Compact Digital Implementation of a Non-Coherent IR-UWB Transmitter and Receiver

Andrew Hennessy and Amirhossein Alimohammad
Department of Electrical and Computer Engineering
San Diego State University, San Diego, USA

Abstract

Non-coherent impulse-radio ultra-wideband (IR-UWB) transceivers are attractive candidates for applications where silicon area and power consumption are relatively limited. This article presents the compact digital architecture design and implementation of a non-coherent IR-UWB transmitter and receiver based on the energy detection scheme, including the synchronizer module. The software models of the designed transceiver are simulated and verified in both floating-point and fixed-point numerical representations. The synthesizable Verilog description of the transmitter and receiver architectures are simulated and verified against their fixed-point simulation model. The transmitter and receiver are implemented on our custom-developed field-programmable gate array (FPGA) board. The bit error rate performance of the transmitter and receiver is measured in real-time on the FPGA, utilizing an accurate on-chip Gaussian noise generator. The characteristics and implementation results of the transmitter and receiver architecture on the FPGA are presented. An ASIC architecture of the IR-UWB transceiver is estimated to occupy 0.0227 mm^2 and dissipate .760 mW from a 1.0-V supply while operating at 82 MHz in a standard 32-nm CMOS technology.

Index Terms– Digital impulse-radio ultra-wideband (IR-UWB) transmitter and receiver, field-programmable gate array (FPGA), ASIC architecture.

I. Introduction

The impulse-radio ultra-wideband (IR-UWB) technology [1] supports high data rate wireless transmission of processed neural data from the implantable device to an external controller

This work was supported (in part) by the Center for Sensorimotor Neural Engineering (CSNE), a National Science Foundation Engineering Research Center (EEC-1028725).

The authors are with the Department of Electrical and Computer Engineering, San Diego State University, CA, USA.

reliably [2], while requiring relatively low power consumption [3], [4]. The main idea behind IR-UWB is to generate a very short pulse or electromagnetic wave (few nanoseconds), which has a very large bandwidth (few GHz), at specific time instants. Since an UWB transmitter sends signals in short pulses, most of its circuits can be shut off during the time when it is not transmitting. Moreover, since the minimum energy per bit decreases when the normalized bandwidth increases, an impulse transmitter can achieve low power consumption as it trades spectral efficiency for energy efficiency [5]. In IR-UWB, data is encoded by modulating a short pulse's position, amplitude, and/or polarity. The multitude of modulation and implementation options for IR-UWB continue to be investigated [6]. A non-coherent receiver is responsible for finding and decoding messages, while it alleviates the need for exact synchronization between the transmitter and receiver, i.e., prior knowledge of the timing of a transmission is not necessary. Moreover, non-coherent receivers need less information about the channel and can be implemented with lower complexity compared to coherent receivers [7]. The combination of the non-coherent reception and short duration of the pulses in the time domain make the IR-UWB scheme particularly suitable for short range communications with extremely low power constraints.

Fig. 1a shows the high-level block diagram of the designed and implemented IR-UWB digital transmitter and receiver over an additive white Gaussian noise (AWGN) channel. The transmitter *Tx* is responsible for creating a signal using a bit stream and a template waveform provided as inputs. The transmitter combines the bit stream with a preamble to create a packet that can be detected and decoded by the receiver *Rx*. The signal created by the transmitter is represented as a series of short pulses in the time domain. The distortion is modeled using an AWGN channel. The modulated signal from the transmitter (i.e., sequence of pulses) is added to the samples from the Gaussian noise generator (GNG) and the noise-corrupted signal is passed to the receiver. The receiver is responsible for taking the noisy signal samples and recreating the bit stream that was presented to the transmitter. Controlling the operations of the receiver is done by a finite state machine (FSM).

We first modeled the IR-UWB transmitter and receiver in floating-point representation in Matlab. The fixed-point representation of the transmitter and receiver is then modeled using a custom-developed library of parameterizable fixed-point operations in MEX-C. The Verilog descriptions of the transmitter and receiver are developed and the cycle-accurate bit-true

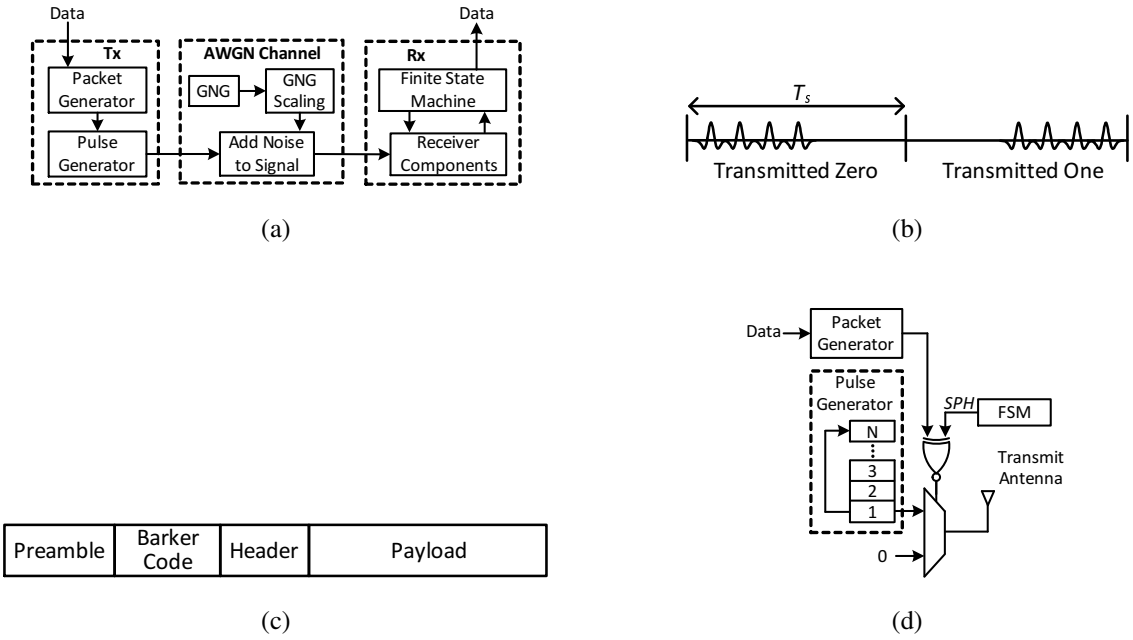


Fig. 1: (a) Block diagram of the IR-UWB transmitter and receiver over AWGN channel. (b) Two binary PPM frames. (c) Packet structure. (d) Datapath of the IR-UWB transmitter.

implementations of the transmitter and receiver are analyzed and verified against the fixed-point model. The transmitter and receiver are implemented on a custom-developed field-programmable gate array (FPGA) board hosting a relatively small Xilinx Spartan-6 FPGA. The on-chip bit error rate performance of the transmitter and receiver using our accurate GNG [8] is measured in real-time and is compared against the fixed-point software simulation results. The transceiver architecture is synthesized in a 32-nm CMOS technology and the silicon area, performance, and power consumption of the transceiver are presented.

