FPGA-Based Transformable Computers for Fast Digital Signal Processing

H. A. Chow H. Alnuweiri

Department Electrical Engineering
The University of British Columbia
Vancouver BC Canada V6T 1Z4

S. Casselman

Virtual Computer Corporation
6925 Canby Ave. Suite 103
Reseda CA 91335 USA

Abstract

FPGA-based computing systems provide a feasible and cost-effective platform for implementing fast parallel arithmetic circuits for digital signal and image processing. This paper reports the results obtained from embedding a highly parallel convolution algorithm on an FPGA-based "transformable" computer. Such a computer is intended to serve as a transformable co-processor for a standard microprocessor system. However, the transformable co-processor is reconfigurable and is capable of exploiting the concurrency of computations more than the "sequential" microporcessor. Our experiments show that a significant gain in speed can be achieved by using the transformable co-processor. We present an example of performing a sequence of (independent) 16-point convolutions on 8-bit data, and show that the speed factor improves significantly as the number of convolutions to be performed increases.

1 Introduction

As the need for high speed real-time DSP grows, fast parallel architectures will be needed to accelerate the normally computationally intensive DSP computations. In this paper, a parallel VLSI architecture which uses Toom's algorithm (based on mapping a modified version of the tensor product factorization of the linear convolution [1]), is implemented in a FPGA-based transformable computer board. In the next section, the basic mathematical formulas for the linear convolution algorithm are presented. Then the possible parallel architectures are outlined.

The advancement of the current tranformable computer technology is the ability to integrate the FPGA configuration file into software programming languages such as C Programming Language, and this is called Hardware Object Programming [3]. This technique allows the designer to implement most of the originally software operations, e.g., convolution, FFT and DSP circuitry, into hardwired integrated circuits in FPGA, and which can be included in the software program. The advantage of being able to do this is that the user can configure a convolution circuit in the FPGA if a line in the program requires the computer to do a convolution, or if there is an FFT calculation in the next line, then the user can create a FFT circuit to do that. Since all the integrated circuits are algorithm specific, they are much faster than the general purpose microprocessors. As a result of the cooperation between transformable computer and the host system's microprocessor, the overall performance is greatly enhanced.

This paper presents an implementation of a highly parallel VLSI linear convolution architecture in a FPGA-based transformable computer - EVC1s. In section (II), a parallel VLSI linear convolution architecture is reviewed, together with all the mathematical models. In section (III), the process of implementing the circuitry is outlined. The hardware object programming techniques are presented in section (IV). In section (V), the performance of the resulting design is evaluated.

2 A Parallel VLSI Linear Convolution Architecture

Let x_n and h_n are two sequences of length n, then their linear convolution $y_{2n-1} = x_n * h_n$ is given by

$$y_i = \sum_k h_{i-k} x_k$$

which can be expressed in a matrix form $Y_{2n-1} = C_n X_n$

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ \vdots \\ \vdots \\ y_{2n-2} \end{pmatrix} = \begin{pmatrix} h_0 & 0 & \dots & 0 \\ h_1 & h_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-1} & h_{n-2} & \dots & h_0 \\ 0 & h_{n-1} & \dots & h_1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_{n-1} \end{pmatrix}$$

The matrix C_n is referred to as an n point linear convolution. Toom's fundamental factorization can be reformulated in the matrix form to become

$$\begin{pmatrix} h_0 & 0 \\ h_1 & h_0 \\ 0 & h_1 \end{pmatrix} = BDA$$

0-8186-7086-X/95 $04.00 © 1995 IEEE

where

$$B = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$$D = diag\left[(A)\begin{pmatrix} h_0 \\ h_1 \end{pmatrix}\right]$$

