# Design Space Exploration of a Cooperative MIMO Receiver for Reconfigurable Architectures

Shahnam Mirzaei
University of California, Santa Barbara
shahnam@umail.ucsb.edu

Ali Irturk, Ryan Kastner
University of California, San Diego
airturk@ucsd.edu, kastner@cs.ucsd.edu

Brad T. Weals, Richard E. Cagley
Toyon Corporation
bweals, rcagley@toyon.com

## ABSTRACT

*Cooperative MIMO is a new technique that allows disjoint wireless communication nodes (e.g. wireless sensors) to form a virtual antenna array to increase bandwidth, reliability and/or transmission distance. It differs fundamentally from other MIMO communication since the signals received from each node have a relative timing and frequency offset due to the distributed nature of their transmitting antennas. Therefore, the receiver must estimate the timing and frequency for each transmitting node, in addition to the MIMO channel. In this paper, we design and implement a receiver for the cooperative MIMO problem using reconfigurable hardware. We discuss the computation required for each stage of the receiver and perform experimental study of the tradeoffs between area, power, performance and quality of results. The end result is an efficient, parameterizable, cooperative MIMO receiver implemented on several different state-of-the-art FPGAs devices.*

## 1. Introduction

A cooperative MIMO network involves a distributed set of transmitting nodes (e.g. sensor nodes) forming a virtual array to transmit a signal to achieve longer range or lower transmit power than would be capable by an individual sensor alone [1-3]. For example, consider a number of densely deployed, low power wireless sensor nodes. Cooperative MIMO techniques can be used to allow these sensor nodes to act as a virtual antenna array to increase the capacity of the wireless channel and enhance the reliability of the transmitted data for long non line-of-site links, e.g. in order to transmit to a distant mobile collector node.

In the following, we describe the design of a cooperative MIMO receiver on a number of different FPGA devices. The Xilinx Virtex FPGAs are perfect platforms for the cooperative MIMO receiver as they provide powerful signal processing architectural features, e.g. shift register LUT (SRLs), Block RAMs (BRAMs) and digital signal processing (DSP) units that can be incorporated to significantly enhance the performance of the cooperative MIMO receiver. We discuss the design decisions that we encountered as we customized our design to utilize the FPGA architectural features. We determined that the timing and frequency offset estimation is a major component of the overall receiver design since each transmitting node in the virtual array requires separate time and frequency offset estimates. Therefore we focus much of our attention on efficiently implementing this core.

The major contributions of this paper include:

1) The design and implementation of a complete wireless receiver for cooperative MIMO applications on Xilinx Virtex FPGAs.
2) Novel design of a reconfigurable computational core for timing and frequency estimation for binary phase-shift keying (BPSK) and quadrature phase-shift keying (QPSK) modulation.
3) An efficient delay line implementation utilizing BRAMs to provide drastic area improvements with limited performance impact compared to a similar design using SRLs;

The remainder of the paper is organized as follows: The next section describes the related work. Section 3 provides an overview of the cooperative MIMO receiver and design flow . Section 4 provides the area and power consumption results when we implement the cooperative MIMO receiver on different Xilinx FPGAs. We conclude the paper in Section 5.

## 2. Related Work

Wireless communication devices traditionally use digital signal processors (DSPs) for implementing their computation. However, in recent years, field programmable gate arrays (FPGAs) have been replacing DSPs due to their ability to customize their functionality to the application at hand. FPGAs have shown orders of magnitude improvements on signal processing applications with regard to speed and power.

For example, Scrofano et al [5] compared a representative FPGA (Xilinx Virtex-II Pro) with a digital signal processor (TI TMS320C6415) and an embedded processor (Intel PXA250). Their results showed that Virtex-II Pro consumes much less energy than any of the other devices. Senoil et al. [6] studied an FFT implementation on an FPGA which design achieved 56% less energy consumption than a DSP. In terms of performance, the FPGA implementation resulted in a 10 fold improvement over an embedded processor. Swaminathan et al. [7] implemented an adaptive Viterbi algorithm on an FPGA, achieving a 29X speed-up compared to the DSP implementation. Rajagopal and Cavallaro [8] presented an implementation of the pipelined multiuser detection algorithm by integrating both FPGA and DSP, and showed that the FPGA achieves 4X speedup over the DSP implementation. Murphy et al [9] implemented a testbed as a prototyping platform for MIMO systems in wireless applications. Tessier et al [10] implemented a MIMO receiver but configured Block RAMs as FIFOs. This could be wasting memory resources on FPGAs since each memory block can be configured to only

one FIFO due to the lack of ports on embedded memory blocks.

