MCC150 - Implementation of Digital Signal Processing Systems

Lab 3: Symbol Timing Recovery

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

Version 1.3 - April 16, 2021

Contents

1 Introduction

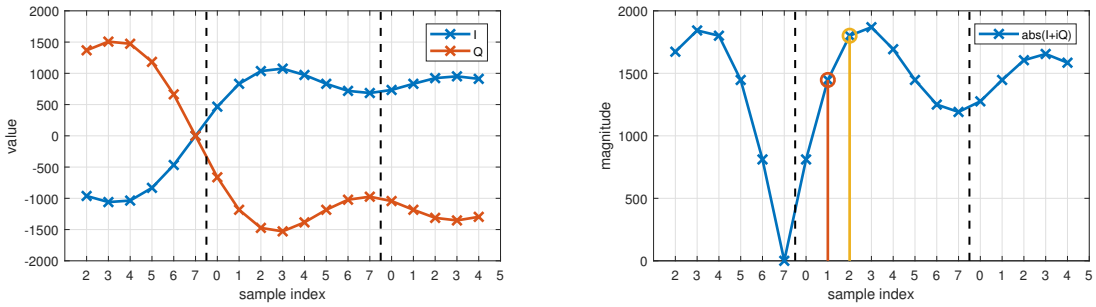
In this lab session you will continue expanding your BPSK transceiver design, by adding support for symbol timing recovery (STR). Previously you set which sample to keep in the decimation unit by manually adjusting an input, but now you will add DSP modules to calculate the best sampling point. You should read the Pre-Lab Preparation section below, and finish the short assignments included there, before you start working on the lab assignments. The instructions in the lab section will be less detailed from now on, since you should know the basics of DSP Builder and Quartus by now.

2 Pre-Lab Preparation

There are many different algorithms available to estimate the best sampling point for BPSK transmissions, such as those described by Gardner [1], and Mueller & Müller [2]. In this lab, you will use a less complex approach where the symbol which has the largest magnitude is selected. Since the I/Q signals can be thought of as complex numbers, this amplitude can be calculated as

$$M = |I + iQ| = \sqrt{I^2 + Q^2}. \tag{1}$$

The magnitude for the signals pictured in Fig. 1a is shown in Fig. 1b, where the maximum amplitude is found at a sample index of 3 for all three oversampled symbols shown. If the red marker in Fig. 1b represents the magnitude of the currently selected sampling point (M_i), it is enough to study the adjacent, yellow, point (M_{i+1}) to calculate if the index i of the selected point should be increased or decreased. If $M_i < M_{i+1}$, the index is increased, and if $M_i >= M_{i+1}$, the index is decreased.



(a) **(b)**

Figure 1: *Graphs showing (a) oversampled I/Q signals, and (b) the magnitude of these samples.*

The signals in a real system will always contain noise, which will affect the magnitudes. The noise can typically be described as additive white Gaussian noise (AWGN), which has a zero mean. Thus, a common way to reduce the impact of noise on the index calculation is to calculate the average magnitude over a number of symbols, and use that average to select the sampling point.

Pre-lab tasks

- When calculating the average of L unsigned values with a word length of N, you will first need to sum L values. How many bits are needed to store this sum in order to avoid overflow?
- After summing the values you need to divide by the number of values, L. Division is typically very hardware intensive, and can be reduced to a right shift if you are dividing with a power of 2. Assuming that L has this property, how would you calculate the number of steps to shift?

3 Lab

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

3.1 Exercise 1 - Updated Decimation and Magnitude Calculation

In the first lab exercise, you will update your *Decimation* subsystem to support STR, and create a subsystem used to calculate the magnitude of the received, complex I/Q samples.

- Start by copying the design files from the last lab into a new folder named `lab3`. In order to find the best sampling point, you will need the modify the *Decimation* subsystem so that it outputs both the selected symbol and one of the adjacent symbols. In this design we choose the adjacent symbol with index $i+1$, if i is the index of the selected symbol. Enter the decimation subsystem and add two new output ports, name them `I_adj` and `Q_adj`. To clean up your design, put the two *Select* blocks with corresponding *SampleDelays* into a subsystem by selecting them, right clicking on one of the blocks and choose **Create Subsystem from Selection**. Name the subsystem `Update1` and the ports `Update`, `I_in` and `Q_in`.