The rest of this article is organized as follows. Section II and III present the transmitter's and receiver's descriptions and their compact hardware architectures, respectively. The components of the transmitter and receiver are individually described and their efficient hardware implementation is elaborated. Section IV presents the simulation results and implementation characteristics of the designed IR-UWB transmitter and receiver on a FPGA and also in a standard 32-nm CMOS process. Section V makes some concluding remarks.

II. Transmitter and AWGN Channel

To transmit information, physical properties of the pulses from the transmitter are modified to reflect the transmission data. The energy to transmit one bit is spread over a series of low-power pulses grouped together as a pulse train [9]. A pulse train for IR-UWB transmission combines ultra-short pulses into a collection of pulses that is easier to detect in the time domain. Data is transferred by manipulating these pulse trains. The collection of pulses that constitute one pulse train have enough energy to be distinguished from the random channel noise.

Transmitted pulses are modulated using the binary pulse position modulation (PPM) scheme [10]. Modulating data using the binary PPM scheme involves moving a data pulse train in the time domain. The pulse train used is a collection of voltage samples used in [11]. The pulse train is represented as 99 consecutive voltage samples. The data pulse is moved into one of the two possible pulse train locations that make up one symbol period. For one symbol period T_s, which composed of 198 consecutive samples, only half of the symbol window will contain data from the transmitter. Placing a series of sequential pulses at the start of T_s and ending at the midpoint indicates a zero while placing the train from the midpoint to the end of T_s indicates a one. Fig. 1b shows an example of two successive PPM symbol periods as transmitted by the transmitter with Gaussian pulses representing the presence of a pulse train.

The success of the transmission depends on the mutual understanding between the transmitter and receiver on both the modulation scheme as well as the overall packet structure of every transmission. The packet structure used for transmitting every symbol is shown in Fig. 1c. The signal for one complete packet of data is made of four sections. The first section is the preamble, which is a long train of all zeros to indicate the beginning of a new transmission. The preamble used in our implementation uses 100 consecutive zeros for detection and synchronization. The preamble is terminated by the Barker code [12]. The Barker code is an 11-bit sequence 00011101101 used as a start frame delimiter (SFD) following the end of the preamble. The presence of a SFD indicates to the receiver where the preamble ends and the data to be decoded begins. The Barker code is followed by the header bits that contain the number of bits in the transmission data. The header is used to indicate where the receiver should transition between decoding the current packet and trying to detect the next packet. While the sizes of the first three blocks are fixed, the length of the transmission data varies based on the number specified

in the header. The final part of the modulated signal is the transmission data.

The implemented transmitter creates a modulated signal, represented as signal strength values, to be detected and decoded by the receiver. These signal strength values represent samples taken from the waveform after a transmitting antenna, but before the corruption that results from the transmission channel. The transmitter consists of a pulse generator and a packet generator, as shown in Fig. 1d. The pulse generator contains a series of discrete sample values that make up a pulse train. The output of the transmitter seen over one symbol period is created by transmitting the samples taken from a pulse train with a matching number of zero-valued pulses. The block of zero-valued samples are transmitted at the front or the back of the pulse train based on the value for the current symbol period provided by the packet generator. The packet generator takes in the transmission data to combine into a packet structure. A complete packet is formed by using the packet structure to sequentially fill symbol periods with pulse trains in the appropriate half of the symbol period.

The transmitter is implemented in hardware by storing pulse samples that are passed to the transmit antenna in the appropriate time slot. The passing of these pulse train samples is managed by a FSM that keeps track of the current bit from the packet generator as well as the last sample passed to the output. The FSM uses counters to manage the current bit position as well as the current pulse position within the symbol period. The pulse train is made of 99 sample values that are passed to the output of the transmitter sequentially. The multiplexer in Fig. 1d is used to pass through either a pulse train sample from the storage location or a zero value. The selection is determined by the current bit from the packet generator and the value of the counter tracking the current position within the symbol period. If the current position of the counter is inside the first half of the symbol period, the SPH (Symbol Period Half) control signal is set low by the FSM. The use of an XNOR gate allows the multiplexer to pass the sample only if the current bit matches the value of SPH.

III. Non-Coherent Receiver

The receiver process starts with the detector that finds the preamble at the start of a packet and decodes all of the data following the preamble. Fig. 2a shows the flowchart of the receiver's operations. The receiver is broken into four main stages. The received pulses are first processed by a correlator after they have been distorted by the channel. The receiver uses an autocorrelation

function to distinguish the incoming pulses from the received noise. The detector is responsible for determining if the incoming samples make up the preamble or are just noise from the channel. The synchronizer is then responsible for finding the boundaries of the symbol period so the decoder can evaluate the symbol contents. The decoder is responsible for converting the incoming pulses into a bit stream. The decoding process is broken into finding the Barker code, decoding the header, and decoding the payload. The output of the receiver, *Output Data*, is the generated bit stream of data created by the decoding of the payload.

The structure of the implemented receiver is shown in Fig. 2b. Because the arrival of the incoming samples is sequential, portions of the structure remain idle while another component is active and using the incoming data. The process of managing the sequential inputs for the separate components of the receiver is controlled using a FSM. The FSM is responsible for deactivating idle components and activating the appropriate components throughout the reception process to manage the power consumption. Note that the autocorrelator, which is the first stage of the receiver, is always active regardless of the state of the receiver FSM. The autocorrelated samples are passed to every component of the receiver as new samples are needed for evaluation in every block of the reception process. The detector and the synchronizer both contain a sub-component, referred to as an integration block, used to integrate the incoming samples of the successive integration windows. The decoder compares the received bit stream with the Barker code.

