

A Tutorial on Using Simulink™ and Xilinx™ System Generator to Build Floating-point and Fixed-point Communication Systems

For EE225c, 2003

By Changchun Shi
Last Updated: March 10, 2003
Berkeley Wireless Research Center
EECS Department, University of California, Berkeley

1. Abstract

A simple communication system using root-raised-cosine filter on both transmitter and receiver will be built. The purpose is to show how to design a system in Simulink™ and Xilinx™ System Generator environment, starting from choosing the algorithm, to build the floating-point system, and to building the fixed-point system.

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. This tutorial will hopefully get you familiar with the design environment to reach this goal. Here Matlab™, Simulink™ and Xilinx™ System Generator form the foundation, on top of which a number of Matlab™ scripts and Simulink™ libraries enable much of the design process automated. We will show how they can be utilized via designing a simplified transmitter-receiver system.

3. How to Start

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

A simple way to solve the problem is to login one of the MS Windows Remote Desktop Servers available in BWRC, namely intel2650-{1, 2, 3}.eecs.berkeley.edu, and a few others. 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 Server has all the necessary tools installed correctly.

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

The demo system in this tutorial can be found at H:\ffc\ffc_tutorial.mdl.

If you have never used one of the three tools before, you might want to read Appendix A of this tutorial before you proceed.

4. Algorithm Study

The system to be built here is a simplified version of that in your HW1[HW1]. Suppose we want to do base-band communication with 2-PAM modulation scheme at 1Mbits/sec. Under 2-PAM input symbols (1 symbol/ 1 μ s), such as sequence choosing from binary integer $\{0,1\}$, 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 received signal into original integer. Suppose the channel impose additive white Gaussian random noise, but otherwise ideal.

Although we have assumed our channel is flat with no fading, in reality it could be band-limited (may also due to RF front-end filtering); thus rectangular base-band pulse in time domain (Sinc shape in frequency domain) through the channel will be clearly distorted. One technique to combating this is to have a low-pass pulse-shaping filter at the transmitter side [Proakis01]. For that one needs to first over-sample the data sequence at R MHz. This is usually done by an upsampler with integer 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 finite-tap structure implementation of the filter, which cause non zero stop-band response and hence aliases after the received signal is downsampled. This is what usually called inter-symbol-interference (ISI). Without higher upsampling rate R this deterioration can be alleviated with the cost of increased 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 need to tune [Chi02, Proakis01].

On the other hand, choosing R too high will result high clock rate on the digital filter, A/D and D/A converters, which is not desirable. In our case, suppose we decide $R=4$.

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 this design. 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 AWGN, one can just feed the 2-PAM modulated signal into the channel. We included more blocks in the design to combat some other channel impairments that are not present here.

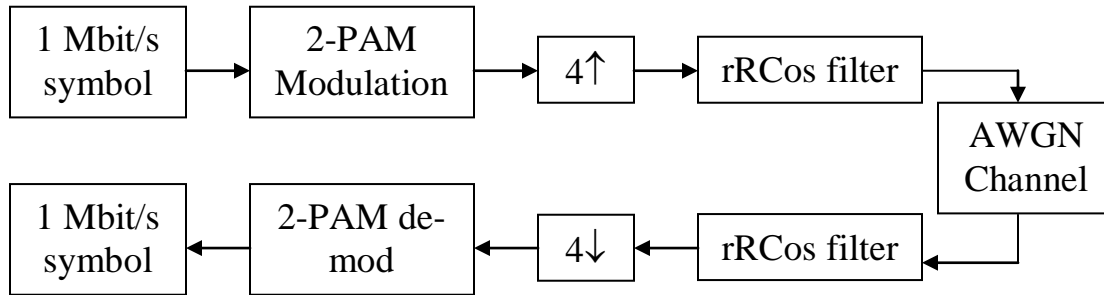


Figure 1. A simple based-band digital communication system

5. Building Floating-point System – Algorithm Validation

With the block-diagram in Figure 1, 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 what conventionally people would do. This is still a good way to understand your 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 Figure 2.

