Application Note: AZD025
IQ Switch® - ProxSense™ Series
I2C Example Code for the IQS222

1. Introduction

The IQS222 uses a 100 KHz bi-directional 2-wire bus and data transmission protocol. The serial protocol is I2CTM compatible. The IQS222 has an optional ready (RDY) pin which indicates when the device enters its communication window period. Communication with the device can only take place in this state, this can be determined by monitoring the RDY line or by using ACK polling. The IQS222 only functions as a slave device on the bus. The bus is controlled by a master device which generates the serial clock (SCL), controls bus access, and generates the START and STOP conditions. The serial clock (SCL) and serial data (SDA) lines are open-drain and therefore must be pulled high to the operating voltage with a pull-up resistor (10kΩ). The RDY pin functions as an open-drain pin and should always be pulled to the operating voltage of the master device via a resistor (100kΩ).

During the communication window period the RDY line will remain low (high for pre-production engineering versions of the IC) for a selectable duration of always/2ms (See datasheet for selection options). If the master does not initiate a data transfer during this time, the device will exit the communication window and continue doing conversions. During the communication window the address pointer will default to the value specified in the DEFAULT_ADDR register. Using this method the user can simply start reading without having to set the address pointer first. The RDY line will remain low for the duration of the communication window period.

1.1 Bus Characteristics

The following bus protocol has been defined:

- Data transfer may only be initiated when the bus is not busy
- During data transfer, the data line must remain stable whenever the clock line is HIGH. Changes in the data line while the clock is HIGH will be interpreted as START and STOP conditions.

The following conditions have been defined for the bus: (refer to Figure 1)

- **Bus Idle (A)** - The SCL and SDA line are both HIGH.
- **START Condition (B)** - A HIGH to LOW transition of the SDA while the SCL is HIGH. All serial communication must be preceded by a START condition.

Listing 1. START Condition.

```
/*!
    Generate Start Condition
*/
void I2CStart (void)
{
    SDA = HIGH;                         // Set data line high
    SCL = HIGH;                         // Set clock line high
    SDA = LOW;                          // Set data line low (START SIGNAL)
    SCL = LOW;                          // Set clock line low
}
```

- **STOP Condition (C)** - A LOW to HIGH transition of the SDA while the SCL is HIGH. All serial communication must be ended by a STOP condition. NOTE: When a STOP condition is sent the device will exit the communications window and continue with conversions.

Listing 2. STOP Condition.

```
/*!
    Generate Stop Condition
*/
void I2CStop (void)
{
    unsigned char input_var;

    SCL = LOW;                          // Set clock line low
    SDA = LOW;                          // Set data line low
    SCL = HIGH;                         // Set clock line high
    SDA = HIGH;                         // Set data line high (STOP SIGNAL)
    TRIS_SDA=1;     // Put port pin into HiZ
}
```

- **Data Valid (D)** - The state of the SDA line represents valid data when, after a START condition, the SDA is stable for the duration of the HIGH period of the clock signal. The data on the line must be changed during the LOW period of the clock signal. There is one clock pulse per bit of data. Each data transfer is initiated with a START condition and terminated with a STOP condition.
- **Acknowledge** - The slave device must generate an acknowledge after the reception of each byte. The master device must generate an extra (9th) clock pulse which is associated with this acknowledge bit. The device that acknowledges, has to pull down the SDA line during the acknowledge clock pulse. NOTE: The IQS222 does not generate any acknowledge bits while it is not in its communication window.