A. Autocorrelator

The correlation of a signal involves multiplying the received samples with a template to reduce the impact of the channel distortion. The suppression of the noise value in the received samples facilitates a more accurate detection and decoding of the received transmitted data. Our non-coherent receiver uses the energy detection scheme [13], or an autocorrelation function with a zero time delay. This process essentially squares all of the input signal samples before they are processed by the detector. Squaring and integrating the samples collected by the receiver creates a sequence of energy samples that can be more reliably detected and decoded. Fig. 2c shows the block diagram of the autocorrelation of input samples passed through an AWGN channel, where (WI, WF) denotes the number of integer bits WI and the number of fractional bits WF of a signal. The input to the receiver, *Incoming Samples*, is an energy sample in the time domain. The corrupted

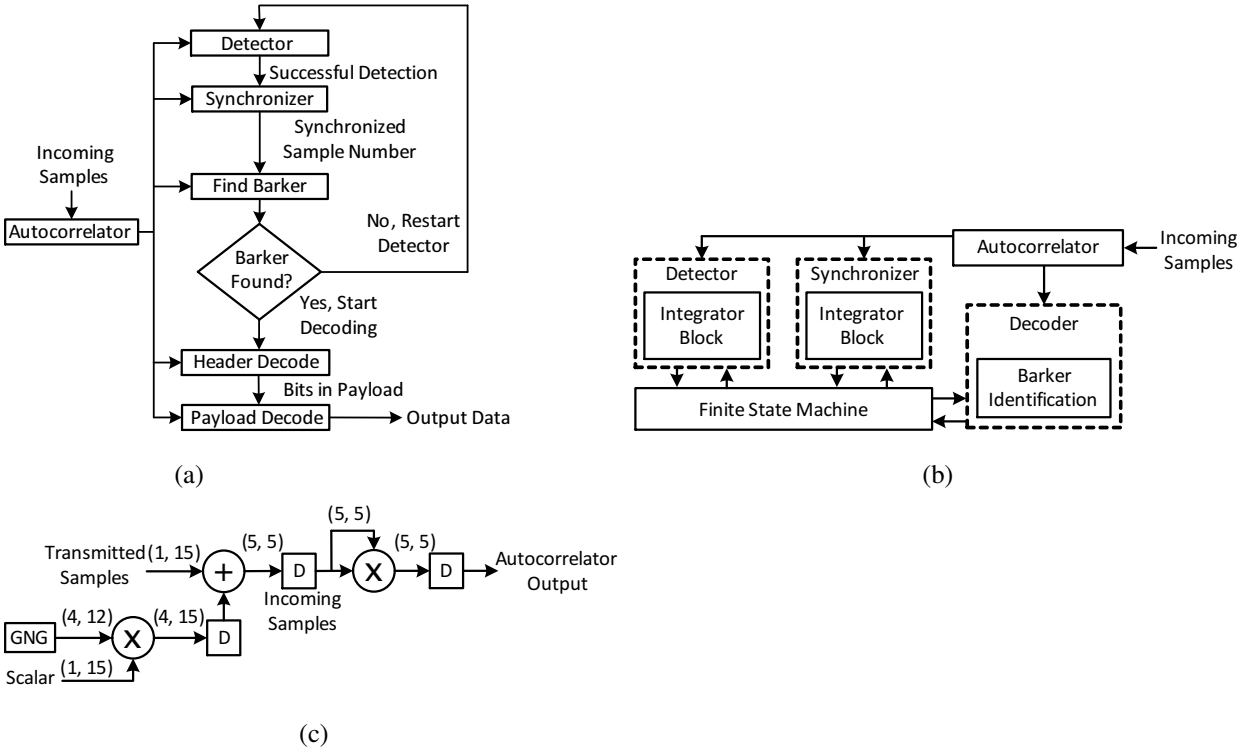


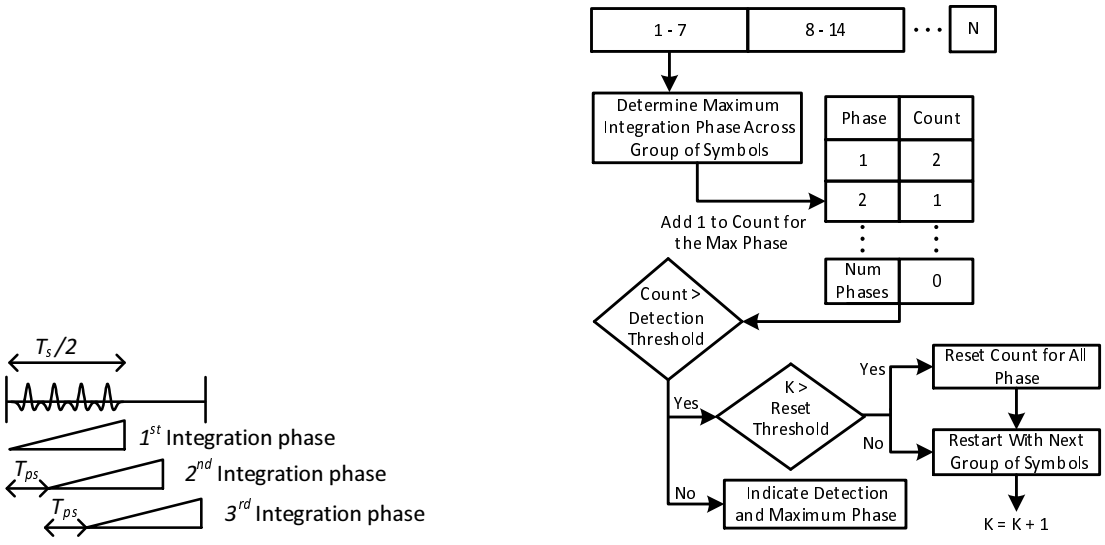
Fig. 2: (a) Flowchart of the IR-UWB receiver's operations. (b) Block diagram of the IR-UWB receiver. (c) Autocorrelation of received samples.

samples are passed to both inputs of a multiplier to perform an autocorrelation. The link between transmitter and receiver is improved by using cuteset retiming to shorten a relatively long critical path delay. Cutset retiming is a transformation technique used to add pipeline registers and/or change the location of the delay elements without affecting the input/output characteristics of the design [14]. A cutset intersects a set of edges of a graph such that if these edges are removed from the graph, the graph becomes disjoint. Cuteset retiming is used to improve the performance of the transceiver throughout the design.