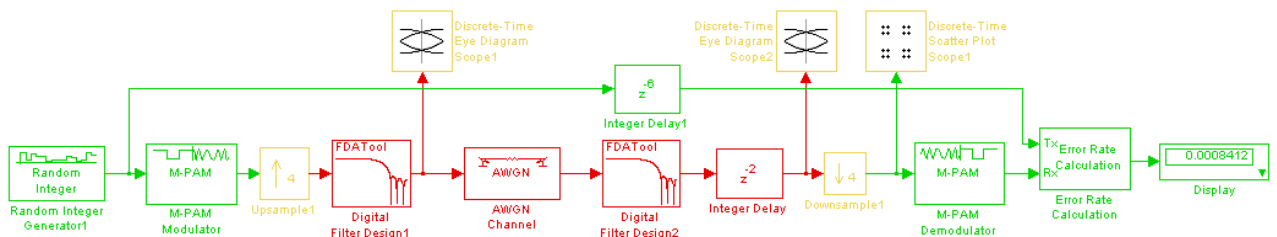


Figure 2. Floating-point system in Simulink™ blockset

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 more detail.

First of all several display 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 Simulink→Sink library and Communication Blockset→Comm Sink library.

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

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 $1/(4\text{MHz}) = \frac{1}{4} \mu\text{s}$) on their center tap. So another 2 delays 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.

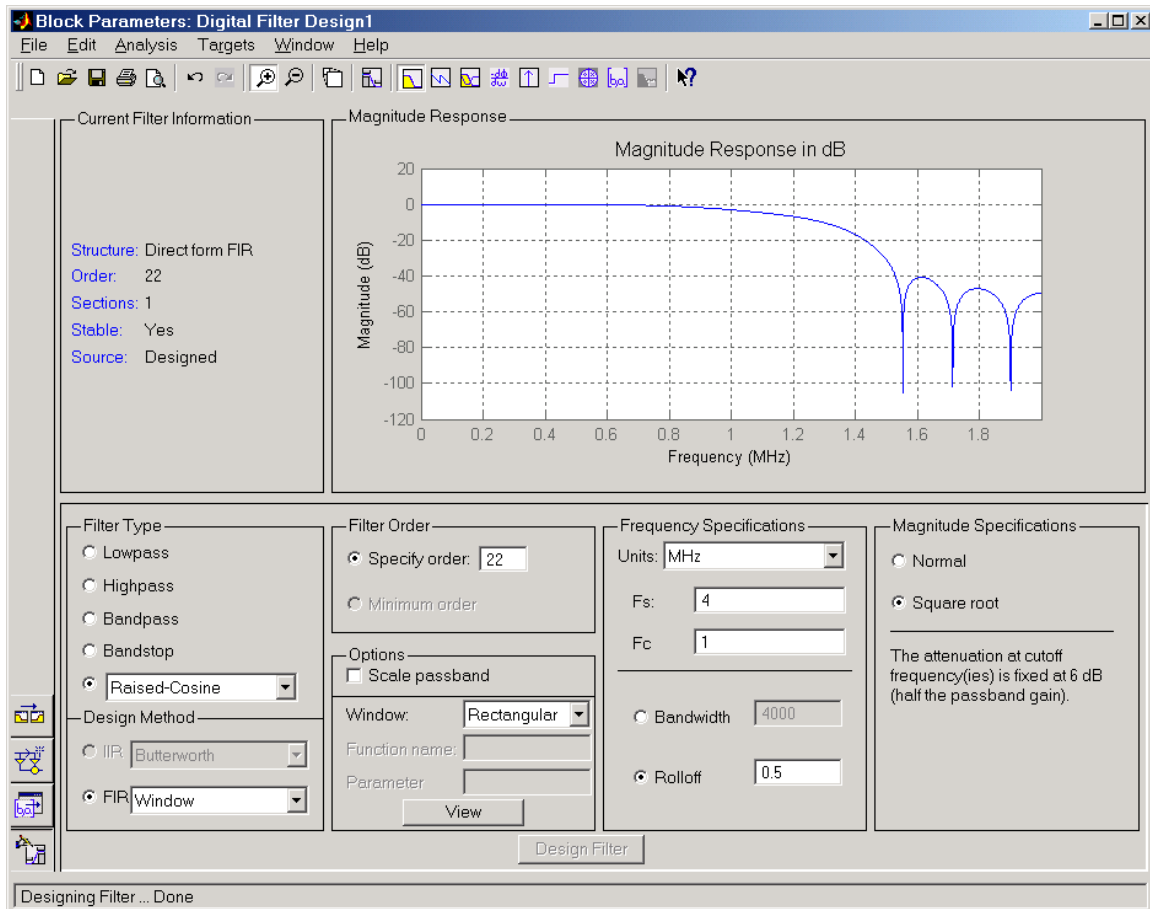


Figure 3. design root-raised-cosine filter

Here we choose Rectangular window method without trying others. To understand window method, 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 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 is another degree of freedom, but let's fit it for now. The final parameter 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 Figure 3.