- Create a new subsystem called `Increment` inside the *Decimation* subsystem with one input and one output, named `In` and `Out`. Connect the input to the `SampleIndex` input of the *Decimation* subsystem. The *Increment* subsystem should calculate the adjacent index, i.e. $i+1$, based on the `In` input. Both the input and the output should be of the type

  ```
  fixdt(false, ceil(log2(sampleRate/dataRate)))
  ```

  and the output should wrap-around, so that when `In = sampleRate/dataRate-1` the output should be 0. Design the block yourself, using blocks from the **Primitive Basic Blocks** library. Make sure that it works for all possible oversampling ratios, i.e. all positive integer values of `sampleRate/dataRate`.

- Connect the `I_adj` and `Q_adj` output so that they output the sample at the position defined by the output of the *Increment* block. An example of how this can be done is shown in Fig. 2.

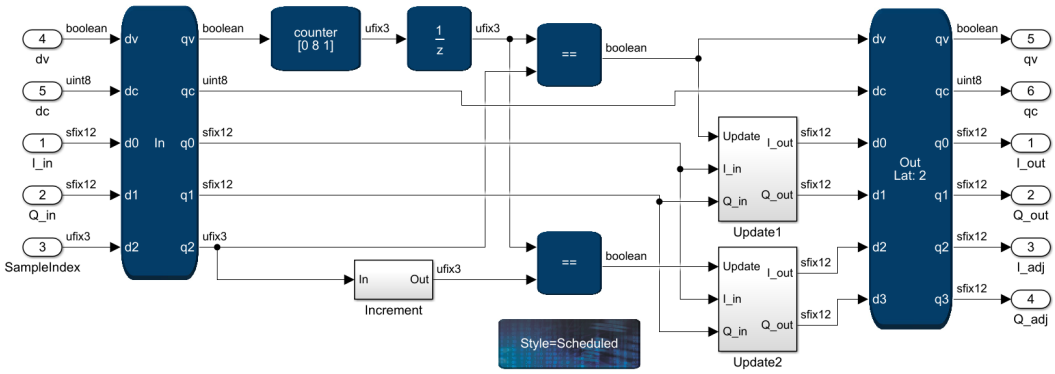


Figure 2: *View of the updated Decimation subsystem.*

- You will now create a block that calculates the magnitude of the complex I/Q symbols. Add a new subsystem and name it `Magnitude1`. This subsystem should have four inputs (`I`, `Q`, `dv` and `dc`) and four outputs (`Mag`, `qv` and `qc`). Add the mandatory *ChannelIn/Out* and *SynthesisInfo* blocks to the subsystem, and connect the `dv` and `dc` input signals directly to the corresponding outputs. Build the system as shown in Fig. 3 and make sure that the output type of the `Mag` signal is an unsigned of `wordLength` bits.

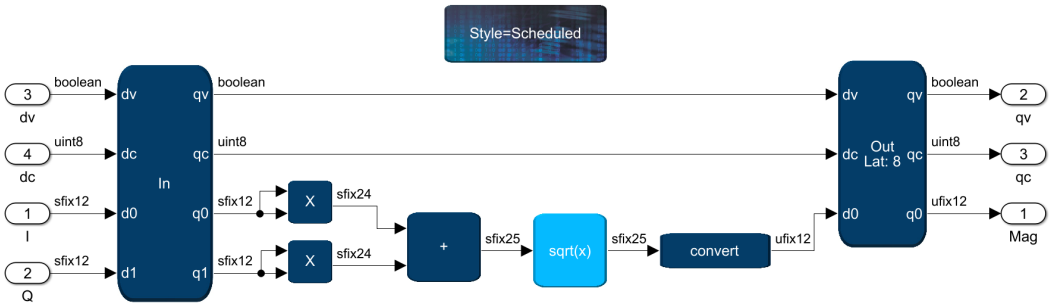


Figure 3: *Contents of the Magnitude1 subsystem.*

- Create a copy of the *Magnitude1* subsystem, and connect the first one to the `I_out` and `Q_out` signals from the *Decimation* subsystem, and the second one to the `I_adj` and `Q_adj` signals. The complete system is shown in Fig. 5, for reference.

