Area and Time Efficient Implementation of Matrix Multiplication on FPGAs[*]

Ju-wook Jang[1], Seonil Choi[2], and Viktor K. Prasanna[2]
[1]Electronic Engineering   [2]Electrical Engineering-Systems
Sogang University   University of Southern California
Seoul, Korea   Los Angeles, CA, U.S.A.
jjang@sogang.ac.kr   {seonilch, prasanna}@usc.edu

Abstract

We develop new algorithms and architectures for matrix multiplication on configurable hardware. These designs significantly reduce the latency as well as the area. Our designs improve the previous designs in [6] and [1] in terms of the area/speed metric where the speed denotes the maximum achievable running frequency. The area/speed metrics for the designs in [6], [1], and our design are 14.45, 4.93, and 2.35, respectively, for 4×4 matrix multiplication. The latency of the design in [6] is $0.57\mu s$, while our design takes $0.15\mu s$ using 18% less area. The area of our designs is smaller by $11\% - 46\%$ compared with the best known systolic designs based on [8] with the same latency for the matrices of sizes $3 \times 3 - 12 \times 12$. The performance improvements increase with the problem size.

1   Introduction

Matrix multiplication is a frequently used kernel operation in a wide variety of graphic, image, robotics, and signal processing applications. Several signal and image processing operations consist of matrix multiplication. Most previous implementations of matrix multiplication on FPGAs focused on trade-offs between area and maximum running frequency [1][6]. In this paper we develop designs which provide improved trade-offs between area and latency.

We significantly reduce the number of registers involved in the data movement. n^2 registers of 16-bit words and $n^2 + 6n^2/s, (1 \le s \le n)$ registers of 8-bit words are involved in the data movement for $n \times n$ matrix multiplication in [8]. Only $4n$ registers of 8-

bit words are involved in the data movement in our design (based on Theorem 1). A closed form function representing the area of a design is derived by incorporating architecture / algorithm details and FPGA vendors' specifications. The function enables designer to make trade-offs between area and latency before launching time-consuming low-level simulation.

Mencer et al [6] implemented matrix multiplication on the Xilinx XC4000E FPGA device. Their design employs bit-serial multipliers using Booth encoding. They focused on trade-offs between area and maximum running frequency with parameterized circuit generators. For a specific example of 4×4 matrix multiplication, 954 CLBs are used to achieve the maximum running frequency of 33 MHz.

Amira et al [1] improved the design in [6] using the Xilinx XCV1000E FPGA device. Modified Booth-encoder multiplication was used along with Wallace tree addition. Emphasis was once again on maximizing the running frequency. For a specific example of 4×4 matrix multiplication, 296 CLBs are used to achieve the maximum running frequency of 60 MHz. Area/speed (the number of CLBs divided by the maximum running frequency) was used as a performance metric.

We improve the previous designs in [6] and [1] in terms of the area/speed metric. The area/speed metrics for the designs in [6], [1], and our designs are 14.45, 4.93, and 2.35, respectively. The design area denotes the number of CLBs and its translation is performed in terms of the equivalent amount of logic on different FPGA devices. Details can be found in Section 3.4.

Prasanna and Tsai [8] achieved the theoretical lower bound in latency for matrix multiplication with a design based on a linear array. Their method provides trade-offs between the number of registers and the latency. For performance comparison, we have implemented their design on the same target FPGA device used in the implementation of our designs. The

areas of our designs for $3 \times 3 - 12 \times 12$ matrix multiplications (based on Theorem 1 in Section 2) are smaller by $11\% - 46\%$ compared with the designs based on [8] with the same latency. Our designs (based on Theorem 2 in Section 2) also improve the performance in terms of area×latency (AT) metric by $53.2\% - 69\%$ for matrices of sizes $3 \times 3 - 12 \times 12$. Experiments on larger matrices show that the performance improvements increase with the matrix size.

The rest of the paper is organized as follows. Algorithms and architectures for area and time efficient implementation of matrix multiplication are presented in Section 2. Section 3 describes implementation and compares performance with previous designs. Section 4 concludes the paper.

2 Fast Algorithms for Matrix Multiplication

Compared with the design in [8], our algorithm significantly reduces the number of registers involved in data movement. n^2 registers of 16-bit words and $n^2 + 6n^2/s, (1 \leq s \leq n)$, registers of 8-bit words are involved in the data movement in [8]. Only $4n$ registers of 8-bit words are involved in the data movement in our design.

We present our algorithms and architectures in two theorems and two corollaries. Pseudo-code for cycle-specific data movement and the detailed architectures are also provided. Theorem 1 improves the best known algorithm for matrix multiplication [8] with respect to the number of registers. This leads to optimal time complexity with a leading coefficient of 1 for matrix multiplication on a linear array. It uses only two I/O ports, which makes our design attractive for hosts with limited I/O capability. Theorem 1 is extended for trade-offs between latency and area using block multiplication (Corollary 1).

