## Exercise 1: Sound Measurement

Goals

* Use the LabVIEW FPGA Toolkit and the NI SPI API to communicate with the microphone
* Simulate the SPI communication before compiling FPGA bitfile to verify functionality
* Plug in the acquisition code into the CompactRIO Waveform Library FPGA Template to handle scaling and data communication between the FPGA and LabVIEW Real-Time

### Part A – Application Description

In this exercise, use LabVIEW and the System on Module to acquire data from a microphone reference board over a SPI bus. Use the build in

### Part B – Code Implementation

#### Step 1

***Reviewing the datasheet***

The PmodMIC3 is a MEMS (Microelectromechanical System) microphone that makes us of an onboard ADC to sample analog audio signals. Once the ADC has digitized the analog signal, it transmits the data via a SPI bus to a receiving host device.

1. **Figure out Physical Connections**

To use the Pmod with SOM, the pin configuration of the Pmod must be known before coding the interface.

* 1. Open the provided datasheet titled *PmodMIC3\_RM.pdf*.
  2. On page 2, the pinout for the Pmod is listed. Take note which pin corresponds to *Signal Select/Chip Select* (SS/CS), *Serial Clock* (SCK), and *Master-In Slave-Out* (MISO). For this application, we do not need to send information to the Pmod, so a MOSI line will not be used.

1. **Timing and SPI Configuration Settings**

With the pin configuration known, the next important piece of information that is required is how the Pmod intends to transmits samples of data using these pins. In a typical SPI application, the toggling of the Signal Select (SS/CS) line initiates the transfer of information by informing all devices to transmit and listen on their corresponding MISO and MOSI lines. SPI is a shared clock protocol, meaning the Master device distributes one clock (SCK) that controls the transfer rate of all communication on the bus. Depending on device’s setup, a bit of data will be transmitted on the MISO and MOSI lines when there is either a rising or falling edge of SCK while the SS line is in its active state.

* 1. On page 2 of the data sheet, view the provided timing diagram and read the description of the Pmod’s communication procedure.
  2. Important pieces of information to note:
     + Each bit is shifted out on each falling edge of SCK after CS is pulled low (active).
     + The ADC is 12-bit, but the transfer uses 16 bits (2 bytes). The first four bits are leading zeroes (unused) and the 12-bit sample fills the remaining 12 bits.
     + The data is sent with the MSB (most significant bit) first.
     + For fast transfers (higher SCK frequencies), SCK should remain at a default high state before SS is pulled low.

