Hardware-Software Co-design of QRD-RLS Algorithm with Microblaze Soft Core Processor

Nupur Lodha, Nivesh Rai, Rahul Dubey, and Hrishikesh Venkataraman

Dhirubhai Ambani Institute of Information and Communication Technology
Gandhinagar, Gujarat, India
{nupur_lodha,nivesh_r,rahul_dubey}@daiict.ac.in,
hrishikesh@ieee.org
http://www.daiict.ac.in

Abstract. This paper presents the implementation of QR Decomposition based Recursive Least Square (QRD-RLS) algorithm on Field Programmable Gate Arrays (FPGA). The design is based on hardware-software co-design. The hardware part consists of a custom peripheral that solves the part of the algorithm with higher computational costs and the software part consists of an embedded soft core processor that manages the control functions and rest of the algorithm. The use of Givens Rotation and Systolic Arrays make this architecture suitable for FPGA implementation. Moreover, the speed and flexibility of FPGAs render them viable for such computationally intensive application. The system has been implemented on Xilinx Spartan 3E FPGA with Microblaze soft core processor using Embedded Development Kit (EDK). The paper also presents the implementation results and their analysis.

Keywords: Hardware-Software Co-Design, QRD-RLS Algorithm, FPGA, Givens Rotation, Systolic Array.

1 Introduction

Adaptive signal processing plays an important role in broadband wireless communications with very high signal transmission rates. The process of calculating adaptive weights for such environments can be computationally very demanding task, as bandwidth of the order of megahertz is required. A processor that can estimate the parameters related to the communication channels on a real time basis is necessary in such applications. QR based Recursive Least Square (RLS) is a well established technique for solving the Least Mean Squares (LMS) problem by calculating adaptive weights and is extensively used in applications like Adaptive Beamforming, Multiple-Input-Multiple-Output (MIMO) and Software Defined Radio (SDR) [1][2].

There is a considerable research devoted to algorithms and VLSI architectures for RLS filtering [3-6], with the aim of reducing computational complexity. Much of this work has been concentrated on calculating the inverse of a matrix, in a more stable and less computational manner rather than simple matrix inversion

[1]. The standard RLS algorithm recursively updates the weights using the matrix inversion lemma. A commonly used alternative solution performs a set of orthogonal rotations on the incoming data thereby transforming the over specified rectangular data matrix into an upper triangular form. The weights are then obtained by back substitution. This method is known as QR decomposition based RLS. It enables the matrix to be retriangularised when new inputs are present, without the need to perform triangularisation from scratch [1]. Good numerical performance is achieved by performing the algorithm using Givens Rotation. An efficient implementation of RLS algorithm achieves much faster convergence than the LMS algorithm; however its complexity increases in proportion to the square of the number of parameters to be estimated [7]. To circumvent this problem our architecture is based on a pipelining technique referred to as systolic array.

The research presented in [1] is part of a project to design a single chip adaptive beamforming system. The core of the project is a QR array processor implementing the Enhanced-Squared Givens Rotation algorithm (E-SGR). The work in [3] demonstrates the implementation of Application Specific Integrated Processors (ASIP), specifically targeting QR matrix decomposition.

Conventionally, architectures address different ends of the performance spectrum. At one end there is a general purpose processor which provides flexibility but implements software algorithms in the order of milliseconds. At the other end exists a fixed dedicated custom hardware which can execute algorithms in the order of nanoseconds, but is highly inflexible. Often what is required is the balance between the high performance obtained from hardware, and the flexibility of software. Reconfigurable logic devices such as FPGAs are most suitable for such applications because they provide speed and are upgradeable. Hence they reduce the risk of depending on evolving standards of industries and are cost effective solutions. Moreover, FPGAs with embedded processors are flexible by nature and allow reconfiguration of logic with optimum use of resources. So programmable logic and reconfiguration is now seen as a suitable solution for several wireless applications.

The hardware software co-design term refers to the integration of hardware and software components at the design and development stages [8]. It deals with the problem of designing heterogeneous systems. One of the goals of co-design is to shorten the time-to-market while reducing the design effort and costs of the designed products. The evolution of hardware-software co-design concepts has also accelerated the developments of systems on single chip. It is based on an architecture that implements an embedded processor and one or more dedicated hardware coprocessors to improve the system efficiency. This approach takes advantage of both the flexibility of processors and power and speed of a dedicated hardware.