B. Detector

The detection process runs continuously until a preamble is detected. The detector uses received energy samples to indicate the presence of the preamble. The detector can find the consecutive zeros of the preamble because the pulse train for a received zero is always placed in the first half of the symbol period. The detector uses a series of integration phases to find

the preamble. As shown in Fig. 3a, multiple integrators are swept over the samples of one symbol period. Every sample in the symbol period is integrated as a part of multiple overlapping integration phases. Rather than evaluating one integration phase during a symbol period, all of the integration phases are totaled during every symbol period. In [13], one integration phase is totaled during every symbol period. Evaluating every integration phase during one symbol period means the integration totals are evaluated using fewer symbol periods and a shorter preamble can be used. Each integration phase is half of one symbol period and spans the time duration of exactly one pulse train. The integration phases are separated in time by the phase space $T_{ps} = \left\lfloor \frac{T_s}{N} \right\rfloor$. Increasing the number of integration phases results in a smaller phase space between phases. As the transmitted pulses are received, overlapping integration phases are processed and the sum of each integration phase is stored. Integrating the energy samples in a time window can be used to detect a signal because the phases containing a modulated signal will have higher integration values than phases only containing noise. The example shown in Fig. 3a would result in the first integration phase having the highest integration value for that symbol period.



(a) (b)

Fig. 3: (a) Overlapping integration phases. (b) Flowchart of the detection process.

Given that the received signal is a train of zeros during the preamble, the integration phase with the highest sum should be the same for every symbol period in the preamble. To ensure the receiver is receiving a train of zeros and not just noise samples, one integration phase must be

the highest integration phase over multiple symbol periods. The number of successive symbol periods checked by the detector is determined by two input threshold values. One input threshold determines how many symbol periods are integrated before the integration phases are compared. The second input threshold determines how many times the integration phases must be the highest integration in the comparison. A third input threshold, the reset threshold, determines how many symbol periods are evaluated before the detection process is restarted.

Fig. 3b shows the flowchart of the detection process where seven bits are used to accumulate the energy values for every integration phase. After the integration of this group of symbols, the integration phases are compared. Following every group comparison, the phase with the highest accumulated energy has the value denoted as *Count* incremented by one. If a value of *Count* is greater than the threshold for detection, the detector indicates the presence of a preamble. The value of K indicates the number of groups of seven symbols that have been checked so far. If K is greater than the reset threshold, then the value of K and the values of *Count* for each phase are reset to zero. Following this comparison, the process is restarted with the next group of symbols and the value of K is incremented.

The detection phase uses a relatively small number of integration phases and has a relatively large phase space between integration phases. Using a smaller number of integration phases is important for the hardware implementation as it means that fewer integration values must be evaluated during the detection process. Moreover, the detector can run for extended periods of time and does not need to accurately return the sample at the start of a symbol period.

Using a pulse train comprised of 99 data samples, the symbol period contains 198 samples. The detection process evaluates all the symbol periods using 9 integration phases. By using 9 integration windows instead of the 8 used in [13], the phase space is $\left\lfloor \frac{198}{9} \right\rfloor = 22$ samples. Using this phase space means that the integration phases span the entire symbol period and the first integration phase always aligns with the first sample of the symbol period. This implementation simplifies tracking the passing symbol periods as the integration phases are swept over.

The detector incorporates an integrator block responsible for accumulating the incoming samples from the autocorrelator. At every clock cycle, the integrator block provides the value of the integration of all the samples that make up one full integration phase. The integration values for each phase are accumulated using groups *NumAve* = 7 bits to calculate the maximum phase. In order to evaluate the integration phases over 7 consecutive bits, the value of each

integration phase is stored in a rotating shift register. Fig. 4a shows the implementation of the energy collection component of the detector. The rotating shift register has an index to store a value for every integration phase. The 6-bit *SymbolCounter* is used to count the number of symbol periods that have been integrated. At every subsequent clock cycle, the contents of the shift register are all shifted down one location. The new input to the top of the shift register is the addition of the bottom element in the shift register with the newest integration phase from the integrator block. For the first symbol period in a group, the values from the integrator block are stored directly in the shift register.

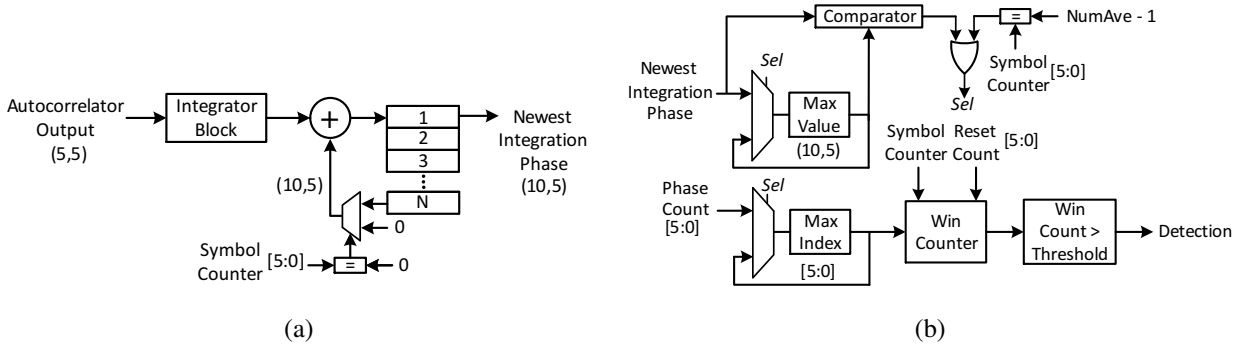


Fig. 4: (a) Datapath of the energy collection component. (b) Datapath of the detection evaluation.

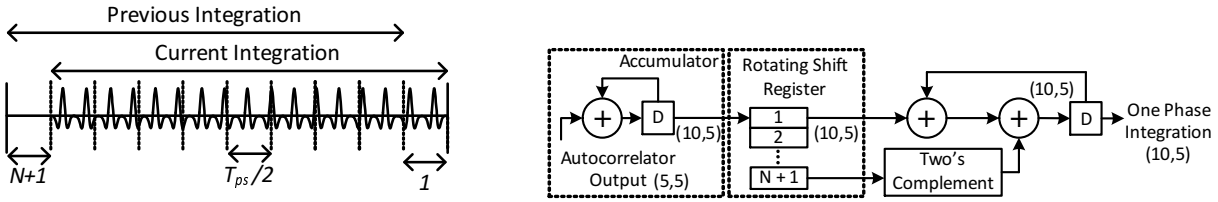
Fig. 4b shows the datapath of the comparison logic used to establish a detection. The input to the comparison logic is the newest integration phase from the detection shift register. Following the integration of a new phase, its value is compared with the current maximum integration phase. If the value of the top location of the shift register is greater than the current maximum, the current maximum is overwritten and the current phase count is stored. Following the evaluation of 7 consecutive bits, the maximum phase over those seven bits is considered the winner for those symbol periods. In addition to a rotating shift register, the detector also includes an array for the storage of the count of number of wins for each phase. When a phase is determined to be a winner, the index matching the current maximum phase register is incremented.

