MCC150 - Implementation of Digital Signal Processing Systems

Lab 2: BPSK Transceiver

Sining An, Erik Börjeson
Per Larsson-Edefors, Zhongxia Simon He

Version 1.2 - April 1, 2021

Contents

1 Introduction

In this lab, you will improve the design of your BPSK transmitter by adding a pulse-shaping filter. You will also construct a simple receiver with a matched filter and a simple symbol timing recovery (STR) unit. During the lab you will perform the following tasks:

- Generate a BPSK signal and filter it using a root-raise cosine (RRC) pulse-shaping filter.
- Construct a receiver that passes the received signal through a matched RRC filter.
- Design a STR subsystem to decimate the signal using a chosen sample offset.
- Use the In-System Sources and Probes to control the sample offset in the STR.
- Observe the output data from the STR in SignalTap Logic Analyzer.
- Save the input and output data from the STR subsystem with different sample offset in SignalTap.

2 Pre-Lab Preparation

In this section a short theoretical background of the DSP blocks that you will implement during the lab session is given, followed by some preparation tasks that should be performed before starting the session.

2.1 Pulse-Shaping Filter

In a communication system, the bandwidth in use is always limited. However, the bandwidth of a square-wave shaped signal is infinite, as shown in Fig. 1, and cannot be transmitted through a realistic communication system. To limit the bandwidth of the signal, a low-pass filter is used on the transmitter side. When the signal passes through the low-pass filter, the shape of each pulse is no longer a square, thus this filter is also called a pulse-shaping filter. A bandwidth-limited signal has a longer time domain response, as shown in Fig. 1.

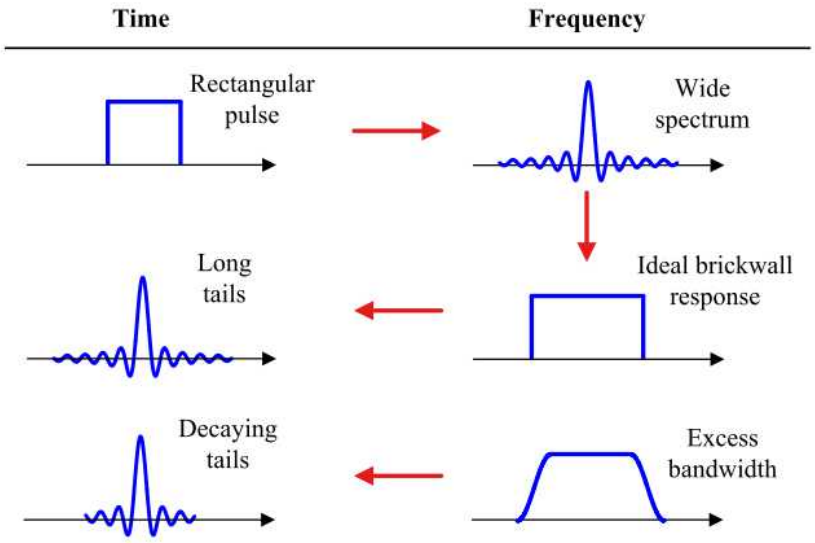
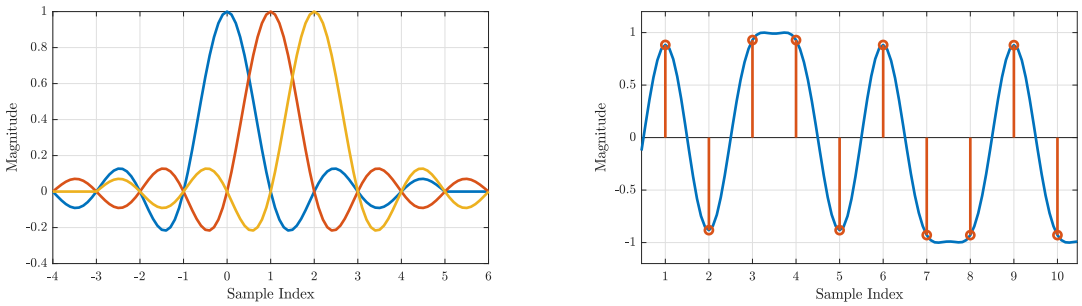


Figure 1: *Illustration of the bandwidth of a square wave signal and the effect of low-pass filtering*

After the signal passes through the pulse-shaping filter, each pulse has a tail that would interfere with the adjacent pulse; this is called inter-symbol interference (ISI). By choosing the right pulse-shaping filter, the interference can be reduced to zero in the middle of each pulse, as shown in Fig. 2a. The middle point of each pulse is the best sampling point with no ISI. A root-raised cosine pulse-shaped signal is shown in Fig. 2b. At the receiver side, the best sampling points need to be found to recover the transmitted data. The orange stems in Fig. 2b are the best sampling points that contain the transmitted data with no ISI. The process of looking for the best sampling points is called symbol time recovery.