Our work explores variations in structure of a specific application in order to optimize the FPGA implementation including logic resources as well as embedded memory. In terms of using on-chip memory resources, our work differs from the previous since it uses the novel *chained buffer technique* introduced by the authors to efficiently implement the correlation function used by timing and frequency estimation unit.

## 3. Cooperative MIMO Receiver Architecture

In this section we will present an overview of the MIMO architecture followed by the design flow that we have used to develop the receiver. Finally we present our architectural optimizations along with implementation details.

### 3.1 Overview

An MxN MIMO system consists of M transmitting and N receiving antennas. In this paper we have implemented a 2x1 system. The larger systems can be built using the same techniques described in this paper. The cooperative MIMO receiver contains a number of computational cores. FIGURE 1 displays a receiver with one antenna that receives data from two transmitting nodes.

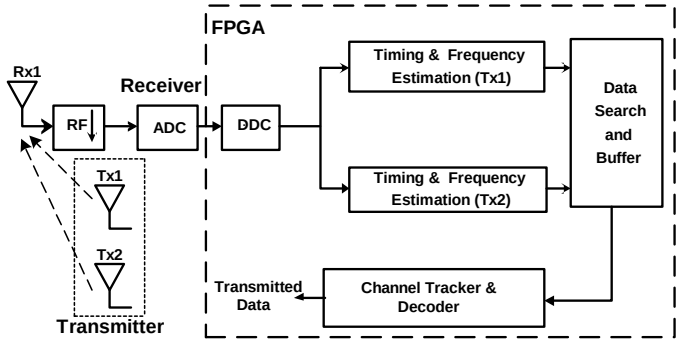


**FIGURE 1 A depiction of the significant computational cores in a 2x1 cooperative MIMO receiver. The signal from two disjoint transmitters (Tx1 and Tx2) is received by one antenna (Rx1) and downconverted to a baseband signal. Timing and frequency estimates for each of the two transmitting nodes are computed, sent to a channel tracker and decoded into the transmitted data.**

The data communication starts from the two transmitting nodes, Tx1 and Tx2. There are several different methods to modulate the transmitted data. Phase-shift keying (PSK) utilizes the phase of the signal to encode the data. Binary phase shift keying (BPSK) is the simplest PSK that uses two phases (0° and 180°) to encode '0' and '1' respectively. Quadrature phase-shift keying (QPSK) uses four phases separated by 90°, e.g. 45°, 135°, -135°, -45°, to encode two data bits. QPSK requires more sophisticated transmitter and receiver hardware, but achieves twice the data rate of BPSK. Our receiver is capable of handling either BPSK or QPSK and we study the tradeoffs between the two in later sections.

The transmitted signal centered at 1350 MHZ arrives at receiver antenna Rx1 and is down converted to a 12 MHz intermediate frequency (IF). The radio frequency (RF) down converters and analog-to-digital converters (ADC) typically reside on a separate RF processing board. The remainder of the processing is done on the FPGA.

The outputs of the ADCs are fed into digital down converters (DDCs) implemented on the FPGA. These convert the signal from its 12 MHz IF to baseband. The baseband output is 500 kilosymbols per second with an oversampling rate of 16 samples per symbol, equivalently 8 megasamples per second. The DDC architecture performs pulse shaping and noise cancellation (FIR filter) in addition to down sampling. The simple nature of the DDC leaves little room for optimization. We have selected to use Xilinx DDC core for this purpose.

This baseband signal is fed into *M* timing and offset frequency estimation cores – one for each of the transmitting nodes that form the virtual antenna array. Since the nodes are not physically co-located, they require unique synchronization and parameter estimation. These nodes do not share a common crystal for mixing the signal. As such, there will be a relative carrier frequency offset that varies from one node to the next. Furthermore, the frequency of a node can change over time due to part degradation and temperature variation. The receiver must also estimate the arrival time of each packet as well. The timing and frequency estimation block provides estimates on channel statistics to a data search and buffering block. The output of this block provides an indication of the degree to which the received signal is correlated with the training sequence (indicating timing) as well as the frequency (indicating offset). This block requires significant resources and we perform a number of architectural explorations to reduce area, increase the performance and lower the power in Section 3.3.