Listing 3. Check for Acknowledge.

```
/*!
    Check for acknowledge
*/
bool I2CAck(void)
{
    SCL=0;          // Acknowledge
    TRIS_SDA=1;     // SDA Input
    SCL=1;
    Delay_us(10);
    return(SDA);    //Test Acknowledge
}
```

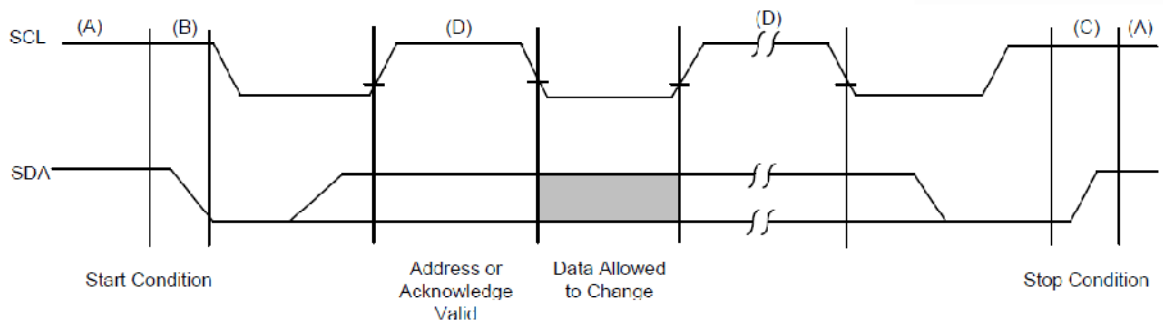

Figure 1. Data Transfer Sequence on the Serial Bus.

2. Access the communication window of the IQS222 according to selected method

Two methods of entering the I2C communication window is described in this section, namely “Acknowledge Polling” and “Using the RDY Line”. Both cases are illustrated in Listing.

2.1 Acknowledge Polling

If the Master device does not have an I/O available for the RDY pin, ACK polling can be used to determine when the device is ready for communication. The device will not acknowledge during a conversion cycle, this can be used to determine when a cycle is complete and whether the device has entered the communication window. Once a STOP condition is sent by the Master the device will perform the next conversion cycle. ACK polling can be initiated at any time during the conversion cycle to determine if the device has entered its communication window. The RDY pin will function normally even if it is not connected to a master device, or being used during communication.

To perform ACK polling the master sends a START condition followed by the control byte. If the device is still busy then no

ACK will be returned. If the device has completed its cycle the device will return an ACK and the master can proceed with the next read or write operation. To summarise, when polling the following procedures are executed:

1. The MCU generates a START condition.
2. The MCU sends the control byte.
3. The MCU checks if an acknowledge was received.
4. If not received the procedure is repeated from step 1.
5. The MCU reads from, or writes to the IQS222.

2.2 Using the RDY Line

When polling is not selected the MCU can simply wait for the RDY line to go low. There is a timeout option on the IQS222 which is selectable (2ms/always wait, see datasheet) to set how long the RDY pin will remain low and wait for communication to initiate, before the IQS222 exits, and continues with conversions.

Listing 4. Entering the I2C Communication Window.

```
void EnterI2CWindow(bool bPolling, uint8_t uAddress)
{
    if(!bPolling)
    {
        //Not polling so we wait for the RDY line to go high
        while(!I2CReadyPin);
         /* Send START condition */
        I2CStart();
        /* Send control byte */
        I2CWrite(uAddress+1); //add 1 to for read operation
    }
    else
    {
    //this is the method we use for polling the device
        do
        {
            /*a bit of a delay*/
            Delay_us(200);
            /* Send START condition */
            I2CSTART();
            /* Send address */
            I2CWrite(uAddress+1); //add 1 to for read operation

        }//Carry on until we get an Acknowledge
        while(!I2CAck());
    }
}
```

3. Control Byte Format

A control byte is the first byte received following the start condition from the master device. The control byte consists of a 7 bit device address and the Read/ Write indicator bit, as shown in Figure 2.

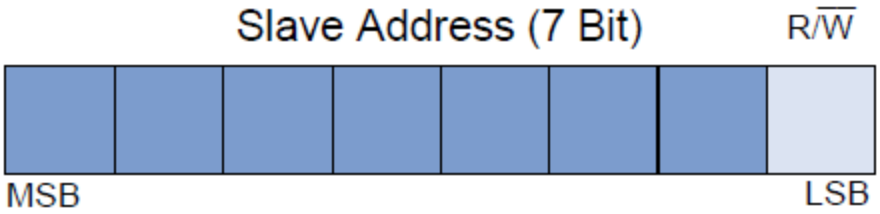

Figure 2. Control Byte Format.

