

A Tutorial on our Floating-point to Fixed-point Conversion (FFC) Tool --on a BPSK Communication System

By Changchun Shi

Last Updated: March 25, 2004

Berkeley Wireless Research Center

EECS Department, University of California, Berkeley

1. Abstract

A simple BPSK communication system is built using root-raised-cosine filters for both its transmitter and receiver sides, which depicts the design process using Simulink and Xilinx System Generator. Starting from choosing the algorithm and building the floating-point system with architecture information, the system is converted into fixed-point using our floating-point to fixed-point conversion (FFC) tool.

2. Motivation

Can we design a digital chip in a day? Research efforts in Berkeley Wireless Research Center (BWRC) and other places have indicated this is achievable. Built on top of MatlabTM, SimulinkTM and Xilinx System GeneratorTM, a number of customized Matlab scripts and Simulink libraries automate our FFC design flow [Shi03] [Shi04_1-7]. By studying on how our FFC tool can be used in designing a simplified transmitter-receiver system, this tutorial will get you familiarized with it as well as the design environment.

3. How to Start

This section of the tutorial may change over time as it is related to some administrative information that changes often. For now, it is assumed that you have a BWRC account to access all the software resources on the remote servers. Alternatively, you can download our source codes and use them at your local machine--please refer to our website for up-to-date installation information [Shi_web]—and you should skip to the next section.

You will need a computer with Matlab, Simulink, Xilinx System Generator (version 2.3 or higher) installed in order to run through this tutorial. You also need read/execute access to BWRC file server [\hitz.eecs.berkeley.edu/designs](http://hitz.eecs.berkeley.edu/designs) to use our Floating-point to Fixed-point Conversion (FFC) Tool. In addition if you want to learn how to map your design to FPGA or BEE, you need to refer to other tutorial such as System Generator Tutorial or the [tutorial on \(Berkeley Emulation Engine\) BEE](#). To map to ASIC, please refer to Tutorial on Insecta.

A simple way to solve the problem is to login the MS Windows Remote Desktop Servers available in BWRC, intel2650-{1, 2, 3}.eecs.berkeley.edu or any other ones with the required software installed. You will need Remote Desktop Connection Client on your local PC to do that. If you are using Linux, you may use Rdesktop [Rdesktop]. Each of the servers has all the necessary tools installed correctly.

Once you have the software ready, you need to map [\\hitz.eecs.berkeley.edu\designs](http://hitz.eecs.berkeley.edu/designs) to your network drive, preferably to “H:” disk.

The example communication system for this tutorial can be found at H:\ffc\systems\ffc_tutorial\ if System Generator installed is version 2.3. Or, if you have version 3.1 or higher, H:\ffc\systems\ffc_tutorial_sg3\. To check the version of System Generator for the server, use

```
>>xlversion
```

at Matlab command window.

If you have never used any one of the Matlab, Simulink and System Generator before, you should read Appendix A of this tutorial before you proceed.

4. Algorithm Study

This section is attempted to get you familiar with floating-point algorithm design. You should skip to the next section if you are only interested in FFC tool usage.

Suppose we want to do base-band communication with 2-PAM modulation scheme at 1Mbits/sec. Under 2-PAM input symbols (1 symbol/ 1us), such as a sequence formed by binary integer {0,1} at each sample, are mapped into a data sequence choosing from {-A, A}. For convenience, we can let $A = 1$. The receiver needs a 2-PAM demodulator to map a received signal into the original integer. For simplicity, let us assume that the channel imposes additive white Gaussian random noise, otherwise it is ideal.

In other words, the assumption says the channel is flat with no fading. In reality, it could be band-limited (may also due to RF front-end filtering); thus, a rectangular base-band pulse in time domain (and SINC shape in frequency domain) through the channel will be clearly distorted. One technique to combat this none-flat band is to have a low-pass pulse-shaping filter on the transmitter side [Proakis01]. To do so, one needs to first over-sample the data sequence at R-MHz. This is usually done by an upsampler with an integer upsampling rate R. A condition $R > 2$ is necessary to satisfy Nyquist criteria.

However, there are multiple reasons to make R even higher. One of them is to minimize the impairment on the frequency response due to the finite-tap structure implementation of the filter, which causes non-zero stop-band frequency response and hence aliasing when the received signal is downsampled. This is what usually called inter-symbol-interference (ISI) [Proakis01]. Without higher upsampling rate R this impairment can be alleviated with the cost of increasing the filter complexity and signal latency. Another reason to have large R is for time and frequency recovery. When the channel together

with RF front-end has a fractional delay of symbol period, one needs upsampled sequences to find out the right fractional delay “adjustment” the receiver needs to tune [Chi02, Proakis01].

On the other hand, an R that is too high will result high clock frequency for the pulse-shaping filters, A/D and D/A converters, which causes implementation difficulty. In our case, without further justification, let us make an engineering decision of $R=4$.

On the receiver side, it is desirable to have a matched filter that matches the pulse-shaping filter on the transmitter side. With this consideration in mind, a commonly used filter shape, called root-raised-cosine filter is used in our system. After the downsampler on the receiver side, the signal will be perfectly reconstructed if the two root-raised-cosine filters are ideal. Figure 1 shows the algorithms we have conceived so far.

It should be pointed out that if the channel is really as simple as being AWGN, one can just feed the 2-PAM-modulated signal into the channel. We included these filters to combat some other possible channel impairments that are not going to be present in our simulation model. In this way, the system becomes complicated enough to take advantage of our FFC, at the same time it is not too complicated to understand.

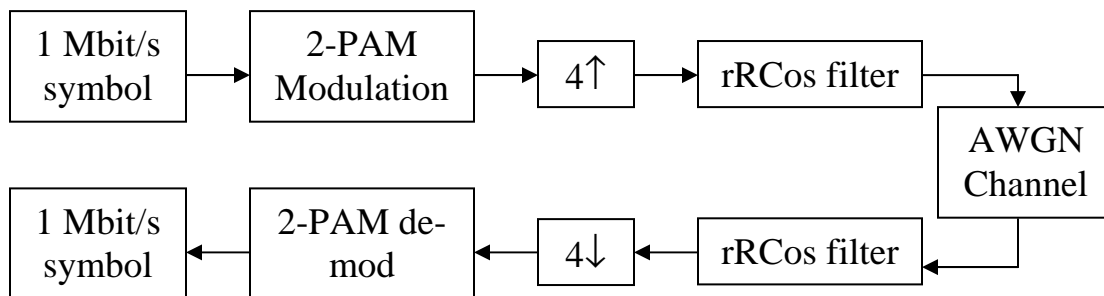


Figure 1. A simple based-band digital communication system

5. Building Floating-point System – Algorithm Validation

Fig. 1 shows the block-diagram of our conceived system so far. We can start to write either C or Matlab codes for each of the blocks, and see if the output symbols agree with the input ones by doing simulations. This is conventionally what people would do. This is still a good way to understand a system, and we encourage you to do so. However, there is a more natural way to validate our algorithm; that is to use existing Simulink library blocks to draw the diagram in Simulink quickly. A snapshot of the completed system is shown in Fig. 2.

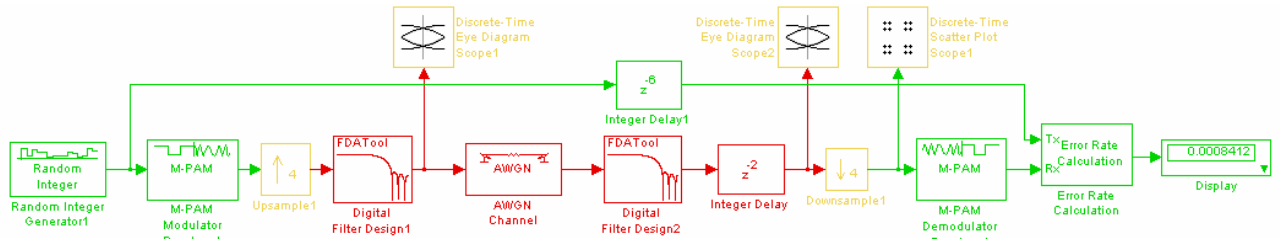


Figure 2. Floating-point system in Simulink blockset

Notice that there is almost a 1-1 correspondence between above Simulink™ system with the block diagram in Figure 1. Different colors of the blocks indicate different clock rate. Here, let's explain some of them in details.

First, several blocks are used to help us debug/understand the system. These include the Display block, Discrete-time Scatter Plot Scope, and Discrete-Time Eye Diagram Scope. A number of other very useful display blocks can be found in the Sink directory of Simulink library and the Comm Sink directory of the Simulink Communication Blocksets.

Secondly, the Error Rate Calculation block is used to compare the Tx signal with the Rx ones. It displays the bit-error-rate (BER) of the system.

A couple Integer Delay Blocks are used to synchronize the Tx and Rx signals. In our design both the Tx filter and Rx filter introduce 11 delays (each delay corresponds to $1/(4\text{MHz}) = \frac{1}{4} \mu\text{s}$) at their center tap. So another 2 delays, each of $\frac{1}{4} \mu\text{s}$, are introduced to make the total delay

$$\frac{1}{4} (11+11+2) = 6 \mu\text{s},$$

which is an integer multiple of the symbol period. Without using the integer delay of 2, a large ISI will be seen on Scatter plot; that is, the down-sampler will not sample at the wide-open instance showed in the eye diagram.