In [1], an efficient tensor product fomulation for the fundamental linear convolution algorithm is presented, which has the form,

$$C_2^{\alpha} = \tilde{B}D\tilde{A}$$

$$D = diag\left(\tilde{A}h\right)$$

where

$$\tilde{A} = \prod_{k=0}^{\alpha-1}(I_{3^{\alpha-1-k}} \bigotimes A \bigotimes I_2^k)$$

$$\tilde{B} = \prod_{k=1}^{\alpha}(I_{3^{k-1}} \bigotimes R_{2^{\alpha-k}}(B \bigotimes I_{2^{\alpha-k+1}-1}))$$

where $\bigotimes$ denotes the tensor-product operation and $R_{2^{\alpha-k}}$ is a special matrix of zeros and ones and of dimension $(2^{\alpha-k+2}-1)$ by $3(2^{\alpha-k+1}-1)$. The [row, column] coordinates of the ones in $R_{2^{\alpha-k}}$ is given by the expression $[2^{\alpha-k}i+j, (2^{\alpha-k+1})i+j]$ where

$$i = 0, 1, 2$$

$$j = 0, 1, ..., (2^{\alpha-k+1}-2)$$

and all other entries are zeros. The actual implementation is based on the following modified formulas:

$$\tilde{A} = \prod_{k=0}^{\alpha-1}(I_{3^{\alpha-1-k}2^k} \bigotimes A)P_{3^{\alpha-k-1}2^{k+1},3^{\alpha-k-1}2^k}$$

$$\tilde{B} = \prod_{k=1}^{\alpha}(I_{3^{k-1}} \bigotimes R_{2^{\alpha-k}}S)$$

$$S = (P_{3(2^{\alpha-k+1}-1),3}(I_{2^{\alpha-k+1}-1} \bigotimes B)P_{3(2^{\alpha-k+1}-1),(2^{\alpha-k+1}-1)})$$

2.1 A Parallel Architecture for Matrix A

For each triple-matrix product in stage k, matrix **A** consists of a permutation matrix, followed by a parallel tensor product and a shuffle permutation. Since matrix **A** in the equation has only ones and zeros, the multiplication can be done by the single adder as shown in Figure (1).

The complete diagram representing one of the α stages forming matrix **A**, including the $(3^{\alpha-k+1}2^k)$ shuffles, the $(3^{\alpha-k+1}2^k)$ parallel adders and the (2^k) shuffles, is shown in Figure (2).

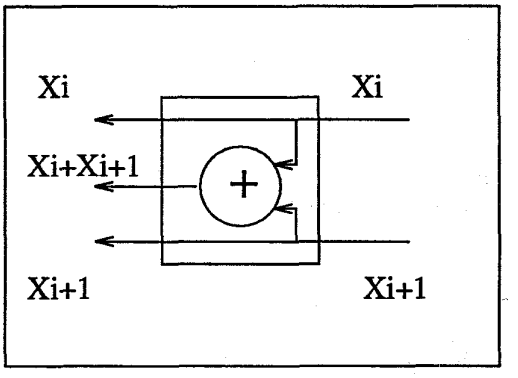

Figure 1: The effect of the matrix **A** on the input vector.

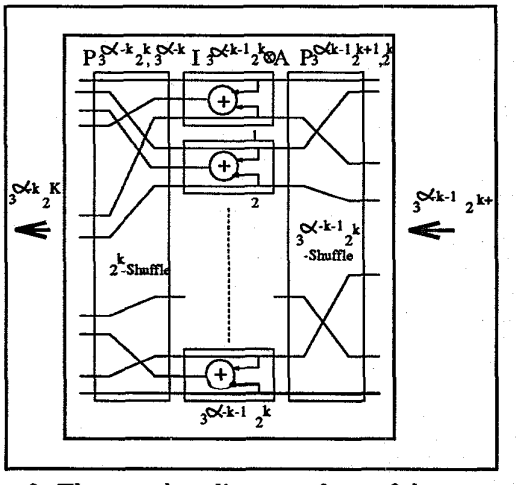

Figure 2: The complete diagram of one of the α stages representing matrix **A**.