You may also try to use Matlab function

```
>>help rcosfir (or firrcos)
```

to do the task. 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 try that approach here.

You can also export the filter coefficients to workspace choosing File→Export in figure 3. That will give figure 4.

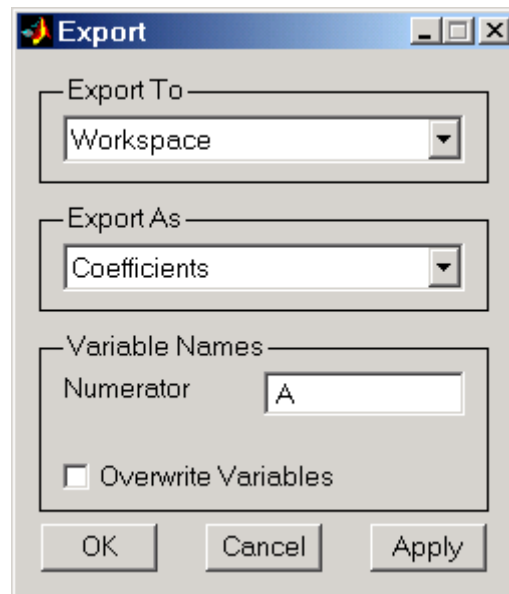


Figure 4. Exporting coefficients to workspace vector A

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

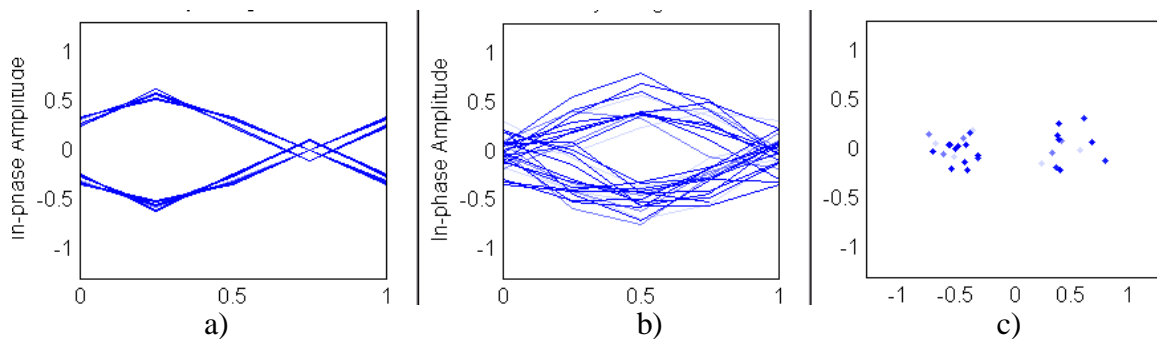


Figure 5. a) Eye diagram of the transmitted signal, b) eye diagram of the received signal, c) scatter plot before the demodulator

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

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

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

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

The simulation takes about 30 minutes to finish.

6. Building Xilinx™ Pseudo Floating-point System – Architecture Validation

The floating-point system above can now serve as our reference system. The next step is to impose the architecture information into the system. Xilinx™ System Generator blockset is used to realize the architecture choice. Historically we have used the granular blocks of Simulink™, such as multiplier, adder etc. in this step. However it turns out it's just easier (for the rest of the BEE or INSECTA flow) to build the system directly in System Generator library. Notice the blocks in SysGen only support fixed-point datatype (but 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]. When this is done, we call the system pseudo floating-point system with architecture information. This is a good way to validate you architecture choice.

Choosing the architecture correctly is a difficult task [Brodersen03]. For example, in our example for the filter structure one can use the built-in FIR block in SysGen 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 too much justification, let's use the Delay, Cmult, AddSub, upsampler, downsampler, and Gateway In/Out block only to build the system. We want to minimize the number of such blocks in our design. Therefore we explore the linear phase property of the rRC filter. Further more the center tap can be normalized to 1 to save another Cmult. The resulting structure is shown in Figure 6 and Figure 7. A gain of A(12) is used in order to bring the total transmitting power the same (one can think it as analog gain, so does not consume Cmult).