The second algorithm is developed to exploit future increases in the density of FPGA devices to realize improvements in latency as larger devices become available (Theorem 2). It uses more multipliers and I/O ports. Corollary 1 and Theorem 2 are integrated into Corollary 2. Corollary 2 provides more comprehensive trade-offs between area and latency.

Theorem 1 $n \times n$ *matrix multiplication can be performed in* $n^2 + 2n$ *cycles using* n *PEs with an MACs (multiplier-and-accumulators), 4 registers, and a local memory of* n *words per PE and 2 I/O ports (Figure 3 shows a PE and Figure 1 shows an array connecting the PEs).*

Proof: The algorithm in Figure 2 and the architecture in Figure 1 and Figure 3 are devised to compute $c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$ for all i, j. a_{ik}, b_{kj}, and c_{ij} represent the elements of $n \times n$ matrices A, B, and C. PE_j denotes the j-th PE from the left in Figure 1, $j = 1, 2, .., n$. PE_j computes column j of matrix C, $c_{1j}, c_{2j}, ..., c_{nj}$ stored in local memory. In Phase k, column k of matrix A ($a_{ik}, 1 \leq i \leq n$) and row k of matrix B ($b_{kj}, 1 \leq j \leq n$) traverse the $PE_1, PE_2, PE_3, ..., PE_n$ in order and allow PE_j to update $c'_{ij} = c'_{ij} + a_{ik} \times b_{kj}$ where c'_{ij} represents the intermediate value of c_{ij}. Once b_{kj} arrives at PE_j, a copy of b_{kj} resides in PE_j until $a_{1k}, a_{2k}, a_{3k}, ..., a_{nk}$ pass through PE_j. We observe that the following two essential requirements should be satisfied: 1) Since a_{ik} stays at each PE_j for just one cycle, b_{kj} should arrive at PE_j no later than a_{ik}, for any $i, 1 \leq i \leq n$. 2) Once b_{kj} arrives at PE_j, a copy of b_{kj} should reside in PE_j until a_{nk} arrives. We show how the above two essential requirements for our implementation are satisfied with minimal number of registers. In addition, we evaluate the number of cycles to finish the operation and the amount of local memory per PE.

1) b_{kj} should arrive at PE_j no later than a_{ik}, for any $i, 1 \leq i \leq n$: Matrix B is fed to the lower I/O port of PE_1 (Figure 3) in row major order ($b_{11}, b_{12}, b_{13}, ..., b_{1n}, b_{21}, b_{22}, ...$). Matrix A is fed to the upper I/O port of PE_1 in column major order ($a_{11}, a_{21}, a_{31}, ..., a_{n1}, a_{12}, a_{22}, ...$), n cycles behind the matrix B. For example, a_{11} is fed to the upper I/O port of PE_1 during the same cycle as b_{21} is fed to the lower I/O port of PE_1. The number of cycles needed for b_{kj} to arrive at PE_j is $(k-1)n + 2j - 1$. a_{ik} needs $n + (k-1)n + i + j - 1$ cycles to arrive at PE_j. The requirement is satisfied since $(k-1)n + 2j - 1 \leq n + (k-1)n + i + j - 1$ for all i, j. For example, we show how b_{2n} arrives at PE_n no later than a_{12} for $c'_{1n} = c'_{1n} + a_{12} \times b_{2n}$. b_{2n} needs $(2-1)n + n$ cycles to arrive at PE_1 and needs $(n-1)$ more cycles to move to PE_n, requiring a total of $3n - 1$ cycles. a_{12} needs $n + (2-1)n + 1$ cycles to arrive at PE_1 and $n - 1$ more cycles to move to PE_n, requiring a total of $3n$ cycles.

2) Once b_{kj} arrives at PE_j, a copy of b_{kj} should reside in PE_j until a_{nk} arrives: We show how to minimize the number of registers to store copies of b_{kj} ($k = 1, 2, .., n$) in PE_j, for each j. For this we prove that only two registers (denoted BM and BL in Figure 3) are enough to hold b_{kj} at PE_j (to store two consecutive elements, $b_{(k+1)j}$ and b_{kj}). For example, when b_{34} arrives at PE_4, b_{14} is in BL and b_{24} is in BM. If we can prove that a_{n1} has arrived at PE_4, b_{34} can replace b_{14} in BL. Note that b_{14} is no longer needed in PE_4 after $c'_{n4} = c'_{n4} + a_{n1} \times b_{14}$ has been performed

using a_{n1} which has already arrived at PE_4. In general, b_{kj} is needed until a_{nk} arrives at PE_j in the $\{n + (k-1)n + n + j - 1\}$-th cycle. $b_{(k+2)j}$ arrives at PE_j in the $\{(k+1)n + 2j - 1\}$-th cycle. Since $(k+1)n + 2j - 1 > n + (k-1)n + n + j - 1$ for all j, k, and n, b_{kj} can be replaced when $b_{(k+2)j}$ arrives at PE_j. This proves that PE_j needs only two temporary registers (denoted as BM and BL in Figure 3) to hold b_{kj} $(k = 1, 2, .., n)$.