This data search and buffer block adjusts the incoming data according to the time and frequency estimates. The output of this block is subsequently fed to the channel tracker and decoder block. To be more precise, for each symbol, the magnitude is calculated and a search is done to find the maximum value which will be compared with the training sequence to calculate the offset.

The channel tracker and decoder block uses the current channel estimates and either known symbols (the training sequence) to calculate a channel estimation error and finally update the channel estimates for the next time period. Our design uses the variable step size least mean square (vLMS) algorithm [4] for tracking.

### 3.2 Design Flow

FIGURE 2 depicts the DSP design flow that we have used in this paper. The methodology here, automates the flow from algorithm to hardware.

AccelDSP™ Synthesis Tool is a high-level MATLAB language based tool for designing DSP functions for Xilinx FPGAs. AccelDSP generates synthesizable HDL as well as testbenches for verification. The HDL file can be fed to Xilinx ISE tools directly or used in conjunction with Simulink Xilinx Blockset library components. System Generator tool translates high level Simulink DSP blocks to RTL language that can be compiled using Xilinx Foundation ISE. Eventually an FPGA bitstream can be generated to download to FPGA using Xilinx Foundation ISE place and route tools. In this work, the data search and buffer unit has been done using Matlab algorithms and synthesized to HDL blocks using Xilinx AccelDSP

synthesis tool and other blocks have been designed with Simulink using Xilinx Blockset libraries and synthesized to HDL by Xilinx System Generator.

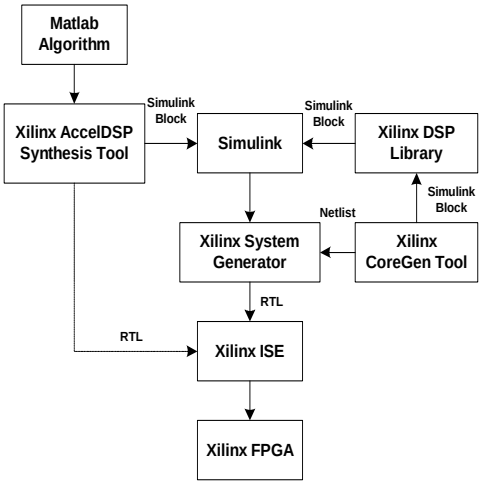


**FIGURE 2 DSP design flow for Xilinx FPGAs**

### 3.3 Time and Frequency Offset Estimation

As we mentioned previously, the time and frequency estimation block requires significant number of resources. In this subsection, we explore a number of architectural optimizations to reduce the resource consumption of this block.

The time and frequency offset estimation block is responsible for estimating the start time and offset frequency of the incoming data from each transmitting node. Since the transmitting nodes in the virtual array are physically separated, and therefore use different onboard crystals for carrier frequency mixing, the data from each node can have significantly different frequency values. Hence the offset frequency of the nodes must be estimated at the cooperative MIMO receiver. Furthermore, the medium access control (MAC) of the individual nodes is not synchronized, which will likely result in a difference in the time when the signals reach the receiver. Therefore, the receiver must also estimate the start of the packet for each of the transmitting nodes in the virtual array.

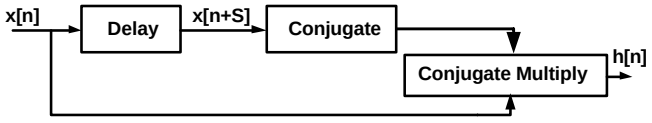


**FIGURE 3 Homodyne block diagram: The incoming signal is delayed by S samples, where S = # samples/symbol, conjugated and multiplied with the underplayed data samples.**

There are several techniques for estimating the time and frequency offset, e.g. the generalized successive interference canceling (GSIC) [1]. Most techniques are quite sophisticated and computationally intensive since they require an FFT to estimate the frequency and timing and consequently they are expensive for FPGA implementation. For instance, the design of FIGURE 1 requires a 1024 point FFT, which needs a minimum of 10282 FFs, 7266 Slices and 10288 LUTs excluding extra control logic. This exceeds the resource utilization of the receiver that we designed using our circular buffer technique (described later) by order of magnitude. The difference is substantial if a MIMO system consisting of multiple channels is implemented. In this work, we strive for a more feasible technique in terms of hardware implementation that centers on a homodyne and correlation. The drawback is losing the accuracy of the calculations but it provides sufficient accuracy for lower bandwidth.