(a) **(b)**

Figure 2: *Graphs showing (a) the interference of three pulse-shaped filtered pulses, and (b) a pulse-shaped filtered signal with the optimal sampling points marked.*

In this lab you will use a root-raised cosine pulse-shaping filter, whose impulse response can be described as

$$h(t) = \frac{\sin(\pi t/T)}{\pi t}\frac{\cos(\pi t\beta/T)}{1-4\beta^2t^2/T^2}, \tag{1}$$

where t is the tine, T is the symbol duration and β is the roll-off factor. During the lab you will use the MATLAB function `rcosdesign` to generate the filter.

Pre-lab tasks

- Read the help document of MATLAB function **rcosdesign** by typing **doc rcosdesign** in the MATLAB command window, or using the MATLAB online help center.
- Draw two eye diagrams of a baseband BPSK signal before and after propagation through a pulse-shaping filter. Indicate the difference between the two eye diagrams.
- What parameter in our setup script should you change if you want to change the symbol rate to 5 Mbps while keeping the sample rate at 40 MSps?

3 Lab

The following sections describe the exercises that are to be performed during this lab session.

3.1 Exercise 1 - Adding a Pulse Shaping Filter

In this exercise you will add a pulse-shaping filter to the transmitter designed in the previous lab session. The filter will be implemented as an FIR filter, which is described in [1, Ch. 3].

- Create a folder named `lab2` in the same place as your project folder from the previous lab was saved, preferrably on the `Z:` drive, if you are using the lab computers. Copy the Simulink project-file (`MCC150.slx`), the MATLAB setup script (`MCC150_setup.m`) and the DSP Builder parameter file (`MCC150_params.xml`) from the `lab1` folder to `lab2`. The parameter file contains all DSP Builder specific settings, such as clock frequency and output folder.

- Open DSP Builder using **DSP Builder - Start in MATLAB 2017b** under **Intel FPGA 18.0.0.614 Standard Edition** in the Windows start menu. Navigate to the `lab2` folder that you've just created, and make sure that it is the current folder in MATLAB. Open the setup script in MATLAB, and then double click the `MCC150.slx` file, to open the Simulink project. Navigate to the top-level Simulink project and rename the *TransmitterBPSK* subsystem to *TransceiverBPSK*, since you will later add additional logic to handle received signals.

- You will need a pulse-shaping filter in your design, to reduce the high frequency content of the output signal. Add the following lines to your MATLAB setup script to calculate the filter coefficients and convert them to fixed-point values:

  ```
  % Filter coefficients
  filter.coefs = fi(rcosdesign(1, 6, sampleRate/dataRate), true, wordLength);
  ```

  While you're editing the settings files, also take to opportunity to change the data rate to `5e6`. This setting corresponds to an oversampling of the data by 8 times (`sampleRate/dataRate`).

- Inside the *TransceiverBPSK* subsystem, add an *InterpolatingFIR* block from the **DSP Builder for Intel FPGAs - Advanced Blockset/IP/Channel Filter And Waveform** library. Open its settings and set the **Input Rate per Channel/MSPS** to `dataRate/1e6`, **Interpolation** to `sampleRate/dataRate`, **Number of Channels** to `1`, and **Coefficients** to `filter.coefs`. Since you will not update the coefficients when running the system, you can also change **Read/Write Mode** to `Constant`, as this will reduce the hardware utilization. Click **OK** when finished. Connect the *InterpolatingFIR* to the *Modulator* submodule as shown in Fig. 3.

- Connect a *Scale* block, from the **DSP Builder for Intel FPGAs - Advanced Blockset/IP/Channel Filter And Waveform** library, after the *InterpolatingFIR* and set it up to output a `fixdt(true, wordLength)` signal. Set **Output scaling value** and **Multiplication factor** to `1`, and **Number of bits to shift left** to `0`. Connect a *Scope* at the output from both scale blocks and the filter and study the waveforms. Simulate the design and adjust the value of the **Number of bits to shift left** parameter of the *Scale* block to utilize as much as possible of the signal resolution that you have (`wordLength` = 12 bits), without distorting the waveform.