In this work, we have adopted hardware software co-design methodology for implementing QRD-RLS, an adaptive filter algorithm. This algorithm is partitioned into two parts. The part involving matrix computations is made as a custom peripheral as it is computationally more expensive. Back substitution and other control functionalities are executed on the processor. Interfacing between the processor and the peripheral is done using interfacing buses. Thus, such a design will enable us to make a complete system on programmable chip (SoPC). Also, it will enable us to achieve an optimum trade off between speed and flexibility.

The rest of the paper is organized as follows. Section 2 mentions the key points of QRD-RLS algorithm. Section 3 presents the architecture of the algorithm including Coordinate Rotation Digital Computer (CORDIC) algorithm and systolic arrays. Implementation methodology and results are presented in section 4 and 5 respectively. Section 6 gives the conclusion.

2 QRD-RLS Algorithm

2.1 QR Decomposition

QR Decomposition is an elementary operation which decomposes a matrix into an orthogonal and a triangular matrix [7]. QR decomposition of a real square matrix U is as $U = Q \times R$, where Q is an orthogonal matrix ($Q^T\mathrm{Q} = \mathrm{I}$) and R is an upper triangular matrix. There are different methods to compute QR decomposition like Gram-Schmidt ortho-normalization, Householder reflections and the Givens rotations.

2.2 Givens Rotations

Givens Rotation is used to find an operator which rotates each vector through a fixed angle and this operator can be represented as a matrix [7]. Givens method is useful when adaptive algorithms like RLS are considered. The reason being that in order to obtain the triangular form, the entries in the matrix are zeroed in a particular manner. This leads to a simple implementation which combined with its numerical stability results in an efficient algorithm.

2.3 QR Decomposition Based RLS

The least squares approach attempts to find the set of coefficients w_n that minimizes the sum of squares of error. Equation (1) refers to the least square solution,

$$U\mathbf{w} = \mathbf{d} + \mathbf{e}. \tag{1}$$

where U is an input matrix ($m \times n$ with $m > n$), $\mathbf{d}$ is a known desired sequence and $\mathbf{w}$ is the coefficients vector to be computed such that the error vector $\mathbf{e}$ is minimized. Direct computation of coefficient vector $\mathbf{w}$ involves matrix inversion, which is undesirable for hardware implementations. QR decomposition based least square methods avoid such explicit matrix inversions and are more robust for hardware implementation.

The QRD is used to solve the least square problem as follows. Equation (2) shows the QR decomposition of the compound matrix [U|$\mathbf{d}$] as,

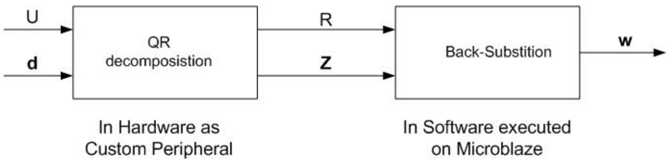
U
d
QR decomposistion
R
Z
Back-Substition
w
In Hardware as Custom Peripheral
In Software executed on Microblaze

Fig. 1. QR Decomposition based RLS

$$[U|\mathbf{d}] = [Q|\mathbf{q}] \cdot \left[\begin{array}{c|c} R & \mathbf{z} \\ \hline 0 & \varsigma \end{array}\right] \tag{2}$$

where [Q|q] is orthogonal and $\left[\begin{array}{c|c} R & \mathbf{z} \\ \hline 0 & \varsigma \end{array}\right]$ is triangular matrix and $\mathbf{z} = Q^T \ \mathbf{d}$. Once the QRD is available, **w** is easily available through the process of back-substitution in (3)

$$R\mathbf{w} = \mathbf{z}. \tag{3}$$

Givens Rotation is applied to compound U matrix to calculate QRD. This gives QRD solution at time k. Now, in order to calculate QRD at time $k+1$ computations do not start from the beginning. QRD is recursively computed at time $k+1$ from time k by updating it as in (4),

$$\left[\begin{array}{c|c} R(k) & \mathbf{z}(k) \\ \hline \mathbf{u}_{k+1}^T & \mathbf{d}_{k+1} \end{array}\right] = Q(k+1). \left[\begin{array}{c|c} R(k+1) & \mathbf{z}(k+1) \\ \hline 0 & * \end{array}\right] \tag{4}$$

where * is a don't care entry, left hand side matrix is a *"triangular-plus-one-row-matrix"* and right hand is an orthogonal times triangular matrix. The next value of coefficient **w** follows from the back substitution in (5)

$$R(k+1)\mathbf{w}(k+1) = \mathbf{z}(k+1). \tag{5}$$

This constitutes the complete QRD-RLS algorithm [7]. The above procedure avoids calculating the inner product U^TU and hence it is numerically more stable. Fig. 1 shows QR decomposition based RLS algorithm.