Following every seven bits, the detector then checks if any of the integration phases have been the maximum integration phase at least 6 times. If one phase has been the highest at least 6 times before 11 checks, then the detector has found the preamble. If the detector successfully finds the preamble, a control signal is asserted to alert the FSM that it can move forward in the

reception process. If no integration phase was the maximum integration phase at least 6 times within the span of 11 checks, then the detection process restarts and the counters tracking the number of times each phase has been a maximum are reset to zero. The FSM does not disable the detector until a preamble is successfully identified.

C. Integrator Block

Detection and synchronization are based on evaluating integration phases that contain the integration of all incoming samples between two points in time. A large number of these samples need to be integrated as part of multiple overlapping integration phases. Because the integration phases overlap, a subsequent integration phase will contain one full phase space of new samples as well as all of the samples from the previous integration, excluding the oldest phase space. Fig. 5a shows the possible contents of the previous integration followed by the contents of the current integration. In the integrator block for the detector, the phase space is 22 samples long. To calculate the current integration window, 22 new incoming samples need to be added while the 22 oldest samples need to be removed from the total.



(a) (b)

Fig. 5: (a) Subsequent integration phases. (b) Datapath of the integrator block.

Fig. 5b shows the datapath of the integrator block. The samples from the autocorrelator are first fed into an accumulator. The output of the integrator block is the accumulation of one full integration phase, which is equal to half of a symbol period or 99 consecutive samples. Thus, the integrator block needs to be able to manage groups of 22 samples to accommodate the phase space and 99 samples to produce the accumulation of one full integration window. The integrator block achieves this by using a rotating shift register to store smaller sub-windows of 11 integrated samples as well as an output register that keeps a running total. These 11-sample sub-windows are pushed into a rotating shift register. The rotating shift register accommodates $N+1$ locations,

where N is the number of integration phases. Whenever a new sub-window is calculated by the accumulator, it is added to the running shift register, pushing the older accumulations down one location. The newest addition to the shift register is added to the current running total at the output. Element $N + 1$ of the rotating shift register represents the oldest sub-window and is subtracted from the running total. The first valid integration phase occurs when the rotating shift register is filled.

For a compact realization of the integration block, a feedback structure is used to prevent repetitive additions. The accumulation of overlapping integration phases is done using the result of the previous integration phase as part of the calculation of the current phase. Although the latency of the accumulation is increased, the use of the feedback register eliminates redundant additions, allowing the integrator block to accumulate overlapping integration windows while remaining compact. At every clock cycle, the integration block updates the output register by adding the newest sub-window of 11 samples and removing the oldest sub-window at the bottom of the rotating shift register. This provides a new integration of the incoming samples shifted by half of the phase space. At every clock cycle, the oldest samples are removed form the running total while the newest samples are added, preventing the redundant additions. Every push onto the shift register covers half of one phase space so the result of adding the newest value with the feedback value creates a valid integration of one window every other clock cycle.

When the integrator block is first started, the accumulator must complete N sub-windows for the output register to contain the integration of the samples of one complete integration window. This point is where the counter in the detector is started as it evaluates a number of consecutive symbol periods. Following the initial filling of the rotating shift register, the integrator block provides integration values for the overlapped integration windows every second clock cycle.

D. Synchronization

After a successful detection, the synchronization stage is started immediately. The accuracy of decoding the transmitted data is largely dependent on the synchronizer's ability to estimate the boundaries of the symbol period. The synchronization process uses a series of overlapping integration phases over the preamble to attempt to find the beginning of a transmitted bit [15]. Fixing the preamble waveforms at the start of the symbol period allows the receiver to establish the start of each transmitted bit. Synchronization follows a process similar to the energy

integration process shown in the detection phase. Synchronization uses the integration process seen in Fig. 3a, but when compared to detection, the phase space is considerably smaller and more integration phases are used per symbol period. During the synchronization stage, integration windows cover series of preamble bits with the aim of finding the integration phase with the highest energy. The integration phase with the most energy accumulated over the preamble is most likely the closest to encompassing the entire modulated waveform in a symbol period. The synchronization stage gives a much better approximation of the start of the symbol period by using several more integration phases. Having a high number of integration phases means that the phase space is smaller and the integration phase is more likely to align with a symbol period.

The process of synchronization is shown in Fig. 6a. Unlike the detection stage, which runs until a preamble is found, the synchronization stage always occurs over a constant number of symbol periods. The maximum integration phase is evaluated after every symbol period for a number of symbols determined by an input parameter. In our implementation, integration phases are evaluated over 22 preamble bits to find the start of the symbol period.

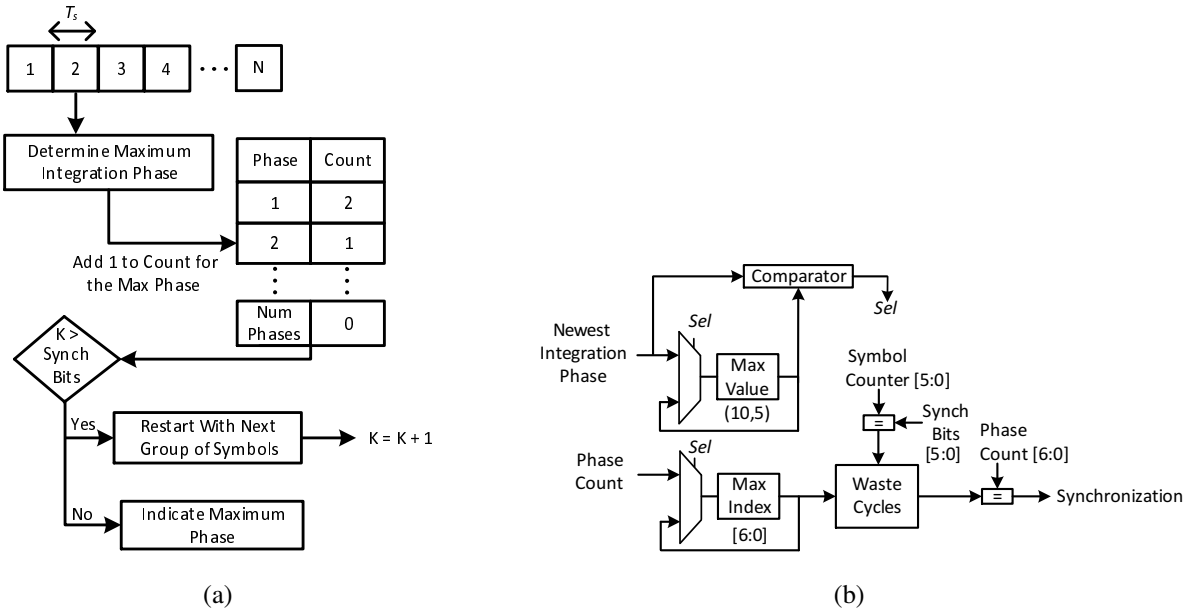


Fig. 6: (a) Flowchart of the synchronization process. (b) Datapath of the synchronizer.