Reflection Questions

- How can the *Increment* subsystem, which you designed inside the *Decimation* subsystem, be simplified if the oversampling ratio is limited to powers of 2, i.e. $2, 4, 8...$?
- The result from the magnitude calculation is converted to an unsigned 12-bit number. Can you be sure that the result will always fit in this data type?

3.2 Exercise 2 - Average Calculation

You will now create a subsystem used to calculate the average magnitude of the selected symbol and its adjacent symbol. This subsystem will use a block averaging method, as opposed to a moving average. This means that you will calculate the average for a block of samples, within a set window, and update the output every time a window is complete. The idea is to use an adder to add each new incoming magnitude to the value of a sum register. This register is then reset each time a new window starts.

- Define a variable in your setup script to store the length of the averaging window:

  ```
  str.averageLength = 64; % Length of STR averaging window
  ```

  Go back to the *TransceiverBPSK* subsystem and add a new subsystem called `AverageMagnitude1`. It should, in addition to the standard valid and channel ports, have one input called `Mag` and one output called `AvgMag`.

- Inside the *AverageMagnitude1* subsystem, you will first need to create an enable signal, which can be used to reset the sum register, to update the output and as a valid signal. Create a *Sequence* with the parameter `[0 str.averageLength-1 str.averageLength]` and connect its input to the valid output of the *ChannelIn* block. This *Sequence* will create a signal that goes high after `str.averageLength-1` pulses on the valid input and low again on the next pulse. To make sure that the signal is high for one clock cycle only, connect its output to an *And* gate, whose other input is tied to the `qv` signal from the *ChannelIn*.
- Construct the rest of the circuit according to Fig. 4. The only blocks that are new are the latches, which can be found in the **DSP Builder for Intel FPGAs - Advanced Blockset/Primitives/Primitive Design Elements** library. Make sure that you don't get overflow

in your addition by setting the data type of the output according to the results from your pre-lab calculations. Do the same for the constant controlling the number of bits to shift, marked **homeAssignment** in Fig 4. The data type of the output from the shift operation should be `fixdt(false, wordLength)`.

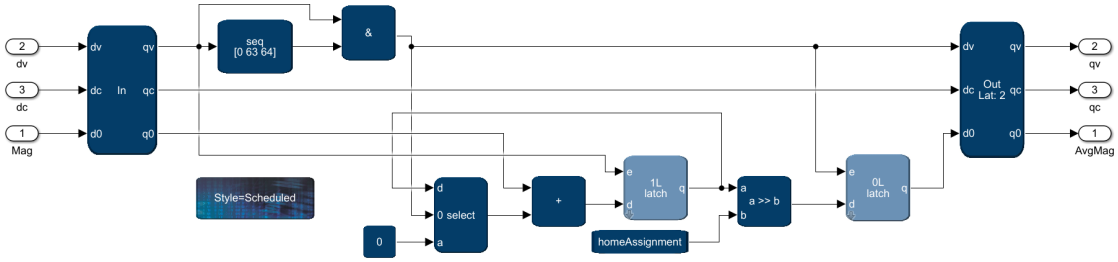


Figure 4: *Contents of the AverageMagnitude subsystem.*

- Go back out to the *TransceiverBPSK* and create a copy of your *AverageMagnitude1* subsystem. Connect one of the copies after each *Magnitude* subsystem, as shown in Fig. 5.

Reflection Questions

- Study the content of the *latch_0L* by pressing the small down arrow in its bottom left corner. What is the difference between this latch and *latch_1L*? Why do we need the *latch_1L* in this block? Can it be removed?
- The division in your average calculation is implemented as a right shift. What type of limitations does this solution incur when selecting the size of the averaging window, compared to using division? What benefits do this solution have?