- Connect the `q` output of the *Scale* block to the `I_out` port. Since you are using BPSK modulation format the `Q_out` signal can be set to 0 by connecting a *Const* block to it. Make sure that you specify the data type of the *Const* block to signed representation using `wordLength` bits.

- Connect a two input *Scope* to the data input and output of the FIR filter, so that you can study it later. The finished transmitter signal path should look similar to Fig. 3.

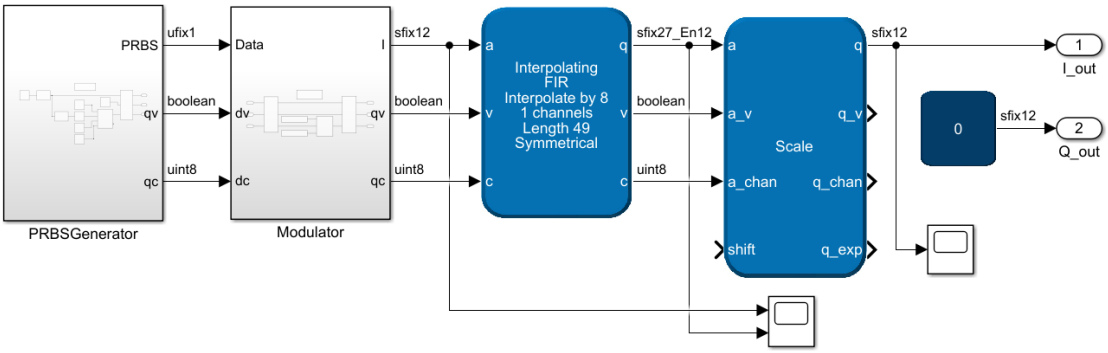


Figure 3: *Signal path of the transmitter.*

Reflection Questions

- Study the waveforms of the input and output of the FIR filter. Do they match the input data? How long is the delay between the input and the output and what causes this latency?
- Why is the wordlength of the output from the FIR filter 27 bits, when the input is 12?
- Why do we have to shift the output from the filter to utilize the full range of our 12 bit output?

3.2 Exercise 2 - Receiver Architecture

You will now implement a simple receiver in your DSP Builder project. The receiver structure contains a pulse-shaping filter and an STR unit, used to decimate the signal. See [1, Ch. 5] for a description of decimation.

- Start by adding two new input ports to the *TransceiverBPSK* subsystem and name these `I_in` and `Q_in`, then go to the top level entity (*txrx_bpsk*) and connect the `I_out` and `Q_out` signals from the *TransceiverBPSK* directly to the corresponding input on the same block. These connections will work like a loopback cable when testing your design in Simulink.
- Inside the *TransceiverBPSK*, add two *SingleRateFIR* filters from the **DSP Builder for Intel FPGAs - Advanced Blockset/IP/Channel Filter And Waveform** library and connect one input port to the `a` port of each filter. Connect a Boolean constant `true` to the valid input and a constant `uint8(0)` to the channel input of the filters. In the settings for both filters, set the **Input Rate per Channel/MSPS** to `sampleRate/1e6`, the **Number of Channels** to 1, **Coefficients** to `filter.coefs` and **Read/Write Mode** to `constant`.
- Add two *Scale* blocks after the filter and set their **Output data type** to `fixdt(true, wordLength)`. Connect a *Scope* to the output of the *Scale* connected to the I input and adjust the **Number of bits to shift left** to utilize as much as possible of the signal resolution without clipping the waveform. Use the same settings for the other *Scale* block. The receiver part should now look similar to Fig. 4.

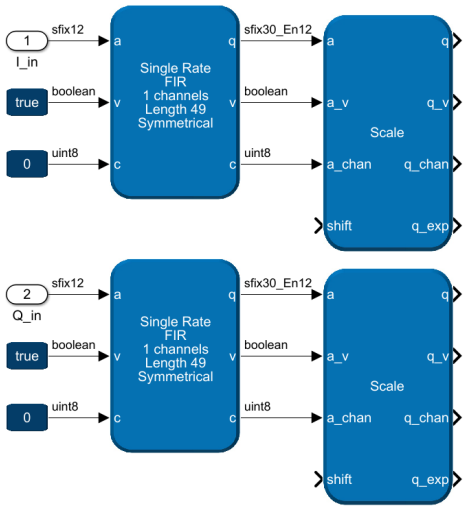


Figure 4: *Connection of the pulse shaping filters in the receiver.*

- The input signal to the receiver is oversampled with a factor of 8, and to recover the data we need to downsample (or decimate) the signals and select the best possible sampling point. The selection should be based on an external input called `SampleIndex`. Create an input port with this name in the *TransceiverBPSK* and in the top-level *tx_bpsk* Simulink project, connect a *Constant* from the **Simulink/Sources** to it. Edit your MATLAB setup file, by adding

  ```
  sampleIndex = 4;  % Sample index used in decimation
  ```