Similar to the detector, the synchronizer pairs an integrator block with a rotating shift register to maintain values for integrated windows. The number of integration phases was increased from

32 as used in [13] to 33 to make the repetition of the first integration phase align with the start of the next symbol period. Using 33 integration phases, the integrator block contains 34 locations, each for half a phase space. Fig. 6b shows the datapath of the synchronizer. The synchronizer is run for a set number of symbol periods *Synch Bits*, indicated as an input. The synchronizer tracks the number of symbol periods that have elapsed by using a phase counter to increment the *Symbol Counter*. The synchronizer also includes a register to keep the current maximum value as well as phase of the integration windows as they are pushed into the shift register. Following the integration of 22 symbol periods, the synchronizer then indicates the integration phase with the maximum value. When the value of *Symbol Counter* matches *Synch Bits*, the synchronizer continues to increment the phase counter until the phase counter matches the value of the register containing the index of the maximum phase, *Waste Cycles*. When the comparison is successful, the synchronizer indicates to the FSM that the reception is now synchronized to the start of the symbol period.

E. Decoder

The decoder is started by the FSM immediately following the indication of the completion of the synchronization. The decoding process is done by integrating over only two integration windows to span the symbol period. The integration windows are separated into the first and the second half of the symbol period, as shown in Fig. 7a. The integration for decoding begins at the start of a symbol period, which is estimated by the synchronizer. After integrating one full symbol period, the integration values are compared and a new binary value is added into the data array. Following binary PPM, a higher integration in the left window indicates a zero and a higher integration in the right window indicates a one.

Fig. 7b shows the datapath of the decoder. The decoder uses two accumulators to integrate the first and second half of a symbol period for comparison. The multiplexers placed before the registers in the datapath allow one of the accumulators to effectively be switched off for a period of time. By tying the autocorrelator output to alternating terminals of the multiplexers, one select line can be used to determine which accumulator receives incoming samples while the other stores its data by continuously adding a zero value. The FSM tracks a sample number used to determine which register receives new values from the autocorrelator to be accumulated. The sample counter in the FSM is also used to extract the result of the comparator when a full

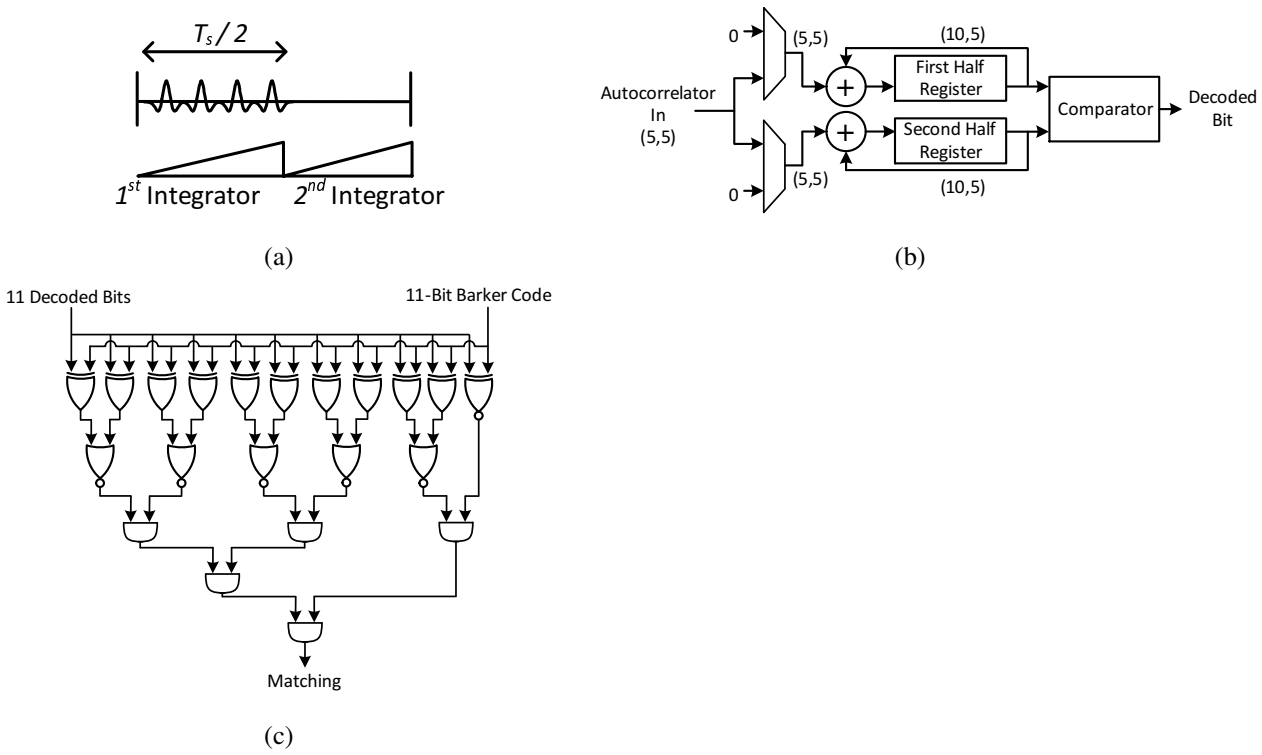


Fig. 7: (a) Two integration windows for decoding. (b) Datapath of the decoder. (c) Logic for comparing decoded bits with the Barker code.

symbol period has been accumulated. The decoder continuously creates an output bit stream until the FSM deactivates the decoder. The FSM maintains a register bank that matches the length of the maximum number of bits in the payload. As the decoder generates bits, these bits are pushed into a register that holds the decoded data. The integrators and comparator in the decoder provide the FSM with a bit stream of decoded data.