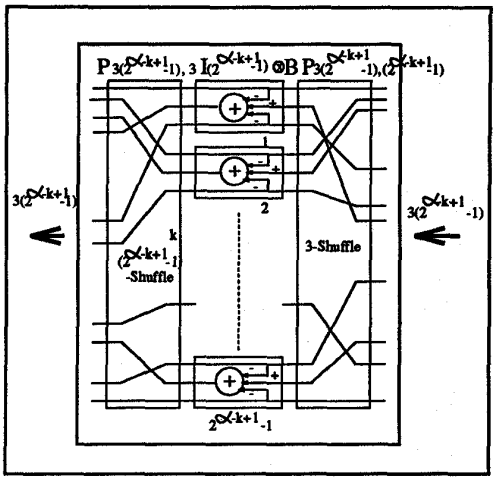

Figure 3: The complete diagram representing **S**.

2.2 A Parallel Architecture for Matrix B

The matrix **B** can be expressed in the following fashion:

$$\widetilde{B} = \prod_{k=1}^{\alpha} (I_{3^{k-1}} \bigotimes R_{2^{\alpha-k}} S)$$

where S has the form

$$S = P_{3(2^{\alpha-k+1}-1),3}(I_{2^{\alpha-k+1}-1} \bigotimes B) P_{3(2^{\alpha-k+1}-1),(2^{\alpha-k+1}-1)}$$

The complete representation of S is shown in Figure (3).

2.3 A Parallel Architecture for Matrix D

The matrix **D** has the form $D = diag[\widetilde{A}h]$. In practical applications, the vector $\vec{h}$ is already known so that the vector multiplication can be pre-computed. Since matrix **D** is a diagonal matrix, it implies an element-wise multiplication.

3 Mapping the Convolution Network

The entire implementation process can be divided into four major stages:

1. Identify the necessary integrated circuits for the software program and enter the design with schematic-entry tools or high-level hardware description languages such as VHDL.
2. Perform design functional verification.
3. Download the design into the Xilinx FPGA Logical Cell Arrays (LCAs).
4. Convert the Xilinx configuration file to a C programming language header file.

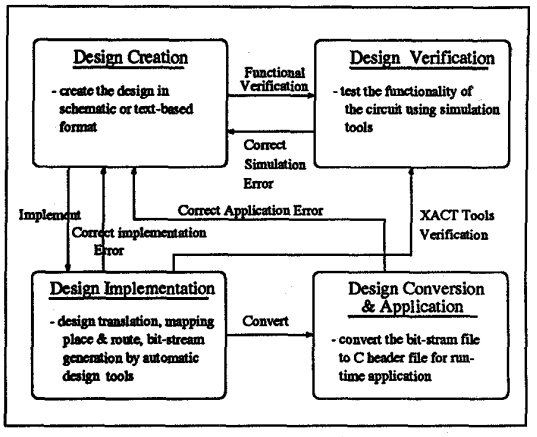

Figure 4: Design flow diagram shows the relationship of design steps.

Figure (4) shows the design flow diagram based on the above steps.

In the following, we provide certain design and implementation details. Basically, our approach was based on converting a schematic design into a bit-stream file that can be downloaded into the FPGA.

Bit-Serial Arithmetic Blocks

All the arithmetic units used in our design are bit-serial. Bit-serial arithmetic has many advantages over bit-parallel (or word-operand) arithmetic. Mainly, bit-serial arithmetic units can be implemented using more compact propagation-free logic circuits, require low I/O bandwidth (because operands are accessed bit by bit), and can be easily modified to accommodate different operand word-lengths. The latency problem of bit-serial designs can be easily overcome by using bit-level pipelining. The low I/O requirements of the individual bit-serial units means that more such units can be used in parallel with overloading the chip I/O or its internal paths.

Figure (5) shows the schematic diagram of a bit-serial adder and figure (6) shows the schematic diagram of a 4-bit serial/parallel multiplier. The detailed structure of each a multiplier cell is shown in the upper portion of figure (6). Bit-serial multipliers are used only in one stage of the convolution circuit, mainly in carrying out the action of the diagonal matrix **D**. Note that since vector $\vec{h}$ is pre-calculated, it is presented as the parallel input to the multiplier.

The arithetic operations needed to implement the actions of matrices **S** and **B** can be also realized using bit-serial adders and subtractors. To illustrate the scale of the convolution circuit, the schematic diagram of the circuit realzing the matrix A is shown in Figure (7).