  and rerunning the script. Edit the settings for the newly added constant and set **ConstantValue** to `sampleIndex`. Switch to the **Signal Attributes** tab and set **Output minimum** to `0` and **output maximum** to `sampleRate/dataRate - 1`. Change the **Output data type** to `fixdt(false, ceil(log2(sampleRate/dataRate)))`. These settings will make sure that the `SampleIndex` input is of the correct type and limit the input values to valid samples. Press **OK** when your are done.

- Enter the *TransceiverBPSK* subsystem and add two new output ports named `Symb_I` and `Symb_Q`. Add a new subsystem and name it `Decimation`. This new subsystem needs five inputs: `I_in`, `Q_in`, `SampleIndex`, `dv`, and `dc`. It also needs four outputs `I_out`, `Q_out`, `qv` and `qc`. Go back out to the *TransceiverBPSK* subsystem and connect the `I_in` and `Q_in` ports of the *Decimation* block to the data outputs of the *Scale* blocks and the outputs to the `Symb_I` and `Symb_Q` ports. Connect the `qv` and `qc` ports from one of the *Scale* blocks to the corresponding input of the *Decimation* block. The resulting connections should look similar to Fig. 5.

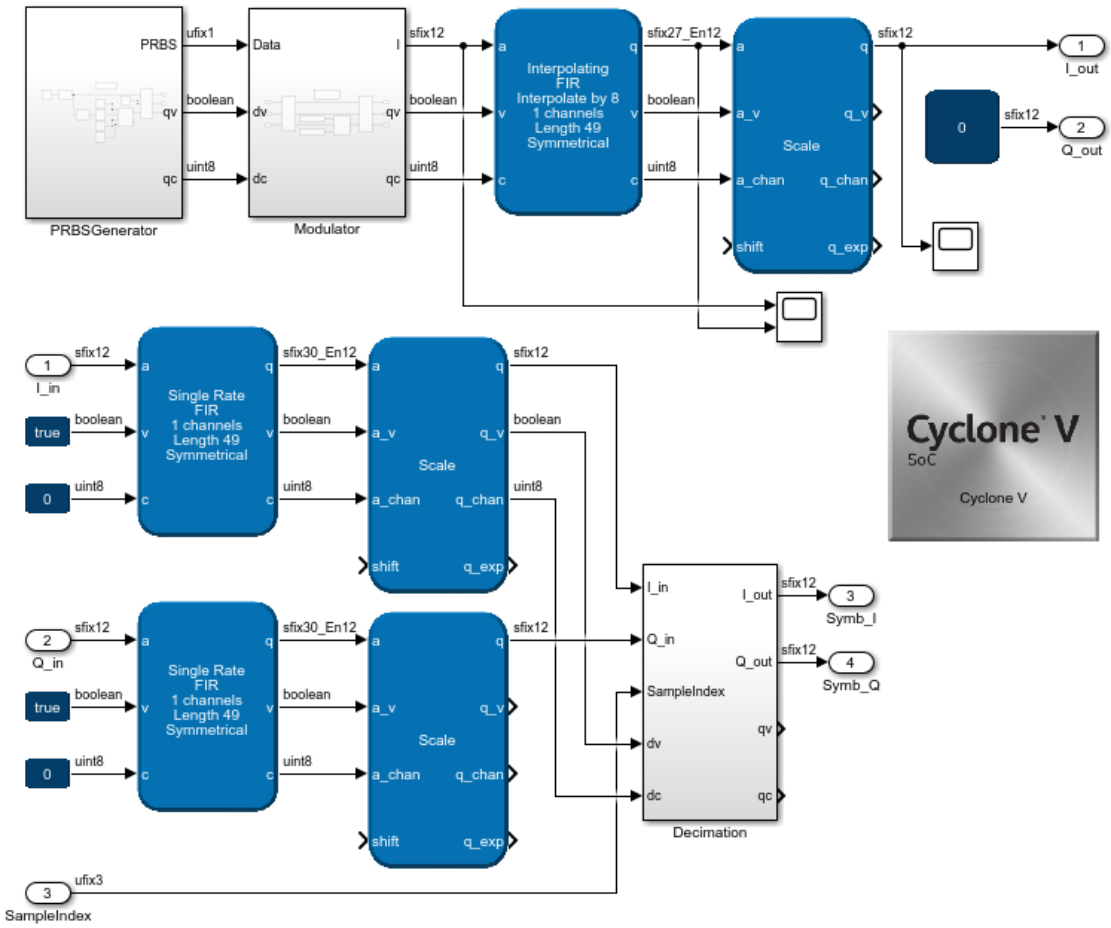


Figure 5: *Complete design of the TransceiverBPSK subsystem.*

- The idea with the `SampleIndex` signal is to select which sample point to use and you will now build the *Decimation* subsystem to achieve this. Double click the *Decimation* subsystem to edit it and a *ChannelIn*, a *ChannelOut* and a *SynthesisInfo* block. Change the settings of the channel blocks to reflect the number of inputs and output that you have, and connect them to your ports.[1]

- Add a *Counter* from the **DSP Builder for Intel FPGAs - Advanced Blockset/Primitives/ Primitive Basic Blocks** library. The counter is going to be used to index the samples and should start counting as soon as the input is valid, so connect the input port of the *Counter* to the `dv` input port. Edit the parameter settings and change **Counter setup** to `[0 sampleRate/dataRate 1]`. Connect a *SampleDelay* to the output of the counter to make sure that the first valid sample is indexed as 0.

- Add an *CmpEQ* block from the **Primitive Basic Blocks** library and connect one of its inputs to the *SampleDelay* and one to the `SampleIndex` input port. The output of this block will be a Boolean that is true when the delayed counter value and the `SampleIndex` are equal. You will

