DEVELOPING A RAPID PROTOTYPING METHOD USING A MATLAB/SIMULINK/FPGA DEVELOPMENT ENVIRONMENT TO ENABLE IMPORTING LEGACY CODE

Okhtay Azarmanesh (The Pennsylvania State University, Pennsylvania, USA; okhtay@psu.edu); and Sven G. Bilén (The Pennsylvania State University, Pennsylvania, USA, sbilen@psu.edu)

ABSTRACT

A rapid prototyping procedure is being developed in this project. As an example, a GMSK demodulator is simulated in a SIMULINK environment and the result is then being programmed in FPGA using Xilinx toolbox's block sets in SIMULINK. This method will enable us to easily develop and test different systems before implementing them completely. It will enable us to study the feasibility of a new SDR system on the hardware and, thus, it will considerably reduce the process time of testing new systems and waveforms on FPGA. This is particularly important when we wish to use legacy codes in a new system and be able to test the system as quickly and efficiently as possible.

1. INTRODUCTION

A conventional design process typically includes four phases, each of which contains certain limitations. These phases and their limitations are:

1. Requirements and specifications capture phase: prevents rapid iteration. These specifications are usually voluminous, making it hard to change a requirement without affecting the whole design.
2. Design phase: it is usually incomplete and cannot predict all the problems in a system and is an expensive process.
3. Implementation phase: human error can affect it significantly.
4. Test and verification phase: by this step, if an error is found it will be too late, meaning that it requires going back and doing the some or all of the previous steps again, raising the expenses for the whole process.

A model-based design process brings a number of improvements to the process of designing a system. Here are the improvements in each step [1]:

1. Specifications become unambiguous, supplemented by text. There is one set of models for all the teams involved. The whole system including its environment can be modeled and an early validation and test can be developed in this stage.
2. Design exploration and optimization become systematic. For this reason, flaws can be found before implementation. Bit- and cycle-accurate simulations of hardware-specific components are possible in this stage. There is an incremental design process from system level to implementation.
3. For implementation, the code is automatically generated, eliminating manual coding errors. This simplifies code portability from one hardware platform to another. This also bridges knowledge between the software and hardware domains.
4. There is continuous test and verification throughout the entire process, which allows for the detection of errors early in the development process and reduces dependency on physical prototypes. Test and verification ensures that the implementation will work the first time. Test suites also can be reused across the development stages.

Most SDR development organizations require incorporation of legacy code (i.e., existing code) into portions of their new applications. Legacy code often represents significant prior investment in design and test, so the ability to reuse is important. Examples of legacy code include math utilities, filters, table lookups, and low-level device drivers. Legacy code integration, however, is a complex problem where no single solution works all cases. Here, we deal with one aspect of the process: importing data into the Simulink models.

One can import legacy code into Simulink for model simulation and code generation. The generated code calls the imported legacy code based on its function call signature and data attributes. If most of the application comprises legacy code, however, it may be easier to export the generated code into the legacy application. (For example, you can apply a delta change to a large code base, based on the new algorithm that you modeled in Simulink).

1.1. Simulink and Model-based Design

Simulink is a very convenient environment for modeling, simulating, and implementing dynamic and embedded systems. Its advantages include the ability to simulate linear, nonlinear, discrete-time, continuous-time, and multi-rate systems. It also has an open architecture for integrating models from other tools. Simulink facilitates applications in

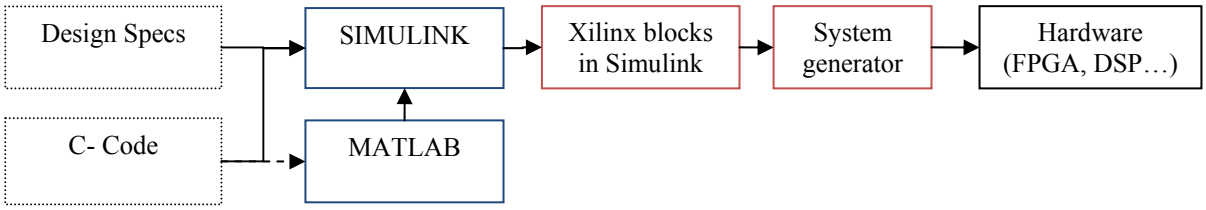

Figure 1. Diagram depicting a model-based design

controls, signal processing, communications, and other systems engineering areas [1].