The homodyne, which performs frequency offset estimation, is depicted in FIGURE 3. The homodyne consists of a delay unit and a complex conjugate multiplier. The incoming complex samples $x[n]$ delayed by one symbol $x[n+S]$ (in our case there are 16 samples/symbol, i.e. $S = 16$), are conjugated and then multiplied, resulting in $h[n] = x[n] \times x[n+S]^*$, where $^*$ denotes conjugation. Assuming that there is a constant frequency and phase offset in each packet, the conjugate multiply provides a constant phase offset for all the incoming symbols which is proportional to the frequency offset that we are trying to estimate. The simplistic structure of the homodyne leaves little room for optimization, and we now turn our attention to timing estimation.

A correlator provides the time estimate. It takes values from the input data stream and matches them with the values of the known training sequence. An adder tree provides a correlation value of the current data with the training sequence. In general, correlation requires a multiplication of the known value with the input sample.

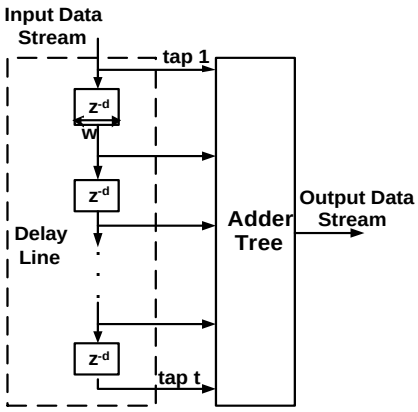


**FIGURE 4 Depiction of the timing estimation core using a delay line and correlation.**

There are three correlator parameters that can be varied as shown in FIGURE 4 - the number of taps *t*, the number of samples in a delay block *d*, and the width of the complex data *w*. These parameters depend on the application. In general increasing the number of taps will increase the accuracy of the timing estimate; we will describe the precise relationship briefly. We need to decide what values to assign to the parameters: *t*, *d* and *w*.

The number of samples in the delay block is determined by the oversampling rate, which is 16 is our case. Therefore, we set $d = 16$. The data width largely depends on the analog to digital converters (ADCs). These converters are usually in the range of 8-14 bits for each in-phase (I) and quadrature (Q) component. Our cooperative MIMO design uses 16 delay samples, and 24 bit complex data - 12 bits for each I and Q, i.e. $w = 24$.

The number of taps determines the quality of correlation; increasing the taps results in better estimates. With an infinite number of taps (infinite SNR), we could estimate the time

offset to within +/- ½ a sample period. FIGURE 5 displays the root mean square (RMS) error for the time estimate as the number of taps increase. The chart shows that increasing the number of taps from 20 to 120 reduces the BPSK RMS error from 0.7 to around 0.3. However, increasing the number of taps past 120 provides diminishing gains. A similar trend occurs for the QPSK error at around 160 taps.

The frequency SNR varies linearly with the number of taps. Assume that $\boldsymbol{r} = \boldsymbol{s} + \boldsymbol{n}$, where $s$ is the desired signal vector, and n is white Gaussian noise with variance $\sigma^2$. A correlator matched to **s** has the scalar output:

$$u = \boldsymbol{s}^t \boldsymbol{r} = \boldsymbol{s}^t \boldsymbol{s} + \boldsymbol{s}^t \boldsymbol{n}$$

$$E\{u\} = \boldsymbol{s}^t \boldsymbol{s} = E_s = P_{av} N,$$

where $P_{av}$ is the power of the samples of $\boldsymbol{s} = [s_1,...s_N]^t$, and $N$ is the length of the signal vector, or number of taps on the delay line. We know that $Var\{u\} = \sigma^2 E_s$. The SNR is defined as $E\{u\}^2/Var\{u\}$, which in this case is:

$$SNR = E_s^2/(\sigma^2 E_s) = P_{av} N/\sigma^2$$

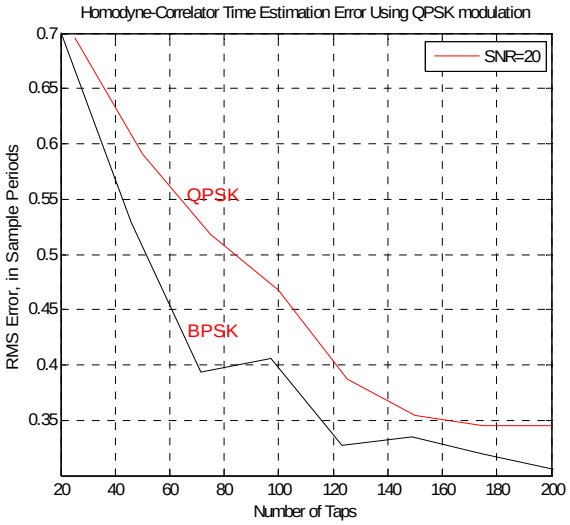


**FIGURE 5 Root mean square (RMS) error of the time estimation versus the number of taps used for correlation for BPSK and QPSK data with 20 dB signal-to-noise ratio (SNR).**

Therefore, for fixed average signal power $\sigma^2$, the SNR of the offset frequency increases linearly with $N$ (the number of taps).

For our experiments, we use 64 taps. This number lets us approach an accuracy that is close to the resolution we can get with the oversampling rate we have chosen. In practice, with 64 taps and a nominal input SNR of about 10 dB, we have found that we are able to get to within +/- 1 sample period.

### 3.3.1 Correlation Using Shift Registers

The modern reconfigurable architectures can implement delay lines using an architectural feature called shift register LUT (SRL). The Xilinx Virtex-4 architecture uses 16 bit SRL (SRL16), while the Virtex-5 has 32 bit SRL (SRL32).

SRL can implement fixed, static or dynamic delay. The shift register LUT contents are initialized by assigning multiple digit binary number to the LUT inputs. These inputs can be used as address lines for the shift register to change the shift amount. There is a separate input to the LUT that is used as the input of the shift register. For instance, if we configure the Virtex-4 LUT in static mode for 16 bit delay by assigning 1111 to the inputs of the LUTs, it will shift the input value for 16 clocks. Consequently, for a 24 bit input, 24 LUTs are equivalent to one delay block (hence implementing $z^{-16}$). This causes significant saving in FPGA area because LUTs can be configured as 16 bit shift registers in the slices of Virtex 4 FPGAs. It is important to note that this configuration does not use any of flip flops in the slice.

FIGURE 6 charts the resource utilization of the delay line in Virtex-4 FPGA as we vary the number of taps $t$, the samples/block $d$, and the data width $w$. These three values are explained in the previous section (see FIGURE 4). As expected, the resource usage increases as you increase each

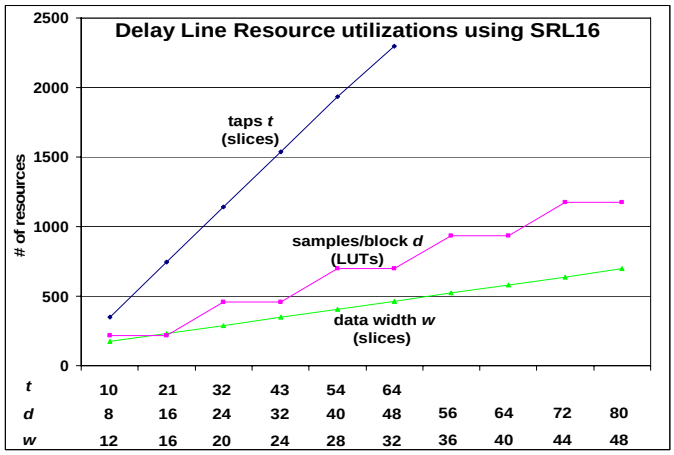


**FIGURE 6 Resource utilizations of the delay line using SRL16. The Graph displays the effects of varying three parameters: the # of taps *t*, the samples/block *d*, and data width *w*.**

of the parameters. The data width and number of taps increase in a linear fashion. As you increase the samples/block, the LUT usage moves in a step fashion at every 16 samples. This is due to the use of the SRL16. A single delay element with 1-16 requires 24 LUTs as described previously. Once we increase to a delay of 17-32, will need 48 LUTs since we now need 2 SRL16s per bit of the delay element.

