Model-Based Design for Software Defined Radio on an FPGA

XIN CAI\(^1\), MINGDA ZHOU\(^2\), AND XINMING HUANG\(^{2,3}\)

\(^1\)State Key Laboratory of ES, School of Telecommunications Engineering, Xidian University, Xi’an, Shaanxi 710071, China
\(^2\)Department of Electrical and Computer Engineering, Worcester Polytechnic Institute, Worcester, MA 01609, USA
\(^3\)Nantong University, Jiangsu 226019, China

Corresponding author: Xinming Huang (xmhuang@ieee.org)

This work was supported in part by the U.S. NSF under Grant 1401121 and Grant 1414250 and in part by the NASA under Grant NNX16AB06A.

ABSTRACT This paper presents an approach of model-based design for implementing a digital communication system on a field programmable gate array (FPGA) for a software defined radio (SDR). SDR is a popular prototyping platform for wireless communication systems due to its flexibility and utility. A traditional SDR system performs nearly all computations and signal processing tasks on the host computer, and then sends the waveform to the RF front end. For complex algorithms or high data rate, the host computer becomes the processing bottleneck and FPGA is often employed as a hardware accelerator. This paper demonstrates the procedure of using model-based design for SDR targeted on FPGA hardware. A complete digital communication system, including a transmitter with convolutional encoder and a receiver with Viterbi decoder, is implemented on an FPGA-based SDR platform and validated by over-the-air demonstration. Synchronization algorithms, such as carrier frequency offset, phase offset, and time recovery, are also optimized for hardware efficiency.

INDEX TERMS Software defined radio, FPGA, convolutional code, Viterbi decoder, synchronization.

I. INTRODUCTION

In traditional development process, a digital communication system is designed, implemented and tested on an application-specific integrated circuit (ASIC). However, the high non-recurring engineering cost of an ASIC make it prohibitive for system prototyping. During the last decade, software defined radio (SDR) has become a popular platform for prototyping digital communication systems. It can configure the hardware to perform different functional operations as needed [1], [2]. SDR is usually described as a middleware which can define the radio and build the connection between the devices in hardware and software interface [3], [4]. This unique capability provide SDR with greater technological flexibility, which aroused great enthusiasm in the field of digital signal processing, digital communication, and RF engineering [5]. For high performance SDR system, field programmable gate array (FPGA) is often employed as a key component that can implement customized hardware logic and perform digital signal processing tasks at real-time. But it requires advanced hardware modeling and designs to target the communication system algorithms onto the FPGA. Traditionally a hardware description language (HDL) such as Verilog is used to model and simulate the digital system and subsequently to target onto the FPGA using synthesis and implementation tools.

Model-based design is a promising approach that can be adopted for rapid development of a communication system on a SDR platform [6]. There are several existing work on the implementations of communication systems using model-based design targeted on SDR for both educational and research purposes [7]–[9]. However, they were mostly focused on theoretical feasibility study without practical implementations. Meanwhile, there are many other work on the FPGA implementations of communication systems but most of them were focused on the hardware realization of a specific algorithm or a section of digital communication system. For instance, a pair of QPSK modulator and demodulator based on Xilinx Zynq Board targeting SDR was designed and implemented in [10]. Modulated the QPSK symbols are generated from a linear feedback shift register (LFSR), passed through root raised cosine (RRC), interpolation filters and then the signal is transmitted. Similar structure was also implemented in the receiver to recover the QPSK symbols. However, only the basic functional modules around QPSK modulation and demodulation instead of an entire communication system are implemented. In [11], hardware
optimization for carrier recovery is presented. The DQPSK signal was loaded onto a generated waveform, transmitted and then resampled by analog-to-digital converters (ADC). Finally the received signal was fed to a Virtex-7 FPGA board and optimized carrier recovery was performed. In addition, design and implementation of error correction coding/decoding are presented in [12], and both LDPC and Turbo coding techniques were tested with noise. Unfortunately, neither [11] nor [12] was involved with an entire communication system. Communication system with convolutional code was prototyped and implemented on WARP SDR platform in [13]. However, most of the computational operations were conducted on the host computer and the SDR boards were mainly used as radio front-end, resulting the fact that most of the FPGA hardware resources were unused.