In model-based design using Simulink, a number of potential advantages are evident. The overall process can be summarized as shown in Figure1**Error! Reference source not found.**. To begin, the design specifications are transformed into Simulink blocks and any C-code may be integrated into Simulink either directly or through MATLAB. There are several methods provided for within Simulink/MATLAB for enabling this integration:

1. S-function API: Real-time workshop in MATLAB. According to MATLAB's definition, an S-function is the computer language description of a Simulink block. They can be written in MATLAB, C, C++, FORTRAN, or Ada. They allow you to create your own blocks in Simulink;
2. S-function builder: Simulink; and
3. Legacy code tool (script-based): Real-time workshop in MATLAB. In Simulink 6, Model Explorer provides a Custom (legacy) code dialog that lets you import functions within generated code. Choosing Real-Time Workshop in Model Explorer will direct you to Custom Code dialog. This dialog enables placing legacy code in
    a. Source file
    b. Header file
    c. Initialize function
    d. Terminate function

After this process comes the implementation of the code in the hardware. The hardware used in this effort is a Lyrtech containing a Xilinx Virtex-2 FPGA and a TI DSP. The issue here is how to divide the tasks implemented in the code between FPGA and DSP. In other words, how do we best take advantage of the strength of each of them in our system?

1.2. DSP vs. FPGA

Although today FPGAs can handle many of the tasks traditionally done by DSPs, there are still a few factors to be considered. One consideration, for example, is the amount of memory an FPGA has access to. The FPGAs available today have some amount of RAM on-board, but still this memory is not comparable to the large external SDRAM that is normally available to a DSP. In spite of all the tools existing today for simulation, a DSP code will always need more memory sooner or later. The new FPGA models have used embedded SDRAM to try solving this problem.

Another consideration is the difficulty in making code changes. While FPGA code can be reconfigured for new modes of operation and feature enhancements, having the system implemented in both the DSP and FPGA generally makes it easier to reconfigure the design. This is due to the fact that the DSP is easily reprogrammable, but making changes on the FPGA can have a profound impact on gate and logic cell topology [4].

By way of example for this paper, model-based design has been put into test by designing a GMSK-based system. In Section 2 we explain how the design works. Section 3 discusses the issues encountered in implementing the design. The results from the simulations so far are presented in Section 4. Section 5 presents concluding remarks.

2. DESIGNING GMSK MODULATOR/DEMODULATOR

The model developed in Simulink includes a GMSK (Gaussian minimum shift keying) modulator and demodulator [2]. The modulator and demodulator are based on the models used in GSM (global system for mobile communications) systems. Figure 2 shows a block diagram of a complex equivalent baseband model of a GMSK modulator/demodulator [2]. Figure 3 shows the details of the completed model.

In a GSM system, there are 124 radio channels. Each channel contains 8 user channels, meaning that using TDMA (time-division multiple access) frames, the radio channel is divided into 8 time slots. In each time slot a burst of data as well as a training sequence can be sent. This training sequence can be used to estimate the channel impulse response. The burst is sent over a 900-MHz carrier using binary GMSK. The bandwidth is normalized so that $BT = 0.3$, where B is the bandwidth parameter, which represents the −3-dB bandwidth of the Gaussian pulse, and T is the symbol duration [2]. In this case where $BT = 0.3$, the GMSK pulse may be truncated at $|t| = 1.5T$ with relatively small error incurred for $t > 1.5T$ [3]. The pulse shape, $g(t)$, for GMSK is

$$g(t) = \left\{ Q\left[\frac{2\pi B\left(t-\frac{T}{2}\right)}{(\ln 2)^{\frac{1}{2}}}\right] - Q\left[\frac{2\pi B\left(t+\frac{T}{2}\right)}{(\ln 2)^{\frac{1}{2}}}\right]\right\}$$

with

$$Q(t) = \int_t^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dt$$

In this design, the complex baseband-equivalent system has been shown and the conversion from pass-band to complex baseband-equivalent model is discussed in Sections 3 and Figure 7.

In the modulator, the data bits are pre-coded differentially, which makes the modulation process Differential GMSK (DGMSK). Differential coding means that a transition from one amplitude level to another occurs only when a 1 is transmitted. The encoding operation is described by the relation [3]

$$\boldsymbol{b_k = a_k \oplus a_{k-1}} \qquad \mathbf{1}$$

where

- $\{a_k\}$ is the binary information sequence into the encoder,
- $\{b_k\}$ is the output sequence of the encoder, and
- $\oplus$ denotes addition modulo 2.