### 3.3.2 Correlation Using *Circular Buffer Technique*

The resource utilization using SRL is quite large, therefore we should explore other options to reduce the area of the correlator. In this subsection, we look towards using on chip memory to store the delay line data.

Modern FPGAs provide plenty of on-chip block RAMs (BRAMs) which is extremely useful for memory intensive applications such as our time and frequency estimation core. We can implement the delay lines through careful utilization of the BRAMs.

Compared to the SRL, BRAMs provide more compact memory storage at the expense of having a limited access interface to the data through two memory ports. Each port has a parameterizable data width and frequency. The write operations are consecutive, and we can design address generator logic to increment the address; this write port is clocked at the same rate of the incoming data. However, the read operations must be done faster. The rate of the read operations depends on the number of taps and the number of

BRAMs that we use. Assume we have 1 BRAM and 64 taps. Therefore, every time we do one write, we must do 64 reads from the BRAM to get the 64 tap values. Now if we increase the number of BRAMs, say to 4, we can do 4 reads in one cycle, meaning that we need 64/4 = 16 reads for every write operations. This scheme is possible in FPGAs since the BRAM has separate ports that can be clocked at different rates using DCM (Digital Clock Manager) units.

The number of Block RAMs that we need is a function of the size of the delay line. The delay size is $O(t \times d \times w)$. We simply divide the size of the delay line by the capacity of a BRAM (18 Kb for Virtex 4) to determine the minimum number of required BRAMs. The required read rate is limited by the maximum operating speed of the Block RAMs. In other words, read operations can not be faster than access time of the on-chip memory.

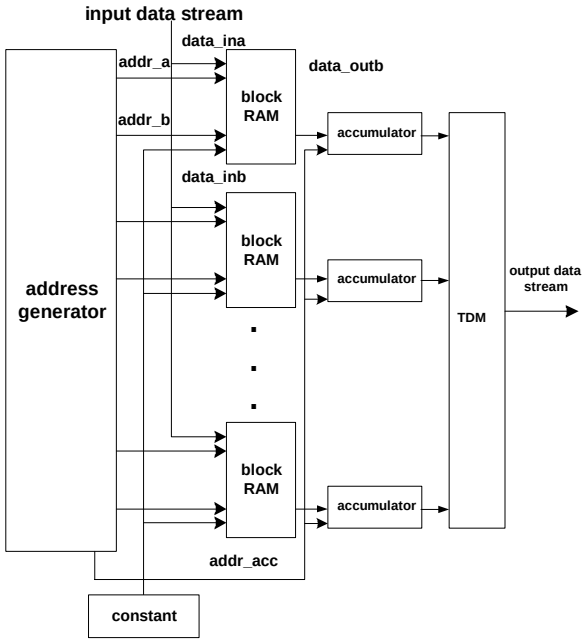


**FIGURE 7 Time estimation core using the circular buffer technique.**

We can cascade multiple block RAMs and use them as a large circular buffer. On the read side, we have to make sure that we add or subtract correctly. The sequence of training bits in the single buffer determines the order of adds/subtracts. This is because the accumulators are associated with each BRAM. Depending on the current location of the "start" of the circular buffer; the start changes by one entry each time a new input sample is received. At some point, for example, BRAM 0 will use training entries *t-1*, *t*, *0, 1, 2,….* At another time it will be another sequence of entries. In circular buffer technique we don't need to chain the BRAMs together; however we do need to connect the data_in to every BRAM. As we increase the number of BRAMs, this can cause significant routing overhead. On the other hand the circular buffer technique requires that the correlator understands the current starting location of the data in the delay line.

FIGURE 7 shows the block diagram of the circular buffer technique. In this technique data is written to the accumulators at the same rate as the read operation and a Time Division Multiplexer (TDM) is placed at the output of the accumulators to pick the data in round-robin manner.

In circular buffer technique, only a subset of the total correlation coefficient set need be applied to the data in each block RAM at any one time. In this experiment, each BRAM is assigned 8 of the 64 coefficients. Also, the 8 coefficients change with time. Thus we have to keep track of which coefficients are being used by each BRAM at each time. Therefore we store 8 copies of the same full set of coefficients in ROM and in each cycle we access these ROMs to read the corresponding coefficient.

## 4. Experiment Results

FIGURE 8 shows area and power consumption for the various blocks of the cooperative 2x1 MIMO receiver respectively. These results were obtained through synthesis flow described in Section 3.2. We have targeted three FPGA architectures: Spartan 3, Virtex 4 and Virtex 5. The goal is to come up with the best platform for receiver implementation in terms of area and power consumption.