[1] There is a bug in DSP Builder that sometimes prevents the *ChannelIn* and *ChannelOut* blocks from updating when you have changed the number of ports. If this happens, double click the block again to open its parameter settings and the blocks should update.

use this Boolean as a new valid signal, so connect the output of the *CmpEQ* block to the `qv` output. Also connect a `uint8` constant with the value `0` to the `qc` output.

- You will now make sure that the `I_out` and `Q_out` output the selected sample. This can be achieved by using a selector that selects between keeping the old value at the output or updating the value when the valid signal is high. Add a *Select* block from the **Primitive Basic Blocks** library and change the **Number of cases** to `1`, the **Output data type** to `fixdt(true, wordLength)`, and the **Output Scaling** to `1` in its settings. Study the information about the *Select* block in the parameter settings and make sure that you understand how it works. More information is available in [2].

- Add a *SampleDelay* and connect the output of the *Select* block both to the input of the *SampleDelay* and to the input of the *ChannelOut* connected to the `I_out` port. You can flip the *SampleDelay* by right-clicking it and selecting **Rotate & Flip** → **Flip Block** for a cleaner layout. Connect the output of the *SampleDelay* to the `d` input of the *Select* block. Connect the valid signal at the output of the *CmpEQ* block to the `0` input, and the output of the *ChannelIn* block corresponding to the `I_in` port to the `a` input of the *Select* block. Repeat the same process for the Q signal, preferably using copy & paste.

- Add a *Demux* block and set the number of **Output Channels** to `[0:sampleRate/dataRate - 1]` and the **Number of Input Channels** to `sampleRate/dataRate`. Connect the `q` input of the *Demux* to the wire connected to the output of the *ChannelIn* corresponding to the `I_in` port, and `v` to the `qv` output of the *ChannelIn*. The channel select signal (`c`) can be taken from the *SampleDelay* succeeding the counter. You must however convert it to an `uint(8)` using a *Convert* block from the **Primitive Basic Blocks** library. Finally, set up a *Scope* to handle eight channels and connect it to the outputs of the *Demux*. The final design should look similar to Fig. 6