3.3 Exercise 3 - Update Sample Index

You will now build a subsystem used to update the selected sample index. It compares the average magnitude of the selected symbol (`AvgMag_out`) with the average magnitude of the adjacent symbol (`Mag_adj`). If `AvgMag_out < AvgMag_adj` the index is increased one position, and if `AvgMag_out >= AvgMag_adj` the index is decreased by one.

- Create a new submodule named `IndexUpdate` with the standard valid and data ports, two input ports named `AvgMag_out` and `AvgMag_adj`, and one output called `SampleIndex`. Connect the submodule as shown in Fig. 5. Don't forget to add the *SampleDelay* in the feedback loop to the *Decimation* unit; it's needed by DSP Builder to determine the timing of the design. Add the necessary *SynthesisInfo* etc. blocks to your new submodule.

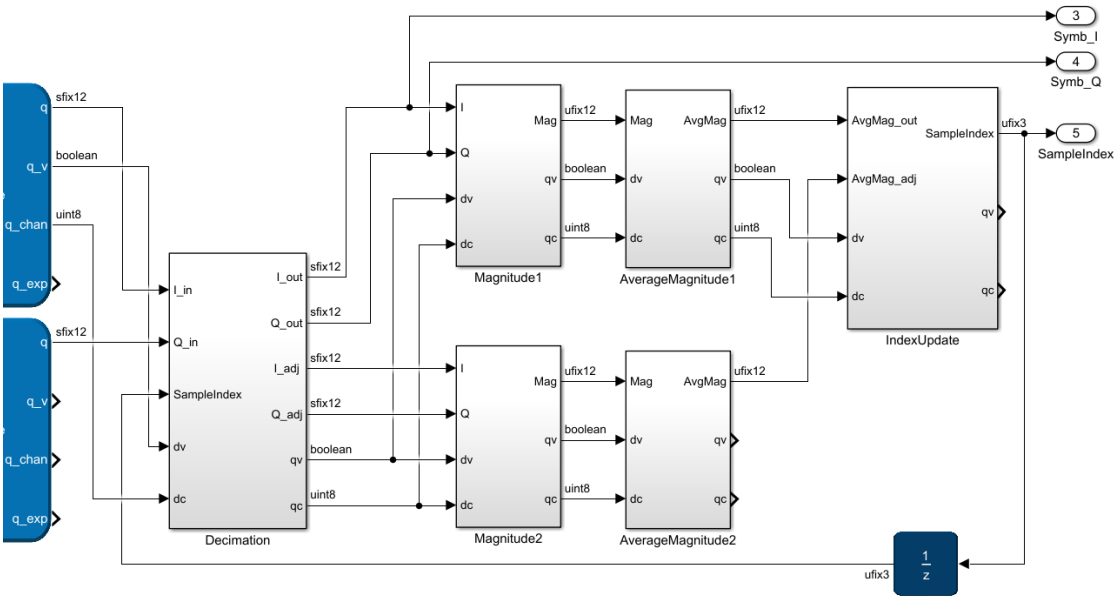


Figure 5: *Overview of the STR part of the receiver design.*

- Copy two instances of the *Increment* block, from the *Decimation* subsystem, to the *IndexUpdate* subsystem. Rename one of the blocks `Decrement` and modify it so that it subtracts one from the index. You will need to manually make sure that it wraps around to the correct value when the input is 0.
- Add a *CmpLT*, a *Mux*, a *SampleDelay* and a *latch_0L* block, and connect them according to Fig. 6. The *Mux* will select which of the `d0` or `d1` inputs that will be connected to the output, based on the value of the `s` input. Open the parameter settings for the *SampleDelay* and check the **Minimum delay** box. This setting will let DSP Builder automatically choose a delay so that the design can be pipelined properly.