Finally two Digital Filter Design blocks are used for the two rRC filters. These two identical blocks are specified using the design mask showed in Figure 3.

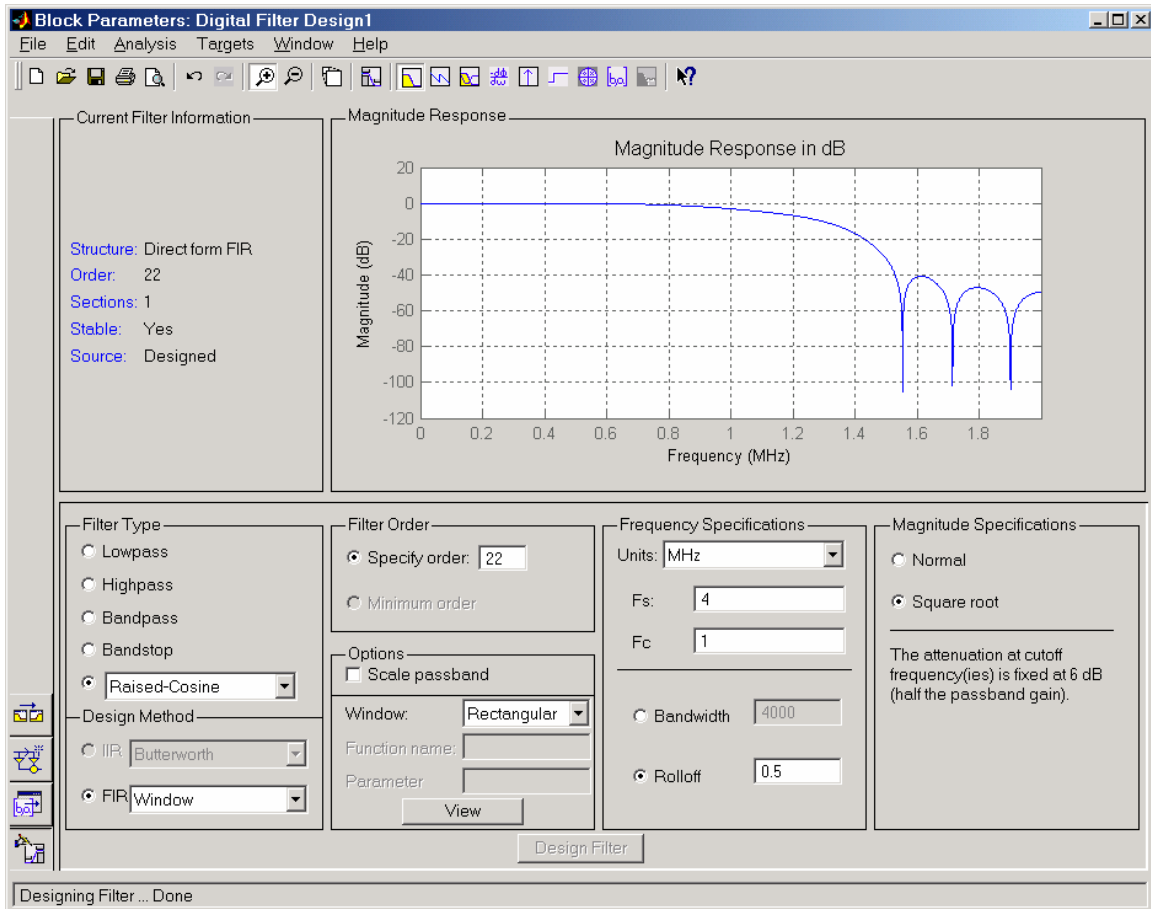


Figure 3. design root-raised-cosine filter

Here we choose Rectangular windowing method in the design without trying others. To understand windowing methods, please refer to [OppenheimShafter99]. The sampling frequency is 4MHz since we choose $R=4$. Rolloff factor is chosen to be 0.5. The higher rolloff factor is, the more relaxed the filter is; thus the less number of taps will be needed. But more excess bandwidth (total bandwidth needed will be $[-(1+\text{rolloff}) \text{ MHz}, (1+\text{rolloff}) \text{ MHz}]$). In our system, this factor brings another degree of design freedom, but for simplicity let us fix it. The final parameter that is adjustable is the filter order, we choose the lowest filter order that satisfies the side-lobe to be 40dB less than the main-lobe, as shown in Fig. 3. You may also try to use Matlab function

`>>help rcosfir (or firrcos)`

to design the filter coefficients. Then you can write a script to automatically determine the lowest filter order given different choices on Rolloff, windowing method, etc. Once the filter coefficients are found you can specify them in a Digital Filter block in Simulink, which basically does the same thing as the Digital Filter Design block. But we won't go try this latter approach here.

You can also export the filter coefficients to workspace choosing File→Export in Fig. 3, which will give Fig. 4.

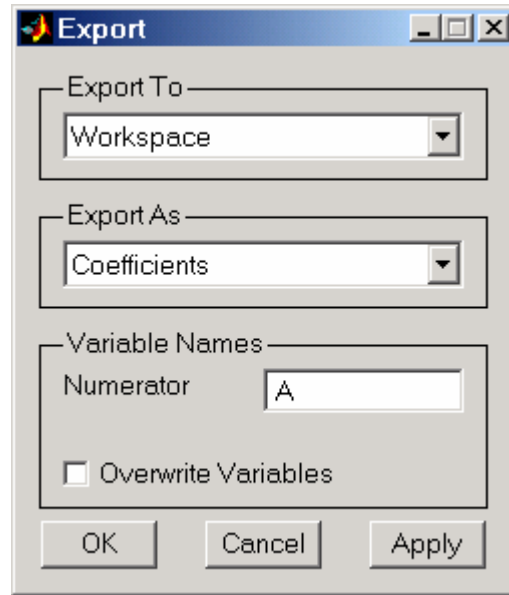


Figure 4. Exporting coefficients to workspace vector A

With the two filters designed above, and a channel noise power of 0.1, we get the following system performance in Figure 5.

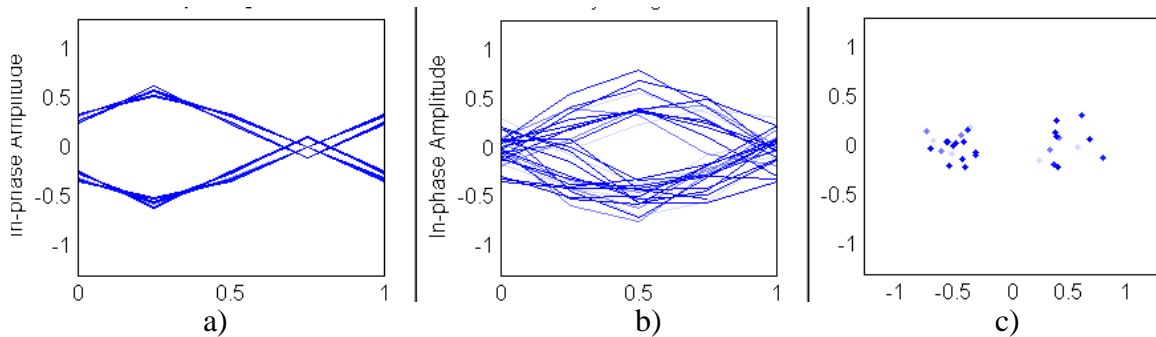


Figure 5. a) Eye diagram of the transmitted signal, b) eye diagram of the received signal, c) scatter plot before the demodulator

It can be seen that the Tx rRC filter caused some ISI as shown in Fig. 5-a. The eye is further closed by the AWGN channel noise as shown in Fig. 5-b. Therefore, the constellation points become blurred in the scatter plot in Figure 5-c. Suppose we wish the

$$\text{BER} < 0.002, \text{ with AWGN of Gaussian distribution } N(0, 0.1).$$

Then our system satisfies above system specification, as indicated in the right-most display of Fig. 2. Note that 100 bit errors are detected before we stop the simulation. Assuming bit error comes in Poisson process, the real BER is in the following interval with .95 confidence [Shi02].

$$\begin{aligned}
& \left[\frac{(100 - 1.96\sqrt{100})}{100}, \frac{(100 + 1.96\sqrt{100})}{100} \right] \times (BE\hat{R}) \\
& = [0.804, 1.196] \times 0.00084 \\
& = [0.00067, 0.00106].
\end{aligned}$$

The simulation takes about 30 minutes to finish.

6. Building a Pseudo Floating-point System in System Generator

The floating-point system we built in the preceding section can now serve as our reference system. The next step is to impose the architecture information into the system. Xilinx System Generator blockset is a design environment that works like part of Simulink library, and it is used to realize the architecture information. In the past, we have used the granular blocks of Simulink, such as multipliers, adders etc. However, it turns out it is just easier (for the rest of the BEE or INSECTA flow) to build the system directly in System Generator library. The blocks in System Generator only support fixed-point data types (with double over-ride functionality in simulation). But that won't cause much difficulties here since we can just choose all the word lengths to be very high whenever possible [Shi03]. Then, the fixed-point system is called pseudo floating-point system with architecture information. This is a good way to validate the architecture choice.