In this paper, a complete communication system with all necessary components, including pulse shaping filters, modulation/demodulation, synchronization and timing recovery, and channel encoder/decoder, is designed and targeted on an FPGA-based SDR using model-based approach. The main contributions of this paper are as follows:

1. The prototype communication system is a model-based design, in which all the functional units are packed in separate models and can be substituted or customized by users.

2. The system is designed in Simulink with visual blocks. To implement their designed system to FPGA based SDR platform, users only need to align the dataflow compatible with FPGA interface and make sure that all blocks placed in their design are compatible with HDL Coder [14] for automatic generation of HDL code. It requires little knowledge of HDL programming and hardware pipeline design.

The rest of this paper is organized as follows. In Section II, the major design and structure of a transmitter is revealed and analyzed. In Section III, details of the receiver are discussed, including all computational units for signal processing. Implementation of transmitter and receiver on the SDR hardware is described in Section IV. Implementation results are addressed in Section V. Conclusion is made in Section VI.

II. MODEL-BASED DESIGN OF TRANSMITTER

System diagram of the QPSK transmitter implemented on the FPGA is shown in Figure 1, mainly including host-FPGA format aligning (see also Section IV-B), convolutional encoder, symbol mapping and pulse shaping. It is worth mentioning that no prior knowledge of HDL coding is required since all the units are designed in Simulink or Matlab code. Data generated from the host computer are fed into the FPGA and then get processed. These units are optimized for HDL code generation and perform baseband signal processing to transform the source data to baseband waveform. The baseband waveform are then upconverted to RF band by the SDR radio board and the RF band signals are transmitted over the air.

A. CONVOLUTIONAL CODING

Figure 2 shows the architecture of convolutional encoder. The convolutional encoder contains five submodules: pilot/payload splitter, encoder, pre-QPSK pilot formation, pre-QPSK payload formation and pilot/payload reunion. When source data, including both pilot and payload, come in this module, the valid signal and data are split first. Then the pilot/payload splitter takes both of them in and split the pilot bits and payload bits to make sure that only the
payload is encoded in the encoder submodule. As an example, the polynomial of the convolutional encoder is (101,111). Considering that the convolutional encoder doubles the data rate of payload, the rate of pilot also needs to be doubled. Both the pilot and payload pre-QPSK formation module pair two sequential bits into an unsigned two-bit output, which is the input format expected by the symbol mapping component. After pre-QPSK formation unit, pilot and encoded data are reframed in the pilot/payload reunion module.

**B. SYMBOL MAPPING**

Gray mapping is a multilevel modulation scheme in which there is only one bit change between two successive values. Gray codes are widely utilized to facilitate error correction by combining with forward error correction (FEC) codes. Due to the special mapping method, a receiver could recover the original constellation point which drifts into the area of an adjacent point. In the symbol mapping module, a baseband QPSK modulator block from communications system toolbox is adopted to realize the Gray mapping, which maps two-bit data [00,01,11,10] into the corresponding complex symbols \([1+i, -1+i, 1-i, -1-i]\).

**C. PULSE SHAPING**

In digital communications, pulse shaping filters are designed to remove the impact of intersymbol interference (ISI) on transmission process. Thus, the filters must have some special requirements: frequency response with sufficient selectivity and attenuation to suppress noise and interference in adjacent channels. Raised cosine (RC) filters are a family of responses that achieves the required zero ISI minimizing the occupied bandwidth [15], [16]. To make the transmitted signal fit in frequency band, the pulse shaping module adopts an FIR interpolation filter with a root raised cosine impulse response. In this module, the raised cosine transmit filter block performs root raised cosine pulse shaping with a roll off factor of 0.5 and an upsampling factor of four. The desired roll off factor ranges from 0.25 to 0.75. In order to make the pulse shaping equally distributed between the transmitter and receiver, transmitter side pulse shaping is often combined with matched filter on the receiver side. In this case, the two filters can achieve optimal tolerance for noise.

**III. MODEL-BASED DESIGN OF RECEIVER**

The detailed receiver design is given in this section, including coarse and fine frequency compensation, timing recovery with fixed-rate resampling, frame synchronization, demodulation and data decoding. The structure of computational units of receiver is shown in Figure 3. These units are also optimized for HDL code generation and implemented on the FPGA. In our demonstration, we choose the baseband sampling rate of 245.76 KHz or the symbol rate of 61.44K symbols per second. Considering the fact that the FPGA can process much faster than that, we buffer each frame of received data first before feed it to the FPGA processing unit. Similarly, the receiver output from the FPGA are also buffered and sent to the host PC to be translated from bits to readable message. This is to accommodate the high speed processing at the FPGA. After going through all the computational processes implemented on FPGA, invalid data are removed and valid data are translated and printed on the screen. The message printed on screen is expected to be a repeated sequence of Hello World ###, where ### are 3 digits increasing from 000 to 099.