In FIGURE 8a, the time and frequency estimator represents a large portion of the design and is ripe for optimization; therefore we focused our optimizations here, which were described in Section 3.3. The SRL architecture consumes a large number of LUTs and slices. This is mainly due to the long delay line in correlator function (see Section 3.3.1). Our novel circular buffer implementation leverages BRAM resources for the delay line (see Section 3.3.2) implementation. Our method shows up to 65% savings in slice usage at 8% drop in clock speed compared to the SRL implementation for this block.

Another observation in FIGURE 8a is larger number of SLICEs in Virtex 5 compared to other devices even though this architecture offers more inputs per LUTs. For instance, consider number of SLICEs for Virtex 5 in FIGURE 8a under SRL technique (26998) as opposed to the similar column for Virtex 4 (20027). This is because of the change in structure of the CLBs on FPGA fabric. In Virtex 5 most of the SLICEs do not offer memory option used for SRL while in other two architectures half of the SLICEs do. FIGURE 8b represents lower dynamic power consumption for Virtex 5 platform since it has lower core voltage and smaller geometry compared to other two architectures.

We also wanted to see how modulation affects our design. We applied our *circular buffer* technique to the time and frequency estimator block of the cooperative MIMO receiver shown in Table 1. For simplicity, we eliminated extra logic in two channel homodyne-correlator since our optimization technique focuses only on correlation function. This included the homodyne block, control logic, complex to real/imaginary converter function and vice versa, input and output logic. Table 2 shows the implementation result after the simplification. This is shown in the first row of Table 2. We simplified the design more by eliminating one of the channels (1x1 cooperative MIMO); the results are shown in the second row. Table 2 also shows QPSK modulation scheme results.

In BPSK modulation, the incoming bits are encoded with a -1 or 1 to represent 0 and 1, respectively while QPSK encodes two separate bits. The first is encoded with a -1 or 1, just like BPSK, and the second is encoded with –j or j,

and these two codes are summed. Thus the set of available symbols is {1, j}, {1, -j}, {-1, j} and {-1, -j}. The multiplications with these constants in QPSK scheme

introduce extra adders/subtractors to the BPSK hardware. These adders and subtractors are inserted between BRAMs and accumulators. The one channel QPSK correlation is only slightly worse compared to one channel BPSK.

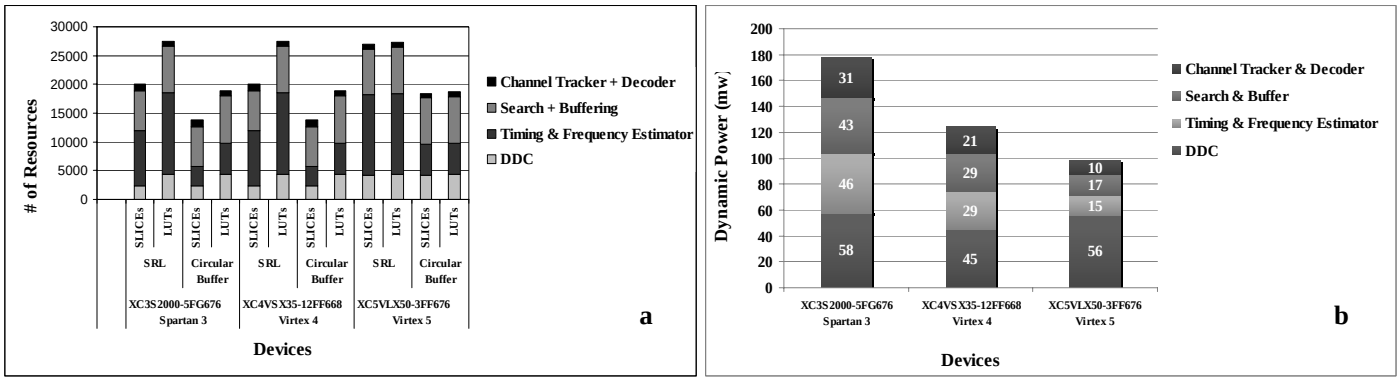


**FIGURE 8 a) Resource utilization of the cooperative MIMO receiver for three FPGA devices by two techniques**

**b) Total dynamic power consumption of the cooperative MIMO receiver for three FPGA devices**

The training sequence of our correlation core can be reconfigured on the fly. The time for reconfiguration depends on the number of accumulators in our design. In BPSK, we have 8 Block RAMs that drive 8 accumulators. There is 8x1 ROM per each accumulator. For QPSK, the ROM size is 8x2 since we have to store two bits per accumulator. Each frame takes 8 cycles for reconfiguration using a 128 MHz clock gives a reconfiguration time 8/(128 x $10^6$) = 62.5 ns for both BPSK and QPSK.