Choosing the architecture is a difficult task [Brodersen03]. For example, the filter structure can be built using the FIR block in System Generator DSP library, which use distributed arithmetic to save area. But, it is often not power-efficient since the pre-stored partial products need to be loaded from the memory block frequently. Without further justification, let us use the Delay, Cmult, AddSub, upsampler, downsampler, and Gateway In/Out block to build the system. We take advantage of the linear phase property of the rRC filter to minimize the number of such blocks in our design. Further more the center tap can be normalized to 1 to save another Cmult. The resulting structure is shown in Fig. 6 and Fig. 7. The gain of A(12) after the filter is used in order to bring the total transmitting power the same (one can think it as an analog gain, so it does not consume Cmult).

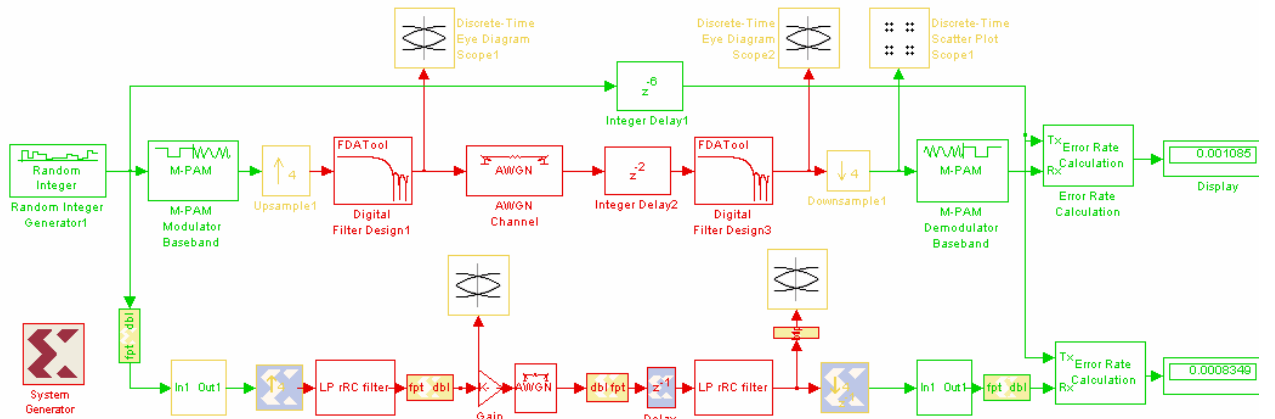


Figure 6. Psuedo-flt system in system generator

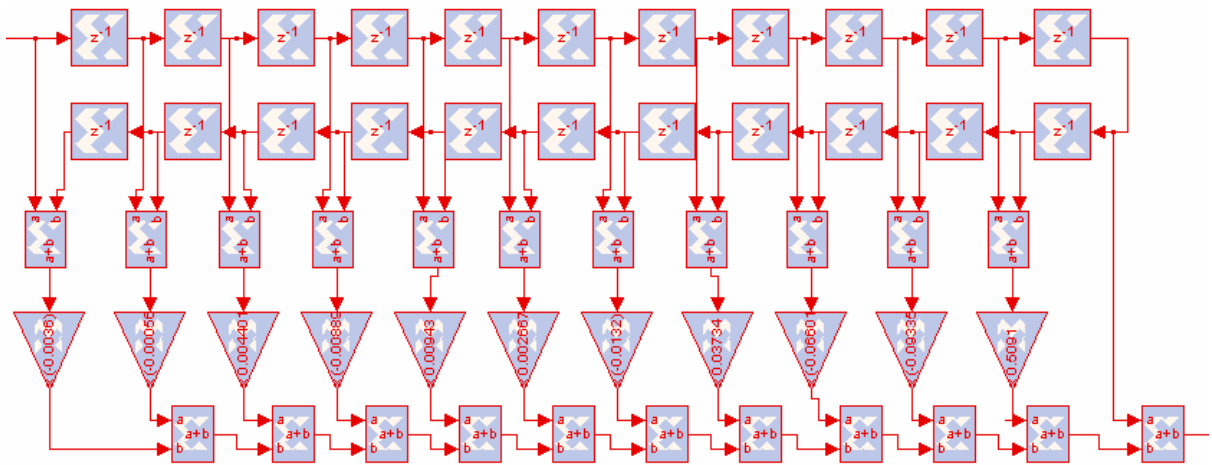


Figure 7. LP rRC filter

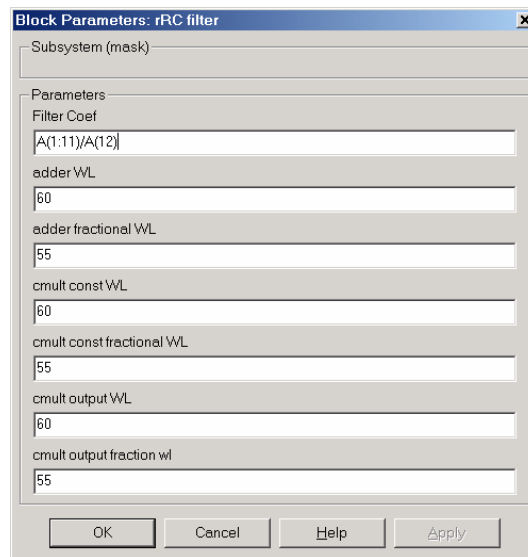


Figure 8. Mask parameters of LP rRC filter

Fig. 7 and 8 show the detailed structure of the low pass rRC filter and its mask parameters. We have set all the WL to be exceptionally high (60 bits). Simulation indicates the pseudo flpt system and original flpt system performs the same within their confidence interval.

Here be careful that since the original system has both I/Q channel, the noise power indicates the sum of I/Q noise. So we should choose noise power be 0.1/2 to get the same BER as previous floating point. The reference system is also modified to contain only the I-channel. With this modification the pseudo-flpt system and flpt system do exactly the same thing upto each cycle.

A long duration [0, 2s] is used for the simulation, the BER is found to be 0.0007795, that is, $2s \times 1\text{MHz} \times 7.7795 \times 10^{-4} = 1559$ errors. So the 0.95 confidence interval estimate of BER is

$$\begin{aligned} & \left[\frac{(1559 - 1.96\sqrt{1559})}{1559}, \frac{(1559 + 1.96\sqrt{1559})}{1559} \right] \times (BER\hat{R}) \\ & = [0.95, 1.05] \times 0.0007795 \\ & = [0.00074, 0.00082]. \end{aligned}$$

The simulation takes about 8 hours to finish.

7. Building Fixed-point System – Fixed-point Data Type Determination

Now all the algorithm and architecture decisions have been made in our design. What is left is to decrease the word lengths presented in the preceding section, and to determine all the overflow and quantization modes. The goal is to have this done automatically, taking advantage of the floating-point to fixed-point conversion (FFC) tool [Shi02][Shi03][Shi_web][Shi04_1-7].

In order to do FFC, one needs to first identify the node where the difference between fixed-point and floating-point systems will be checked. This is practically done by inserting a Specification Marker block from the FFC library that is also located in H:\ffc\ffc_package\ffc_lib.mdl (which can be opened using

>>ffc_lib

in Matlab)—see appendix B for some detail. A natural node to place the marker is the one after gateway-out block of the receiver. At this node the bit error rate solely caused by quantization noise can be detected. Suppose it has probability p , i.e. BQER (bit quantization error rate) = p ; then the

$$\text{MSE (flpt-fxpt)} = p \cdot 1^2 + (1-p) \cdot 0^2 = p = \text{BQER}.$$

So theoretically, placing the Spec Marker here is a good choice. However, in practice, it is often less attractive due to the long simulation time to fulfill the estimate of a BQER accurately. In fact, since it is normally necessary to have BQER less than BER, the same number or more input samples as in the preceding section are needed to get a high-confidence estimate. That corresponds to long simulation duration for each run, which is

too long as iterations of runs are needed. As a side note, in general, the total BER with both channel noise and quantization noise is not the sum of the flpt BER (without QN) and this BQER, because a slicer (demodulator) block is a nonlinear function of noise power (it is a Q-function of SNR).

A much more robust node to place the Specification Marker block is the one before the 2-PAM demodulator. One reason is that we know the rest of the receiver following this node (the only block left is just a demodulator, i.e. a slicer) does not have word lengths to be determined. Another reason is that the MSE(flpt-fxpt) at this node gives a good indication of the BER performance after the demodulator. In fact if we think QN and channel noises propagate to this node as uncorrelated Gaussian noises independently, it is equivalent to think their sum as a total noise power. So one just needs to make sure the QN power much less than the channel noise power at this node to quantify the statement “fxpt system differs only little from flpt system”.