3) We show that $n^2 + 2n$ cycles are needed to complete the matrix multiplication. The computation finishes one cycle after a_{nn} arrives at PE_n, which is the $\{n + (n-1)n + n + n - 1\}$-th or $\{n^2 + 2n - 1\}$-th cycle.

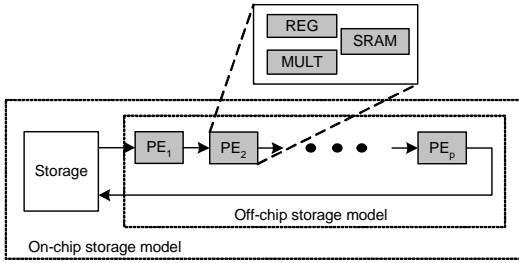4) Local memory of n words is used to store intermediate values for a column of matrix C in each PE.

∎

Figure 1: Family of Architectures

```
For t=1 to n do
For all j, 1≤j≤n, do in parallel
    PE_j shifts data in BU right to PE_{j+1}
    If (BU=b_{kj}), copy it into BM
For t=n+1 to n²+n do
For all j, 1≤j≤n, do in parallel
    PE_j shifts data in A, BU right to PE_{j+1}
    If (BU=b_{kj}), copy it into BL or BM (alternately)
    If (A=a_{ik}), c_{ij}' := c_{ij}' + a_{ik} x b_{kj}
        (b_{kj} is in either BM or BL)
```

Figure 2: Algorithm used in the proof of Theorem 1

Corollary 1 $n \times n$ matrix multiplication can be performed in $(rn^2 + 2r^2n)$ cycles using n/r PEs with 1 MAC (multiplier-and-accumulator), 1 local memory of n/r words and 4 registers per PE and 2 I/O ports where n is divisible by r.

Proof: $n \times n$ matrix multiplication can be decomposed into r^3 $n/r \times n/r$ matrix multiplications. Using Theorem 1 with n replaced by n/r, the result follows. ∎

Corollary 1 provides trade-offs between area and latency. A smaller value of n/r reduces the number
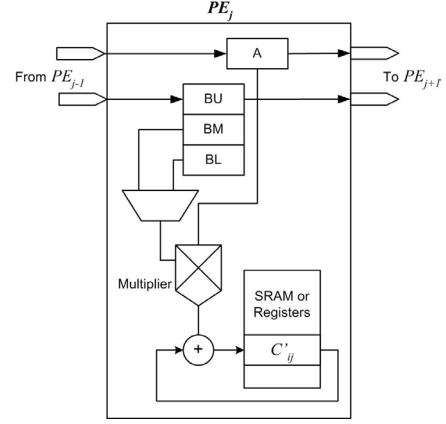

Figure 3: Architecture of PE_j used in the proof of Theorem 1

of PEs resulting in lesser area. However, it increases the number of cycles to complete the matrix multiplication.

Theorem 2 $n \times n$ matrix multiplication can be performed in $(n^2/r + 2n/r)$ cycles using n/r PEs with r^2 MACs (multiplier-and-accumulators), r^2 local memories of n/r words and $4r$ registers per PE and $2r$ I/O ports (Figure 4 shows a PE for r=2) where n is divisible by r.

Proof: The $n \times n$ matrices A, B, and C are divided into r^2 submatrices of size $n/r \times n/r$, respectively. Let A_{xy}, B_{xy}, C_{xy}, $1 \le x, y \le r$ denote the submatrices. Then we have $C_{xy} = \sum_{k=1}^{r} A_{xk} \times B_{ky}$, $1 \le x, y \le r$. The basic idea is to perform $C_{xy} = \sum_{k=1}^{r} A_{xk} \times B_{ky}$ in parallel for all x, y, $1 \le x, y \le r$ using r^2 MACs per PE. $A_{xk} \times B_{ky}$ for each $x, y, 1 \le x, y \le r$ is performed using an MAC per PE as in Theorem 1. However, a direct application of Theorem 1 would require $4r^2$ registers per PE and $2r^2$ I/O ports. Later we will show how to reduce it to $4r$ registers per PE and $2r$ I/O ports. PE_j denotes the j-th PE from the left in Figure 1, $j = 1, 2, .., n/r$. PE_j computes column j of all submatrices C_{xy}, $1 \le x, y \le r$. Each local memory of n/r words in PE_j is used to store intermediate results for column j of a submatrix C_{xy} for any x, y, $1 \le x, y \le r$, requiring a total of r^2 local memories of n/r words per PE. An MAC is used to update column j of each submatrix C_{xy}, $1 \le x, y \le r$, requiring a total of r^2 MACs per PE.

