EMBEDDED RECONFIGURABLE SOLUTION FOR OFDM DETECTION OVER FAST FADING RADIO CHANNELS

Mihai SIMA and Michael McGUIRE

University of Victoria – Department of Electrical and Computer Engineering
P.O. Box 3055 Stn CSC, Victoria, BC V8W 3P6, CANADA
Email: {msima,mmcguire}@ece.uvic.ca

ABSTRACT

OFDM demodulation under fast fading radio channels is very computationally demanding, making the implementation of Software Defined Radio (SDR) solutions problematic. A sub-optimal demodulation algorithm based on *QR* decomposition of blocks of the channel transfer matrix offers near optimal performance at lower computational cost, but hardware support is still needed. We first propose a COordinate Rotation DIgital Computer (CORDIC) rotator in reconfigurable hardware to expose and then exploit at software level the intra-block paralellism of the *QR* decomposition. In particular, we show that although the rotator is deeply pipelined, the scale factor inherent to CORDIC algorithm can still be distributedly compensated throughout the pipeline at no additional cycle time penalty. Then, for a Nios II processor augmented with a Reconfigurable Functional Unit (RFU) that incorporates the proposed CORDIC rotator, we also propose a computing scenario that keeps all the data to be processed inside the RFU, to minimize overhead of the data trafic between the Register File and the CORDIC rotator. Overall, we show that OFDM demodulation under fast-fading can be performed in fixed-point arithmetic and in real-time on a Nios II reconfigurable embedded system, proving that an SDR solution for OFDM demodulation under fast fading is possible.

Index Terms**—** Software-defined radio, OFDM, fast fading, *QR* decomposition, CORDIC.

1. INTRODUCTION

The goal of an Orthogonal Frequency-Division Multiplexing (OFDM), or any digital communications, receiver is to detect with the lowest possible Bit Error Rate (BER) the transmitted symbols based on the received signal. For a stationary radio channel, the channel state matrix, $\mathbf{H}$, is diagonal, and the optimal detection process is a division operation followed by a hard decision for each OFDM subcarrier. If the radio channel is subject to fast fading, then the channel state matrix, $\mathbf{H}$, is a general matrix resulting in interference between different parts of the transmitted signal [1]. In this case, more complex channel state matrix estimation and symbol detection algorithms are required. In this paper, we assume that the channel state matrix, $\mathbf{H}$, is known perfectly at the receiver. Several techniques are available for channel estimation based on the transmission of pilot symbols [1, 2]. For the fast fading channels of greatest interest, the computational burden for channel estimation is much less than that of symbol detection, which we address in the paper.

To mitigate the interference effects, symbol detection algorithms for OFDM signals in fast fading require decompositions of the channel transfer matrix [1]. Since standard OFDM frames include at least 128 symbols, the order of this matrix is greater than 128, and therefore the cost of these decompositions is prohibitive. To reduce the computational burden required for symbol detection, suboptimal detection schemes with BERs close to the optimal algorithm with computational costs of $\mathscr{O}\left(N^2\right)$ have been proposed [3]. These algorithms are inherently sequential so they have unavoidable latencies. This limitation makes the implementation of Software Defined Radio (SDR) [4] solutions problematic.

To increase the computing power, a full-custom circuit can be designed, but this approach is expensive, requires a large time-to-market, and is prone to rapid obsolence due to its lack of flexibility. To achieve hardware-like performance with software-like programmability, an increasing number of embedded systems are being built using Field-Programmable Gate Arrays (FPGA), also known as reconfigurable hardware. In a common scenario, a soft-core processor along with programmer-defined functional units are deployed onto FPGA. The basic idea is to exploit both the processor flexibility to implement basic tasks (e.g., memory management and interfacing, load and store operations), and FPGA capability to implement application-specific computations. For our analysis, we use Stratix II FPGA [5] onto which a Nios II processor [6] is deployed. The resulting reconfigurable computing engine is software programmable, and thus it can provide SDR solutions to wireless and mobile communication tasks. However, since application-specific functional units are to be deployed onto FPGA, a digital design phase has still to be carried out.