A system with the marker specified is displayed in Fig. 9. Compared with the previous design in figure 6 many of the unnecessary blocks have been eliminated at this stage. For example since we already have the pseudo flpt system in System Generator blocks, the first version of the system designed in pure Simulink block set has been deleted. You can leave those blocks there without any major problem, except for a slow-down of simulation speed. This newer version is named ffc_tutorial_v2.mdl. One can see that a specification marker has been placed before the demodulator. In addition you can find some supporting Matlab files in the same directory (H:\ffc\systems\ffc_tutorial1\ or H:\ffc\systems\ffc_tutorial1_sg3); they are:

System_init.m
ffc_setting.m
and A.mat.

In order to continue the demonstration yourself you need to copy ffc_tutorial_v2.mdl, system_init.m, ffc_setting and A.mat into a working directory of which you have write-access. To prevent possible hazard, H:\ffc is secured as read-access only. After the copy you can go to that directory and start the conversion tool yourself by running

```
>>ffc
```

If you have your own design to FFC, you might want to have a directory for that design specifically. In that directory you should have a file named “system_init.m” that initializes your pseudo flpt system, and a file named “ffc_setting” to save FFC design parameters. In our example here, system_init.m basically loads the filter coefficients A.

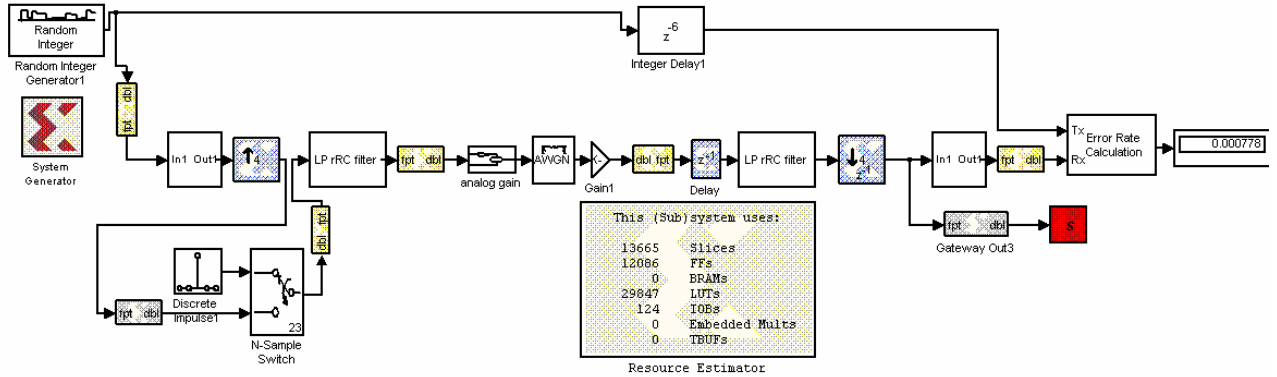


Figure 9. ffc_tutorial_v2.mdl file. Comparing with figure 6 a specification marker and a resource estimator block are included. Furthermore some blocks supporting the floating-point design are eliminated/added to speed up/support simulation.

The Resource Estimator looks different in System Generator 3.1 or higher version.

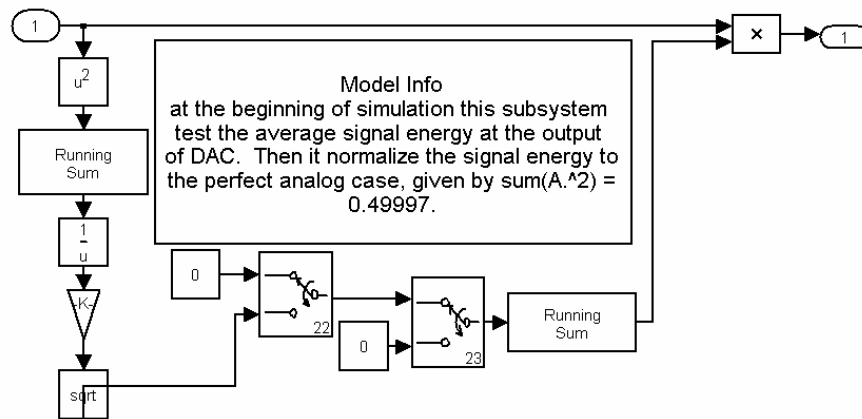


Figure 10. Analog gain subsystem in ttc_tutorial_v2.mdl

Another modification is replacing the first A(12) gain to the analog gain block and adding the impulse input in front of the Tx rRC filter. Altogether they make sure the average signal power to be transmitted is the same as the floating-point system. Here the average power of the transmitted signal $x(n)$ is

$$\begin{aligned}
 & E[x(n)^2] \\
 &= E\left[\left(\sum_m S(n-m) \cdot A_m\right)^2\right] \\
 &= \sum_m \sum_l E[S(n-m) \cdot S(n-l)] \cdot A_m A_l \\
 &= \sum_m \sum_l \delta_{ml} \cdot A_m A_l \\
 &= \sum_m A_m^2 = \|A\|_2^2
 \end{aligned}$$

Here we have used the fact that the signal after the modulator is a zero mean random process choosing from {1, -1}. Thus the signal power is just the 2-norm square of the filter coefficients. Sometimes the quantization of transmitter filter could increase the transmitting power. Without normalizing the transmitting power, the comparison of performances between floating-point and fixed-point systems is unfair.

To FFC this small system takes about 5 minutes. FFC tool sequentially asks you to input some important information you want to choose, such as design names. On the other hand it might be too lengthy to answer all the questions sequentially. Then you need to create an `ffc_setting.m` file in the directory and set the parameter “ask_question” to be 0. At one point it also asks you to change the model simulation time. You can input the simulation start and stop time as [0, 1/1e3] (to change simulation time right click your model window, and click Simulation Parameters, where you can see the parameter Start Time and Stop Time). This will make the simulation duration to be 1ms, which results in 1001 output samples at the Spec Marker. That is enough to have a good MSE estimation, assuming the flpt-fxpt difference error is a stationary random process. The assumption is justified since each quantization error is assumed to be stationary. If you use `ffc_setting.m`, you can see the parameters “sim_start_time” and “sim_stop_time” are set to be 0 and 1/1e3 separately.

Another required important user input is the MSE level you want to choose. You can either manually do a couple tries to understand the relationship between your system performance (e.g. BER) and MSE. Some papers [Shi04_1-7] explained in detail about how this can be done. On the other hand you can do the following calculation to determine it (refer to [Proakis01]). Since the signal power before the demodulator is at about 0.6 (you can estimate it by placing a eye-diagram scope before the demodulator, and see the signal power), the physical noise (PN) power before the slicer of the BPSK system is about

```
PN power =
(Matlab command line input)
>>fzero('1/2*erfc(1/sqrt(2)*sqrt(.6/x)) - 7.795e-4', 0.1)
= 0.0600.
```

So suppose the BER deterioration due to quantization noise is 10% of the original BER, i.e. the final BER to be less than $7.995 \times 10^{-4} \times (1+10\%) = 8.57 \times 10^{-4}$, we need the quantization noise power (QNP) to be

```
QNP power =
(Matlab command line input)
>>fzero('1/2*erfc(1/sqrt(2)*sqrt(.6/(x+.06))) - 7.795e-4*(1+.1)', 0.1)
= 0.001.
```

Thus we need $MSE < QNP \text{ power} = 0.001$. To be robust we assume there could be modeling error and estimation error, thus we set

$$\text{MSE} = \frac{1}{2} \text{ QNP power} = 0.0005.$$

Next the grouping rules need to be defined. Without grouping all Xilinx blocks are independently adjustable to find the minimum hardware cost. That would result a problem of too many design variables to be optimized (and turns out to be unnecessary in turns of design optimality, see section 9). You can type

```
>>help_ffc('rules')
```

to understand more about the rules. The rules used for the current conversion in this section is [1, 2.1].

With the setting described above we achieve a fxpt system of about 356 slices as shown in Fig. 11. A final simulation of duration [0, 1s] produces 811 errors; so the BER of the final fxpt system BER is within .95 confidence interval

$$\begin{aligned} & \left[\frac{(811 - 1.96\sqrt{811})}{811}, \frac{(811 + 1.96\sqrt{811})}{811} \right] \times (\widehat{BER}) \\ & = [0.93, 1.07] \times 0.000811 \\ & = [0.00076, 0.0087]. \end{aligned}$$

It is therefore of high chance the resulting fxpt system has BER less than the targeted 8.57×10^{-4} .