1) For each $1 \le x, y \le r$, $A_{xk} \times B_{ky}$ for all $k, 1 \le k \le r$ can be performed in $(n^2/r + 2n/r)$ cycles using n/r PEs with an MAC and 4 registers per PE and 2 I/O ports: $A_{xk} \times B_{ky}$ for each $1 \le x, k, y \le r$ can be performed in $(n^2/r^2 + 2n/r)$ cycles using n/r PEs
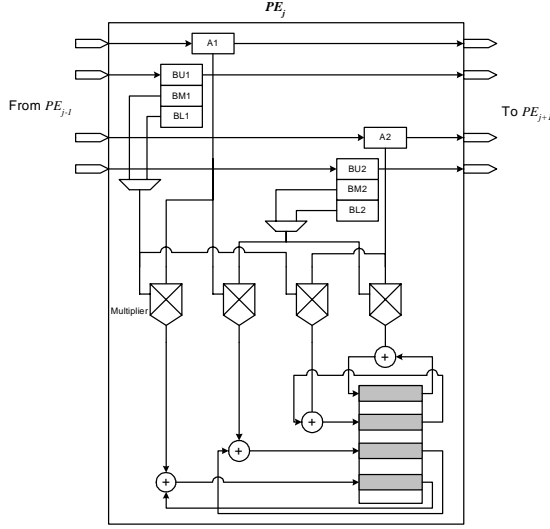
Figure 4: Architecture of PE_j used in the proof of Theorem 2

with an MAC and 4 registers per PE and 2 I/O ports using Theorem 1. Input of last column in submatrix A_{xk} to the array can be overlapped with input of the first row in submatrix $B_{(k+1)y}$ for $k = 1, 2, ..., r - 1$. Using the overlapping, $A_{xk} \times B_{ky}$ for $k = 1, 2, 3..., r$ can be performed in a pipelined fashion, each taking n^2/r^2 cycles. At the start, n/r cycles are needed to input the first row of submatrix B_{1y}. At the end, after the last column of submatrix A_{xr} is input, n/r more cycles are needed to move it through the array of n/r PEs to complete the updates for C_{xy}. This discussion leads to $n/r + r \times n^2/r^2 + n/r = n^2/r + 2n/r$ cycles.

2) $C'_{xy} = C'_{xy} + A_{xk} \times B_{ky}$ for all $1 \leq x, k, y \leq r$ can be performed in $(n^2/r + 2n/r)$ cycles using n/r PEs with r^2 MACs (multiplier-accumulators), r^2 local memories of n/r words per PE, $4r$ *registers* per PE, and $2r$ I/O ports. C'_{xy} is the intermediate results for C_{xy}. In stage k, $1 \leq k \leq r$, $C'_{xy} = C'_{xy} + A_{xk} \times B_{ky}$ is performed in parallel for all $1 \leq x, y \leq r$. I/O ports, $IOA_1, IOA_2, ..., IOA_r$ are used to feed submatrices $A_{1k}, A_{2k}, ..., A_{rk}$ to the array. I/O ports $IOB_1, IOB_2, ..., IOB_r$ are used to feed submatrices $B_{k1}, B_{k2}, ..., B_{kr}$ to the array. An MAC, MAC_{xy}, is used to perform $C'_{xy} = C'_{xy} + A_{xk} \times B_{ky}$ for each $1 \leq x, y \leq r$. A_{xk} can be shared among r MACs, MAC_{xy}, $1 \leq y \leq r$ in each PE while B_{ky} can be shared among r MACs, MAC_{xy}, $1 \leq x \leq r$. This sharing allows us to reduce the number of registers required per PE and the number of I/O ports from $4r^2$ and $2r^2$ to $4r$ and $2r$, respectively. A direct application of Theorem 1 without sharing would use 4 registers per PE and 2 I/O ports to feed each MAC,

requiring $4r^2$ registers per PE and $2r^2$ I/O ports as a total. Column j of a submatrix C'_{xy} is updated by MAC_{xy} and stored in a local memory of size n/r, LM_{xy}, for each $x, y, 1 \leq x, y \leq r$ in PE_j.