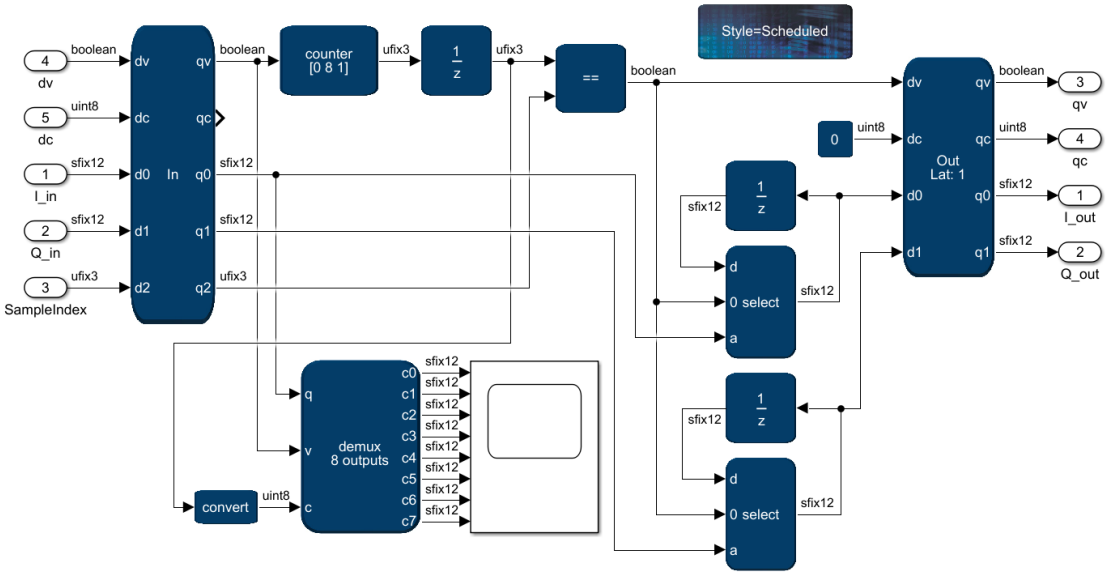


Figure 6: *Design of the Decimation subsystem.*

- Set the **View** → **Layout** of the *Scope* to 4×2 and start a simulation. The *Scope* should now show you the downsampled waveforms and how they differ depending on which sample you choose.

- You are now ready to connect your channel model in the loopback path in your top-level Simulink project. Remove the wires connected to the input of the *TransceiverBPSK* and add a *Convert* block. In the *Convert* parameters, change **Output minimum** to `-(2^(wordLength-1)-1)`, **Output maximum** to `2^(wordLength-1)-1` and **Output data type** to `fixdt(true, wordLength)`. Add a *Complex to Real-Imag* block and connect it as shown in Fig. 7. Finally, add a *Real-Imag to Complex* block and a *To Workspace* block and connect them to the `Symb` outputs of the transceiver. Set the **Variable name** of the *To Workspace* block to `SymbSave`

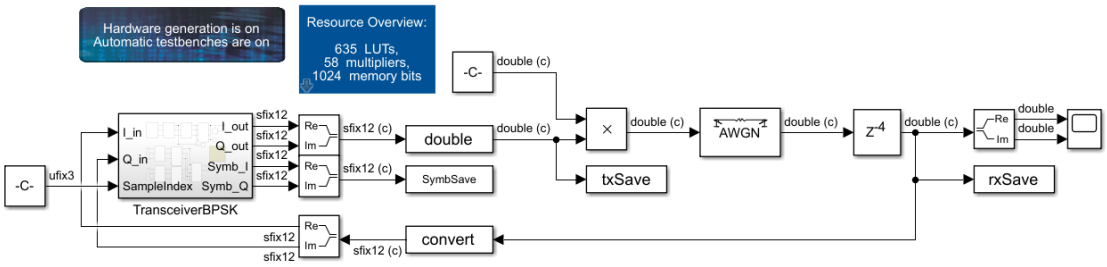


Figure 7: *Connection of the channel model to the transceiver.*

- Set the `chan.snr` to `30`, `chan.phase` to `pi/2*rand() - pi/4`, and `chan.dly` to 4. Find the best sampling point by varying the `SampleIndex`. Run a simulation with that value of `SampleIndex` and save your workspace in MATLAB using `save('exercise2')`. You will need this file for the home assignment.
- Verify with your TA that the output from the previous step is correct.

Reflection Questions

- In what way is the optimal `SampleIndex` value affected by the channel delay, which you set by adjusting the `chan.dly` variable?
- How can you be sure that two *Scale* blocks at the receiver side are set properly? What values did you choose for the **Number of bits to shift left**?

3.3 Exercise 3 - In-System Sources and Probes

You will now import your transceiver design into Quartus Prime, using the same method as in the previous lab. In this exercise you will also familiarize yourself with the **In-System Sources and Probes Editor**, which can be used to control internal FPGA signals from Quartus Prime. For the items marked with a red bullet (•), you will need to have access to a computer connected to the DE10-Standard board. We have provided two servers, with connected hardware, which you can access remotely. See Canvas for instructions on how to book your time slot and how to connect to these machines. More detailed instructions for how to set up SignalTap is provided in the instructions for the first lab.