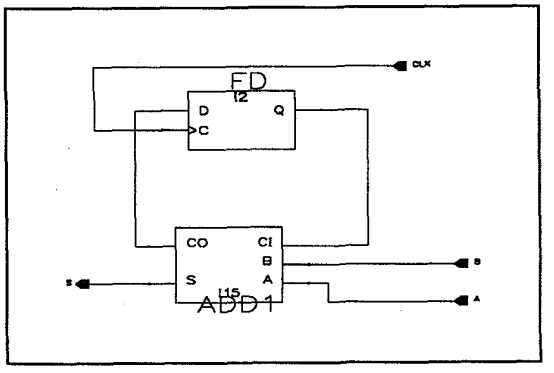

Figure 5: The schematic diagram of the bit-serial adder.

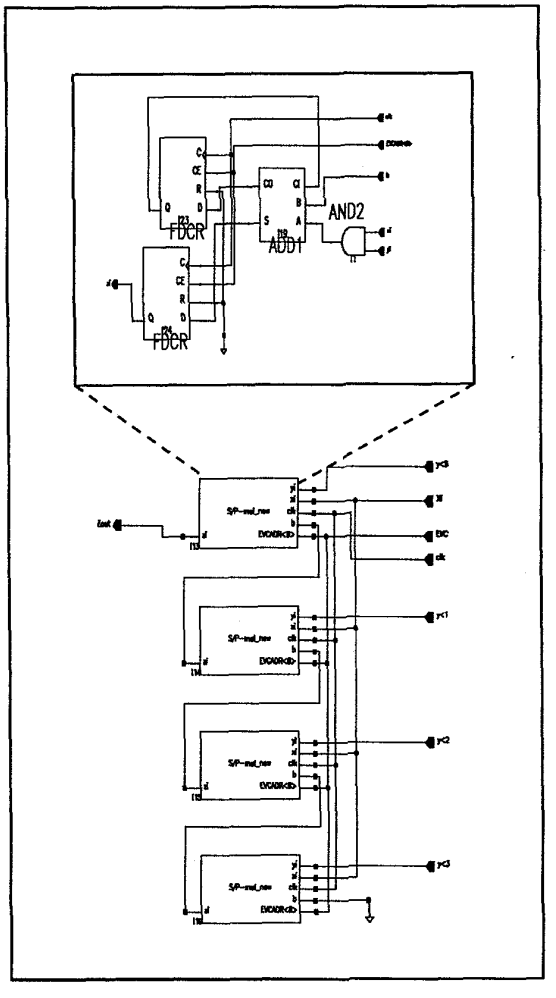

Figure 6: A 4-bit bit-wise multiplier.

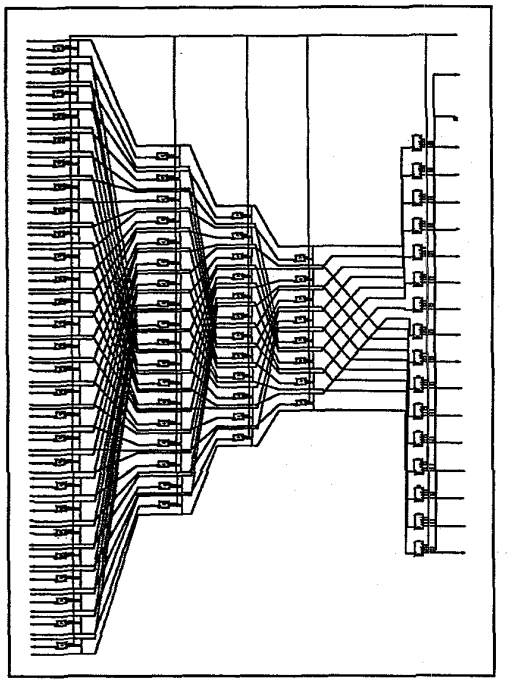

Figure 7: The complete schematic diagram for matrix A.