Recently, we proposed a detection algorithm with the computational cost of $\mathscr{O}\left(N^2\right)$, that performs block *QR* decomposition instead of a full *QR* decomposition of the channel transfer matrix [2]. Since each block can be decomposed independently, our strategy exhibits *inter-block parallelism*, enabling the use of Single-Instruction Multiple Data (SIMD) processors [7]. In this paper, we propose a pipelined COordinate Rotation DIgital Computer (CORDIC) rotator in reconfigurable hardware to expose and then exploit the *intra-block parallelism* of the *QR* decomposition. We then present a detection scenario that makes use of both inter-block parallelism and intra-block parallelism, and show that, for the OFDM frame lengths of greatest interest (256 to 1024), the OFDM detection under fast fading can be performed in real time on FPGAs using strictly fixed-point arithmetic. FPGA logic capacity requirements, as well as BER versus word-length curves are also disclosed. To summarize, the paper contributions are as follows.

- Exposing the intra-block parallelism of the demodulation process to the programmer by assisting block-wise *QR* decomposition in reconfigurable hardware with a pipelined CORDIC unit.
- Proposing a deeply pipelined CORDIC-based Givens rotator with unity scale factor that fits into the Nios II processor's pipeline.
- Proposing a Reconfigurable Functional Unit (RFU) that incorporates the proposed CORDIC rotator, along with its associated RFU instructions that augment the instruction set architecture of the Nios II processor.
- Proposing a computing scenario that keeps all the data to be processed within the RFU, in order to minimize the data traffic between the register file and the CORDIC rotator.
- Showing that the synergism of inter-block parallelism, intra-block parallelism, and reconfigurable hardware capability to implement application-specific computations makes SDR solutions for OFDM detection tasks under fast fading feasible on embedded platforms.

Overall, we prove that the resulting workload for a typical OFDM detection task under fast-fading can be performed on Stratix II FPGAs. The general techniques presented in this paper should also be useful with other FPGA families. Given the fact that low-price FPGA devices are widely available on the market, the approach we propose is a cost-effective SDR solution for OFDM detection under fast fading.

The paper is organized as follows. In Section 2, we recall the CORDIC algorithm. In Section 3 we review OFDM detection under fast fading. In Section 4 we present the new CORDIC rotator. In Section 5 we describe a reconfigurable functional unit that incorporates the proposed CORDIC rotator, along with the computation scenario. In Section 6 we provide figures for the number of required operations to perform OFDM detection under fast fading. Section 7 completes the paper with some conclusions and closing remarks.

2. CORDIC ALGORITHM BACKGROUND

A Givens rotation [8] can be described by a 2-by-2 orthogonal matrix $\mathbf{G}(\theta)$. The multiplication by $\mathbf{G}(\theta)$ of a vector $[x,y]^T$ amounts to a plane rotation of an angle θ. Historically, the Givens rotation has been used in *QR* factorization, since it can annihilate matrix elements selectively. Clearly, the second entry of the vector $[x,y]^T$ is forced to zero by properly choosing the angle θ [8].

Software implementation of Givens rotation is computationally demanding. Given an arbitrary vector $[x,y]^T$, the evaluation of the rotation matrix, $\mathbf{G}(\theta)$, requires two square operations, one addition, one square root operation, and two divisions [8]. When an arbitrary angle θ is given, a large memory storing the cosine and sine tables can be used to evaluate $\mathbf{G}(\theta)$. Alternatively, a sequence of multiplications, additions, and memory look-up operations can be used to evaluate Taylor series expansions for cosine and sine. While a desktop computer has sufficient hardware resourses to support these operations, implementing the Givens rotation on an embedded platform can be difficult.