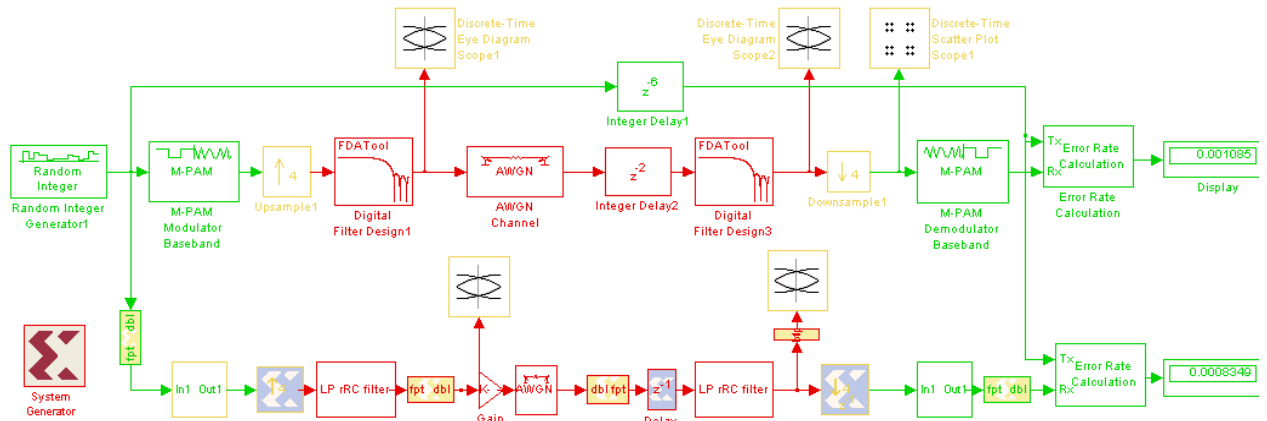


Figure 6. Psuedo-flt system in system generator

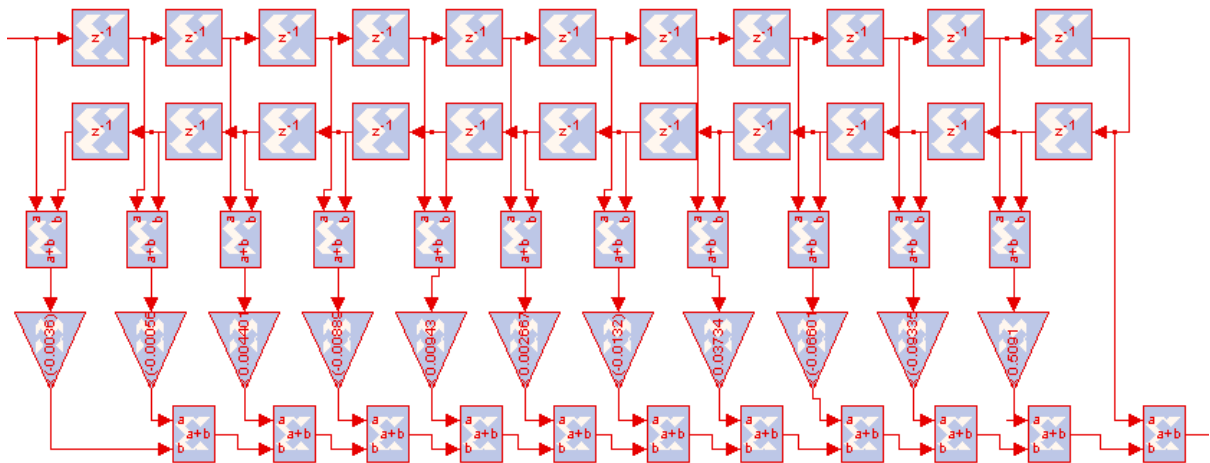


Figure 7. LP rRC filter

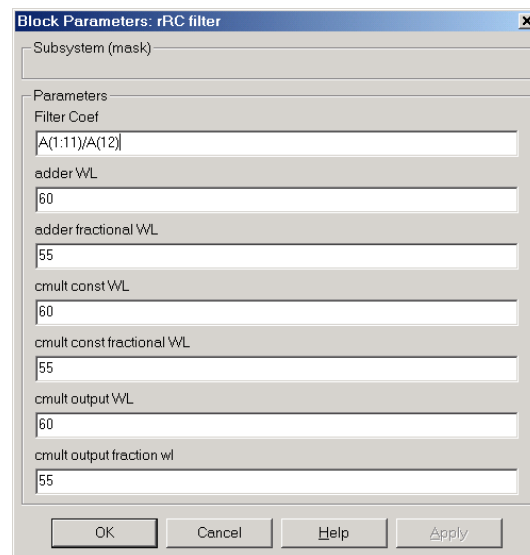


Figure 8. Mask parameters of LP rRC filter

Figure 7 and 8 show the detailed structure of the LP rRC filter, and its mask. 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.