Schematic Entry and Conversion

The schematic entry and conversion process is architectural dependent. In our experiment, since Xilinx's FPGA is used, a standard Xilinx design conversion procedure [4] [5] is adopted. The conversion process consists of seven steps: design entry, XNF translation, optimization and mapping, merging, LCA translation, place and route, and bitstream generation. Figure (8) shows the schematic entry and conversion process flow.

3.1 Design Implementation Using EVC1s

The EVC1s (Engineers' Virtual Computer, SBus) is one of the transformable computing products using the reconfigurable ability of the Field-Programmable Gate Arrays (FPGA). It is produced by Virtual Computer Corporation (VCC) and is delivered in the form of a standard single slot SBus card that can be plugged into any SBus compatible systems. The EVC1s uses Xilinx reconfigurable logic device XC4013 Logical Cell Arrays (LCA) which provides a versatile hardware platform.

In order to use EVC1s for hardware object programming, the `design.rbt` file produced in the design conversion process must be changed into the C header file. The program used in this step is `r2h` which is supplied by VCC. The result file is called `design.h` which can be included in the source code file, just as other header file in the C Programming Language.

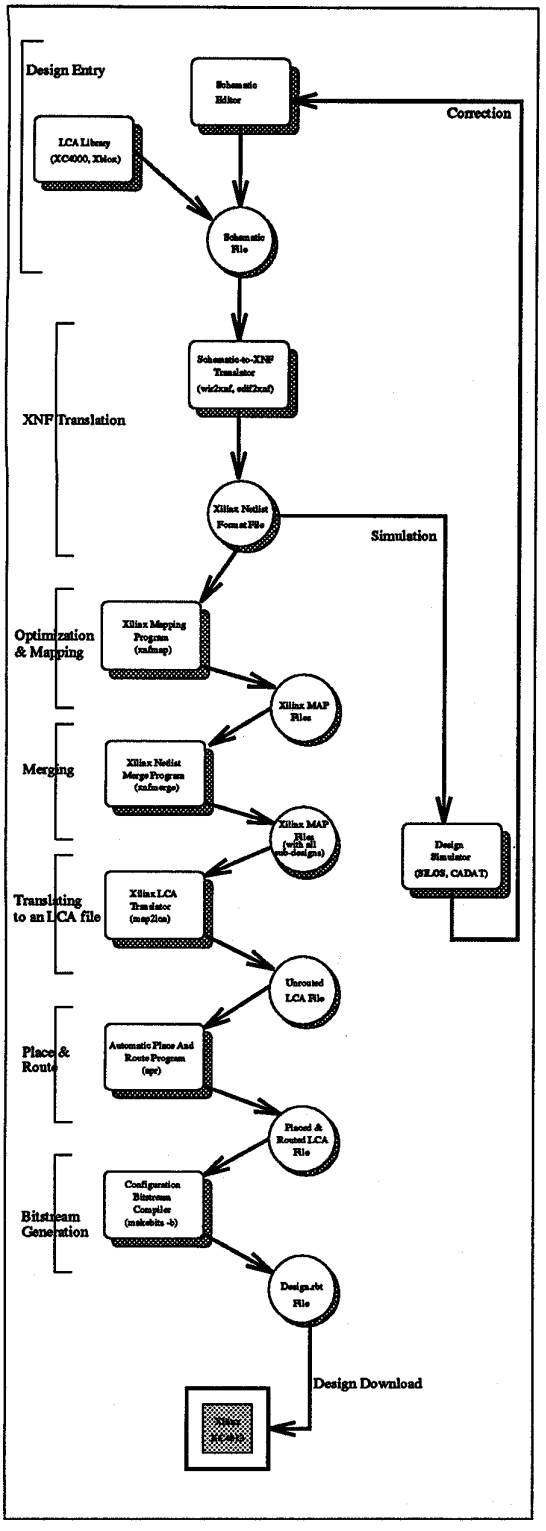

Figure 8: FPGA Design Conversion Flow Chart.