COordinate Rotation DIgital Computer (CORDIC) algorithm has been proposed to implement Givens rotations [9, 10]. CORDIC is an iterative method for performing vector rotations by arbitrary angles using only shifts and additions, as shown in Equation 1.

$$\begin{cases} x(j+1) = x(j) - \sigma(j)2^{-j}y(j) \\ y(j+1) = y(j) + \sigma(j)2^{-j}x(j) \\ z(j+1) = z(j) - \sigma(j)\arctan\left(2^{-j}\right) \end{cases} \tag{1}$$

For rotation mode $\sigma(j) = +1$ if $z(j) \geq 0$, otherwise is -1. For vectoring mode, $\sigma(j) = -1$ if $y(j) \geq 0$, otherwise is $+1$.

The CORDIC algorithm is operated in one of two modes: rotation or vectoring. In **rotation mode**, the angle accumulator is initialized with the desired rotation angle. The rotation decision at each iteration is made to decrease the magnitude of the residual angle. In **vectoring mode**, the CORDIC unit rotates the input vector to align the result vector with the x axis, such that y approaches 0.

The CORDIC algorithm scales up the operands by a factor of about 1.64676025812 [9, 10]. Compensating for this scale factor has to be performed in order to maintain the initial vector magnitude. Finally, it is worth mentioning that in order to prevent round-off errors from contaminating the final result, at least $\log_2 n$ additional low-order bits are necessary in CORDIC for intermediate values [10].

3. OFDM DETECTION UNDER FAST FADING

The received signal vector, $\mathbf{y}$, on the N subchannels of a single OFDM symbol under fast fading radio channels can be described as $\mathbf{y} = \mathbf{Hx} + \mathbf{v}$, where $\mathbf{x}$ is the transmitted signal vector, and $\mathbf{v}$ is a noise vector. The matrix $\mathbf{H}$ is determined by the state of the radio channel during the current OFDM symbol [1]. For the remainder of this paper, it will be assumed that the channel state matrix $\mathbf{H}$ is known perfectly at the receiver. Several techniques are available for channel estimation based on the transmission of known pilot symbols and filtering algorithms [1, 2].

The symbol detection task selects the vector $\mathbf{x}$ which is the most likely to have produced the observed vector $\mathbf{y}$ given the (estimated) $\mathbf{H}$ matrix. A useful approach to symbol detection is to calculate the QR decomposition of the matrix $\mathbf{H} = \mathbf{QR}$, where $\mathbf{Q}$ is a unitary matrix and $\mathbf{R}$ is an upper-triangular matrix. Detection is then performed on $\mathbf{y}' = \mathbf{Q}^H\mathbf{y} = \mathbf{Rx} + \mathbf{Q}^H\mathbf{v}$, for which a sequential least squares detection method is easily formulated. Unfortunately, a QR decompositon on the full-size matrix $\mathbf{H}$ is very computationally expensive.

Recently, we proposed a detection algorithm that performs block QR decomposition instead of a full QR decomposition of the matrix $\mathbf{H}$, followed by an iterative technique to estimate and remove the interference between the subsets [2]. The basis of this algorithm is that, for moderate fading, the entries of the channel transfer matrix, $\mathbf{H}$, with the highest absolute values are concentrated on a band around the main diagonal [3], as shown in Figure 1. This means that M $(d \times d)$ matrices placed along the main diagonal of $\mathbf{H}$ will contain most of the significant values of $\mathbf{H}$. The non-zero entries of $\mathbf{H}$ outside of these sub-matrices are cancelled by an iterative interference cancellation technique as described next.

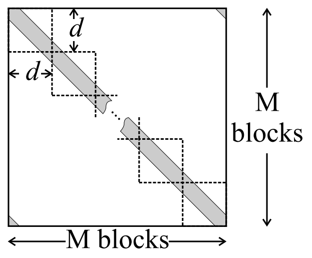

Fig. 1. Channel State Matrix Decomposition (gray regions contain values with largest magnitudes) – adapted from [2, 3].