4. Current Address Read Operation

Once the communications window is entered and a data transfer is initiated the required data can simply be read. The read transfer is in the format shown in Listings 5 and 6. Once all the required data has been read the Master must send a NACK. The Master can then either generate another START condition to perform another read or write operation or it can generate a stop condition at which point the device will exit the communication window and continue with conversions.

Listing 5. Read Operation.

```
uint8_t I2CRead (void)
{
    uint8_t i=0, ReceivedData;

    i = SDA;                            // Put data pin into read mode

    ReceivedData = 0x00;
    for(i = 0; i < 8; i++)              // Send 8 bits to the I2C Bus
    {
        ReceivedData <<= 1;             // Shift the byte by one bit
        SCL = HIGH;                     // Clock the data into the I2C Bus
        ReceivedData |= SDA;            // Input the data from the I2C Bus
        SCL = LOW;
    }

   return ReceivedData;
}
```

Listing 6. Read I2C Data.

```
/*!
    Read I2C data

    @param uAddress The IQS222 address
    @param uLength The number of bytes to read
    @param bPolling If true the IQS222 is polled to determine if it is ready to read
    @param bSendStop If true a Stop condition is generated after read operation is complete
    @param upDataBuf The buffer to which the data is written
*/
void I2CRead(uint8_t uAddress, uint8_t uLength, bool bPolling, bool bSendStop, uint8_t *upDataBuf)
{
    uint8_t i;

    EnterI2CWindow(bPolling, uAddress);

    //now read the data from the IQS222
    for(i =0;i<uLength;i++)
    {
        *(upDataBuf++) = I2CRead();
    }

    if(bSendStop)
    //if a stop is sent the IQS222 will exit the communication window and continue with converions
    I2CStop();
}
```

5. Write Operation

Once the communication window is entered and a data transfer is initiated, a write operation can be executed. Write operations are in the format shown in Listings 7 and 8. Once the Master is finished writing, the Master can then either generate another start condition or it could generate a stop condition. Another start condition will allow the Master to perform another read or write operation. A stop condition will exit the communications window and the IQS222 will continue with conversions.

Listing 7. Write Operation.

```
/*!
    write I2C data

    @param uAddress The IQS222 address
    @param bPolling If true the IQS222 is polled to determine if the device is ready to write
    @param bSendStop If true a Stop condition is generated after read operation is complete
    @param uRegAddr The Address to write to
    @param uRegData The Data to write
*/
uint8_t I2CWrite (uint8_t uAddress, uint8_t bPolling, bool bSendStop, uint8_t uRegAddr, uint8_t uRegData)
{
    uint8_t i;

    EnterI2CWindow(bPolling,uAddress);

    //The first byte written is always the data pointer
    I2CWrite(uRegAddr);
    //Now we write the data
    I2CWrite(uRegData);

    if(bSendStop)
    //if a stop is sent the IQS222 will exit the communication window and continue with converions
    I2CStop();
}
```

Listing 8. Write I2C Data.

```
/*!
    Write the Byte to the I2C device, bit by bit

    @param uByte The data to be written
*/
void I2CWrite(uint8_t uByte)
{
    unsigned char i;

    for(i = 0; i < 8; i++)                  // Send 8 bits to the I2C Bus
    {
                                            // Output the data bit to the I2C Bus
        SDA = ((output_data & 0x80) ? 1 : 0);
        uByte  <<= 1;                       // Shift the byte by one bit
        SCL = HIGH;                         // Clock the data into the I2C Bus
        SCL = LOW;
    }

    i = SDA;                                // Put data pin into read mode
    SCL = HIGH;                             // Clock the ACK from the I2C Bus
    SCL = LOW;
}
```