**Table 2**: **Correlation implementation results on Virtex4SX FPGA**

| Design Technique | FF | LUT | BRAM | SLICE | Delay (ns) |
|---|---|---|---|---|---|
| Two Channel BPSK | 3730 | 3177 | 14 | 2695 | 9.59 |
| One Channel BPSK | 2858 | 2164 | 14 | 1930 | 8.78 |
| One Channel QPSK | 3098 | 2420 | 14 | 2074 | 9.68 |

## 5. Conclusion

In this paper we designed and implemented a cooperative MIMO receiver for reconfigurable architectures. We discussed the architecture of the overall system, and described a technique to optimize the time and offset frequency estimation block, as it consumed a large number of the overall resources. We developed a circular buffer technique to implement a correlation functionis based on using BRAMs to implement long delay lines and optimize the area on FPGA. Our technique provides significant area savings with limited increase in delay compared to an SRL implementation. We described how to extend the time and frequency estimation core to handle BPSK and QPSK modulation formats. Our results show that the QPSK implementation is only slightly larger than an equivalent BPSK implementation. Our final receiver implementation uses memory resources efficiently and is prameterizable.

## 6. References


[1] R. A. Iltis, S. Mirzaei, R. Kastner, R. E. Cagley, and B. T. Weals, "Carrier Offset and Channel Estimation for Cooperative MIMO Sensor Networks," *IEEE Global Telecommunications Conference (GLOBECOM)*, 2006.

[2] J. N. Laneman and G. W. Wornell, "Distributed space-time-coded protocols for exploiting cooperative diversity in wireless networks," *IEEE Transactions on Information Theory*, vol. 49, pp. 2415-25, 2003.

[3] C. Shuguang, A. J. Goldsmith, and A. Bahai, "Energy-efficiency of MIMO and cooperative MIMO techniques in sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 1089-98, 2004.

[4] T. Aboulnasr and K. Mayyas, "A robust variable step-size LMS-type algorithm: analysis and simulations," *IEEE Transactions on Signal Processing*, vol. 45, pp. 631-9, 1997.

[5] R. Scrofano, C. Seonil, and V. K. Prasanna, "Energy efficiency of FPGAs and programmable processors for matrix multiplication," *IEEE International Conference on Field-Programmable Technology (FPT).* pp. 422-5, 2002.

[6] C. Seonil, G. Govindu, J. Ju-Wook, and V. K. Prasanna, "Energy-efficient and parameterized designs for fast Fourier transform on FPGAs," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. II-521-4 vol, 2003.

[7] S. Swaminathan, R. Tessier, D. Goeckel, and W. Burleson, "A dynamically reconfigurable adaptive Viterbi decoder," *ACM International Symposium on Field-Programmable Gate Arrays*, pp. 227-36, 2002.

[8] S. Rajagopal, S. Bhashyam, and J. R. Cavallaro, "Real-time algorithms and architectures for multiuser channel estimation and dectection in wireless base-Station receivers," *IEEE Tran. on Wireless Comm.*, pp. 374-377, July 2002.

[9] P. Murphy, F. Lou, A. Sabharwal, and J. P. Frantz, "An FPGA based rapid prototyping platform for MIMO systems," *Asilomar Conference on Signals, Systems and Computers*, pp. 900-4 Vol, 2003.

[10] R. Tessier, V. Betz, D. Neto, and T. Gopalsamy, "Power-aware RAM mapping for FPGA embedded memory blocks," *ACM International Symposium on Field-Programmable Gate Arrays*, 2006.

[11] Y. Meng, T. Sherwood, and R. Kastner, "Leakage power reduction of embedded memories on FPGAs through location assignment," *Design Automation Conference*, pp. 612-17, 2006.