Each block $m = 1, \ldots, M$, $\mathbf{Q}_m^H$, calculated during block QR decomposition is multiplied by the corresponding q-element subvector $\mathbf{y}_m$, to give an estimated $\hat{\mathbf{x}}_m$ through a standard least squares detection. Then, a matrix $\bar{\mathbf{H}}$ containing only $\mathbf{H}$ elements outside the $d \times d$ blocks is multiplied by the whole N-element vector $\hat{\mathbf{x}}$ to give an updated interference vector, $\bar{\mathbf{y}}(i) = \bar{\mathbf{H}}\hat{\mathbf{x}}$, where i is the iteration index. Then, a new iteration is performed on the difference measurement, $\mathbf{y} - \bar{\mathbf{y}}(i)$.

For the same conditions assumed in previous work [3] (QPSK modulation, signal-to-noise ratio $E_b/N_0 = 30\,\text{dB}$, four propagation paths, window length $M = 11$ carriers, and a normalized Doppler frequency of $f_dT = 0.05$), to obtain a comparable BER of $2.5 \cdot 10^{-4}$ we need a word-length of 19 bits with 12 fractional bits for fixed-point implementation [2]. An additional 5 bits are necessary only for CORDIC internal values [10]. The mentioned BER was obtained with two decoding iterations following the block QR decomposition. As we show in Section 6, the comparable BER value is achieved with much lower computing latency.

4. CORDIC IMPLEMENTATION ON FPGA

Field-Programmable Gate Arrays (FPGA) mainly consist of Look-Up Table (LUT) based programmable logic blocks and reconfigurable interconnections build with nMOS-tree multiplexers. Compared to Application-Specific Integrated Circuit (ASIC) designs, the speed of FPGA designs is slower due to the significant extra delay introduced by interconnections [11]. Thus, the latency of an FPGA circuit is determined by two factors: the delay in LUTs and the delay in the interconnection paths. A good understanding of the FPGA's architecture, the synthesis tool, and the routing and mapping software is essential in obtaining satisfactory system speed.

To improve computation, most FPGA architectures provide dedicated resources for specific operations. For example, addition is supported by carry chains that use a faster dedicated routing rather than the general routing. This means that addition is typically faster than LUT-based Boolean function evaluation. State-of-the-art FPGAs, e.g., Stratix II [5], also include a number of small-capacity Random Access Memories (RAM) to support data-intensive applications, embedded multipliers to provide hardware support for filter implementation, and additional logic to support conditional operations such as $(X > Y)\ ?\ Z : W$ at no additional delay penalty.

CORDIC algorithm contains only additions and shift operations. While addition is well supported in FPGA by means of dedicated carry chains, variable shift operations are difficult to build due to the high cost of multiplexing logic [12]. For this reason, the CORDIC recursion is completely unrolled. Since the shift operations are carried out over a known number of positions, they can be hardwired, and thus have the delay only because of general routing.

A feature of the CORDIC algorithm is an increase of the magnitude of the vector by a factor of 1.64676025812 [9, 10]. Compensating the scale factor by a final plain multiplication by its inverse, 0.607252935, increases the latency and reduces the throughput of the CORDIC rotator.

To simplify the scaling operation, several schemes that expand the inverse of the scale factor into a product of elementary factors have been proposed. These schemes belong to either of two approaches: (i) repeating some of the CORDIC iterations to make the scale factor a power of the machine radix [13], so that it can be cleared by a simple shift, and (ii) accomodating addititional scaling iterations to compensate the scale factor [14], or a combination of both [15–17]. For the suboptimal OFDM decoding where small matrices are to be processed, these extra iterations should be avoided if the CORDIC pipeline were to be kept full of data, thus running at full speed. To remove the extra iterations, CORDIC recurrence has been modified to merge single iterations of CORDIC and scaling into a single iteration [18]. However, this merging scheme requires long shift operations over 10 to 16 bits, and this does not make it attractive for mapping on FPGA, where going through long interconnections should be avoided if the propagation time is to be kept low. Even if iterations of the other proposed CORDIC schemes were to be merged to accomodate scale factor compensation, long shift operations would still be needed.