#### Step 2

1. Add and setup the SOM hardware in LabVIEW
   1. Open the provided LabVIEW project *SOM Hands-On.lvproj*
   2. Add a SOM sbRIO-9651 to the project. Right-Click on **Project: SOM Hands-On.lvproj** (one line above My Computer) and select **New** > **Targets and Devices**.
   3. In the window that appears, leave the default options selected and expand the **Real-Time Single-Board RIO** folder to discover the connected SOM sbRIO. When it appears select it and press **OK**. This will add a SOM target to your project view with a default name of **RT Single-Board RIO Target**.
   4. Test to make sure you can connect to the target. Right-click on the SOM in the project and select **Connect**. A deployment window will appear and should mention that the deployment was successful. If it is not successful, alert your instructor.
   5. Add the provided *RT Main.vi* to your target. Right-click on the SOM target and select **Add** **>** **File**, then choose *RT Main.vi* and press **Add File**.
   6. Add an FPGA Target. Expand the SOM target, right-click on **Chassis**, and then select **New > FPGA Target**.
   7. Add the SOM’s I/O. Under the FPGA Target, right-click on the **sbRIO-9651 Socket** and select **Properties**. In the new window that appears, within the General category, select **DevKit** from the **sbRIO-9651 Socketed Component Level IP Declaration** drop-down combo box. Press **OK**. In the project, you can now see all of the available I/O from the SOM daughter board populate under the Socket item.
   8. Add a new 120 MHz derived clock. Under the FPGA Target, right-click the 40 MHz Onboard Clock and select **New FPGA Derived Clock**. In the window that appears, set the desired frequency to **120 MHz**, and press **OK**. This clock will be used later in code to drive logic on the FPGA for the SPI communication.
   9. Add the provided FPGA VIs to the project. Right-click on the FPGA Target and select **Add > File**, then select the *FPGA Main\_SPI API.vi* and press **Add File**.
2. Modify the SPI Example Code
   1. Open the block diagram of *FPGA Main\_SPI API.vi*.
   2. View the layout of the code. There are three main parts to be aware of. First, the initialization code in the left-most section of the block diagram is responsible for designating which hardware I/O lines the SPI API will use to communicate. It also initializes FPGA memory items and FIFOs that the SPI API needs to operate. Next, the *SPI Engine* SCTL (single-cycle timed loop) is where the SPI Engine code will be placed. The SPI Engine is responsible for actually controlling, reading, and writing the I/O lines that were selected. Finally, the *Command Queuing Loop* is where commands will be prepared and transferred to the SPI Engine for execution.
   3. Modify the *DIO Lines* and *Enable Lines* constants. In the initialization portion of the code, there are two cluster constants containing I/O references that can be used to select which pins the SOM should use for the SPI communication. In *Step 1*, the pinout of the PmodMIC3 was defined by the Pmod’s datasheet. This pinout will be used to determine the configuration of these I/O constants.
   4. Turn on *Context Help* (**Ctrl+H**) and hover over each of the cluster constants. This will provide more information as to what each of the I/O constants is for. By default, the first I/O item of the *DIO Lines* cluster is **SCLK** followed by **MISO**, then **MOSI**, etc.
   5. Press the drop-down arrow for each I/O item and select the resource that corresponds to the pin configuration from the PmodMIC3’s datasheet. For example, if the Pmod is plugged into the PMOD1 connector of the SOM’s daughter board, for SCLK the selected I/O item should be **PMOD1\_Pin4\_out**. This is because the PmodMIC3’s datasheet defines the 4th pin as SCLK. Repeat for *MISO* and *CS 0* lines.
   6. There are multiple CS lines available in the SPI API, but for this exercise only one CS line is required.
   7. Wire up the SPI Engine as shown in the diagram using VIs from the SPI API palette.
   8. Wire up the Command Queuing Loop as shown in the diagram.
3. Simulate the VI using a Desktop Execution Node (DEN)
   1. In the project, open the *SPI API – SPI Simulation.vi* located underneath *My Computer*.
   2. Open the block diagram and view the code. For this portion of the exercise, the SPI code will be simulated and the behavior will be verified to ensure the message being sent to the ADC is formatted correctly.
   3. Add a **Desktop Execution Node** from the **FPGA Interface** palette to the block diagram beneath the ***Desktop Execution Node*** note.
   4. Select the VI to be simulated. In the window that appears, click the folder icon next to the VI entry textbox. This will pop up a small window showing the FPGA targets available in the LabVIEW project. For this exercise the simulation will be done with the second hardware target titled **Simulation Target**. Expand this target and select the VI titled **FPGA Main\_SPI API – Solution.vi**.
      * Note: The SOM currently cannot be simulated in LabVIEW 2014, but support will be available in future releases. However, simulation with another target will perform the same.
   5. To the right of the VI selection box are the **Reference Clock** and **Clock Ticks** fields. The Desktop Execution Node simulates the FPGA by moving time forward a set amount of clocks ticks (**Clock Ticks** field) of a specified clock (**Reference Clock**) each time a call to the node is made. For applications with lower frequency logic, larger amounts of time could be skipped between calls to increase the simulation speed. However, for simulating SPI communication, each tick of the SPI Engine clock needs to be simulated to correctly validate the engine’s logic.
      * Select **120MHz** for the **Reference Clock**. This is the derived clock that was previously made to drive the SPI Engine.
      * Leave the **Clock Ticks** entry set at **1**, signifying that the Desktop Execution Node should only move time forward one tick of the 120 MHz clock each time it is called.
   6. In the Terminal Configuration section, the inputs and outputs that the Desktop Execution Node should use to interact with the FPGA need to be selected. Move the following items from the *Available Resources* window to the *Selected Resources* window by highlighting the item and clicking the **Right Arrow**.
      * CS
      * SCLK
      * MOSI
      * SPI PHY Settings
      * Sample Rate (Hz)
      * Chip Select
      * Tx
   7. Each resource selected has an associated direction: **In**, **Out**, or **In/Out**. By default, LabVIEW will automatically set FPGA controls as **In** and FPGA indicators as **Out**, but I/O items can be either and show up as **In/Out**. For resources that are inputs, highlight the item and select **In**. Similarly, items that are outputs of the FPGA need to be set as **Out**. Ensure the resources have the following directions:
      * CS: **OUT**
      * SCLK: **OUT**
      * MOSI: **OUT**
      * SPI PHY Settings: **IN**
      * Sample Rate (Hz): **IN**
      * Chip Select: **IN**
      * Tx: **IN**
   8. Press **OK** to accept the changes.
   9. Wire up the Desktop Execution Node as shown.
   10. Switch back to the front panel, and run the VI.
       * Ensure the **Engine Rate (MHz)** control has a value of **120**, matching the clock chosen for the Desktop Execution Node. This ensures the **Timing Characteristics** are calculated correctly.
       * Modify the **Sample Rate (Hz)** control and observe how the FPGA responds.
       * Change the **Divide Value** and **Transfer Length** located in the **SPI PHY Settings** control and observe the change in the SPI message’s characteristics.

#### Step 3

1. Compile the bitfile and test to see if the VI is operational
2. Step 4