**A. AUTOMATIC GAIN CONTROL**

The stability of the signal amplitude at the inputs of the carrier and timing recovery loops must be considered in the receiver design. The automatic gain control (AGC) is an adaptive subsystem and accounts for maintaining the amplitude and strength of the received signal. To make the signal’s amplitude stable, the gain of AGC usually drops when the received signal strength is strong and increases when received signal strength is weak [17]. The structure of AGC is shown in Figure 4. The product of the input signal and the value from the loop gain generates the output signals. The error signal comes from the difference between a reference value and the product of value from the loop gain and the modulus of the input. Once received signal is fed in, AGC is performed to guarantee the accuracy of all the operations. Considering the signals are upsampled, the AGC output power in this module
is set to $1/\sqrt{\text{upsampling factor}}$ so that the input amplitude of the coarse frequency compensation subsystem is stable and roughly one. Thus, the equivalent gains of the phase and timing error detectors can keep nearly constant over time.

**B. COARSE FREQUENCY COMPENSATION**

The coarse frequency compensation is designed to roughly estimate and compensate the frequency offset of the received signal. Figure 5 shows the coarse frequency compensation module. The received signals can be expressed as $e^{i(\pi n/2 + \Delta f t + \Delta \phi)}$, where $n$ is a designate phase index, $\Delta f$ is the frequency offset and $\Delta \phi$ is the phase offset, $n = 0, 1, 2, 3$, for QPSK. During coarse frequency compensation, the signal which comes out of AGC is raised to the power of four and the transformed signal $e^{i(4\Delta f t + 4\Delta \phi)}$ is obtained, which excludes QPSK modulation information. This power-raising operation is implemented by two product blocks. Luise algorithm [18] is implemented afterward to acquire $\Delta f$. This algorithm is based on autocorrelation method, which is implemented by an FIR filter and an accumulator. Actually, the filter is an autocorrelator in the structure of an FIR filter, which calculates autocorrelation value. Compared with an FFT method, this algorithm uses much less hardware resources. In addition, pipeline registers are used in the autocorrelation path to improve the speed. However, even if the coarse frequency compensation module estimates the major frequency offset, there is still a residual frequency offset. Usually the remaining frequency offset causes a rotation of the constellation.

**C. FINE FREQUENCY COMPENSATION**

Fine frequency compensation needs to be implemented after coarse frequency compensation. The fine frequency compensation module is shown in Figure 6, which further estimates the residual frequency offset. This unit is designed with a phase-locked loop (PLL), including a numerically controlled oscillator (NCO), phase error detector (PED) and a loop filter [19]. Since the majority of the frequency offset is compensated, the small residual frequency offset leads to a constant drifting of signal’s phase. Therefore a maximum likelihood PED is placed to estimate the phase difference between the current signal and the expected one. The phase difference is fed into a tunable proportional-plus-integral loop filter where the normalized loop bandwidth and damping factor are usually tuned as 0.06 and 2.5 [19], respectively. This configuration ensures that signal can quickly be locked to the expected phase while introducing little phase noise. The NCO block takes the loop filter output as input and acts as a complex exponential signal generator and therefore compensates the residual frequency offset.

**D. TIMING RECOVERY**

The timing recovery module calculates and compensates the timing error. The timing recovery subsystem is designed to choose the instants at which the incoming signals are sampled in the receiver. The timing recovery module is shown in Figure 7. It is also implemented with a PLL, which includes an interpolation filter, a zero-crossing timing error detector (ZCTED), a loop filter and an interpolation control module [19], [20]. The ZCTED detects the timing errors with the interpolators generated from the interpolation filter, cascaded by a tunable proportional-plus-integral loop filter [19], [21], whose normalized loop bandwidth is set to 0.01 and damping factor is set to unity to make PLL quickly lock to the correct timing. The loop filter’s output is fed into the interpolation control module to update the timing error. The purpose of the interpolation control block is to provide the interpolators with the basepoint index and fractional interval for each desired interpolant. The underflow signal will be activated and indicates one extra or missing interpolant when the timing error reaches symbol boundaries. The underflow signal also controls the ZCTED to decide when the ZCTED calculates the timing errors. The interpolation control block also updates the timing difference for the interpolation filter and generates the interpolants at the optimum sampling time. The interpolation filter is a farrow parabolic filter and is simplified with special parameters [19]. When the signal is fed into timing recovery module, the interpolation filter...
interpolates the signal according to the estimated timing error and corrects it.