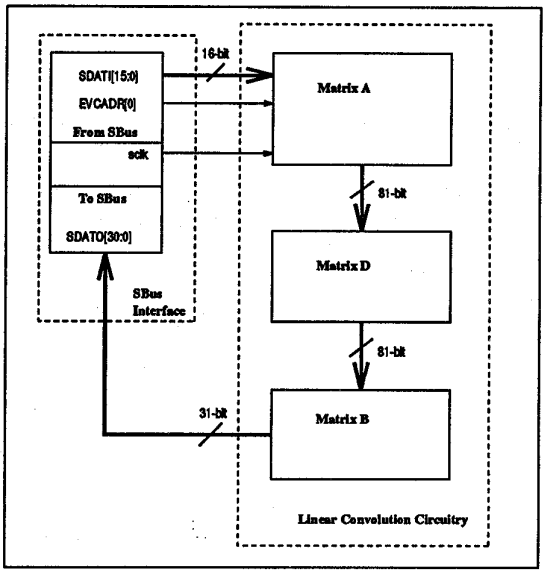

Figure 9: Convolution circuit block diagram with the SBus interface macro.

3.2 SBus Slave Interface

The SBus Slave interface of the EVC-1 unit provides an intermediate circuitry for the user's design and SBus. A Slave Interface *marco* is used in all the design examples. Figure (9) shows the convolution circuit block diagram with the interface marco. In the SBUS interface macro, ports `SDATI[15:0]` are used by the program to write data to the designed convolution circuit, while ports `SDATO[30:0]` are used by the program to read data from the circuit. Also, `EVCADR[0]` is the trigger line and `SCLK` is the SBus clock. The SBus Slave Interface marco symbol is stored in the file `EVCSLVIF.1`. The SBUS interface can be either placed directly into the schematic design (such as in the schematic entry tools form Viewlogic), or use the `evcaddif` program to add the macro symbol to the design's `.xnf` file. More details are covered in [2]. Each time the `EVCdownload()` function is used, it returns a pointer to the address `EVCADR0` which, in the memory map of the slave interface, is corresponding to address 0x10000. If the user adds 1 to the base pointer in the program, then the pointer will point to `EVCADR1` or 0x10004.

4 Using the Convolution Circuit in a Software Program

The `design.h` file contains an array with the name `design`, which has all the configuration data that can be downloaded into the XC4013 inside EVC1s. If we have more than one design, we just have to repeat the above steps for a couple of times and we will have a number of different header files that can all be included in the C source file. So, each time when we want to download a particular integrated circuit to the EVC1s, we just need to use the following function

call in our program:

```
EVCadd = EVCdownload(conv, 1, 1);
```

In this statement, `design` is the array which contains all the LCA configuration data while the second argument specifies the device type (0 = XC4010 or 1 = XC4013) and the third argument specifies the SBus slot number. The return value is a pointer which points to the base address of the SBus Slave interface `EVCADR0`. Then we can use the downloaded circuit to carry out the parallel linear convolution operations. After the operations, we can use `EVCreset(1)` to clear the LCA configuration so a new circuit can be downloaded next time. The integer argument `1` indicates the SBus slot number.

The following table [2] shows the available functions that can be used in C source code.

Function	Description
EVCdownload	Downloads configuration data to EVC1s with 3 arguments: array name, device identifier and SBus slot no.
EVCreset	Resets EVC1s with 1 argument: SBus slot no.
EVCrbtrig	Toggles RDBK.TRIG on the XC40xx device.
EVCreadback	Returns a single bit of the readback data stream from the XC40xx device.

5 Performance Evaluation

In general, most of the programs would have their computation speed increased significantly after adopting the EVC1s. In [2], the Fibonacci Sequence Generator in the EVC1s almost doubles computation speed of a Sun SPARC 2 workstation. As the design becomes more specialized for a certain computation algorithm, the magnitude of improvement in computation speed increases accordingly. The linear convolution circuit gives a very good example in how much speed increase can be achieved when the EVC1s is used to carry out the convolution computation.