3 QRD-RLS Architecture

3.1 CORDIC in QRD

CORDIC describes a method to perform a number of functions including trigonometric, hyperbolic and logarithmic functions. The algorithm is iterative and uses only additions, subtractions and shift operations [9]. This makes it more suitable for hardware implementations. CORDIC cells are used to calculate and apply the unitary transformation represented by Givens rotation.

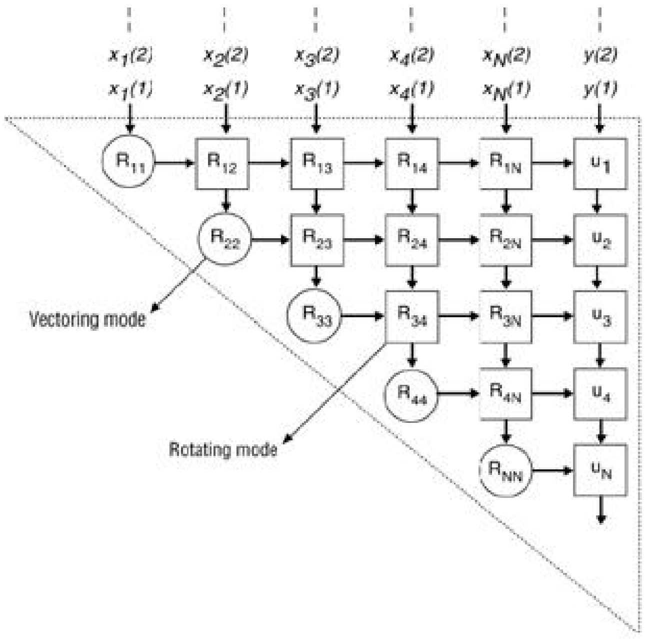

Fig. 2. Systolic Array Architecture

3.2 Systolic Arrays

Systolic arrays have two types of processing nodes: Boundary cells and Internal cells. Boundary cells are used to calculate the Givens rotation that is applied across a particular row in a matrix. The unitary transform which is calculated by the boundary cells, is taken as an output and applied to the remainder of the row containing internal cells [5-6].

Fig. 2 shows the use of systolic array architecture for performing the QR decomposition of the input matrix X. The rows of matrix X are fed as inputs to the array from the top along with the corresponding element of the vector **u**, which is the desired sequence. The data is entered in a time skewed manner. The calculations for a particular decomposed matrix R propagate through the array. The R and **u** values held in each of the cells once all the inputs have been passed through the matrix are the outputs from QR-decomposition. These values are subsequently used to derive the coefficient vector **w**. Each of the cells in the array can be implemented as a CORDIC block.

Direct mapping of the CORDIC blocks to the systolic array in fully parallelized mode consumes a significant number of logic blocks but at the same time yields very fast performance. The resources required to implement the array can be reduced by trading throughput for resource consumption via mixed and discrete mapping schemes, which map multiple nodes on a single instantiation of hardware [1][4].

3.3 Back Substitution

The back substitution procedure primarily involves multiplication and division operations. The Microblaze embedded soft core processor can be configured with its optional functionalities like hardware multiplier and division unit. This feature makes it ideal to implement back substitution. The CORDIC block performs the QR decomposition and stores the R and $\mathbf{z}$ values in registers accessible to the Microblaze processor, which then calculates coefficient values and stores the results back into memory.

4 Implementation Methodology

4.1 Profiling and Hardware-Software Partitioning

The complete algorithm has been programmed in C. It was executed on a 2 GHz Intel Dual Core high performance processor. As the profiler showed, matrix decomposition part took almost 94% of the total time, indicating that it is the most expensive operation in terms of computational cost, while back-substitution represented remaining 6% of the time. So, we chose to implement matrix decomposition in hardware by means of a custom peripheral. Back-substitution was executed in software on Microblaze.