- It's now time to import your transceiver system into Quartus Prime. Download the project template from Canvas and import your DSP Builder generated HDL descriptions in the Quartus project by performing the same steps as you did in Exercise 2 in the first lab. Use your new transceiver system instead of the simple transmitter used in the first lab, the connections are shown in Fig. 8. Synthesize your design and open the SignalTap menu. Set the SignalTap clock to the `clk_40M` output from the AD9361 block, the sample depth to `2 K`, and to sample the `I_out`, `Q_out`, `I_in`, `Q_in`, `Symb_I` and `Symb_Q` signals of the *TransceiverBPSK* block. Save the SignalTap settings and answer **Yes** when asked if you want to enable SignalTap.

- You can now study the output of the system, but you also need to be able to adjust the `SampleIndex` signal while the FPGA is running. You can either connect the input to the switches on the DE10-Standard board or, as you will be doing here, use the **In-System Sources & Probes** IP block. This block is added to the project by locating it in the **IP Catalog** in the right-most pane in Quartus. It is found under **Basic Functions** → **Simulation; Debug and Verification** → **Debug and Performance** → **Altera In-System Sources & Probes**. Double click on the name and the **IP Parameter Editor** should open. Change **Entity name** to `DebugSource` and click **OK**. The block can work both as a signal source and as a probe. In this lab, you will only use the source functionality, thus set the **Probe Port Width** to `0`. Set the **Source Port Width** to `3`, since the `SampleIndex` port is 3 bits wide, and click the **Generate HDL** button and then **Generate** in the **Generation** window. This will start the generation of the IP. Once the generation is completed, close the pop-up and the **IP Parameter Editor**.

- You must now add the generated IP block to your project. Change the view to **File** in the **Project Navigator** pane, right-click **Files** and choose **Add/Remove Files in Project**, press the **...** button and select the file `DebugSource/synthesis/DebugSource.qip`. Open the schematic of your top-level design (*MCC150_top*), add the *DebugSource* symbol and connect it to the the `SampleIndex` input of the *MCC150_TransceiverBPSK*. Rename the instance to `DebugSource_inst` and save the schematic, which should now look similar to Fig. 8.

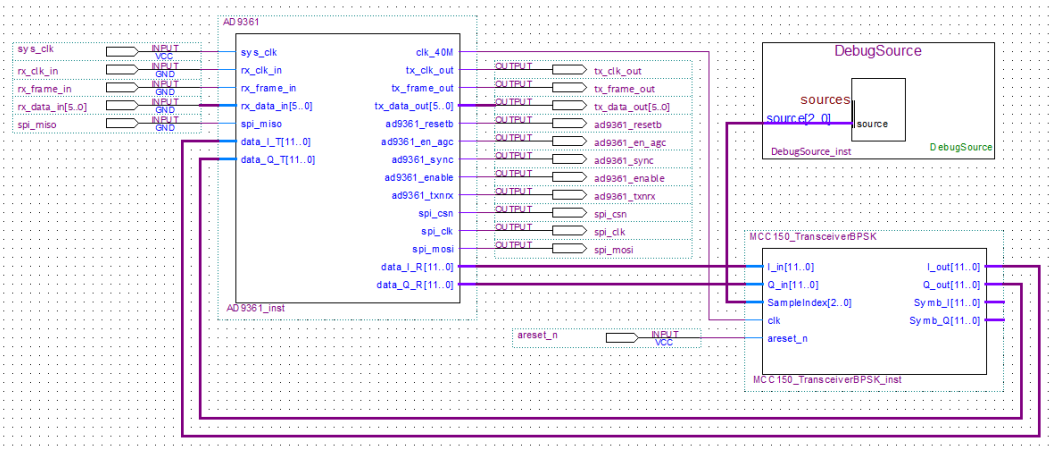


Figure 8: *Schematic of the top-level MCC150_top design.*

- Compile the design by selecting **Processing** → **Start Compilation** and wait for the process to finish.

- Program the FPGA using the SignalTap Logic Analyzer, in the way described in the instructions for lab 1, and start a continuous capture. You should now see the transmitted and received waveforms, as well as the downsampled I/Q signal.

- To be able to adjust the `SampleIndex` signal, you need to start the **In-System Sources and Probes Editor** from the **Tools** menu in the original Quartus window. The editor will probably complain multiple times that **No instances found...**, just press **OK** if this window appears. Select `DE-SoC` in the **Hardware** drop-down list and `5CSEBA6` in the Device list, the **Instance Manager** should now say **Ready to acquire**.