In our approach, we still merge rotating and scaling operations, but in a way that minimizes the delay over the interconnection network. We propose to expand the inverse of the scale factor into elementary factors of the form $1+\sigma_k(k)2^{-k}$, where k is the shift length of a scaling operation, and $\sigma_k(k) = \pm 1$. Then, the rotating and scaling operations are merged according to Equation 2.

$$\begin{aligned} x(j+1) &= (1+\sigma_k(j)2^{-k})x(j)+ \\ &\quad +\sigma(j)2^{-j}(1+\sigma_k(j)2^{-k})y(j) \\ y(j+1) &= (1+\sigma_k(j)2^{-k})y(j)- \\ &\quad -\sigma(j)2^{-j}(1+\sigma_k(j)2^{-k})x(j) \end{aligned} \tag{2}$$

In order to keep the delay over interconnections as low as possible, shift operations over a small number of bits are preferred. Thus, small values for both k and $k+j$ are sought. The expansion of the scale factor inverse presented in Equation 3 provides a precision of 20 bits with only 11 merged rotation-scaling iterations, and with a shift over a maximum length of 12 bits per merged iteration. To the best of our knowledge, this is superior to what has been proposed in the literature.

$$\begin{aligned} &(1-2^{-2})\ (1-2^{-3})\ (1-2^{-4})\ (1-2^{-5})^2 \\ &\quad (1+2^{-6})^2\ (1+2^{-7})^3\ (1-2^{-8}) \approx 0.607253779 \end{aligned} \tag{3}$$

Once the values of k's have been determined, lower values for $k+j$ can be achieved by merging short-shift rotating iterations with long-shift scaling iterations. For example, the 0th rotating iteration is merged with the largest-shift scaling iteration, the 1st rotating iteration with the next largest-shift scaling iteration, and so on.

A merged rotating-scaling iteration requires three adders to implement the four argument addition. In Section 3, we have determined that CORDIC-based rotation for OFDM under fast fading requires a word-length of 24 bits. Since two out of three additions of a merged iteration can be carried out in parallel, the latency of four 24-bit argument addition is the sum of one carry propagation time over 24 bits, the propagation time across a LUT, clock-to-output time, and setup time. As mentioned, conditional operation based on the sign of component y presented in Equation 1 is supported by Stratix II with no additional delay [5]. Quartus II (the Altera's FPGA mapping tool) reports that the latency of the longest-shift merged iteration is 3.5 ns with a variation of ± 10% depending on various synthesis options. Given that the Nios II processor can run up to 185 MHz [6], there is still a positive slack of more than 1.5 ns that can be used to accomodate the other sequential delays, such as setup or hold time. These general design techniques should also be useful with other FPGA families.

To conclude, the resulting CORDIC rotator has a unity scale factor, and reliably fits into the Nios II pipeline. This means that the throughput can be as high as one Givens rotation per cycle. In the next section, we present the details of a complete reconfigurable functional unit that integrates the CORDIC rotator along with FPGA's embedded memory.

5. CORDIC-BASED FUNCTIONAL UNIT

To avoid any additional processing per pipeline stage (e.g., selection between the CORDIC vectoring and rotation modes), and thus force the pipeline cycle time to the minimum possible, two distinct pipelines, V-CORDIC and R-CORDIC, for the *vectoring* and *rotation* CORDIC operation modes, respectively, are deployed, as presented in Figure 2. An important decision was to store the rotation angle calculated during a *vectoring* operation both into FPGA's embedded memories for future calculations, and as a trace along the pipeline. Since the result of the first vectoring iteration (or of the first angle-restore iteration) is readily available in place to drive the rotation pipeline, the true-dependency between an R-CORDIC operation and the corresponding V-CORDIC operations is reduced to a single clock cycle. Therefore, an R-CORDIC operation can be performed back-to-back to corresponding subsequent V-CORDIC operation. The benefit of this result is that deep software pipelines [19] can be launched into execution, with a large positive impact on the computing power. It also should be noticed that since there is no need to calculate the unitary matrix, **Q**, explicitly, as we show in Section 6, the angle can be represented directly in the *arctan* base [9, 10]. Therefore, the last identity of Equation 1 is not implemented.