4.2 Hardware Flow

Hardware platform describes the flexible, embedded processing subsystem which is created according to the demands of the application. The hardware platform consists of one or more processors and peripherals connected to the processor buses [10]. Fig. 3 shows the block diagram of the complete system configured for our design.

Microblaze: Microblaze is 32 bit Reduced Instruction Set Computer (RISC) architecture. A "Harvard" style bus architecture is used which includes 32 bit general purpose registers and separate instructions and data buses. It features a five stage instruction pipeline. As it is a soft core processor, the functional units incorporated into its architecture can be customized as per the needs of the application. Thus, the barrel shifter unit, hardware divider unit, data cache and instruction cache can be optionally instantiated along with the processor. Extra peripherals like UART, Ethernet controllers or other IP cores can be configured using EDK [10].

Fast Simplex Link (FSL) Bus: FSL is a unidirectional point-to-point communication channel bus used to perform fast communication between any two design elements on the FPGA when implementing an interface to the FSL bus. Microblaze can be configured with up to 16 FSL interfaces, each consisting of one input and one output port. FSL provides mechanism for unshared and non-arbitrated communication mechanism. This can be used for fast transfer of data words between master and slave implementing the FSL interface [10]. One FSL link was configured for communicating between Microblaze and custom peripheral.

4.3 Software Flow

A software platform is a collection of software drivers, libraries and, optionally, the operating system on which to build an application. The Library Generator tool configures libraries, device drivers, file systems and interrupt handlers for the embedded processor system by taking Microprocessor Software Specification (MSS) file as an input [11].

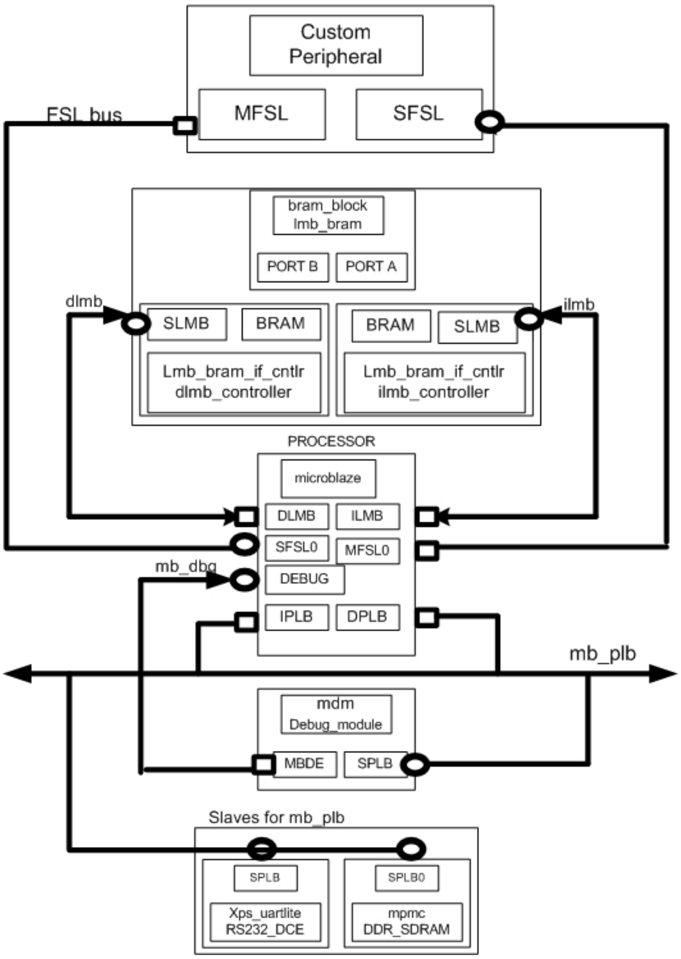

Fig. 3. Block Diagram

5 Implementation Results

5.1 MATLAB Results

QRD-RLS algorithm was implemented in Matlab, in order to verify its functionality. The desired data vector was obtained by reading a wave file. The input matrix was obtained by filtering this data through a channel and adding white Gaussian noise. The outputs obtained were the coefficient vectors and the error vector in modeling the channel.