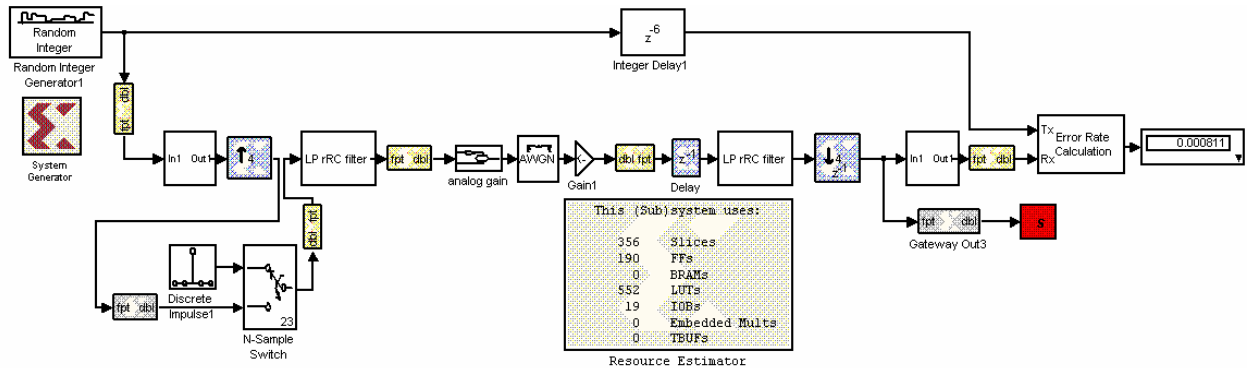


Figure 11. The final fxpt system with $\text{BER} \sim 8.11 \times 10^{-4}$, and ~ 356 FPGA slices.

You can choose Format \rightarrow show Port Data Types to see the fxpt data-types used for the final system. In fact, it is probably surprising to find that some of the constant multipliers have zero coefficients now (since the constant value is too small to be represented by the small wordlength fxpt data types). But you don't need to worry too much about it since these logics will be automatically eliminated in the final placement-and-routing stage.

8. Multiple Specifications

The FFC conversion done in previous section is subject to one MSE specification constraint. The FFC tool set can actually handle multiple specifications. Recall that the transmitter filter frequency response in [1.5MHz, 2MHz] should be less than -40dB, it is natural to set this as the second specification. Thus one more Spec Marker block is inserted in the system, and saved to ffc_tutorial_v3.mdl as shown in Fig. 12. This Spec Marker chooses “user specified spec. calculation function” as the specification type, and use “simulation_function” as the specification calculation function name. A snap shot of the block mask is shown in Fig. 13. Thus, there is an associated Matlab function “simulation_function.m”. This function calculates the highest frequency response in interval [1.5MHz, 2MHz]. Fig. 14 shows the frequency response of the resulting system. The simulation to show BER is again about 8 hours for duration [0, 1s].

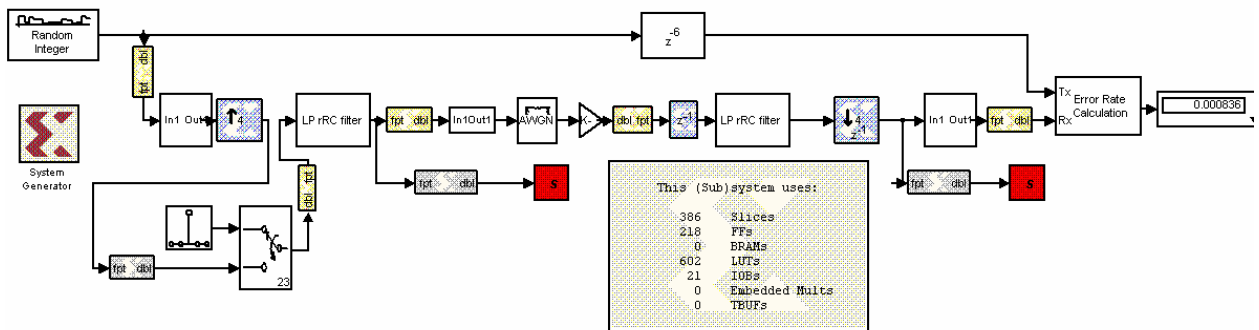


Figure 12. 386 slices and BER $\sim 8.36 \times 10^{-4}$, or .95 confidence interval of $[7.8 \times 10^{-4}, 8.9 \times 10^{-4}]$; still of good chance within 8.57×10^{-4} spec.

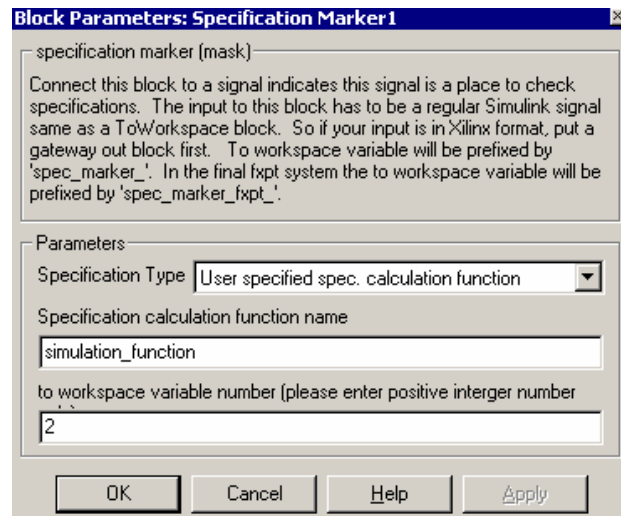


Figure 13. New specification Marker parameters

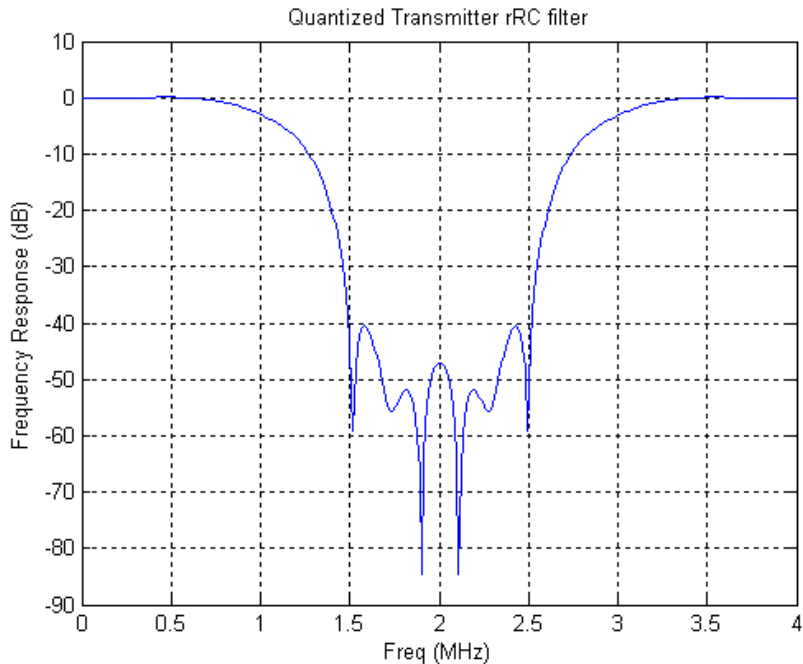


Figure 14. Tx rRC filter frequency response satisfies the -40 dB spec.

The conversion takes about 20 minutes, which is considerably more time than the case if there is MSE spec only. This is in general the situation. The speed-up for MSE spec results from the careful analysis of the relationship between the MSE and wordlengths [Shi03][Shi04_1-7].

9. Some further Analysis

Rules [1 2.1] results eight groups of fractional word lengths. Let's see what if we apply more or less rules.

If rules [1.1 2.1] are chosen, only 4 groups are resulted (notice from help_fcc that rule 1.1 supercedes rule 1) and the conversion only takes about 2 minutes. The system as shown in Fig. 15 it has 493 FPGA slices, which is about 30% increase over the design in section 7. The performance is compatible. This proves that more rules will speed up the conversion, but they also introduce a loss of design optimality at the same time.

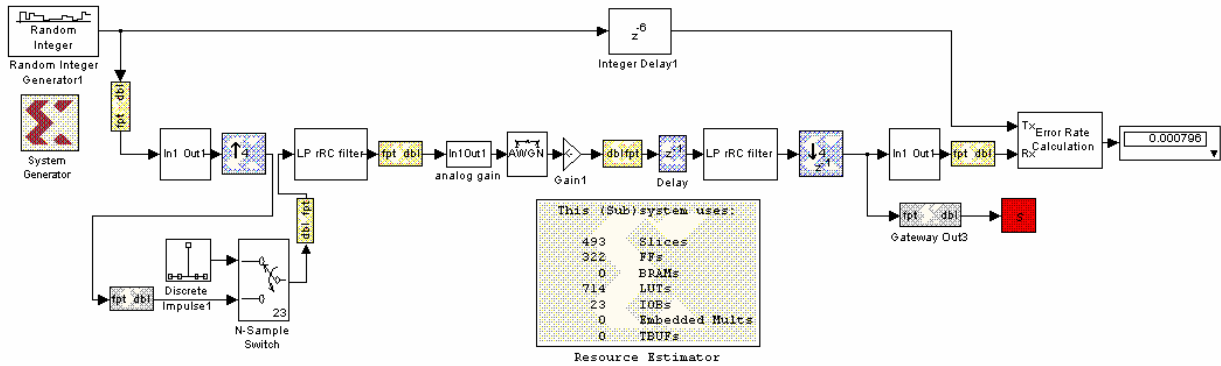


Figure 15. The final fxpt system with BER $\sim 7.95 \times 10^{-4}$, and ~ 493 FPGA slices.