5.1 Linear Convolution Circuit Testing Program

Before the linear convolution circuit can be used in a C program, it has to be compiled according to the steps outlined in previous sections. The design took 1 hour and 30 minutes to compile and the resulting circuit occupied 37 percent of the space available in a Xilinx XC4013 device. Then a C program is written to test the performance of the linear convolution circuit. Basically, the program takes two 16-point vector x and h and convoluted them. The structure of the program looks like this:

- Download the linear circuit configuration file into the EVC1s using `EVCdownload()` function.
- Shift the least significant bit of each number in vector x to form a 16-bit integer, then shift the next significant bit to form another 16-bit integer. If the values in vector x has 32-bit precision, then we will have 32 16-bit integers.
- Write those 16 or 32 16-bit integers continuously into the memory location returned by the function `EVCdownload()` and read back the resulting values from the next memory location.
- Carry out the reverse operation as mentioned in step 2 so that we can obtain 31 16-bit or 32-bit integers.

We also included a timing function to record the computation time used by the program.

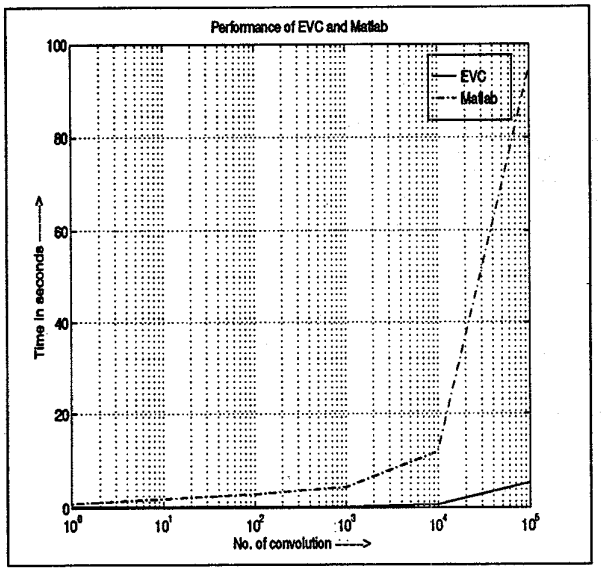

Figure 10: Performance of C program with EVC1s and Matlab convolution function.

5.2 Computation Time Comparison

Meanwhile, we have used the convolution function in a very common computation program – Matlab to do the 16-point convolution. Both the C program using the EVC1s and the convolution function in Matlab are used to compute the convolution for 1, 10, 100, 1000, 10000 and 100000 times. The machine we used was a Sun SPARCStation10 workstation. A total of 5 sets of timing data are collected and the averages are used to plot the graph. The resulting graph is shown in Figure (10).

As we can see from Figure (10), the improvement in the computation speed is not that significant when only a small number of convolutions are involved. But when the number of computation increases, the gain in speed becomes very large. Although it is not an exact comparison since Matlab is a high-level application and there are a lot of overheads which will not appear in a C language convolution program, the graph still shows a very promising improvement in computation

speed. It is expected that most DSP programs using EVC1s will have similar improvement.

6 Conclusions

This paper presents one of the versatile applications of the FPGA-based transformable computer. The use of FPGA-based transformable computers in DSP, modeling and simulation, etc. is a simple and fast method to achieve much higher computational speed. FPGA-based transformable computers allow the software algorithms to be hardwired into silicon. With more and more software algorithms implemented in FPGA-based transformable computer, a hardware macro library can be created and the software performance can be greatly accelerated.

References

[1] A.Elnagger, H.M. Alnuweiri, M.R. Ito, "Highly Modular Parallel VLSI Architectures for Digital Filtering and Linear Convolution", First International Conference on Electronic, Circuits and Systems, Cairo, Dec. 19-22, 1994.

[2] Michael Thornburg, Steven Casselman, John Schewel, "Engineers Virtual Computer Users Guide, EVC1s", Virtual Computer Corporation, August 12, 1994.

[3] John Schewel, Mike Thornburg, Steve Casselman, "Programming Hardware Objects - Placing digital designs into Software Applications for use in Rapid Product Development and Software Acceleration", Virtual Computer Corporation, 1994.

[4] The XC4000 Data Book, Xilinx Corporation, 1993.

[5] The XACT Development System Reference Guide, Xilinx Corporation, 1993.