7. Building Fixed-point System – Arithmetic Datatype 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 previous section, and to determine all the overflow and quantization modes. The goal is to have this done automatically, which is done using our floating-point to fixed-point conversion (FFC) tool [Shi02, Shi03, ShiBrodersen03].

In order to have the conversion, one needs to identify the node where the fixed-point and floating-point system difference will be checked. This is practically done by inserting a Specification Marker block from FFC library that is also located in H:\ffc directory. A natural place to place the marker is the node after gateway out block of the receiver. At this node one can detect the number of bit errors that caused solely by quantization errors. Suppose it has probability p , i.e. BQER (bit quantization error rate) = p ; then the

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

Theoretically this is fine. However in practice it is often a bad choice, not because perturbation theory fails, but due to the long simulation time to estimate this error accurately. In fact since we normally wish BQER less than BER we need about the same number of samples as the one in previous section to get a small confidence interval. That corresponds to minutes to hours simulation duration for each estimate, which is too long as we also need iterations. BTW, the total BER with both channel noise and quantization noise is not the sum of the BER(without QN) and this BQER, because slicer (demodulator) block is a nonlinear function of noise power.

A good choice to place the Specification Marker block is before the 2-PAM demodulator. One reason is that we know the rest of the receiver does not have word lengths to be determined. Also the MSE(flpt-fxpt) at this node gives a good indication of the BER system performance after the demodulator. In fact if we think QN and channel noise cause uncorrelated Gaussian noise at this node separately; then it is equivalent to think the total noise power as the sum of them. So one just needs to make sure QN power much less than channel noise power to quantify the statement “fxpt system differs only little from flpt system”.

A system with the marker specified is displayed in Figure 9. This newer version is named ffc_tutorial_v2.mdl. One can see that a specification marker has been placed after

the demodulator. In addition you can find some supporting Matlab files in the same directory (H:\ffc\); they are:

System_init.m
and A.mat.

In order to continue the demonstration yourself you need to copy ffc_tutorial_v2.mdl, system_init.m and A.mat into your personal directory of which you have write-access. To prevent possible hazard H:\ffc is read-access only. After the copy you can go to that directory and try the conversion tool yourself by typing in

```
>>ffc
```

Later when you have your own design you might want to have a directory for that design. In that directory you should have a file name system_init.m that initializes your pseudo-flpt system. In our example it load the filter coefficients A.

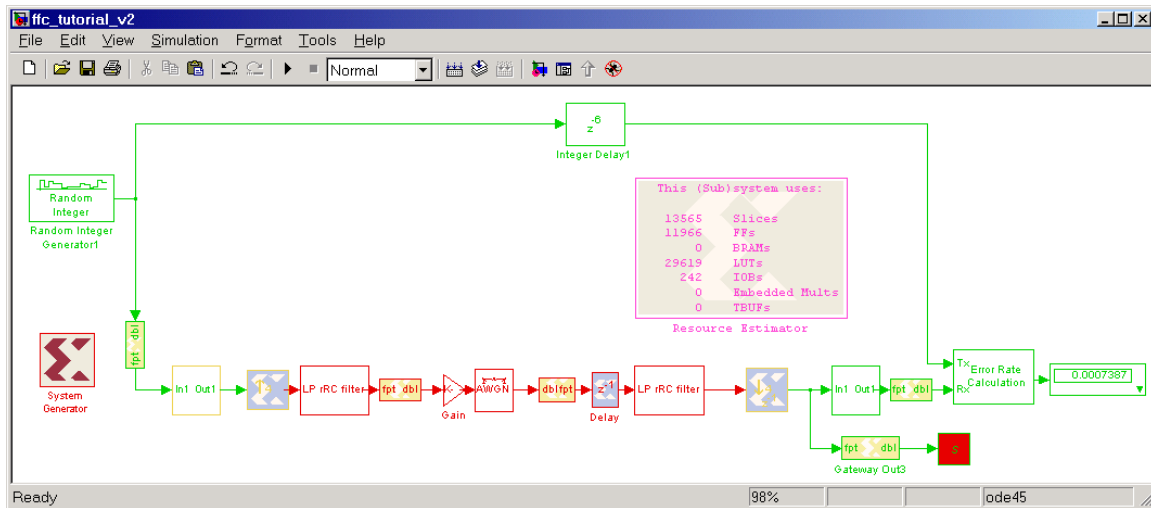


Figure 9. ffc_tutorial_v2.mdl file, with specification marker and resource estimator in. some blocks supporting the floating-point design are eliminated to speed up simulation time.