Fig. 4 shows the error curve corresponding to the number of samples. It shows that as the number of samples increase, the error value decreases and converges to a very small value of 4.9×10^{-6}.

5.2 Verification of Our Design

An example scenario of 3×3 matrix ($M = 3, N = 3$) was considered because of the ease in implementing the algorithm, targeting the given Spartan 3E FPGA device. 16 bit real inputs to CORDIC blocks were considered. Fixed point representation was used to represent real numbers. The input data was the same which was used for Matlab implementation. Table I shows the comparison between the output coefficient values obtained from our design and the values obtained from Matlab for two updates of the coefficients. Update implies when all cells in the systolic array are updated with their new R and $\mathbf{z}$ values.

We see that the coefficient values obtained from our design i.e. Hardware Software (H/W-S/W) Co-design, do not match exactly with Matlab values. Table 1 also shows the error computed between both results. There is a small error value

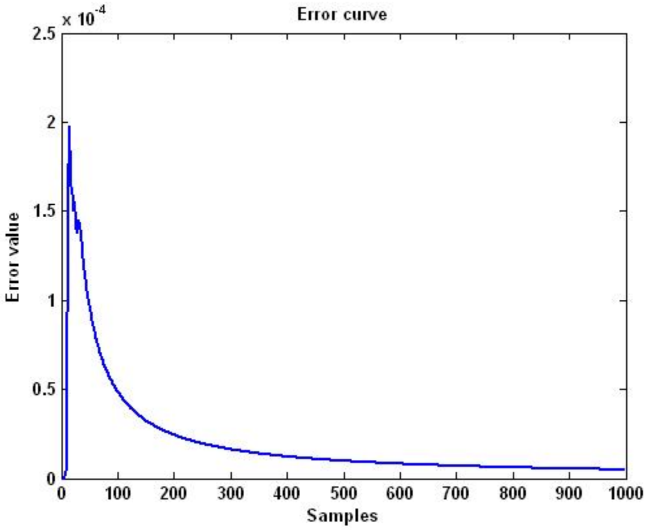

Fig. 4. Error Curve

Table 1. Comparison between Matlab and H/W-S/W Co-design Results

Weight Coefficients	MATLAB Results	H/W-S/W Co-design Results	Error
w1 (update1)	0.58e-2	0.65e-2	-6.071e-4
w2 (update1)	0.275e-1	0.269e-1	6.767e-4
w1 (update2)	-0.89e-2	-0.88e-2	-1.245e-4
w2 (update2)	0.255e-1	.0254e-1	1.371e-4

because our design is tailor made to work for four decimal places, while Matlab is a simulation software which uses IEEE double precision. Such a precision is very difficult to obtain in hardware due to resource constraints.

5.3 Analysis of Our Design

The custom peripheral was described in a high level description language like Verilog. The design was synthesized on a Xilinx Spartan-3E $XC3S500E-4FG320$ FPGA. CORDIC blocks were used with the speed of 170 MHz.

Two mapping schemes were used:- Direct mapping and Discrete mapping. In Direct mapping, where each cell of the systolic array was mapped to a CORDIC block, 5 CORDIC blocks were required for 3×3 matrix. The update time obtained was $387ns$. Update time refers to the time required before all the cells in the systolic array are updated with their R and $\mathbf{z}$ values. But it led to huge consumption of resources, amounting to 5262 slices. This number even exceeded the number of slices available in targeted FPGA device (4656). So discrete mapping was adopted. In this scheme, two CORDIC blocks are used; one for translation operations and the other for rotation operations. The resource consumption reduced to 1689 slices, which is less than 40% of the available resources. The update time obtained was $454ns$, which is slightly higher than direct mapping.

Table 2 gives the resource estimates of *'Pure Hardware'* approach compared with our *'Hardware-Software Co-design'* approach for two matrix sizes of 3×3 and 4×4.