Some of the advantages of a differential encoder are (a) reliability close to that of theoretical limits for AWGN, and (b) improved resistance to pulsed noise and both continuous and time-varying narrow-band interference.

The next part is modulation. GMSK modulation is a continuous phase-modulation technique. Its prominent characteristics are its constant envelope (like all other phase modulations) and narrow bandwidth. It also deliberately introduces controlled ISI (inter-symbol interference) to improve spectral efficiency. The information is carried by the phase of the transmitted signal and the total phase signal is a linear function of the data sequence. We can approximate the baseband GMSK signal with a linear modulation approximation. Making use of the linear approximation, the received signal sampled at the symbol rate may be represented as

$$r_k = \sum_{n=0}^{L} (j)^{\mathrm{n}} d_n h_{k-n} + n_k \qquad \mathbf{2}$$

where

- $\{d_n\}$ is the original binary (±1) data sequence,
- $\{h_n\}$ represents the complex overall impulse response of the channel,
- $\{n_k\}$ is AWGN (additive white Gaussian noise) with variance N_0,
- $L + 1$ is the span of the channel response, and
- j $\sqrt{-1}$.

After the channel comes the demodulator part of the system. Due to the differential pre-coding performed at the transmitter, direct restoration of the original data sequence $\{d_n\}$ from the in-phase component of the received signal is possible. This can be done by a constant phase rotation of $\pi/2$ on Equation (2), which corresponds to multiplication by $(-j)^n$. This multiplication operation effectively performs differential decoding, so that after matched filtering the in-phase components will contain the information needed to restore the transmitted sequence. Thus, we may ignore the quadrature component and subsequently process only the real part of the received signal, treating it as a BPSK-type (binary phase-shift keying) signal. Therefore, the detector itself can be a real and computationally much simpler than its complex counter-part. This is known as a serial receiver as opposed to a parallel (in-phase and quadrature) receiver.

Since the coherence times of the mobile radio channels in a GSM system typically are much greater than the duration of a TDMA time slot, these channels can be characterized as slowly time-varying (flat fading), so the channel can be considered fixed during the burst period and, consequently, we only need to compute the channel estimate only once per burst. The estimation of the channel basically is cross correlating the middle part of the received burst after phase rotation. The position of the correlation peak is utilized for burst synchronization. This channel estimate is then used by the matched filter. The optimal receiver for this system consists of a continuous-time filter matched to the overall channel and then a symbol-space sampler and an MLSE (maximum likelihood sequence estimation) detector. The combination of the phase rotation and matched filtering performed on the received signal produces an output whose real component is used for estimating the data sequence $\{d_n\}$. For designing the optimal receiver, we need a continuous-time filter that is matched to the overall channel, followed by a symbol-space sampler and an MLSE detector. In our design, a discrete-time matched filter is used, being adaptively set up once per burst. The impulse response of such a filter is the time-reversed complex conjugate of the impulse response of the channel, which is expected.

We assume that the channel response spans in $L + 1$ symbol intervals, making $L + 1$ the largest number of symbols affected by the ISI. The simplified recursive metric that is maximized by the maximum likelihood estimate of data symbols $\{d_n\}$ is

$$L_n(d_n) = L_{n-1}(d_{n-1}) + \mathrm{Re}\left\{ d_n \left(y_n - \sum_{k=1}^{L} d_{n-k} x_k \right) \right\} \qquad \mathbf{3}$$

This suggests the use of Viterbi algorithm. Since the discrete-time impulse response estimate made available by channel estimator has length $L + 1$, the number of states in the Viterbi algorithm is 2^L. In this case, the complexity of MLSE grows exponentially with L.

2.1. Bit Error Rate

After the completion of the system, a number of simulations were performed to obtain the bit-error rate (BER) of the

system with respect to the amount of noise present in the AWGN channel. Simulation was performed with the signal-to-noise ratio (SNR) ranging from 4 dB to 26 dB and the result can be seen in Figure 6. The theoretical BER in AWGN channels for coherently detected GMSK is given approximately by [7]

$$\text{BER} = \frac{1}{2}\text{erfc}\left(\sqrt{\alpha\left(\frac{E_b}{N_0}\right)}\right) \qquad \textbf{4}$$

and for non-coherently detected GMSK is

$$\text{BER} = \frac{1}{2}\exp\left(-\alpha\left(\frac{E_b}{N_0}\right)\right) \qquad \textbf{5}$$

where

- α is a constant related to BT and for $BT = 0.3$, it is ~0.68,
- erfc corresponds to the error function, and
- E_b is the energy of the signal and N_0 is the energy of the noise.

