Monitor

Remove the unwanted sensor codes.

5.7.2. I²C Interface Module

- Remove sensor to the polling state machine
- Remove registers for the following signals from the sensor monitor:
  - I2c_start
  - Read_write_n
  - Read_byte_count
  - Reg_address
  - Write_data
- Disconnection the following signals to the sensor monitor:
  - Read_data_valid
  - Read_data
  - I2c_done
- Make sure *i2c_reg_ctrl* module still has valid connection.

5.7.3. SPI Interface Module

- De-assign a unique 3-bit value that can be used by *device_no* bus for the sensor.
- Change VERSION value, if needed.
- Change INTR_TIMEOUT counter, if needed.
- Remove the sensor from the following logic section:
  - Register Write Interface
  - Register Read Interfact
  - FIFO Data Read
- Remove device from the *intr_arb* module.
  - Remove device from the interrupt monitor poll.
  - Remove device from the ISR assertion logic.

5.8. HDL Simulation Waveform

Figure 5.4, Figure 5.5, and Figure 5.6 show the simulation waveform for the I²C sensor pool and Application Processor Interface.

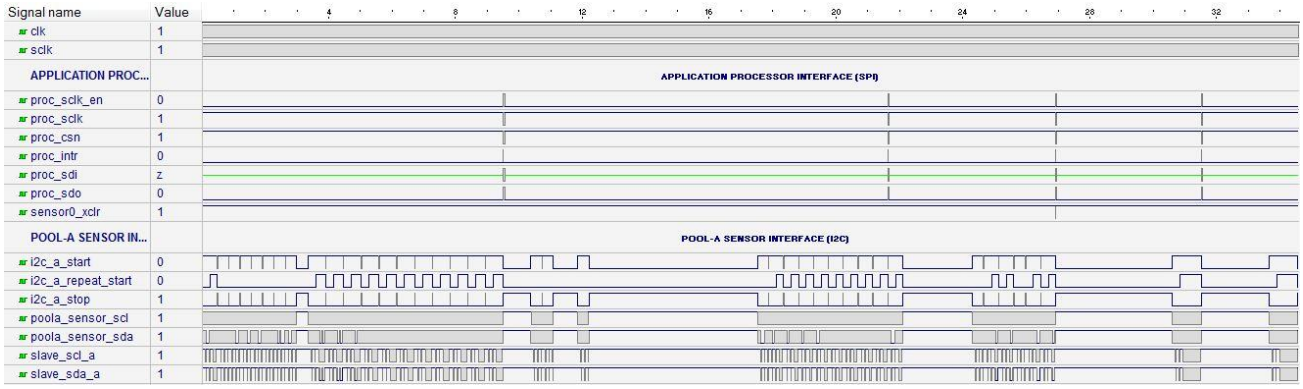


Figure 5.4. HDL Simulation Waveform

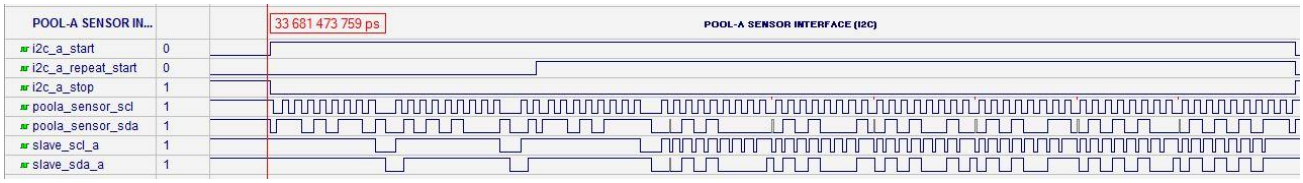


Figure 5.5. Simulation Waveform Excerpt Showing the I²C Transactions for the LSM330DLC Gyroscope Sensor

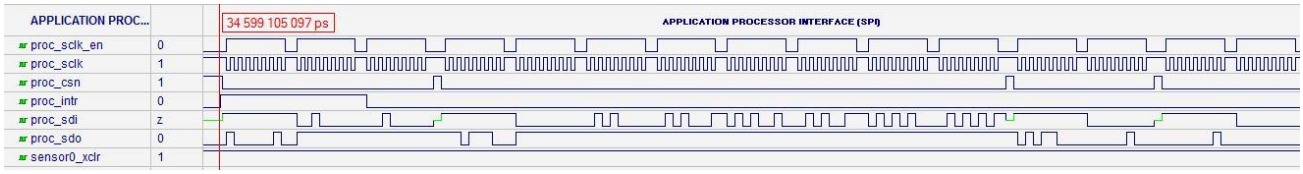


Figure 5.6. Simulation Waveform Excerpt for the Application Processor Interface Showing the Interrupt Signal (proc_intr) from the FPGA and the SPI Transactions for the LSM330DLC Gyroscope sensor.