It can be seen that for $N = M = 3$ the resource estimates of "H/W-S/W Co-design" approach and "Pure Hardware" approach differ by only a small

Table 2. Resource Estimates of Complete Design

Matrix size	Method	Slices	LUT's	Flip Flops
3x3	Pure Hardware	3416	4588	6259
	H/W-S/W Co-design	3791	5247	5685
4x4	Pure Hardware	3844	5322	7473
	H/W-S/W Co-design	3821	5286	6190

number. For $N = M = 4$, the resource estimates of our approach are less than "Pure Hardware" approach. Thus our approach is scalable to larger matrix sizes with only a minor increase in resources, while there is a significant increase in resources for "Pure Hardware" approach. This is because in "Pure Hardware" approach back substitution part also takes resources, but this does not hold true for "H/W-S/W Co-design" approach where back substitution is executed in software. The software part of the design also lends flexibility, which is an important requirement for continually evolving standards of design cycle.

The back substitution part of the algorithm was coded in C language and it was executed on Microblaze soft core processor with clock frequency of 50 MHz. For $N = 3$, the Microblaze processor took 64 clock cycles or 1.28 μs which is acceptable for many applications. Though it is slower than a pure hardware approach, the presence of Microblaze processor lends flexibility to the whole system. Moreover, as matrix size increases, the execution time of custom peripheral which contains complex logic increases considerably as compared to software time, thereby allowing Microblaze to implement other data and control functions on the FPGA.

This QRD-RLS architecture achieves a throughput of 1.68 μs or 0.59M updates per second.

The design is easily extendable to other matrix sizes of $N \times M$ by changing the control unit. Also, there is a trade off between area of the design and throughput by using different mapping schemes. The abundant resources of newer and bigger FPGA families support the realization of a fully parallel hardware design, should the throughput requirements of the target application demand extremely high performance.

6 Conclusion

A novel design methodology of Hardware Software Co-design has been proposed in this paper. QRD-RLS algorithm has been implemented using Xilinx Spartan 3E FPGA with embedded Microblaze soft core processor. The use of systolic array and CORDIC architecture makes the algorithm suitable for hardware implementation. This QRD-RLS architecture achieves a throughput of 1.68 μs or 0.59M updates per second. The interfacing of the peripheral with Microblaze embedded processor provides an element of flexibility, which cannot be achieved in pure hardware approach. The flexibility will allow the system to configure adaptively for varying wireless conditions and external requirements. Moreover, it facilitates to create a SoPC. The independent custom unit which does the decomposition of the matrix can be used in any application depending upon different conditions. For example, the same hardware can be used in beamforming, MIMO, SDR or any such application where decomposition of matrix is required.

References

1. Lightbody, G., Walke, R., Woods, R., McCanny, J.: Linear QR Architecture for a Single Chip Adaptive Beamformer. Journal of VLSI Signal Processing Systems 24, 67–81 (2000)
2. Guo, Z., Edman, F., Nilsson, P.: On VLSI Implementations of MIMO Detectors for Future Wireless Communications. In: IST-MAGNET Workshop, Shanghai, China (2004)
3. Eilert, J., Wu, D., Liu, D.: Efficient Complex Matrix Inversion for MIMO Software Defined Radio. In: IEEE International Symposium on Circuits and Systems, Washington, pp. 2610–2613 (2007)
4. Gao, L., Parhi, K.K.: Hierarchical Pipelining and Folding of QRD-RLS Adaptive Filters and its Application to Digital Beamforming. IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing 47 (2000)
5. Walke, R.L., Smith, R.W.M., Lightbody, G.: Architectures for Adaptive Weight Calculation on ASIC and FPGA. In: Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers, California, vol. 2, pp. 1375–1380 (1999)
6. Yokoyama, Y., Kim, M., Arai, H.: Implementation of Systolic RLS Adaptive Array Using FPGA and its Performance Evaluation. In: 2006 IEEE Vehicular Technology Conference (VTC 2006 Fall), Montreal, Canada, vol. 64, pp. 1–5 (2006)
7. Haykin, S.: Adaptive Filter Theory, 4th edn., pp. 513–521. Prentice Hall, Englewood Cliffs (2001)
8. Gupta, R.K., Micheli, G.D.: Hardware-Software Cosynthesis for Digital Systems. Design and Test of Computers 10, 29–41 (1993)
9. Andraka, R.: A survey of CORDIC algorithms for FPGA based computers. In: 1998 ACM/SIGDA sixth International Symposium on FPGAs, Monterey, pp. 191–200 (1998)
10. Xilinx Inc., Microblaze Processor Reference Guide (2004), `http://www.xilinx.com`
11. Xilinx Inc., Embedded System Tools Reference Manual (2004), `http://www.xilinx.com`