- To reduce the amount of traffic on the JTAG bus used to communicate with the FPGA, change **Write source data** to `Manual`. The list at the bottom of the screen shows the source signals

available in our system. Right click on the signal name (`source[2..0]`) and select **Bus Display Format** → **Unsigned Decimal**, since that is the type of the `SampleIndex` signal. You can now change the **Data** value in the bottom list and send that value to the FPGA by pressing the **Write Source Data** button, circled in red in Fig. 9.

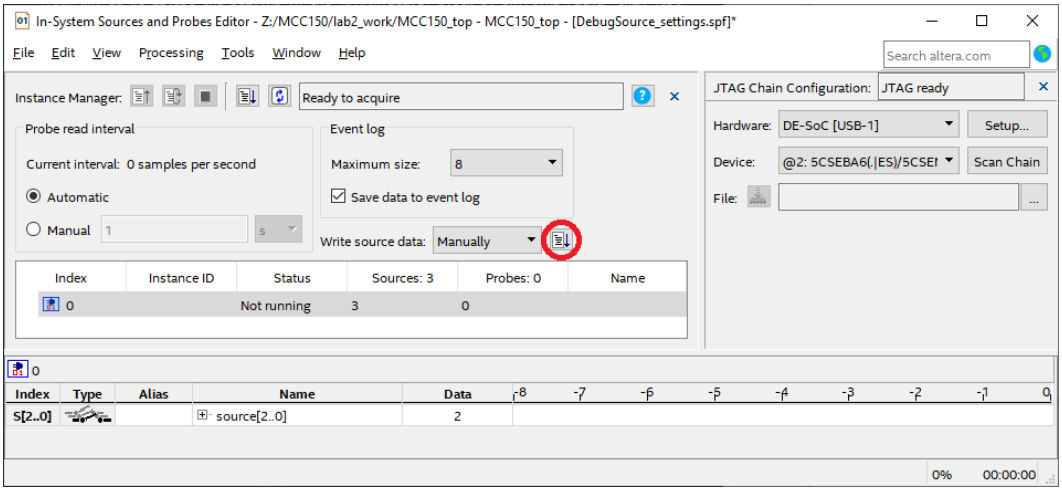


Figure 9: *In-System Sources and Probes Editor settings*

- Try all possible values for the `SampleIndex` signal, capture the corresponding waveforms in SignalTap and export each value in a separate `.csv` file. You will need these files for the home assignments. Don't forget to click **Write Source Data** in the **In-System Sources and Probes Editor** for each changed value, and to capture new SignalTap data for each run.
- Read through the home assignment below, and make sure that you have generated the data files and screenshots you need to write the report.

4 Post-Lab Home Assignment

The following tasks should be performed before the next lab session, and a lab report should be handed in on Canvas before 23:59 on Sunday, April 25. The report should be maximum 4 pages, and should contain descriptions of the blocks designed in DSP Builder and their functionality, answers to the reflection questions at the end of Sections 3.1–3.2 in the lab instructions, and solutions to the home-assignment questions below, including the resulting figures. When handing in your report, you will need to submit both a `.pdf` version of your report and a `.zip` archive containing your DSP Builder and Quartus projects.

- Show a **SignalTap Logic Analyzer** window. Include the following captured signals: `I_in`, `I_out`, `Q_in`, `Q_out`, `Symb_I`, `Symb_Q`, output signals from the two *Single Rate FIR* filters and output signals from the two *Scale* blocks at the receiver side. Use the Signed Line bus display format for signals where this is applicable. Explain the differences between these signals.
- Load the data saved in Exercise 2. Plot eye diagrams for the received data (`rxSave`) and plot the data processed by the transceiver (`SymbSave`) as a function of time. What is the relationship between these two signals?
- Load all `.csv` files into MATLAB and plot the down-sampled signals (`Symb_I` and `Symb_Q`) as a function of time. Which one contains the best sampling points?

- Load one of the `.csv` files containing the received BPSK signals into MATLAB and plot an eye diagram of the oversampled received signal (`I_in` and `Q_in`). Find the best sampling point and calculate which sample that is. Explain how you can find the best sampling point in a hardware implementation.
- Estimate the phase offset between the transmitted (`I_out` and `Q_out`) and received signals (`I_in` and `Q_in`). Explain how do you estimate the phase offset.

References

[1] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 4th ed. Berlin, Heidelberg: Springer, 2014.

[2] Intel Corporation, *DSP Builder for Intel FPGAs (Advanced Blockset) Handbook*, 2018.