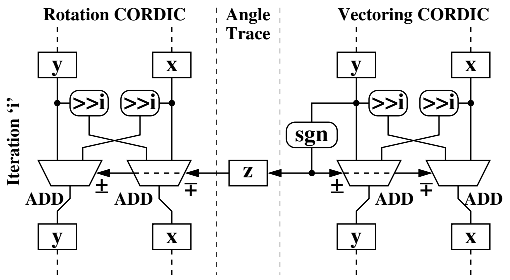

Fig. 2. A CORDIC pipeline stage and the angle trace.

Four η-way SIMD custom instructions are investigated: V_CORDIC_RF, R_CORDIC_RF, ANGLE_RESTORE, and R_CORDIC_FPGA, where η, the SIMD way size, is the number of data items to be processed in parallel by a single instruction. These instructions are defined as follows:

- **V_CORDIC_RF** performs η vectoring operations in parallel on η groups of arguments x and y from register file, and stores the resulting η coordinates x into FPGA embedded memory, and the η angles, z, into both FPGA embedded memory and mentioned traces.

- **R_CORDIC_RF** performs η rotation operations in parallel on η groups of arguments x and y from register file using the η angles stored on mentioned traces, and stores the resulting η coordinates x and y back into FPGA embedded memory.

- **ANGLE_RESTORE_WB** loads η angles from FPGA embedded memory into traces, or writes back to register file values stored into FPGA embedded memory.

- **R_CORDIC_FPGA** performs η rotation operations in parallel on η groups of arguments x and y from FPGA embedded memory using the η angles stored on mentioned traces, and stores the resulting η coordinates x and y back into FPGA embedded memory.

It is important to notice that both R_CORDIC_RF and R_CORDIC_FPGA instructions need valid angle values preloaded on the traces. Thus, appropriate V_CORDIC_RF or ANGLE_RESTORE instructions must be launched prior to the R_CORDIC_RF and R_CORDIC_FPGA instructions. A second important observation is that all the data to be processed are transferred from register file into FPGA only once, during V_CORDIC_RF and R_CORDIC_RF operations. Since the subsequent operations ANGLE_RESTORE_WB and R_CORDIC_FPGA use data within the FPGA only, the data trafic between the register file and the FPGA is minimized.

With the CORDIC-based functional unit and its associated custom instructions, the parallelism of our OFDM detection algorithm is exploited. The FPGA capability to implement application-specific computations allows the deployment of a unity scale factor CORDIC with the throughput of one rotation per cycle. By storing the rotation angle as a trace, a R_CORDIC operation can be launched back-to-back to a V_CORDIC operation, exploiting the intra-block parallelism. In turn, this facilitates the generation of deep software pipelines that are best executed on VLIW architectures [20]. As mentioned, the inter-block parallelism of our algorithm [2] is exploited by the use of SIMD operations.

The computing scenario is as follows. Each subdiagonal element of the channel state matrices, $\mathbf{H}_m$, is forced to zero by a complex Givens rotation described in terms of three real Givens rotations [8]. There is no need to calculate the global unitary matrices, $\mathbf{Q}_m$, explicitly. Instead, this matrix will be represented implicitly by a set rotation angles that are stored into FPGA embedded memory. These angles are readily available on FPGA for interference cancellation.

6. NUMERICAL RESULTS

As mentioned, a Nios II processor [6] and a CORDIC-based reconfigurable functional unit are deployed on a Stratix II FPGA [5]. Nios II is a 32-bit general-purpose processor supporting custom instructions in reconfigurable hardware. To accomodate the large amount of data to be processed, we use a Very Long Instruction Word (VLIW) version of Nios II [21]. In a VLIW processor, load/store instructions can be issued in parallel with arithmetic or logic operations. Thus, the data traffic between memory and register file is carried out in parallel with useful computation, and thus it contributes only a little to the total cycle count.