FFC this small system takes about 10 minutes. Ffc tool will sequentially ask you to input some important information you want to choose, such as design names. At one point it also asks you to change the model simulation time. You can let it to stop at $1/1e3$. This will make the simulation duration to be 1ms, which results in 1001 output samples after the downsampler. That will be enough to have a good MSE estimation.

Another important input is the MSE level you want to choose. You can either manually do a couple try to understand the relationship between your system performance (e.g. BER) and MSE. On the other hand you can do the following calculation to decide. The flpt system has $BER \sim 8 \times 10^{-4}$; so its SNR is about 8.4dB (refer to [Proakis01] chapter 5, figure 5.2-4 for the waterfall curve of BPSK system). Since the signal power is at

about .5 (you can estimate it by placing a eye-diagram scope before the demodulator, and see the signal power), the PN power is about

$$\text{PN power} \sim 0.5/10^{(8.4/10)} = 0.07.$$

We further want $\text{MSE} \ll \text{PN power}$. Notice at 8.4dB SNR the waterfall curve has derivative about 1.8dB/decade. If we hope the BER increase about 10% due to Q-noise, then the total SNR should become

$$8.4\text{dB} \times 1/10^{(1.8 \times \log_{10}(1+0.1)/10)} \sim 8.4\text{dB} \times (1-1.8 \times 0.1);$$

so the QN power is about

$$\text{QN power} = \text{PN power} \times (1.8 \times 0.1) = .012.$$

Using this MSE level, we achieve a fxpt system with BER of 9×10^{-4} . The total resource used is ~270 FPGA slices. The final system is save as ws_ffc_tutorial_v2.mdl, as shown in figure 10.

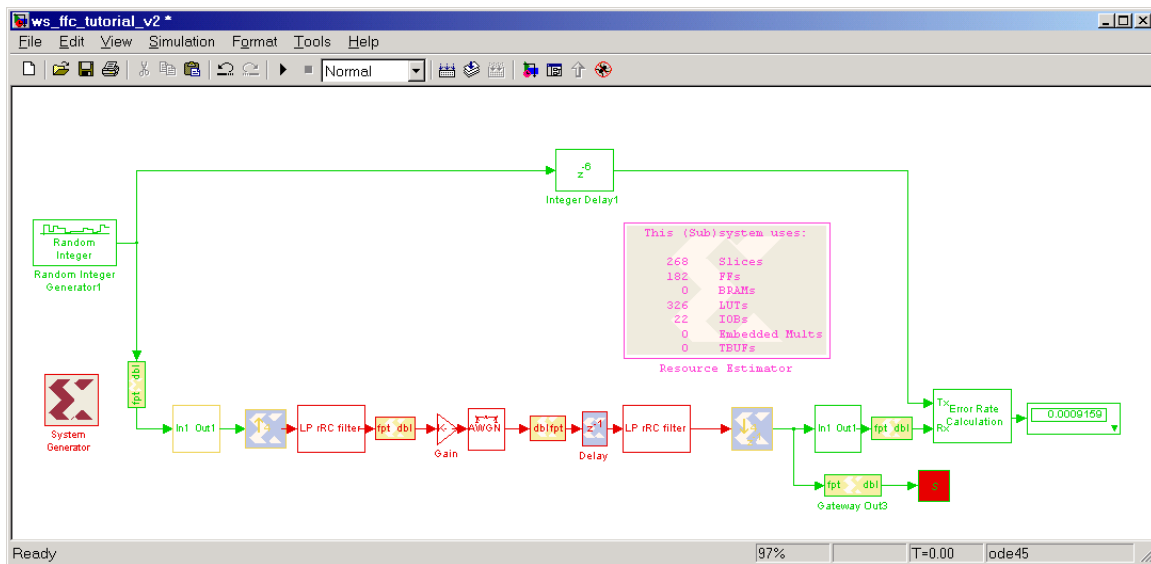


Figure 10. The final fxpt system having BER~ 9×10^{-4} , and ~270 FPGA slices.

You can choose Format->show Port Data Types to see the fxpt data-type used for the final system. In fact you can see some of the constant multipliers have coefficients zero now (since constant cannot be represented by the small WL fxpt datatype). But you don't need to worry too much about it since these logics will be eliminated in the final place and route stage.