**E. FRAME SYNCHRONIZATION AND DEMODULATION**

Figure 8 shows the frame synchronization and demodulation module. This module is designed to find the header of a frame so that the data can be located and demodulated. In this work, a 13-bit Barker code is adopted as the pilot [22]. With the characteristics of the Barker code, auto-correlation function is executed in the matched filter block to correlate the Barker code against the received signal [23]. Then the modulus of the matched filter output is compared with a threshold. The threshold value has an important influence on miss probability and frame synchronization accuracy. Usually a larger value increases the miss probability whereas a smaller value may result less accurate frame alignment. Thus, it is a trade-off especially at lower signal to noise ratio (SNR). For high SNR, a larger threshold value can be used. In our design, the threshold is set to 16, which is a compromised value between miss probability and accuracy for the consideration of both low and high SNR cases. Once the output value of modulus block exceeds the threshold, it is indicated that a frame is detected and therefore its pilot and payload can be accurately located. The frame control block is also enabled, which splits a frame into two parts: pilot and payload. The demodulation module demodulates the payload and each symbol is mapped into two bits based on the Gray mapping method. Before feeding the output of matched filter to frame control block, a conjugate operation is adopted to resolve the phase ambiguity issue since passing signal through the matched filter brings extra phase shift. Therefore the matched filter output is fed into a conjugate block to neutralize the extra phase shift.

**F. Viterbi DECODER**

Viterbi decoder in the receiver is the correspondence to the convolutional encoder in transmitter. Detailed structure of Viterbi decoder is demonstrated in Figure 9. This is a general hard decision Viterbi decoder, which has three main components: branch metric computation (BMC), add-compare and select (ACS), and traceback decoding (TBD) [24]. The BMC calculates the branch metrics, which are Hamming distances between possible symbols in the code alphabet and the received symbols. The ACS adds the branch metrics to possible decoding paths, compares the results and then selects the most possible decoding path. The selected decoding path is the maximum-likelihood path and is restored in TBD. A TBD utilizes a first-in-last-out (FILO) buffer to reorder the decoded signal.

Since the preamble Barker code at beginning of each frame is not encoded by the rate 1/2 convolutional encoder, the Viterbi decoder need to be bypassed for the preamble and re-enabled for decoding the payload. Practically,
we introduce an external enable signal to the Viterbi decoder unit. When enable is low, the Viterbi decoder freezes at current state, skip those invalid data and continue once data is valid again. Therefore, the enable signal in the Viterbi decoder is built around the general logic and freezes all intermediate states when enable signal is low. Meanwhile, to avoid the impact of pilot on decoding process, another block is placed right before the Viterbi decoder, as can be seen in Figure 3. This block sets all detected pilot data to zero to guarantee that the decoder starts fresh at the beginning of each frame. To measure the performance of this Viterbi decoder, 10 million bits are coded with SNR ranging from 1 to 10 dB and fed into the decoder. The bit error rate (BER) versus SNR is illustrated in Figure 10.

IV. IMPLEMENTATION AND INTEGRATION

In spite of all the computational units implemented on FPGA as illustrated in Section III, the host PC is still responsible for some functional parts of the SDR system, such as data generation, host-FPGA interfaces and data presentation. Figure 11 and 12 are Simulink models running on the PC when performing over the air transmission, for transmitter and receiver respectively. Figure 12 (b) shows the functional units inside QPSK Receiver block. As shown in Figure 11, the host PC generates pilot and payload data, aligns them (both format and data rate) with FPGA interface, and then feeds data to the radio (FPGA and AD9361). Similar structure can also be found in Figure 12, the PC receives processed data from radio, aligns them with PC itself, digs out the valid data and print them out in the workspace. The functional units running on PC are mainly data preparation and presentation since all the computational units are already implemented on FPGA.