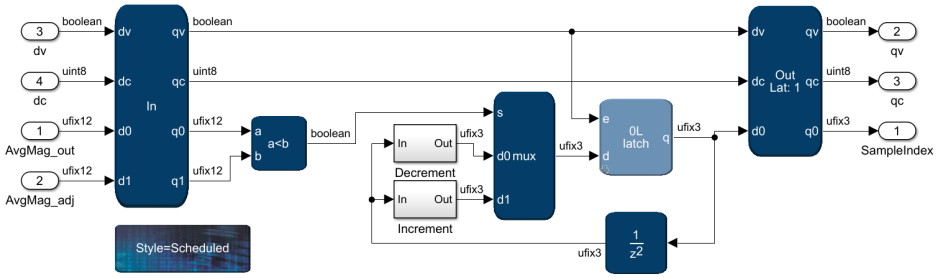


Figure 6: *Contents of the the IndexUpdate subsystem*

- Go out to the top-level *MCC150* project, add a *ToWorkspace* block, and connect it to the SampleIndex output port of the *TransceiverBPSK*. Connect a single channel scope to the same output and verify that the system can find a good sampling point for different settings of the `chan.dly` parameter.
- You can now check what parts of your STR design that utilize the most FPGA resources. Verify that **Hardware generation** is turned on in the *Control* block and rerun a simulation. Select **Resource Usage** → **Design** from the **DSP Builder** menu and expand the

TransceiverBPSK folder. Study especially the number of LUTs, multipliers and memory bits used. In the right-most column you can also see the latency of each processing block, in number of clock cycles.

- Verify with your TA that the output from the *IndexUpdate* subsystem is correct.

Reflection Questions

- Considering the resource usage of your design, where would you put your efforts in order to optimize the system?
- What operations inside the *Magnitude* subsystems are likely to consume the largest amount of FPGA resources?
- Once your STR system has found the best sampling point, the `SampleIndex` is not kept constant but shifts up and down between the two closest indices. Why is that, and what can you do to fix it?

3.4 Exercise 4 - Synthesis and Verification

In this exercise you will upload your design to the FPGA and verify that your system works by using the SignalTap Logic Analyzer. 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.

- Import your DSP Builder design into the Quartus project template provided on Canvas, using the same method as in the previous labs, and synthesize the design.
- Set up SignalTap to record the binary data stream generated in the PRBS generator, the `I_out` and `Q_out` signals from the transmitter, the `I_in` and `Q_in` signal coming in to the receiver, and the `Symb_I`, `Symb_Q` and `SampleIndex` signals at the output of the receiver. Use the `clk_40M` from the `AD9361` as the SignalTap clock. Compile the design when you have completed the SignalTap settings.
- Upload the design to the FPGA and capture data from a SignalTap run. Take a screenshot of the SignalTap window and save the data as a `.csv` file; you will need these files for the home assignment.

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, May 2. The report should be maximum 4 pages long, and should contain answers to the pre-lab questions, descriptions of the blocks designed in DSP Builder and their functionality, answers to the reflection questions at the end of Sections 3.1–3.3 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 project.

- Show a screen shot of the SignalTap window with the captured signals mentioned in Exercise 4. Explain what each signal represents.
- Plot eye diagrams of the received signals and plot the signals after STR as a function of time, using the SignalTap data. Do the signals match what you expect to see compared to the DSP Builder simulation?

- Plot a graph of the `SampleIndex` signal captured using SignalTap. Does it match the simulation?
- Optimize the *Magnitude* subsystems so that they consume less resources. Explain your solution and show the resulting resource utilization and latency in your report.
- Update the STR system so that it keeps the `SampleIndex` signal constant once the optimum sampling point is found. Show your solution and its effect on the resource utilization and latency.
- How will the choice of `str.averageLength` affect the quality of the STR estimation and the resource utilization of your system? How do the SNR of the input signal affect the best choice of averaging length?

References

[1] F. Gardner, "A BPSK/QPSK timing-error detector for sampled receivers," *IEEE Transactions on Communications*, vol. 34, no. 5, pp. 423–429, 1986.

[2] K. Mueller and M. Muller, "Timing recovery in digital synchronous data receivers," *IEEE Transactions on Communications*, vol. 24, no. 5, pp. 516–531, 1976.