3. DESIGNING USING XILINX BLOCKS

In this part, the model developed in the previous step is built again using the Xilinx toolbox.

3.1. Band-pass to Complex Base-band Conversion

The design of the models above is done assuming that we are dealing with complex base-band signals. To achieve this signal, however, we have to use a quadrature demodulator after our antenna to extract the complex envelope. We can write the received band pass waveform as follows

$$r(t) = \tilde{r}_I(t)\cos 2\pi f_c t - \tilde{r}_Q(t)\sin 2\pi f_c t \qquad \textbf{6}$$

where we wish to have the following

$$\tilde{r}_I(t) = [r(t)\cdot 2\cos 2\pi f_c t]_{LP} \qquad \textbf{7}$$

$$\tilde{r}_Q(t) = [-r(t)\cdot 2\sin 2\pi f_c t]_{LP} \qquad \textbf{8}$$

where "*LP*" indicates the low-pass filter to reject the double frequency term after demodulation.

For designing the filters, an already-designed filter was imported into Simulink using Simulink's Legacy Code tools to test the use of legacy code in our design. The in-phase and quadrature components of the received signals are being fed into a Viterbi decoder. For this part, the Viterbi decoder block that is present in the Xilinx block set has been used. The designed model is shown in Figure 4.

4. SIMULATION RESULTS

As mentioned earlier, this project has used MATLAB and Simulink to develop this method. The versions being used have been: MATLAB R14 (ver. 7.0.1); Simulink ver. 6.1 (R14SP1); Xilinx ISE 6.3i; Xilinx Sys Gen 6.3. We first designed the model in the Simulink environment and, after achieving the desired results, built the same model using the Xilinx block sets. Then, we used a Lyrtech SignalWAVe board as the hardware target to test the system with real signals.

Figure 5 shows the output of the modulator and the received signal on the demodulator. Comparing the data being sent in the modulator with the data that is being retrieved in the demodulator we have calculated the BER, which can be seen in Figure 6. In the BER curves in Figure 6, the theoretical limit is very different from that of our system, which is due to assumption of coherent detection in the theoretical BER calculations.

Figure 7 shows the method used for transforming the pass band signal in a phase-shift block to extract the in-phase and quadrature components. The phase rotation for transferring the data to the in-phase component is also done in this block.

5. CONCLUSION

It is been shown that a rapid prototyping system is an excellent method to deal with challenging system design problems. A model-based design flow reduces the complexity of the design considerably and makes it easy to perform simulations and modify the design based on simulation results. For future work, this project is going to further develop and test the model based design in real-time scenarios and with more complex systems.

ACKNOWLEDGMENT

This work was performed in part under a contract with Raytheon Intelligence and Information Systems, State College, PA.

REFERENCES

[1] E. Mayo and J. Bryan, "Model-Based Design of Communication Systems," *Mathworks Presentation*, 2008.

[2] B. Bjerke and J. Proakis, "A Comparison of GSM Receivers for Fading Multipath Channels with Adjacent- and Co-Channel Interference," *IEEE J. on Selected Areas in Comm.,* Vol. 18, No. 11, pp. 2211–2219, November 2000.

[3] J. Proakis, *Digital Communications*, McGraw–Hill, 2001.

[4] "FPGAs for Software Radios," Pentek, Inc., www.pentek.com.

[5] G.L. Stuber, *Principles of Mobile Communications*, Kluwer Academic Publishers, 2001.

[6] Matlab Help function, version 7.0.1.

[7] S. Elnoubi *et al*., "BER Performance of GMSK in Nakagami Fading Channels," 21st National Radio Science Conference (NRSC2004), 2004.