6. Software Requirements

The following software are required in order to design and use the iCE40 Low Power Sensor Hub Solution:

- **iCEcube2 2014.08** (or higher) – This is used to build the iCE40LM and iCE40 Ultra versions of this reference design.
- **Diamond Programmer® 3. 8** (or higher) – This is used to program the bitstream to the External SPI Flash on the board.
- **Radiant Software 1.0** (or higher) – This is used to build the iCE40 UltraPlus version of this reference design.
- **Radiant Programmer tool** – This is used to program the bitstream to the External SPI Flash on the board.

7. Hardware Requirements

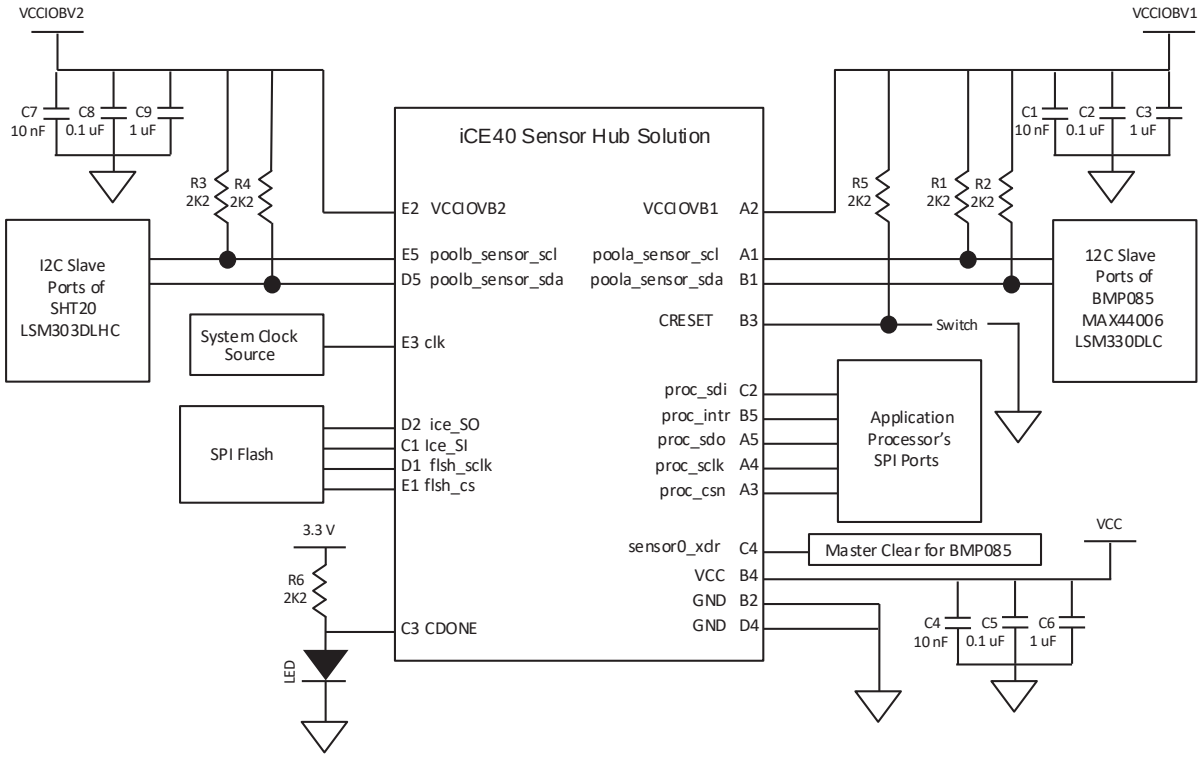
The standalone solution for this reference design has been targeted to support the iCE40LM4K Sensor Evaluation Kit but can be modified for use in other HW as needed. Project files for iCE40LM, iCE40 Ultra, and iCE40 UltraPlus are included in the reference design package.