3) Figure 4 shows our architecture for $r = 2$ and Figure 5 shows the accompanying algorithm. Let A_{xy}, B_{xy}, and $C_{xy}, 1 \leq x, y \leq 2$ denote submatrices of size $n/2 \times n/2$, respectively. In the first stage, $C'_{xy} = A_{x1} \times B_{1y}$, are performed in parallel for all $1 \leq x, y \leq 2$ by feeding A_{11}, A_{21}, B_{11} and B_{12} into the 4 input ports. In the second stage, $C'_{xy} = C'_{xy} + A_{x2} \times B_{2y}$, is performed in parallel for all $1 \leq x, y \leq 2$. Each stage takes $n^2/4 + n$ cycles from Theorem 1. Since overlapping is possible between the end of the first phase and the start of the second phase, the total number of cycles is $n^2/2 + n$. For other values of r, the proof follows using the same idea. ∎

For t=1 to n/2 do

For all j do in parallel

  PE_j shift words in BU1 & BU2 to the right(to PE_{j+1})*

  If (BU1 = b_{11kj}), copy it into BM1

  If (BU2 = b_{12kj}), copy it into BM2

For t=n/2+1 to (n/2)²+n/2 do

For all j do in parallel

  PE_j shift words in A1,A2,BU1,BU2 to the right(to PE_{j+1})*
  If (BU1 = b_{11kj}), copy it into BM1 after moving the word in BM1 into BL1
  If (BU2 = b_{12kj}), copy it into BM2 after moving the word in BM2 into BL2
  If (A1= a_{11ik}), $REG11_i = REG11_i + a_{11ik} \times b_{11kj}$ $REG12_i = REG12_i + a_{11ik} \times b_{12kj}$
  If (A2= a_{21ik}), $REG21_i = REG21_i + a_{21ik} \times b_{11kj}$ $REG22_i = REG22_i + a_{21ik} \times b_{12kj}$
        (b_{11kj} is in either BM1 or BL1)
        (b_{12kj} is in either BM2 or BL2)

*Matrix B11 and B12 enters the third and fourth I/O port of
      PE_1 in row major order

*Matrix A enters the first and second port of PE1 in column major
      order n/2 cycles behind Matrix B

Figure 5: Algorithm used in the proof of Theorem 2

With emphasis on the number of MACs used, Theorem 2 and Corollary 1 can be combined into Corollary 2.

Corollary 2 $n \times n$ *matrix multiplication can be performed in* $(n^3/m + 2n^2/m)$ *cycles using* $\min(m, n^2/m)$ *PEs with* $\max(1, m^2/n^2)$ *MACs (multiplier-and-accumulators),* $\max(1, m^2/n^2)$ *local memories of* $\min(m, n^2/m)$ *words,* $\min(4, 4m/n)$ *registers per PE and* $\max(2, 2m/n)$ *I/O ports where* n^2 *is divisible by* m *and* $1 \leq m \leq n^2$.

Proof: For $1 \leq m \leq n$, the proof follows from Corollary 1 by setting $n/r = m$. For $n \leq m \leq n^2$, the proof follows from Theorem 2 by setting $nr = m$. ∎

A smaller value for m reduces the number of MACs, number of registers, and number of I/O ports resulting in lesser area. But it increases the number of cycles. Corollary 2 provides trade-offs between latency and area for $1 \leq m \leq n^2$. Corollary 2 provides more comprehensive trade-offs between area and latency than Corollary 1 and Theorem 2. Note that Corollary 1 and Theorem 2 provide trade-offs between latency and area for $1 \leq m \leq n$ and $n \leq m \leq n^2$, respectively.

3 FPGA Implementation

Our approach is to use a domain-specific modeling technique proposed in [4]. The model is applicable only to the design domain spanned by the family of algorithms and the architectures to be considered. The family represents a set of algorithm-architecture pairs which share a common structure and similar data movement. The domain is a set of point designs resulting from unique combinations of algorithm and architecture level changes. The abstraction is independent of the commonly used levels such as gate, register, and system level. For example, if the number of multipliers and the number of registers change values in a domain, a domain-specific model is built using them as key parameters. The detailed approach can be found in [4].

We choose a Xilinx Virtex-II FPGA device to evaluate our designs. We estimate the area by deriving domain-specific area functions. Each domain represents a set of architectures and algorithms spanned by algorithm level changes. We identify key components and the area for each component to make an approximate estimate of the total area required. Performance comparison with previous designs are made in terms of area/speed metric and area×latency (AT) metric.