When the decoder is first activated, so is a sub-component of the decoder that receives the output bit stream and looks for the Barker code. The decoder fills the data array from the least significant bit and the array is compared with the Barker code array following the integration of every symbol period. The sub-component to find the Barker code compares the last 11 bits decoded by the decoder with the known Barker code. The logic used to make the comparison of the decoded bit stream with the Barker code is shown in Fig. 7c. The output of this gate configuration goes high only when all eleven bits match the input SFD bits. While this sub-component is active, the FSM keeps track of the number of bits added to the bit stream that

are compared with the input SFD. This sub-component continues to run until the Barker code is found or there is a timeout. If the Barker code is found, the sub-component looking for the SFD is deactivated while the decoder continues to run. If the Barker code is not found before the timeout, the FSM stops the decoder and restarts the detector.

The Barker code indicates the end of the preamble and the start of the payload header. The header is decoded using a similar process to the Barker code. Following the positive indication to the FSM generated by the logic shown in Fig. 7c, the integrators are used to fill a data array with a length specified as a parameter to the receiver. The process of decoding the header uses the same two integration windows, but stops after the number of symbols indicated by the input parameter. The decoder uses the number of bits to repeat the energy collection process to decode the payload. When the payload is decoded, the receiver returns to the detection phase to look for the next transmission.

IV. Performance Evaluation and FPGA Implementation Results

We modeled the IR-UWB transmitter and receiver in both floating-point and fixed-point representations in Matlab/C as well as in a fully-parameterizable synthesizable Verilog hardware description. We used a library of custom developed, parameterizable fixed-point operations. Fig. 8a shows the bit error rate (BER) performance of the transmitter and receiver over a range of signal-to-noise ratio (SNR) values modeled in software as well as the results of the FPGA simulation. We used our custom-developed FPGA board hosting a relatively small Spartan-6 Xilinx FPGA. The BER performance is evaluated using our on-chip accurate and scalable GNG presented in [8]. The hardware results are created using the fractional bitwidths of components as depicted in their respective figures with no component ever keeping more than 5 fractional bits. Fig. 8b shows the BER performance of the receiver for various numbers of fractional bits. While using a smaller number of fractional bits will reduce the size of the transmitter and receiver architecture, the use of fewer fractional bits causes a degradation in the error rate performance, as shown in Fig. 8b.

Table I gives the characteristics and implementation results of the IR-UWB transmitter and receiver on a Spartan-6 FPGA. The size of this implementation is based on maintaining 5 bits of fraction in the receiver's modules. Tables II and III give the power consumption and the silicon area of the transmitter and receiver, respectively, synthesized in a standard 32-nm CMOS

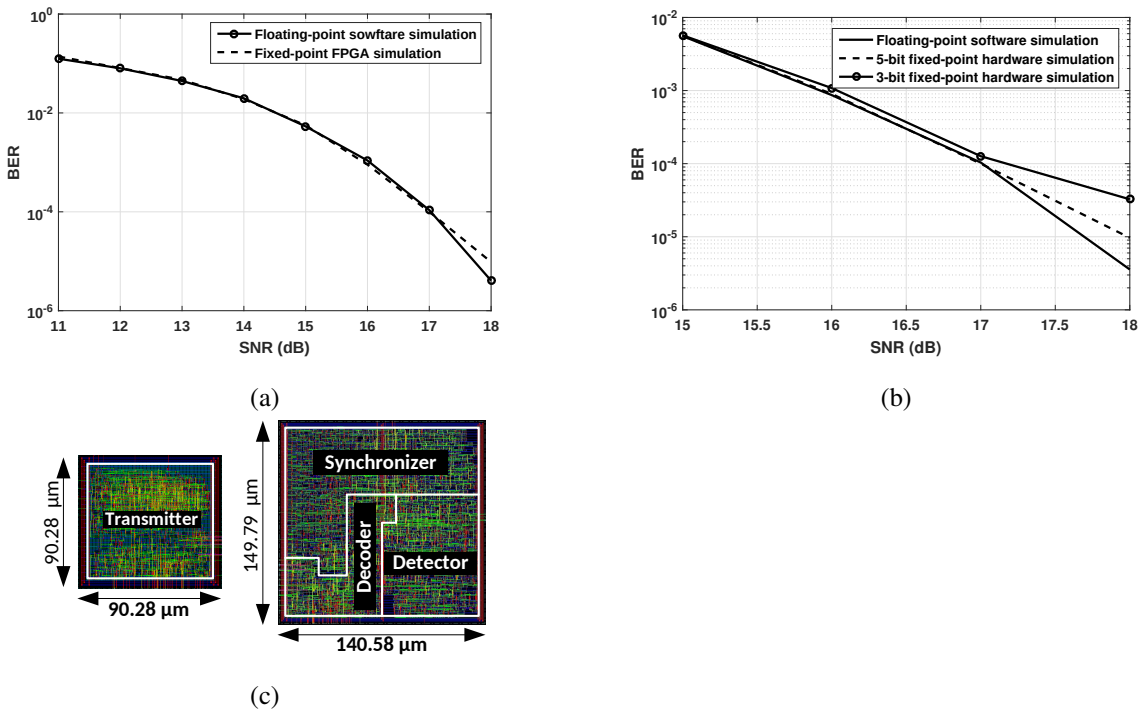


Fig. 8: (a) BER performance of the designed IR-UWB transceiver in software and implemented on a Spartan-6 FPGA. (b) BER performance of the implemented IR-UWB transceiver for different fractional bitwidths. (c) Chip layouts of the IR-UWB transmitter and receiver in a 32-nm CMOS technology.

process. The power consumption was measured after place and route using IC Compiler by synopsys. The receiver is about seven to eight times the size of the transmitter and consumes about six times more power at 82 MHz from a 1.0-V supply. Fig. 8c shows the chip layouts of the transmitter and receiver.

To the best of our knowledge, the only other reported digital implementation of an IR-UWB transmitter and receiver in a 90-nm CMOS technology in [16] dissipates 222 μW from a 1.0-V supply and occupies 30,800 μm^2 while operating at a relatively low frequency of 15 MHz. Our digital receiver achieves an improved BER performance by using a BPPM modulation scheme and reduces the transmitter power dissipation by using a parallel energy collection scheme during detection and synchronization. The presented non-coherent IR-UWB transmitter and receiver architectures were utilized in the design and implementation of our secure code-shifted reference

UWB transceiver in [17].