8. Directory Structure

Figure 8.1 shows the directory structure when unzipping the iCE40 SensorHub Reference Design ZIP File. The figure explains what files are contained in each folder.

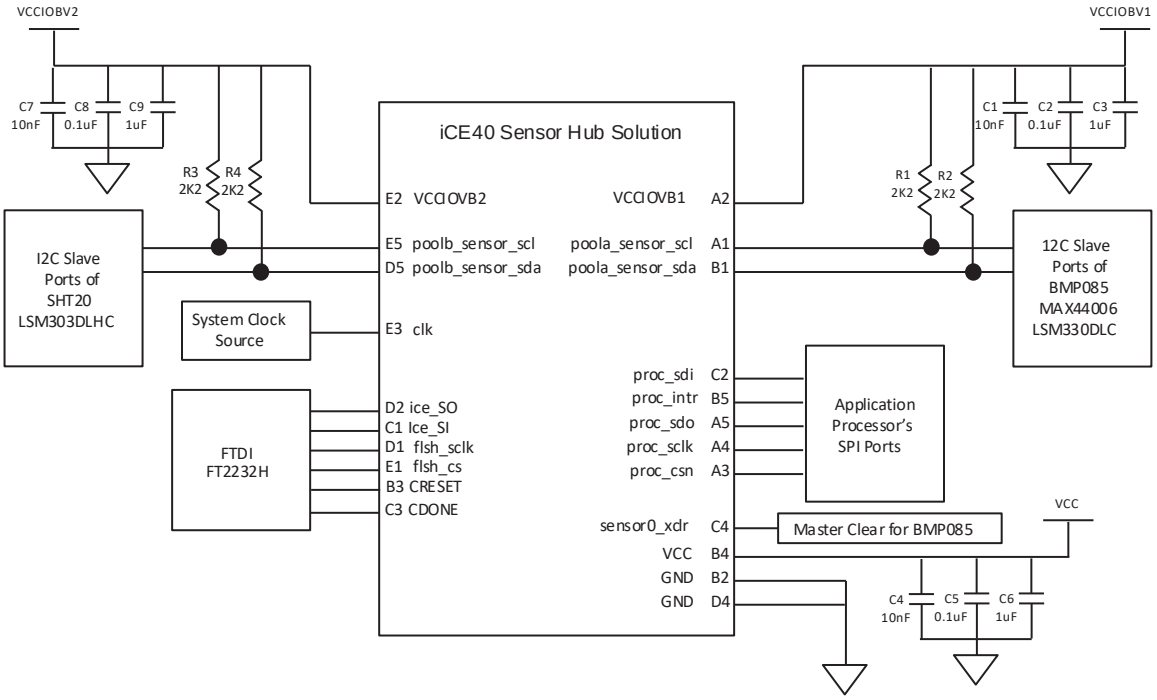
Figure 8.1. iCE40 Sensor Hub Reference Design Directory Structure