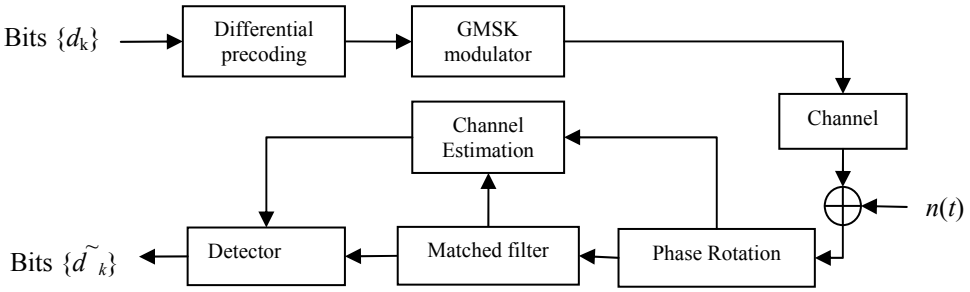

Figure 2. Complex equivalent baseband system model

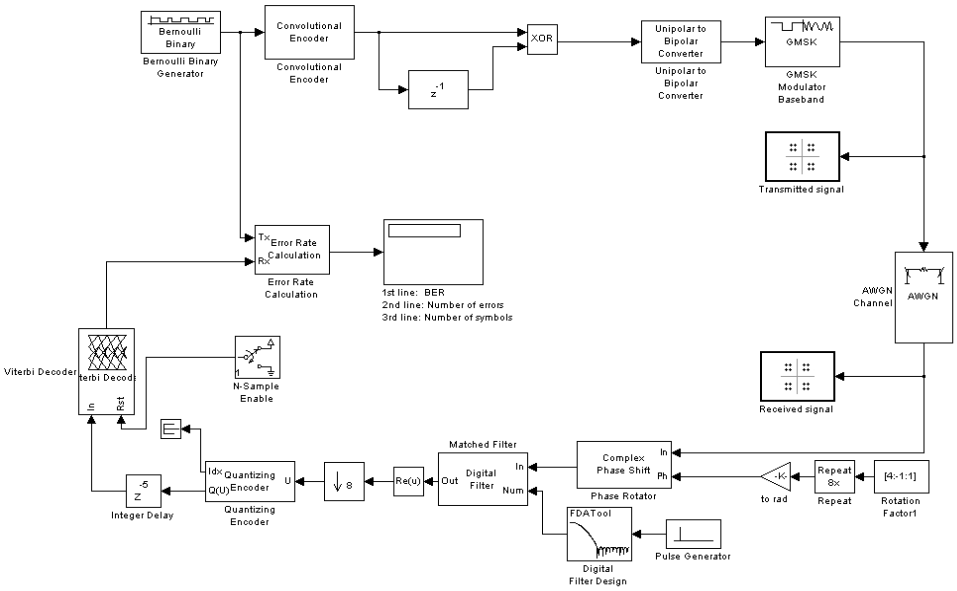

Figure 3. SIMULINK model of a GMSK modulator/demodulator

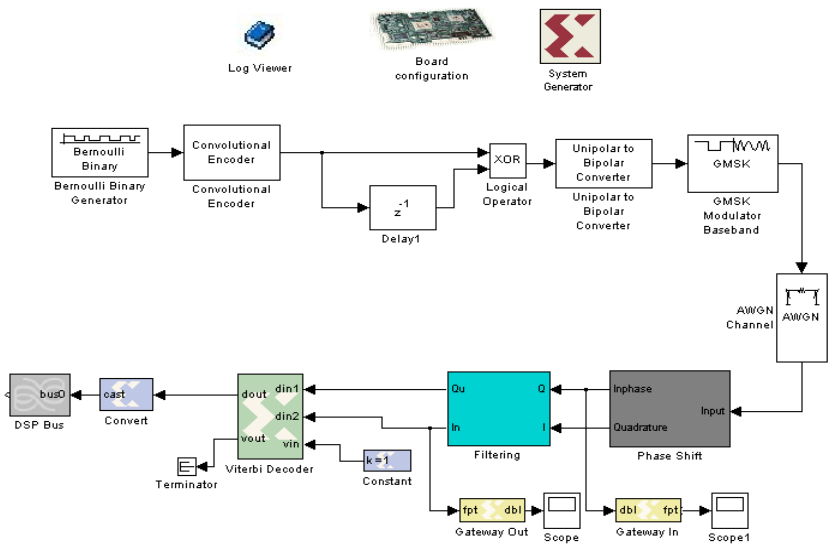

Figure 4. Demodulator built using Xilinx block sets

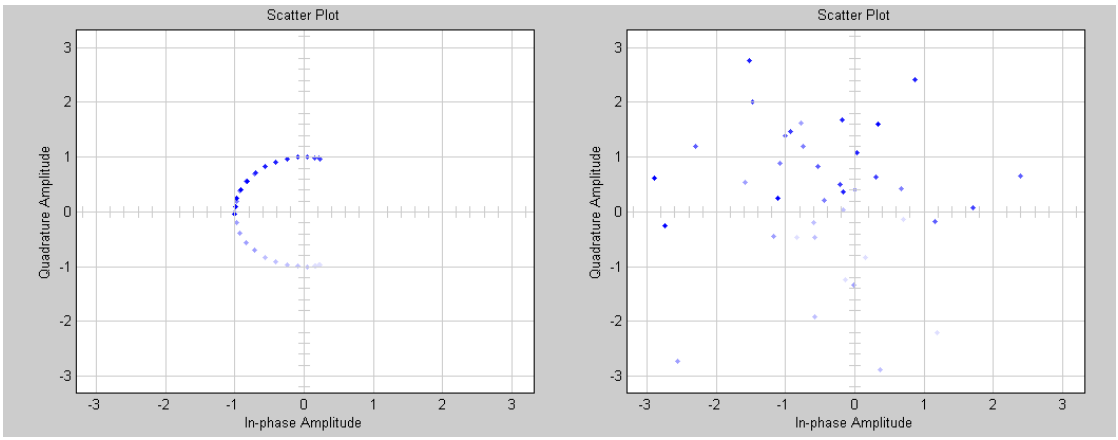