However, that does not mean we should include no grouping rules in our design. In fact if we chose [1 2] as the rules, 30 groups are resulted, and conversion takes about 20 minutes to finish. As shown in Fig. 16 the result system has 377 slices and BER $\sim 8.36 \times 10^{-4}$. This system actually consumes about 6% more resource than the one in section 7! The reason behind it is when groups are so small, the modeling of their analytical hardware-function and MSE-function suffers high error. The error causes uncertainties in optimal decision. It should be emphasized that small 6% difference almost gives the conclusion that having 30 groups won't improve the fxpt conversion much than having 8 groups, at least for this system.

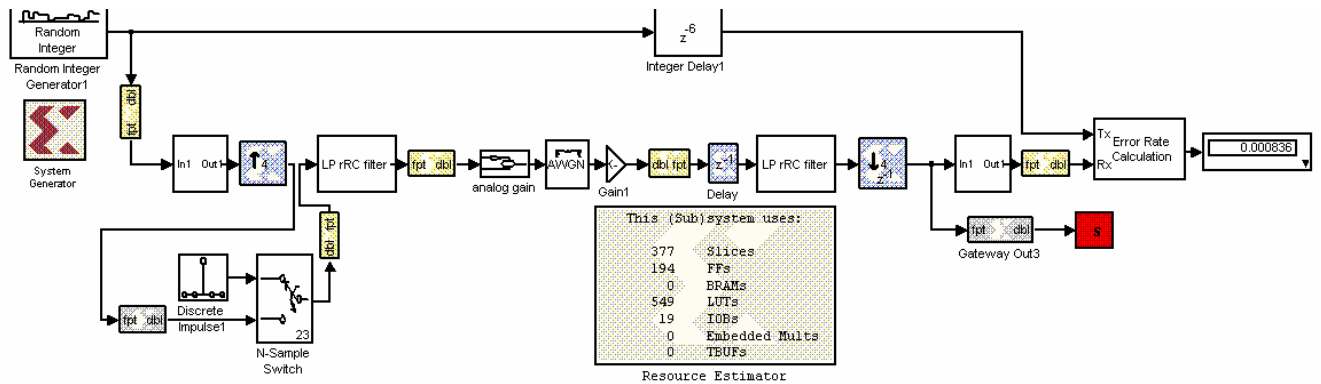


Figure 16. The final fxpt system with BER $\sim 8.36 \times 10^{-4}$, and ~ 377 FPGA slices.

Similar simulation is done for multi-criteria FFC case. Choosing [1.1 2.1] as the rule a system of 650 slices with BER $\sim 8.2 \times 10^{-4}$, as shown in Fig. 17. So there is a 60% reduction in hardware cost by applying rules [1.1 2.1] instead of [1.1 2.1]. The gain here is conversion time, only 5 minutes as opposed to previous 20 minutes.

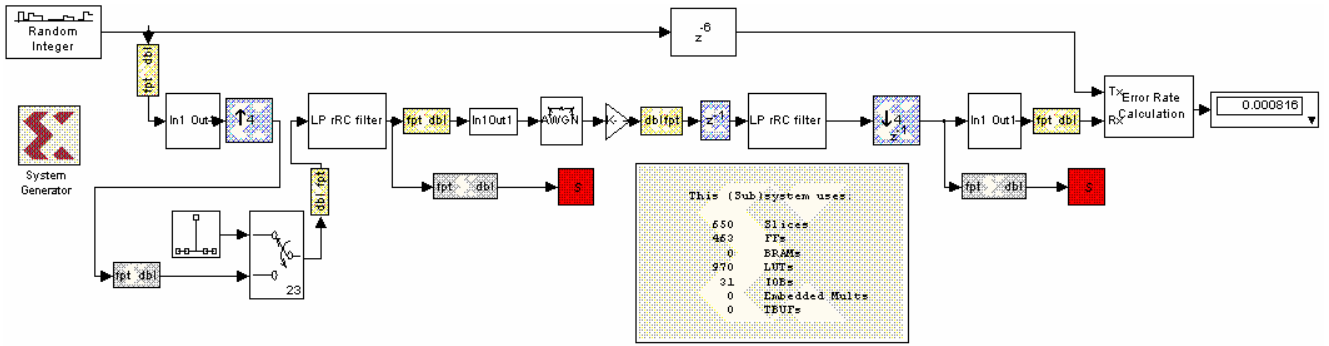


Figure 17. 650 slices and BER $\sim 8.2 \times 10^{-4}$

All the experiments so far were using rule 8 that sets “saturation” as overflow mode everywhere. However, since the integer word lengths are chosen conservatively already, it is in general not necessary to use rule 8. In stead, it is good to use rule 8.1, which sets “wrap-around” as the overflow mode everywhere. The resulting two systems are shown in Fig. 18 and 19. Comparing with the previous conversions that uses rule 8, the new results saves about 25% slices! The simulation results are however exactly the same. So these two results should be our final designs.

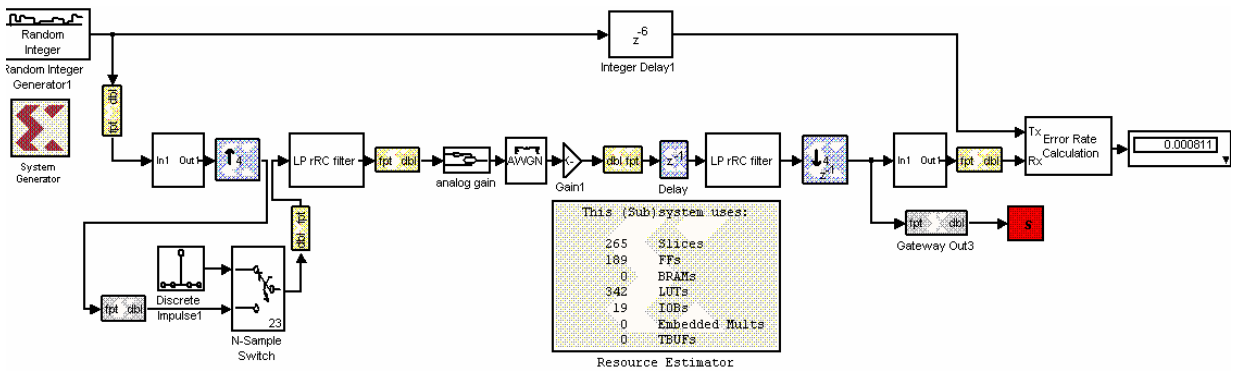


Figure 18: 256 slices and BER $\sim 8.11 \times 10^{-4}$ (rule [1 2.1 8.1])

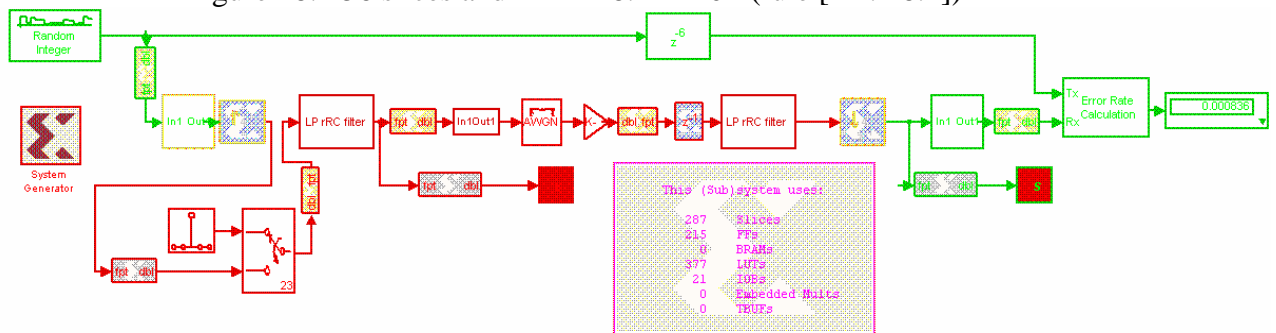


Figure 19: 287 slices and BER $\sim 8.36 \times 10^{-4}$ (rule [1 2.1 8.1])

Another subject we mentioned in Section 7 is robust programming. A robust MSE spec has been chosen there. Let’s see what if MSE is chosen to be 0.001 directly. Fig. 18 shows the result. About 7% reduction in hardware resources causes the .95 confidence interval of BER to be [0.00079, 0.00090]. The BER becomes much more likely to be greater than 0.000857 (recall this is 10% more than that of the floating-point system).

The small hardware reduction is usually not worth the risk of breaking the spec. The other possible situation is the spec. is also flexible in the first place, in which case having a constraint on the spec is in some sense robust programming itself. The bottom line here is to be careful on choosing the MSE spec level: you should understand there could be both under-modeling error and estimation error associated with the spec. function; therefore it is usually wise to be a little conservative. After all, this example showed 3dB “relaxation” in MSE only causes a variation on hardware about 7%. This is almost always the situation, due to the characteristics of objective function and constraint functions, being quadratic and exponential respectively.