8. An Important Remark

The most important remark I want to point out here is 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 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 sounds reasonable while we make the decision since it saves hardware and result less latency. However it’s fairly possible that with fixed-point datatype, we need too high wordlength to make the 40dB attenuation still true 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 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 truly “best” design. The bad news is a problem in this magnitude 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 in some sense (different WLs 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.

9. Conclusion

Via 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.

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 systems there. Notice that Xilinx demos are located at Blocksets→Xilinx directory in the demo window. An example is shown in Figure A1.

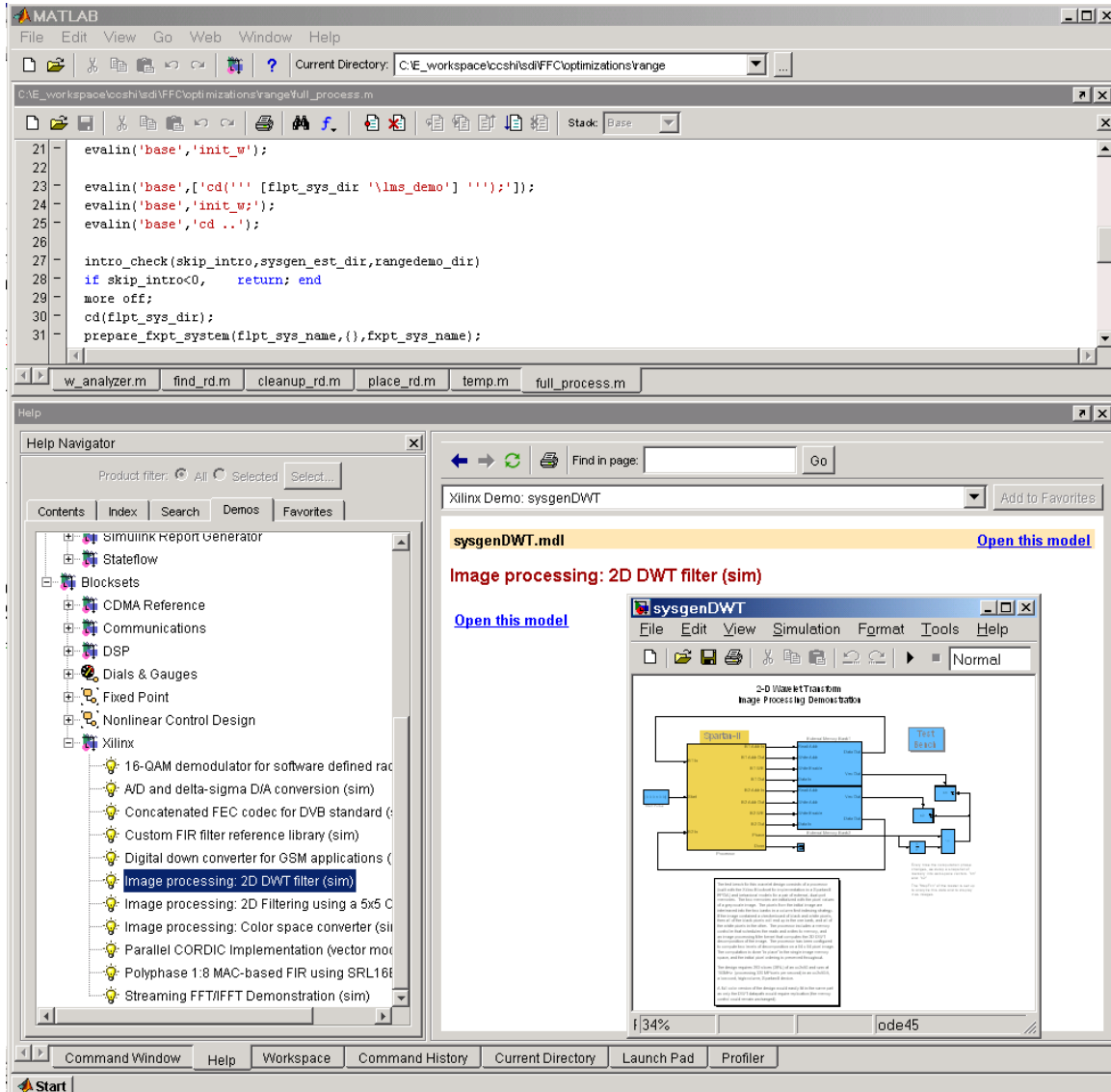


Figure A1. Using Matlab™ demos

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.