9. Typical Application Circuits



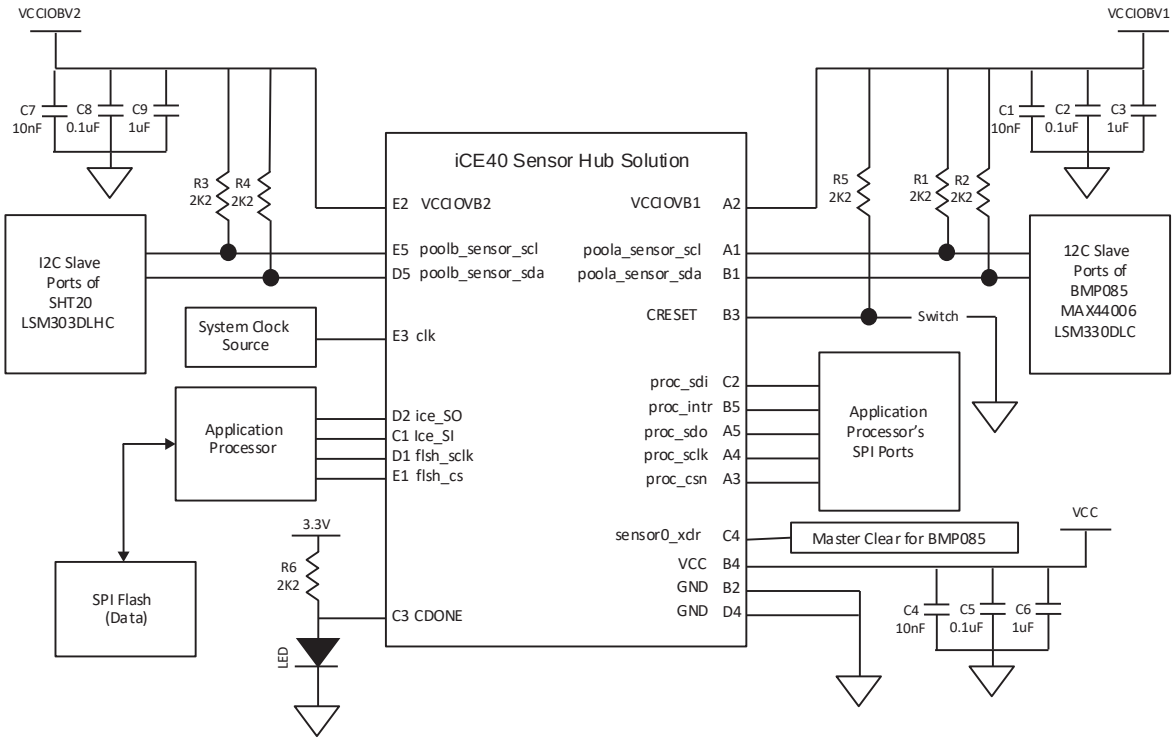
Note: Pad Names may vary depending on the device family and package type used in this customizable solution.

Figure 9.1. Sensor Hub with Pre-programmed SPI Flash – iCE40LM



Note: Pad Names may vary depending on the device family and package type used in this customizable solution.

Figure 9.2. Sensor Hub with Direct Programming through FTDI – iCE40LM



Note: Pad Names may vary depending on the device family and package type used in this customizable solution.

Figure 9.3. Sensor Hub with Programming through Application Processor – iCE40LM

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.3, September 2018

Section	Change Summary
All	• Changed document title to Sensor Interfacing and Preprocessing. • Changed document number from RD1189 to FPGA-RD-02048. • Added support for iCE40 UltraPlus. • Updated document template.
Related Documentation	Added new section to the document.
Pin Configuration	• Updated note in Figure 3.1. Bottom View of iCE40LM4K-SWG25TR (Balls Up). • Updated note in Table 3.1. Pin Function Description – iCE40LM*.
HDL Simulation	Added new section to the document.
Software Requirements	Updated section.
Hardware Requirements	Added new section to the document.
Directory Structure	Added new section to the document.
Typical Application Circuits	Updated note in Figure 9.1. Sensor Hub with Pre-programmed SPI Flash – iCE40LM, Figure 9.2. Sensor Hub with Direct Programming through FTDI – iCE40LM, and Figure 9.3. Sensor Hub with Programming through Application Processor – iCE40LM.

Revision 1.2, June 2014

Section	Change Summary
All	• Changed document title to iCE40 Low Power Sensor Hub Solution for Mobile Devices. • Added support for iCE40 Ultra.

Revision 1.1, October 2013

Section	Change Summary

Theory of Operations	• Updated the following sections in Functional Description:
	  • BMP085 Sensor Monitor
	  • LSM303DLHC Sensor Monitor
	  • LSM330DLC Accelerator Sensor Monitor
	• Corrected typographical error in Table 4.1. Ports To/From the Sensor Monitor.
	• Changed table title to Table 4.4. Ports To/From i2c_reg_ctrl.
	• Changed table title to Table 4.7. Ports To/From Interrupt spi_slave Module.

Revision 1.0, October 2013

Section	Change Summary
All	Initial release.

7th Floor, 111 SW 5th Avenue
Portland, OR 97204, USA
T 503.268.8000
www.latticesemi.com