3.1 Identify Key Components, Parameters and Domain

The family of architectures in Figure 1 and the PEs in Figures 3 and 4 are used for our algorithms. We identify registers of 8-bit words, MACs (multiply-and-accumulates), SRAMs, BSRAMs [9] as key components. SRAM is a CLB-based local memory and BSRAM is a built-in on-chip storage. In the off-chip model, the storage for input matrices is assumed to be outside the FPGA. I/O ports are used for data

access. The on-chip model uses BSRAM (Block Select RAM) in the Xilinx Virtex-II FPGA devices as on-chip storage.

We build four domains for Theorem 1, Corollary 1, Theorem 2 and Corollary 2. Only one parameter, n, is used for Theorem 1. Two parameters, n and r, are used for Corollary 1 and Theorem 2. n denotes the size of matrices. r is introduced for block multiplication using submatrices of size n/r. Two parameters n, m are used for Corollary 2. n denotes the size of matrices and m denotes the number of MACs. Our algorithms ensure that the number of all the key components depends on only two parameters.

Table 1 lists the key parameters and the number of key components represented by the parameters for each domain. For given matrices of size $n \times n$, the space spanned by the parameters forms a domain over which trade-offs between area and latency are made. Each unique combination of parameter values form a point design. For example, in the domain for Corollary 1 off-chip model, $n = 16, r = 4$ represents a point design where 4 PEs are used for 16×16 matrix multiplication. Each PE has 4 registers, 1 MAC (multiplier-and-accumulator) and 1 SRAM and there are 2 I/O ports connecting the array of PEs to off-chip memory.

3.2 Estimation of Area for Key Components

We estimate the area needed for implementing k instances of each key component. For simplicity, we assume that each component consumes same amount of area regardless of its location. As an example, Figure 6 shows the estimation of the area needed for storing k 16-bit words on Xilinx XC2V1500 device with 166MHz clock.. Each sample point represents a low level simulation to estimate the area. The curve for registers can be approximated by a linear function while the curve for SRAMs can be approximated by a step function. Since SRAMs can have only multiples of 16×2^i words, the curve for the area of SRAMs can be approximated as a step function. 1 CLB in the XC2V1500 FPGA device has 4 slices. For other components, area can be similarly characterized. Components with non-linear characteristic can be characterized by tables. Let $A_{R8}(k), A_{R16}(k), A_{MULT}(k), A_{SRAM}(k), A_{BSRAM}(k)$, and $A_{IO}(k)$ denote area estimation for 8-bit registers, 16-bit registers, 8-bit multipliers, SRAMs,BSRAMs and I/O ports.

Table 1: Key parameters and key components for the domains (note: BSRAMs and I/O ports are used for on-chip model and off-chip model, respectively)

Domain	Parameters	No. of PEs	Reg/PE	MACs/PE	BSRAMs	I/O ports	SRAMs/PE
Theorem 1	n	n	4	1	$2n^2$	2	1 of n words
Corollary 1	n, r	n/r	4	1	$2n^2$	2	1 of n/r words
Theorem 2	n, r	n/r	$4r$	r^2	$2n^2$	$2r$	r^2 of n/r words
Corollary 2	n, m	$\min(m, n^2/m)$	$\max(4, 4m/n)$	$\max(1, m^2/n^2)$	$2n^2$	$\max(2, 2m/n)$	$\max(1, m^2/n^2)$ of $\min(m, n^2/m)$ words

Note: * n is divisible by r, $(1 \leq m \leq n^2)$. * n^2 is divisible by m.

  Minimum unit size of BSRAMs and SRAMs are 1024 and 16.

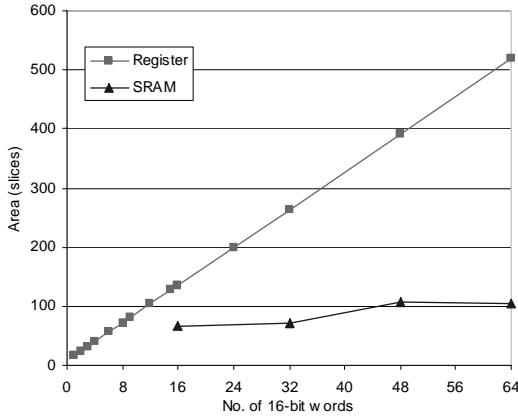 For example, $\lceil 2n^2/1024 \rceil$ BSRAMs are needed to store $2n^2$ words.