TABLE I: Implementation Characteristics of the IR-UWB Receiver on a Spartan-6 FPGA

Module	Slice LUTs	Regs.	DSP48s	Freq. (MHz)
Transmitter	171 (2%)	105 (0.9%)	0 (0%)	173
Detector	224 (3%)	229 (2%)	2 (12%)	142
Synchronizer	160 (2%)	231 (2%)	2 (12%)	197
Integrator	67 (1%)	90 (.07%)	1 (6%)	82
Decoder	143 (2%)	102 (0.8%)	1 (6%)	257
Receiver	621 (10%)	687 (6%)	4 (16%)	82

TABLE II: Power Consumption of the Synthesized IR-UWB Transceiver in a 32-nm CMOS Process

Power (μW)	Transmitter	Receiver
Total Dynamic Power	36.1	228.6
Cell Leakage Power	59.5	436.1
Total Power	95.6	664.7

TABLE III: Silicon Area of the Synthesized IR-UWB Transceiver in a 32-nm CMOS Process

Area (μm^2)	Transmitter	Receiver
Interconnect Area	419	4000
Cell Area	2522	18031
Total Area	2941	22031

V. Conclusion

The digital implementation of a non-coherent impulse-radio ultra-wideband (IR-UWB) transmitter and receiver was demonstrated. Utilizing a non-coherent transmitter and receiver reduced

the design complexity by alleviating the need for the transmitter and receiver to be perfectly synchronized in time with compact detection and synchronization stages. The receiver uses the energy collection scheme to find and to synchronize with incoming transmissions before decoding data. A finite state machine controlled the enabling and disabling of various components preventing them from idly consuming power while not having valid input on which to perform operations. Our compact architecture proposal for the integration block allows the integration of a variable set of integration windows using only three adders. The bit error rate performance of the transmitter and receiver implemented on the field-programmable gate array (FPGA) over a range of signal-to-noise ratio values were verified against the software models of the transmitter and receiver implemented in Matlab/C in double-precision floating-point and fixed-point formats. The transmitter and receiver used about 12% of the configurable resources, and 16% of the DSP48 blocks, and run at 173 MHz and 82 MHz, respectively, on a relatively small Spartan-6 FPGA. This is equivalent to a chip area of 0.0227 mm^2 in a standard 32-nm CMOS process, as estimated from chip synthesis.

VI. Acknowledgement

This work was supported (in part) by the Center for Sensorimotor Neural Engineering (CSNE), a National Science Foundation Engineering Research Center (EEC-1028725).

References

[1] M. Win and R. Scholtz, "Impulse radio: How it works," *IEEE Commun. Lett.*, vol. 2, no. 2, pp. 36–38, 1998.

[2] M. Yuce, H. Keong, and M. Chae, "Wideband communication for implantable and wearable systems," *IEEE Trans. Microw. Theory Techn.*, vol. 57, no. 10, pp. 2597–2604, 2009.

[3] A. P. Chandrakasan *et al.*, "Low-power impulse UWB architectures and circuits," *Proc. of the IEEE*, vol. 97, no. 2, pp. 332–352, 2009.

[4] V. Sze, "An Energy Efficient Sub-Threshold Baseband Processor Architecture for Pulsed Ultra-Wideband Communications," Master's thesis, Dept. Elect. Eng. Comp. Sci., Massachusetts Inst. Technol., Cambridge, 2006.

[5] Y. D. Chen, "Reflective Impulse Radios: Principles and Design," Ph.D. dissertation, Univ. California, Berkeley, 2013.

[6] L. Stoica, "Non-coherent energy detection transceivers for ultra wideband impulse radio systems," Ph.D. dissertation, University of Oulu, 2008.

[7] M. E. Sahin, I. Guvenc, and H. Arslan, "Optimization of energy detector receivers for uwb systems," in *IEEE 61st Vehicular Technology Conf.*, vol. 2, May 2005, pp. 1386–1390.

[8] A. Alimohammad, S. Fard, B. Cockburn, and C. Schlegel, "A compact and accurate Guassian variate generator," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 5, pp. 517–527, 2008.

[9] M. Ghavami, L. Michael, and M. Kohno, *Ultra Wideband Signals and Systems in Communication Engineering*, 2nd ed. Chichester, UK: John Wiley & Sons, 2007.

[10] A. Yousif, M. Rashdan, J. Haslett, and B. Maundy, "A low power and high speed PPM design or ultra wideband communications," in *Canadian Conf. Elect. Comput. Eng.*, 2008, pp. 1055–1058.

[11] A. Hennessy, "Implementation of physical layer security of an ultra-wideband transceiver," Master's thesis, Dept. Elect. Eng., San Diego State Univ., San Diego, CA, 2016.

[12] P. Pace, *Detecting and Classifying Low Probability of Intercept Radar*, 2nd ed. Boston, Mass., USA: Artech House Books, 2008.

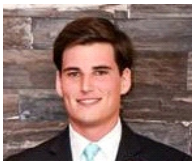
[13] S. Vitavasiri, "A non-coherent ultra-wideband receiver: Algorithms and digital implementation," Master's thesis, Dept. Elect. Eng. Comp. Sci., Massachusetts Inst. Technol., Cambridge, 2007.

[14] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York, N.Y., USA: John Wiley & Sons, 1999.

[15] A. Rabbachin and I. Oppermann, "Synchronization analysis for uwb systems with a low-complexity energy collection receiver," in *Ultra Wideband Systems, 2004. Joint with Conf. Ultrawideband Systems and Technologies. Joint UWBST IWUWBS. 2004 Int. Workshop*, May 2004, pp. 288–292.

[16] Z. Zou, F. Jonsson, L. Zheng, and H. Tenhunen, "A digital back-end of energy detection UWB impulse radio receiver," in *IEEE Norchip Conf.*, 2009, pp. 1–4.

[17] A. Hennessy and A. Alimohammad, "Design and implementation of a digital secure code-shifted reference UWB transmitter and receiver," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–10.



Andrew Hennessy is a graduate student working in the VLSI Design and Test Laboratory in the department of Electrical and Computer Engineering at San Diego State University, San Diego, California. He received the B.S. degree in computer engineering from the University of California, Riverside, in 2014. His research interests include field-programmable gate arrays, ultra-low power VLSI architectures for digital communication, and data security algorithms.

Amirhossein Alimohammad is an Associate Professor in the Electrical and Computer Engineering Department at the San Diego State University. He was the Co-Founder and Chief Technology Officer of Ukalta Engineering in Edmonton, Canada, from 2009-2011. He was a Postdoctoral Fellow at the University of Alberta between 2007-2009. He obtained a Ph.D. degree in Electrical and Computer Engineering from the University of Alberta in Canada and a M.Sc. degree from the University of Tehran in Iran. Before starting his Ph.D., he was a Hardware Engineer at Get2Chip GmbH, a Research Fellow in the Institute of Microelectronics at the University of Ulm and Atmel Wireless in Germany. His research interests include digital VLSI systems, reconfigurable architectures, wireless communication circuits, and signal processing algorithms.