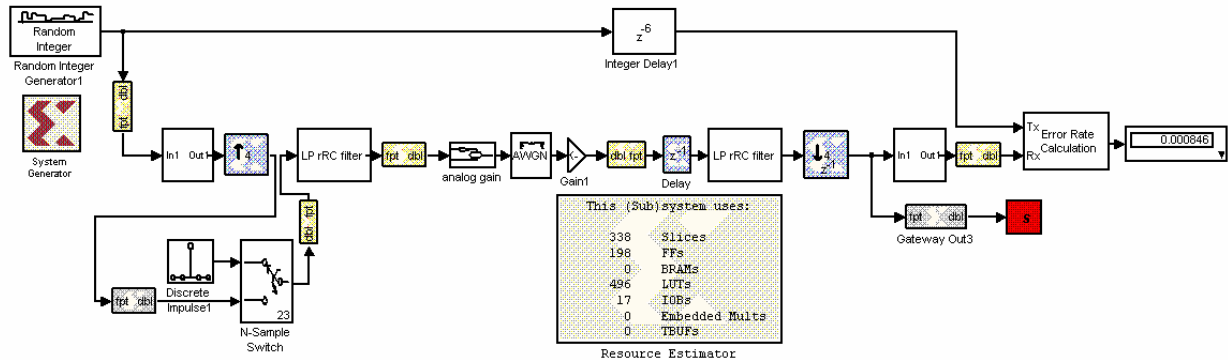


Figure 20. 338 slices and BER $\sim 8.46 \times 10^{-4}$

10. An Important Remark

The most important remark to be pointed out here is that in our design procedure above, we only did qualitative justification on choosing the algorithms (say data modulation scheme, upsampling rate R, filter type, etc.) and architectures (say filter specification, filter form). To have a good design these “parameters” need to be justified using careful analysis or simulations. Nevertheless the purpose of this tutorial is to get you familiar with the design process, and mainly on using FFC tool. So these design dimensions have not been explored fully here.

On the other hand higher-level decision (such as algorithm) made without considering the lower-level discrepancies could turn out to be unfavorable when the lower-level design (such as choosing circuit) space is explored. For example, we decided the number of filter taps to be the smallest one satisfying the 40dB attenuation requirement. This is reasonable since it saves hardware and result less latency. However it’s fairly possible that with fixed-point data types, too high word lengths are needed to maintain the 40dB attenuation because there is not much room left for WL reduction. By relaxing the number of taps to a few more, one might dramatically drop the number of bits needed for each tap, therefore save the total hardware cost.

So ideally algorithm, architecture, and fixed-point datatypes should be optimized jointly, maybe with other design variables such as circuit level flexibility, in order to get the “best” design. The bad news is a problem like this could become too hard to solve. That’s exactly the reason design of a large system is almost always divided into different

levels, and different blocks. One always tries to reduce the inter-dependency between these levels and blocks to make each of the smaller problems more tractable. Our introduction of MSE specification as a global justification on FFC problem is based on this argument.

Of course one needs to bear in mind that quite often by considering the inter-dependency more carefully one can achieve large improvements. Examples include Trellis-coding (coding and modulation jointly considered), our approach on FFC problem (since the data types of different signals are jointly considered), channel coding (where algorithm is directly done in number theory, which is already fixed-point), etc. But this interesting trade-off is beyond the scope of this tutorial.

Another important point is some analysis of the time needed if conventional FFC methods are used as mentioned in the past techniques in [Shi03]. First of all, the hardware resource calculation would take much more time cause by the lack of understanding of the functions involved in the optimization. Secondly, each spec check takes a simulation at least 8 hours as BER as done in the final validation step in section 6 and 7 used as the spec directly. It is usually much more than 8 hours to discern the BER changes more clearly. Thirdly at least $4 \times (5 \text{ or } 6) \sim 20$ such simulations are needed in those techniques in order to done the optimization. Here 4 is the number of wordlength groups. This would result a total conversion time

$$\gg 8 \text{ hours} \times 20 \sim 7 \text{ days,}$$

as opposed to a few minutes in our case.

11. Conclusion

By designing a simple base-band digital communication system, we showed a design procedure, starting from algorithm to fixed-point implementation, in our design environment combined with Matlab, Simulink, Xilinx System Generator. One major topic is on how to use our floating-point to fixed-point conversion tool. Table I summarizes the conversions done in this tutorial.

	Grouping Rules	MSE spec level	Max(H(1.5MHz, z, 2MHz))	Conversion time (minutes)	# of Frac. WL groups	Slices	BER (ML estimate)
Flpt sys	-	-	<-40dB	-	-	~13500 (or -)	~0.00078
	[1 2.1 8]	0.0005	-	5	8	~356	~0.00081
	[1 2 8]	0.0005	-	20	30	~377	~0.000836
	[1.1 2.1 8]	0.0005	-	2	4	~493	~0.000795
*	[1 2.1 8.1]	0.0005	-	5	8	~265	~0.00081
	[1 2.1 8]	0.001	-	5	8	~338	~0.000846

	[1.1 2.1 8]	0.0005	<-40dB	5	4	~650	~0.00082
	[1 2.1 8]	0.0005	<-40dB	20	8	~386	~0.000836
*	[1 2.1 8.1]	0.0005	<-40dB	20	8	~287	~0.000836

Table I. Summary of conversions in this tutorial. Targeted BER for converted systems is 0.000857. “-“ means not applicable. “*” means the design is considered to have good performance while relatively less conversion time.

Appendix A: Getting Familiar with Matlab™, Simulin™, and Xilinx™ System Generator

The quick way to get started on these tools is to see an existing design. You can do so by type in

```
>>demo
```

in Matlab command line, and start to play around the demo systems there. Notice that Xilinx demos are located at Blocksets→Xilinx directory in the demo window. An example is shown in Figure A1.

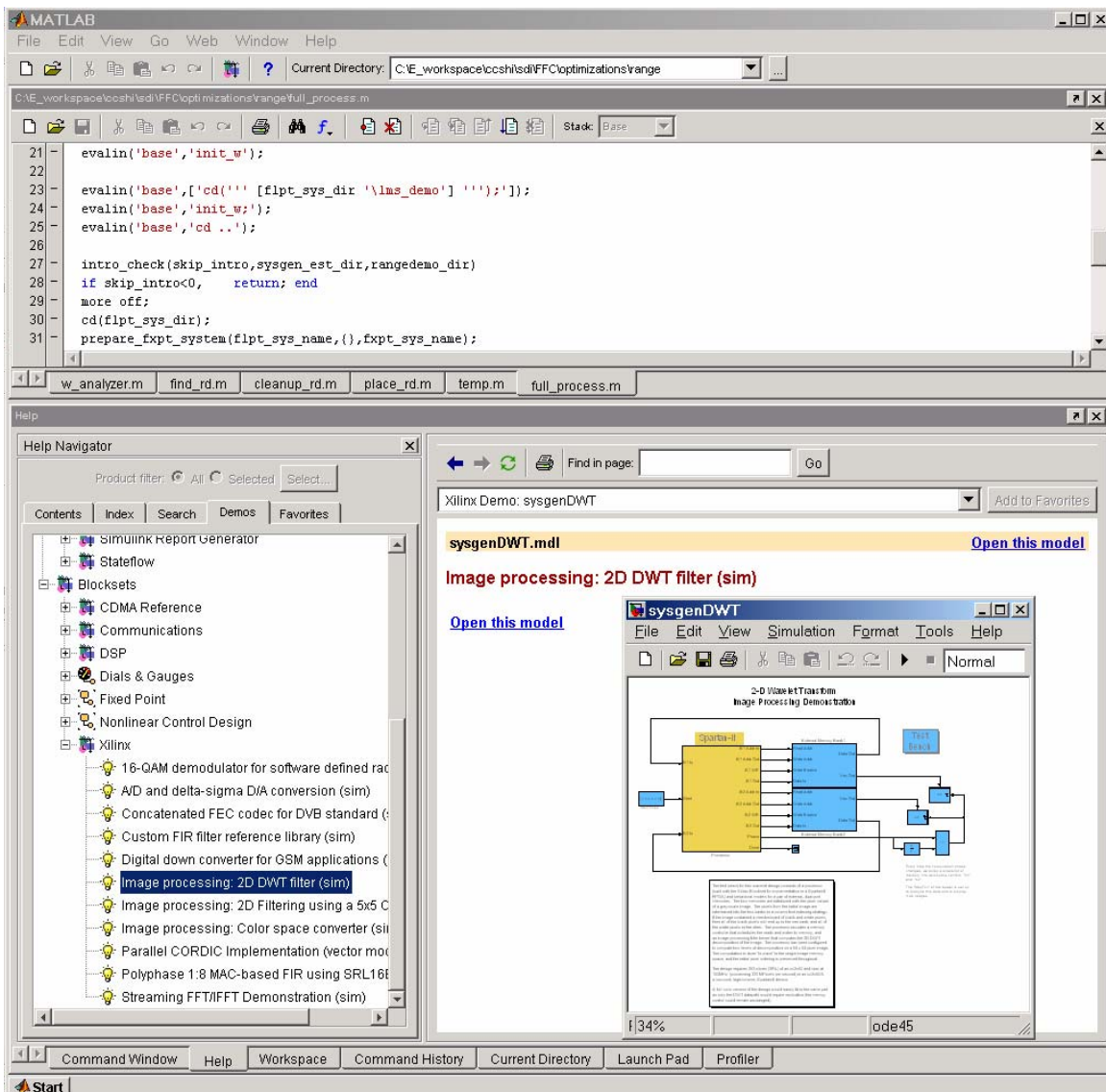


Figure A1. Using Matlab™ demos

If you wish to learn these tools in more a systematic way, we encourage you to pay more attention on the help file, with a window somewhat like Figure A2.