Figure 6: Comparison of areas needed to store 16-bit words (Xilinx XC2V1500)

3.3 Functions to Estimate Area and Latency

Functions to represent area and latency are derived in the domains for Corollary 1, Theorem 2, and Corollary 2. The area is obtained by $A = \sum_i A_i(k_i)$ where $A_i(k_i)$ denotes the area estimation for k_i components of type i in the design. The values of k_i depend on the parameters for each domain (refer to Table 1). For example, in the domain for Theorem 2, k_i is determined by the specific values assigned to n, r. If $n = 16, r = 4$, then $k_{R8} = 16$ which implies that 16 8-bit registers are used. Note that assigning specific values to the parameters not only determines a point design in the domain but also determines the number of components for each type. The latency is determined by the algorithm and is represented using the parameters chosen for each domain.

Table 2 illustrates the functions to represent area and latency for the domain of Corollary 1. Functions for other domains are obtained in the same way. For

the off-chip model, I/O ports are used to fetch elements from outside the FPGA. In the on-chip model, BSRAMs of 16-bit 1024 words are used for on-chip storage of input matrices.

3.4 Performance Comparison

Our designs improve the previous designs in [6] and [1] in terms of the area/speed metric (Refer to Table 3). Since each CLB has different amount of logic on different FPGA devices used in the designs in [6] and [1], translation should be made in the number of CLBs used for fair comparison. 954 CLBs of the Xilinx XC4000E FPGA device used in the design of [6] for 4×4 matrix multiplication can be translated into 477 CLBs of the Xilinx XCV1000E FPGA device used in [1]. 140 CLBs of the Xilinx XC2V1500 used in our design of 4×4 matrix multiplication to achieve 166 MHz can be translated into 280 CLBs of the Xilinx XCV1000E FPGA device used in [1]. In addition, our designs use 4 dedicated multipliers which can be translated to 110 CLBs. After translation, the area/speed metrics for the designs in [6] and [1] are 14.45 and 4.93, respectively. The area/speed metric for our designs is 2.35.

We also implemented the best known linear array based design by Prasanna and Tsai [8] on the same FPGA device we used for our design. The implementation occupies 155 CLBs and the area/speed metric adjusted as indicated above is $(155 \times 2 + 110)/166 = 2.53$. It is 7% higher than our design. The gap widens as the size of the matrices grows. For example, 12×12 matrix multiplication, our design uses $(417 \times 2 + 110 \times 3)/166 = 7.0$. The design based on [8] uses $(921 \times 2 + 110 \times 3)/166 = 13$. Our design improves the design in [8] by 46% in terms of the metric.

Since the latency of the design in [1] is not clearly stated, we compare the latency of our design against that of the design in [6] for 4×4 matrix multipli-

Table 2: Functions to represent area and latency for the domain of Corollary 1

Functions		For the domain of Corollary 1
Area (slices)	on-chip	$\frac{n}{r}A_{MULT}+\frac{n}{r}\lceil\frac{n}{r}/16\rceil A_{SRAM}+4\frac{n}{r}A_{R8}$ and $\lceil 2n^2/1024\rceil$ BSRAMs
	off-chip	$\frac{n}{r}A_{MULT}+\frac{n}{r}\lceil\frac{n}{r}/16\rceil A_{SRAM}+4\frac{n}{r}A_{R8}$ and 2 8-bit I/O ports
Latency (cycles)		$(r^3)(\frac{n^2}{r^2}+\frac{2n}{r}+1)$

Table 3: Area/speed comparison of previous designs against our designs for 4×4 matrix multiplication

Design	FPGA	Area (CLBs)	Area (equivalent)	Speed (MHz)	Area/Speed	Latency (μs)
Mencer et al [6]	XC4000E	954	477	33	14.45	0.57
Amira [1]	XCV1000E	296	296	60	4.93	n/a
Proposed (Theorem 1)	XC2V1500	140	390	166	2.35	0.15
Prasanna and Tsai [8]	XC2V1500	155	420	166	2.53	0.15

cation. The latency of the design in [6] is 19 cycles $\times$ 1/33MHz $= 0.57\mu s$. Our design takes 25 cycles $\times$ 1/166MHz$= 0.15\mu s$. One cycle is added for flushing the 2-stage pipeline multipliers used in our design($4^2+2\times 4+1=25$). The equivalent area used by our design is smaller by 18% compared with that of the design in [6].

For sizes of the matrices other than 4×4, no data is provided in [6] and [1]. Table 4 compares area and latency of our designs (based on Theorem 1 and Theorem 2) against the design in [8] for various sizes of matrices. The area of our designs based on Theorem 1 is smaller by $11\%-46\%$ compared with the designs based on [8] with the same latency. The gap widens as the size of the matrices grows. Our designs based on Theorem 2 reduce the latency with modest increase in area as compared with the designs based on [8] and Theorem 1. Comparison of performance based on the AT (area$\times$latency) metric shows that designs based on Theorem 2 are better than the designs based on [8] by $53.2-69\%$ for matrices of sizes $3\times 3-12\times 12$. Figure 7 shows the trade-off curve between latency and area. Experiments on larger matrices show that the reduction becomes larger with increase in the sizes of matrices. Details are omitted due to space limitations.