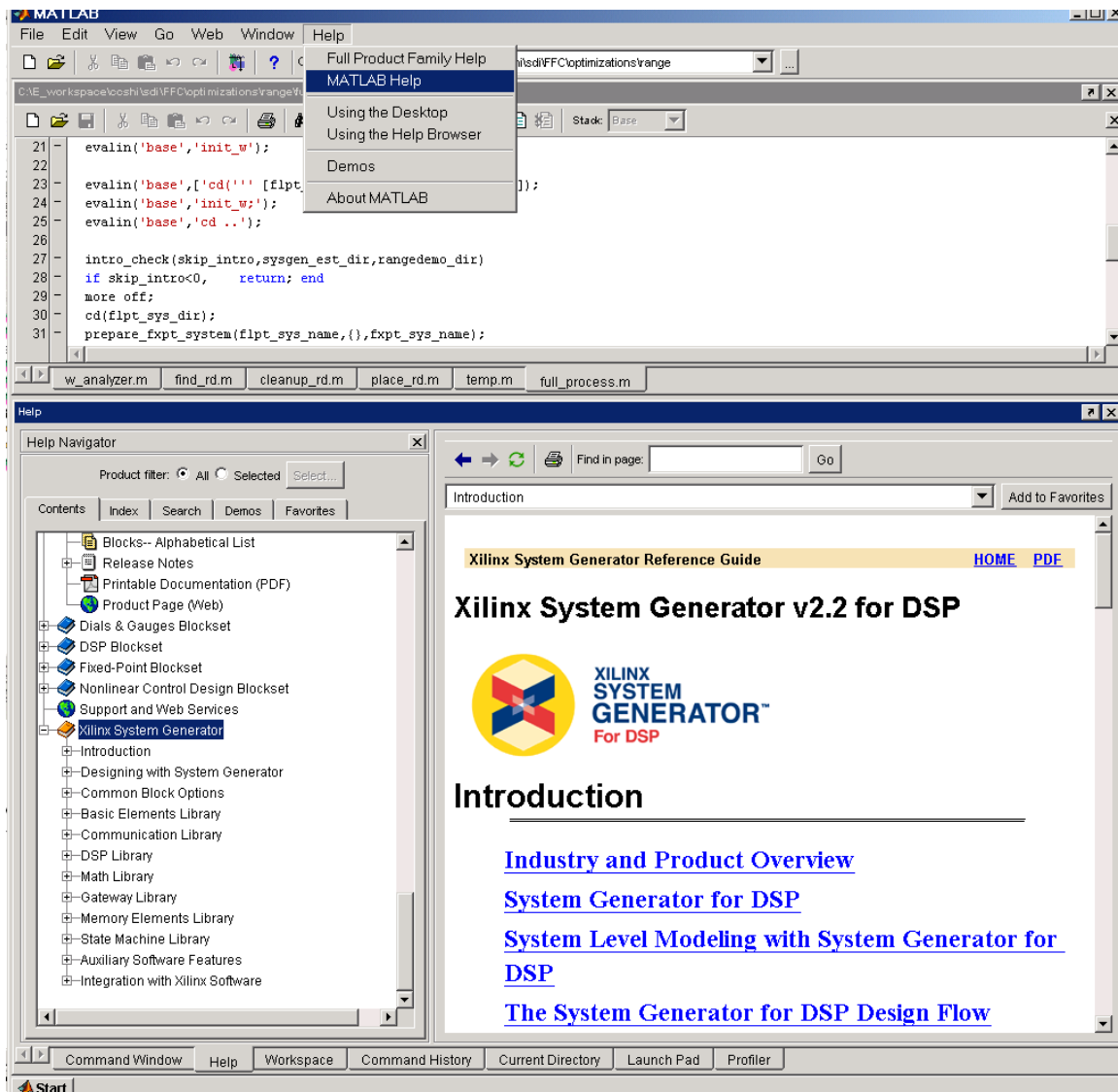


Figure A2: Using Matlab™ help system

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 swap the Xilinx library to a version that is prepared for 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.

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

```
>>ffc
```

This will 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.

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.

[OppenheimShafterBuck99]. Alan V. Oppenheim and Ronald W. Schafer 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.

[ShiBrodersen03]. C. Shi, R. Brodersen, “An Automated Floating-point to Fixed-point Conversion Methodology”, to appear in *ICASSP-2003*. Also available at <http://bwrc.eecs.berkeley.edu/Classes/EE225C/paper/shi.pdf>

[HW1]. [Assignment 1](#) of EE225c, Spring 2003. Taught by Prof. Brodersen, EECS Department. University of California, Berkeley.