Figure 5. Left side shows the input to the channel and right side is the output feed to the demodulator

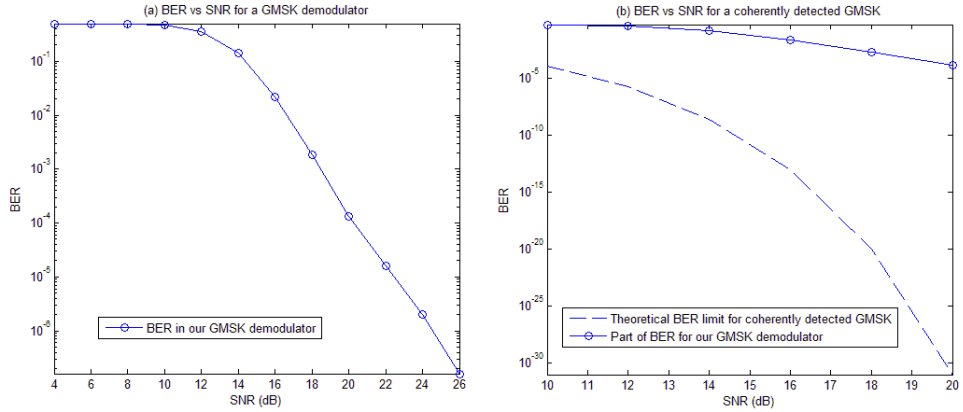

Figure 6. (a) BER vs SNR in our GMSK demodulator, and (b) part of BER vs. SNR being compared to a theoretical limit in an AWGN channel

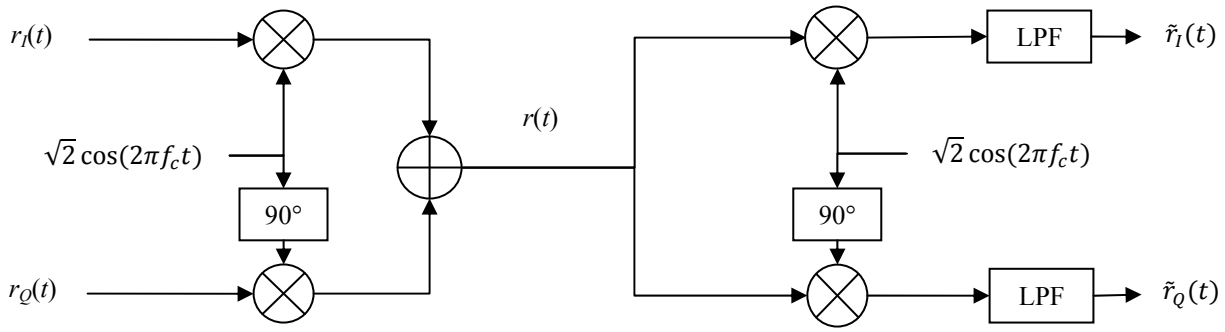

Figure 7. Transformation between band-pass and base-band components