The QR decomposition has a computational cost of $\mathscr{O}\left(N^3\right)$ [8], which means that for block processing the computational cost is $\mathscr{O}\left(Md^3\right) = \mathscr{O}\left(\frac{N}{d}d^3\right) = \mathscr{O}\left(Nd^2\right)$. Our approach allows parallel processing of all M blocks; thus, the computational latency decreases to $\mathscr{O}\left(d^3\right)$. The interference calculation is essentially a matrix multiplication; thus it has a computational cost $\mathscr{O}\left(N^2\right)$, but it can also be parallelized.

Assuming $N = 256$ and $d = 16$, there are $M = 16$ (16×16) blocks in a 256×256 channel state matrix. To calculate the upper-triangular matrix of a block, $\mathbf{R}_m$, where $m = 1, \ldots, 16$, a number of 1360 complex Givens rotations (3×1360 real Givens rotations) are needed. The unitary matrix of a block, $\mathbf{Q}_m$, is the product of 120 elementary unitary matrices, and does not have to be generated explicitly. The 120 rotations are applied directly on the d-element subvector $\mathbf{y}_m$ and to all subsequent d-element subvectors $\mathbf{y}_m - \hat{\mathbf{y}}_m(i)$. In Section 3, we determined that $I = 2$ iterations are needed for a good BER. For a rate of 2000 OFDM symbols per second, the instruction count per second to perform OFDM detection is $3 \times (1360 + 120I) \times 16 \times 2000 \approx 154 \cdot 10^6$. For the OFDM frame symbol lengths of $N = 512$ and $N = 1024$, the instruction counts per second are $308 \cdot 10^6$ and $616 \cdot 10^6$, respectively.

For the assumed four propagation paths, the path gains and their first-order derivatives are estimated. This is performed by measuring eight pilot symbols per frame, followed by inverting the resulting 8×8 system matrix. This requires $2 \cdot 10^6$ instructions per second to carry out the matrix inversion by a QR decomposition [22]. We assume an overhead of $2 \cdot 10^6$ instructions per second to accomodate the trashing routines. Therefore, the total instruction counts per second for 256-OFDM, 512-OFDM, and 1024-OFDM demodulation tasks are $158 \cdot 10^6$, $312 \cdot 10^6$, and $620 \cdot 10^6$, respectively.

Since each block can be decomposed independently, the OFDM detection process exhibits also inter-block parallelism. Therefore, the utilization of Single-Instruction Multiple Data (SIMD) instructions [7] is enabled. Assuming a 4-way SIMD for 1024-OFDM, and a 2-way SIMD for 512-OFDM, the instruction count per second for OFDM demodulation reduces to $158 \cdot 10^6$ in both cases. Since the Nios II/f processor runs at 185 MHz, the OFDM demodulation under fast fading can be performed in real time for 256, 512, and 1024 symbol lengths.

7. CONCLUSIONS

We have shown that the synergism of inter-block parallelism, intra-block parallelism, and reconfigurable hardware capability to implement application-specific computations, allows the OFDM detection under fast fading radio channels to be performed in real time on reconfigurable hardware. Although the computational complexity of our detection algorithm is already available from previously known detection algorithms, its inherent inter-block and intra-block parallelism implies that low latency implementations can be created, that the sequential nature of the prior algorithms do not allow.

8. REFERENCES

[1] Y.-S. Choi, P. Voltz, and F. Cassara, "On Channel Estimation and Detection for Multicarrier Signals in Fast and Selective Rayleigh Fading Channels," *IEEE Trans. Commun.*, vol. 49, no. 8, pp. 1375–1387, Aug. 2001.

[2] M. McGuire and M. Sima, "Block-wise Parallel Detection for OFDM with Fast Fading," in *Proc. 15th IEEE Int'l Conf. Digital Signal Processing (DSP 2007)*, Cardiff, Wales, U.K., July 2007, accepted.