A. DATA GENERATION

Before baseband processing, the host PC needs to prepare the well-organized data for transmitter. Figure 13 shows the structure of generated data frames. A 13-bit Barker code is adopted as the pilot. With the consideration of getting the best likelihood of detection and synchronization while preserving the autocorrelation properties of the original sequence, the Barker code is mapped to 26 bits in transmitter simply by upsampling and allocating the same sequence for both I and Q channel. The pilot is used to overcome channel impairments and locate the frame header in the receiver. The payload includes 105 bits of data and 69 bits Bernoulli random bits. The 105 bits are the ASCII representation of Hello world ###, where ### is a repeating sequence increasing from 001 to 099. As can be seen in Figure 13, with half rate convolutional encoder, the data rate of payload is doubled.

B. INTERFACE BETWEEN THE HOST COMPUTER AND THE FPGA

Different from host PC, the FPGA board of the SDR platform processes data streamingly. Hence, data fed into FPGA is accompanied by its valid signal so that the FPGA know when to process the received data. Since the FPGA board in the SDR platform requires only one 32-bit data bus: 16 higher bits for real part and 16 lower bits for imaginary part, data format alignment between PC and FPGA is necessary. An example of host PC to FPGA interface alignment is shown in Figure 14. After a frame is composed, the PC to FPGA alignment block converts every data bit into a 16-bit integer (higher 16 bits) and accompanied by lower 16 bits containing the valid bit. Then the data and valid bits are combined to a complex signal that is compatible with FPGA bus.

Similarly, we split the data and valid signal in FPGA for the purpose of signal processing and computation. When FPGA finishes its task, it is also required to align the format of data and valid signal to fit the bus. An example of FPGA to PC format aligning is shown in Figure 15.

C. DATA REPRESENTATION

After the Viterbi decoder finishes decoding, the decoded frame data will be sent to the dataframer block, as shown in Figure 12 (b). It receives the streaming data coming out of the FPGA and reframes the data. This block processes the decoded data only when it is indicated that the data is valid. Then in data print block (also shown in Figure 12 (b)), the pilot and 69 bits redundancy are discarded and every 7 bits in the payload will be converted to a corresponding ASCII symbol. While the simulation is running, the data print block also compares the received and expected payload bits to compute accumulative bit error rate.

V. IMPLEMENTATION RESULTS

To implement all the signal processing units on FPGA and run the prototype communication system, we utilize the Workflow Advisor in Simulink [25]. It helps generate the HDL code of our design and corresponding FPGA bit file so that all the computational units of Tx and Rx can be targeted onto FPGA. As described earlier, the Simulink models of retargeted transmitter and receiver are shown in Figure 11 and 12 respectively. Since our design is aimed to transmit and receive signals in all directions, an omni directional antenna is selected. In this paper, each SDR platform consists of an FPGA board (ZedBoard), an AD9361/9364 RF transceiver and an omni directional antenna. The center frequency of each SDR platform is 2.4 GHz (tunable from
70 MHz to 6.0 GHz) and baseband sample rate is 245.76 KHz (tunable from 200 KHz to 56 MHz). The host PC and SDR platforms are connect through CAT-5 Ethernet cable. Both Tx and Rx of the SDR hardware are shown in Figure 16.

When running the targeted models of both Tx and Rx, the data are first generated by the PC, sent to Tx, processed in the FPGA board of Tx and transmitted over the air. Then the received signal at Rx are processed in the FPGA board and sent back to PC. The translated messages are displayed on screen. The average bit error rate (BER) is also displayed on screen every 50 frames while the BER itself is always calculated cumulatively. The SDR platform and its real time implementation results are demonstrated in Figure 16.
To examine the performance of our implementation, when we run the communication system and perform over-the-air transmission, 100,000 frames are transmitted (105 bits payload data per frame, roughly 10 million bits in total) and the cumulative BER is located between $3 \times 10^{-4}$ and $3.5 \times 10^{-4}$. The experimental results are satisfactory and match with the BER performance at 10dB SNR in the simulation.

VI. CONCLUSION

In this paper, a model-based design of a complete communication system with channel coding is presented and all computational units of the system are implemented on an FPGA-based SDR platform. All the units within our design are modular and can be modified by users as long as the dataflow and interface are aligned with corresponding FPGA board. The performance of the model-based design is evaluated through simulations and also demonstrated by over-the-air transmission using the SDR hardware.

ACKNOWLEDGEMENT

The authors would like to thank The Mathworks, Inc. for their generous technical support and Dr. Yanjie Peng for his assistance during this research.

REFERENCES