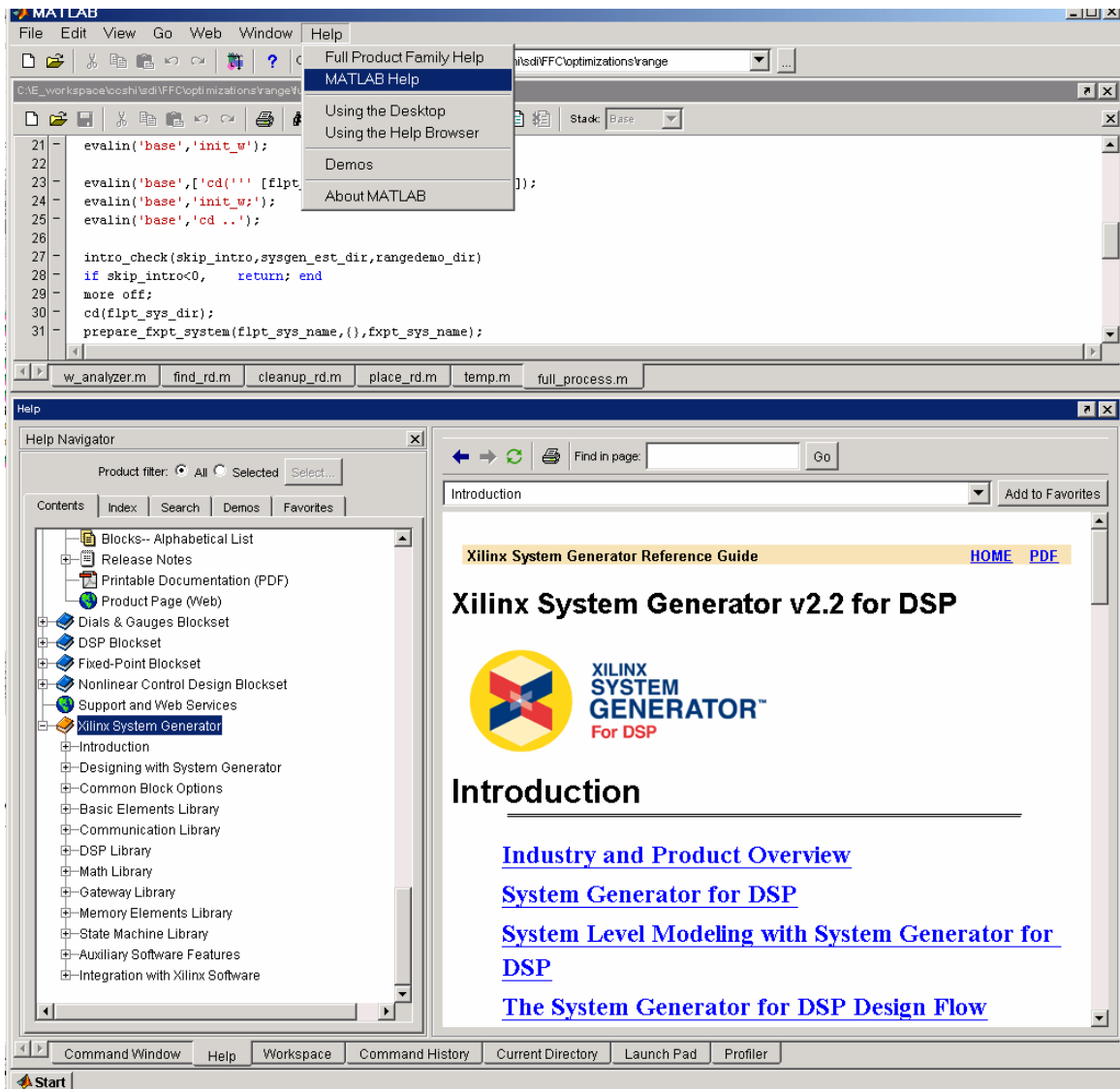


Figure A2: Using Matlab™ help system

Some other information is available if you want to know understand how to using Matlab™ scripts to build Simulink™ system. (see Appendix of [Shi02]).

If you still have questions related to Matlab™ and Simulink™, and could not be answered by anybody around you, you might contact help@mathworks.com. They usually respond within the same day.

Appendix B: Using Floating-point to Fixed-point Conversion Tool

You should have already mapped [\\hitz.eecs.berkeley.edu\designs](http://hitz.eecs.berkeley.edu/designs) to H: disk. Now go to H: disk in Matlab:

```
>>cd H:  
>>cd ffc  
>>ffc_init
```

The last command above swaps the Xilinx library to a version that is prepared for fast hardware resource estimation. You should see some library opened and closed. In addition a few Matlab paths containing FFC scripts are added to the path file. A good way to check that you have successfully done this initialization is to open Xilinx blockset, and see whether you have the resource estimator block in the Basic Elements.

To place the Specification Marker block in your design as mentioned in section 7 the FFC library can be opened by the following command:

```
>>ffc_lib
```

Now you can go to your own directory where your pseudo-floating point system is located, and type in:

```
>>ffc
```

The tool itself will then lead you sequentially through the ffc process. This ffc.m script is located at H:\ffc\ffc_package directory that you have linked to in the initialization step. The definition of many variable names and functions can be found using:

```
>>help_ffc('keyword').
```

Reference

[BEE03]. <http://bwrc.eecs.berkeley.edu/research/bee/doc/designflow/tutorials.htm> and <http://bwrc.eecs.berkeley.edu/Research/BEE>

[Brodersen03]. R. Brodersen, “EE225c Lecture 7: Architectural Transformation. And lecture “EE225c Lecture 3: System on Chip design.

[Oppenheim99]. Alan V. Oppenheim and Ronald W. Schaffer with John R. Buck. *Discrete-Time Signal Processing*. 2nd Edition, Prentice Hall, 1999.

[Rdesktop]. <http://www.rdesktop.org/>

[Chi02]. Peimin, Chi, “Time and Frequency Synchronization”, M.S. Thesis, EECS Department, University of California, Berkeley. 2002. Advised by Prof. Brodersen.

[Shi03]. C. Shi, “EE225c Lecture 6: Floating-point To Fixed-point Conversion”, [lecture slides](#)

[ProakisSalehi]. J. Proakis, M. Salehi, *Contemporary Communication Systems Using MATLAB*.

[Proakis01]. J. Proakis, *Digital Communication*. 4th Edition, Boston : McGraw-Hill, c2001

[Shi02]. C. Shi, “A Statistical Floating-point to Fixed-point Conversion Methodology”, M.S.Thesis, EECS Department, University of California, Berkeley. 2002. Advised by Prof. Brodersen.

[Shi03]. C. Shi, R. Brodersen, “An Automated Floating-point to Fixed-point Conversion Methodology”, to appear in *ICASSP-2003*.

[Shi_web] C. Shi, FFC website with all the source codes and related documents. Available [online] http://bwrc.eecs.berkeley.edu/people/grad_student/ccshi/research/ .

[Shi04_1] C. Shi, and R. W. Brodersen, “Floating-point to fixed-point conversion,” To be published, *IEEE Trans. Signal Processing*. 2004.

[Shi04_2] C. Shi, “Floating-point to fixed-point conversion,” 2004, Ph.D. Thesis, Department of EECS, Univ. of California, Berkeley. (Advisor: Robert W. Brodersen).

[Shi04_3] C. Shi, and R. W. Brodersen, “Floating-point to fixed-point conversion with decision errors due to quantization,” *Proc. IEEE Int. Conf. on Acoust., Speech, and Signal Processing*, 2004, Canada.

[Shi04_4] C. Shi, and R. W. Brodersen, “A perturbation theory on statistical quantization effects in fixed-point DSP with non-stationary input,” *Proc. IEEE Int. Sym. Circs. and Sys.*, 2004, Canada.

[Shi04_5] C. Shi, R. W. Brodersen, “Automated Fixed-point Data-type Optimization Tool for Signal Processing and Communication Systems,” *Design Automation Conference*, San Diego, June 2004.

[Shi04_6] C. Shi, *et. al.*, “An Automated Pre-netlisting FPGA-Resource Estimation Tool,” Submitted to *International Conference, Field Programmable Logics and Its Applications*, 2004

[Shi04_7] C. Shi, and R. W. Brodersen, “A perturbation theory on quantization effects in digital signal processing,” In preparation. *IEEE Trans. Circuits and Systems II: Analog and Digital Signal Processing*.