[3] X. Cai and G.B. Giannakis, "Bounding performance and suppressing intercarrier interference in wireless mobile OFDM," *IEEE Trans. Commun.*, vol. 51, no. 12, pp. 2047–2056, Dec. 2003.

[4] M. Dillinger, K. Madani, and N. Alonistioti, Eds., *Software Defined Radio: Architectures, Systems and Functions*, John Wiley & Sons, July 2003.

[5] Altera Corporation, Inc., *Stratix II Device Handbook*, San Jose, California, Aug. 2006.

[6] Altera Corporation, Inc., *Nios II Processor Reference Handbook*, San Jose, California, May 2006.

[7] D.A. Patterson and J.L. Hennessy, *Computer Architecture. A Quantitative Approach*, Morgan Kaufmann, second edition, 1996.

[8] G.H. Golub and C.F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, 3rd edition, 1996.

[9] J.E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sept. 1959.

[10] J.S. Walther, "A unified algorithm for elementary functions," in *Proc. Spring Joint Computer Conf. the Am. Federation of Information Processing Soc. (AFIPS)*, Arlington, Virginia, 1971, vol. 38, pp. 379–385.

[11] S. Brown, M. Khellah, and Z. Vranesić, "Minimizing FPGA Interconnect Delays," *IEEE Design & Design of Computers*, vol. 13, no. 4, 1996.

[12] P. Metzgen, "A High Performance 32-bit ALU for Programmable Logic," in *12th ACM/SIGDA Int'l Symp. Field Programmable Gate Arrays (FPGA 2004)*, Monterey, California, Feb. 2004, pp. 61–70.

[13] H.M. Ahmed, J.-M. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," *IEEE Computer*, vol. 15, no. 1, pp. 65–82, Jan. 1982.

[14] G.L. Haviland and Al A. Tuszynski, "A CORDIC Arithmetic Processor Chip," *IEEE Tran. Comput.*, vol. C-29, no. 2, pp. 68–79, Feb. 1980.

[15] J.-M. Delosme, "VLSI Implementation of Rotations in Pseudo-Euclidian Spaces," in *Proc. IEEE Int'l Conf. Acoust., Speech, and Signal Process. (ICASSP 1983)*, Boston, Massachusetts, Apr. 1983, vol. 8, pp. 927–930.

[16] J.R. Cavallaro and F.T. Luk, "CORDIC Arithmetic for an SVD Processor," *J. Parallel and Distrib. Computing*, vol. 5, no. 3, pp. 271–290, June 1988.

[17] B. Yang and J. Böhme, "Reducing the Computations of the SVD Array given by Brent and Luk," in *Advanced Algorithms and Architectures*, vol. 1152 of *Int'l SPIE Proc.*, San Diego, California, Aug. 1989, pp. 92–102.

[18] A.A.J. de Lange, A.J. van der Hoeven, E.F. Deprettere, and J. Bu, "An optimal floating-point pipeline CMOS CORDIC processor," in *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS 1988)*, Helsinki, Finland, June 1988, vol. 3, pp. 2043–2047.

[19] V.H. Allan, R.B. Jones, R.M. Lee, and S.J. Allan, "Software Pipelining," *ACM Computing Surveys (CSUR)*, vol. 27, no. 3, pp. 367–432, Sept. 1995.

[20] M. Lam, "Software pipelining: an effective scheduling technique for VLIW machines," in *Proc. CM SIGPLAN 1988 Conf. Programming Language Design and Implementation (PLDI '88)*, New York, NY, U.S.A., June 1988, pp. 318–328.

[21] A.K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster, "An FPGA-based VLIW Processor with Custom Hardware Execution," in *Proc. ACM/SIGDA 13th Int'l Symp. Field-Programmable Gate Arrays (FPGA '05)*, Monterey, California, Feb. 2005, pp. 107–117.

[22] M. Sima and M. McGuire, "CORDIC Scenario for Kalman-Based Channel Estimation," in *Proc. 10th IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim 2005)*, Victoria, B.C., Canada., Aug. 2005, pp. 165-168