4 Conclusions

New algorithms and architectures were developed for matrix multiplication to provide improved trade-offs between area and latency as compared with the state-of-the-art FPGA-based designs. A function to represent the impact of algorithm change on the total area is derived to enable the designer to understand the trade-offs before time consuming low level sim-
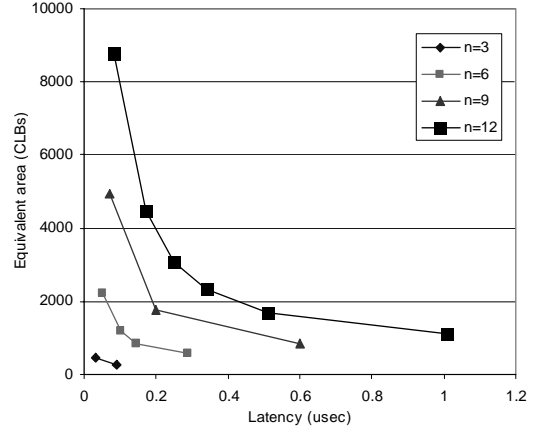

Figure 7: Trade-offs between latency and area for matrices of chosen sizes

ulation. Low level simulation using Xilinx ISE 4.1i and Mentor Graphics ModelSim 5.5e and recent Xilinx XC2V1500 as a target FPGA were performed to evaluate the chosen designs.

References

[1] A. Amira, A. Bouridane, and P. Milligan, "Accelerating Matrix Product on Reconfigurable Hardware for Signal Processing," *Field-Programmable Logic and Applications (FPL)*, pp. 101-111, 2001.

[2] A. Bogliolo, L. Benini, and G. Micheli, "Regression-based RTL Power Modeling," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 5, no. 3, 2000.

Table 4: Performance comparison of various designs against the Xilinx design.

Design	Metric	Matrix size							
		3×3		6×6		9×9		12×12	
Proposed (Theorem 1)	latency		0.09		0.29		0.60		1.01
	area		254		558		837		1,116
	AT		22.9		161.8		502.2		1127.2
Proposed (Theorem 2)	latency	$r=3$	0.03	$r=2$	0.145	$r=3$	0.2	$r=2$	0.51
	area		444		843		1767		1686
	AT		13.3		122.2		353.4		859.9
	latency		n/a	$r=3$	0.1	$r=9$	0.07	$r=3$	0.34
	area				1178		4957		2356
	AT				117.8		347.0		801.0
	latency		n/a	$r=6$	0.05		n/a	$r=4$	0.25
	area				2233				3051
	AT				111.7				762.8
	latency		n/a		n/a		n/a	$r=6$	0.17
	area								4466
	AT								759.2
	latency		n/a		n/a		n/a	$r=12$	0.08
	area								8761
	AT								700.9
Design based on [8]	latency		0.09		0.29		0.60		1.01
	area		326		795		1,427		2,172
	AT		29.3		238.5		884.7		2258.9

Note: The unit of latency is μ sec. The area is the number of equivalent CLBs.
AT is the area×latency metric.

[3] B. Bass, "A Low-Power, High-Performance, 1024-Point FFT Processor," *IEEE Journal of Solid-State Circuits*, Vol. 34, no. 3, pp. 380-387, 1999.

[4] S. Choi, J. Jang, S. Mohanty, and V. K. Prasanna, "Domain-Specific Modeling for Rapid System-Wide Energy Estimation of Reconfigurable Architectures," *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2002.

[5] W. Luk, P. Andreou, A. Derbyshire, F. Dupont-De-Dinechin, J. Rice, N. Shirazi, and D. Siganos, "A Reconfigurable Engine for Real-time Video Processing," *Field-Programmable Logic and Applications (FPL)*, pp. 169-178, 1998.

[6] O. Mencer, M. Morf, and M. Flynn, "PAM-Blox: High Performance FPGA Design for Adaptive Computing," *Proc. PDCS*, 2001.

[7] P. Master and P. M. Athanas, "Reconfigurable Computing Offers Options For 3G," *Wireless Systems Design,* pp. 20-23, 1999.

[8] V. K. Prasanna Kumar and Y. Tsai, "On Synthesizing Optimal Family of Linear Systolic Arrays for Matrix Multiplication," *IEEE Transactions on Computers*, Vol. 40, no. 6, 1991.

[9] Xilinx Application Note, Virtex-II Series and Xilinx ISE 4.1i Design Environment, http://www.xilinx.com, 2001.

[10] http://www.xilinx.com.