# Generalized Matrix Multiplication (GEMM)

Matjaz Zupancic Muc
University of Ljubljana, Faculty of Computer and Information Science
Večna pot 113, SI-1000 Ljubljana, Slovenia
Email: mm1706@student.uni-lj.si

## I. Introduction

IN this seminar we implement a General Matrix Multiply (GEMM) operation RTL block. GEMM is a commonly used in linear algebra, machine learning and statistics. We implement a basic version and optimize it using Vivado High level synthesis (HLS). C Synthesis is targeting *NexysA7 100t* board, part: *xc7a100tcsg324-1*(1). Source code and synthesis results are available at https://github.com/Matjaz12/GEMM-FPGA.git

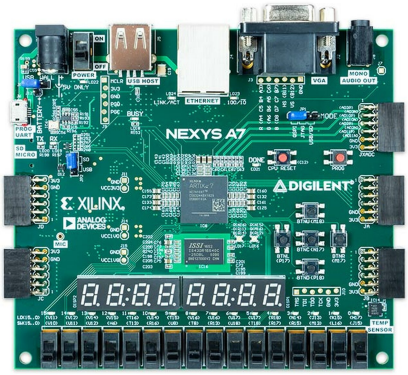


Fig. 1: NexysA7 100t Board

August 26, 2015

## II. Methods

### A. Background

General matrix multiplication (GEMM) is defined as follows (note that in these seminar we use square matrices):

$$\underset{N\times N}{C} = \underset{1\times 1}{\alpha}\ \underset{N\times N}{A} \times \underset{N\times N}{B} + \underset{1\times 1}{\beta}\ \underset{N\times N}{C} \tag{1}$$

where $\alpha$ and $\beta$ are scalars, matrices $A$ and $B$ are multiplied. Matrix $C$ on the right hand side of the equation is the current matrix $C$.

### B. Baseline implementation

Basic implementation of *gemm* consists of three sub functions: *matrix_mult*, *matrix_add* and *scalar_mult* (Fig 2 displays the function hierarchy). Matrix multiplication is implemented using the standard algorithm with complexity $\mathcal{O}(N^3)$. We use a matrices A, B, C of size $16 \times 16$ to compare different methods, best method is than benchmarked on larger matrices of size $32 \times 32$ and $64 \times 64$.

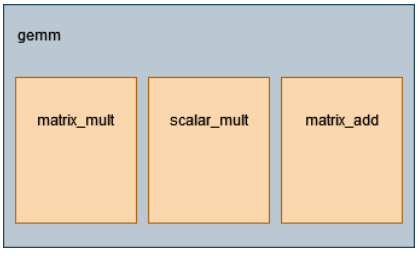


Fig. 2: RTL Block, function hierarchy

Fig. 3 in appendix shows synthesis results of basic implementation (with no parallelism). We can see that the clock frequency below 20.00 ns has been met. The top function takes 9832 clock cycles (interval) to compute all the outputs. New inputs can be applied after 9832 clock cycles (latency). The biggest contributor of latency is the *matrix_mult* part of *gemm* with a latency of 8737 (about 88 % of total latency). *Matrix_add* and *scalar_mult* each contribute 545 clock cycles of latency (about 11 % in total). Baseline implementation utilizes roughly 2 % of hardware resources, mainly so called *DSP48E* modules, which are multiply and add operators.

### C. Pipeline loops

To pipeline the loops we use the **pipeline** directive. We could place the directive it in either first (row) loop, second (column) loop or inner (dot product) loop with the goal of achieving different resource-throughput tradeoffs. In these section we apply a **pipeline** directive to the second (column) loop in *matrix_mult*. This results in fully unrolled inner most loop (dot product loop). Also the row loop is flattened. We expect the resulting circuit to include $N$ multiply add operations. **Pipeline** directive is also used on second (column) loop in *matrix_add* and *scalar_mult* functions.

Fig. 4 in appendix shows that the latency and interval dropped down to 2572 clock cylces with a small increase in resource utilization (*DSP48E* utilization increases to 3 %). The *matrix_mult* operation still contributes to the most latency. We can also see that target initiation interval (II) of 1 was not achieved. We achieve II of 8. What we can achieve greatly depends on size of matrix (in case of 3x3 matrix we can get II of 2). The latency of add and *matrix_add* and *scalar_mult* operations also about halved.

### D. Reshaping arrays

In order to increase number of accesses that can be performed on each memory we can use array partitioning. This increases the number of array elements that can be read each clock cycle. We will use **array_reshape** directive to perform array partitioning. This directive partitions the address space of the memory in to separate memory blocks just like **array_partition**, but it also recombines the memory blocks into a single memory. According to [KMN18] **array_partition** directive makes each individual memory smaller, which can sometimes result in inefficient memory usage. The **array_reshape** directive results in larger memory blocks, which can sometimes be mapped more efficiently into primitive FPGA resource. Both arrays used in matrix multiplication are partitioned along their respective $k$ dimension: For matrix $A$, this is dimension 2 since we iterate over columns and for matrix $B$ this is dimension 1 since, we iterate over rows. Note that we keep **pipeline** directives on column loop in all three functions.

Fig. 5 in appendix shows that the latency and interval dropped down to 780. The total latency decreased by a factor of $2572/780 = 3.29$. Surprisingly *matrix_mult* now contributes only 258 clock cycles of latency which is just as much as *matrix_add* and *scalar_mult*. Furthermore we can see that the initiation interval (II) of 1 has been reached. The block now utilizes $21\%$ of available *DSP48E* resources.

### E. Pipeline entire function

In these section we attempt to pipeline the entire function. This should results in highest performance since all loops contained in the function are unrolled. Fig. 6 shows that *matrix_mult* has a latency of 129 and interval of 128. Taking a look at hardware utilization, we realise that 4862 % of *DSP48E*s are used. This means that when using a matrix of size $(16 \times 16)$ there is not enough *DSP* resources to implement that many multiplications every clock cycle and/or enough external bandwidth to get get data on and off the chip.

Using the matrix of size $16 \times 16$ we ran out of available resources in particular there is not enough *DSP* resources to implement that many multiplications every clock cycle and or enough external bandwidth to get get data on and off the chip.

### F. Method evaluation

So far we have used a matrix of size $16 \times 16$, we have shown that pipelining the whole function is infeasible. In this section we benchmark the best method (column pipeline and array reshape) on matrices of three different sizes as shown in Table I. We can see that the number of *DSP48E*s and LUTs used in the RTL block roughly doubles as matrix doubles in sizes. The latency and interval increases roughly by a factor of 4 each time. The clock frequency seems to drop only when matrix size changes from $16 \times 16$ to $32 \times 32$.

| Matrix dimension | Clock frequency (ns) | Latency (ns) | Interval (ns) | BRAM_18K (%) | DSP48E (%) | FF (%) | LUT (%) |
|---|---|---|---|---|---|---|---|
| $16 \times 16$ | 16.772 | 780 | 780 | 0 | 21 | 0 | 2 |
| $32 \times 32$ | 14.984 | 3085 | 3085 | 0 | 41 | 0 | 4 |
| $64 \times 64$ | 14.984 | 12301 | 12301 | 2 | 81 | 1 | 7 |

TABLE I: Performance and resource utilization for various matrix sizes

## III. Conclusion

We implemented and analysed the basic implementation of GEMM. We analysed the use of **pipeline** directive on different sections of loops. We also used array partitioning in particular **array_reshape** directive, to increase number of possible reads in a clock cycle. Finally we benchmark the best implementation on matrices of different size. We show that pipelining the whole function does yield best performance but is not feasible when dealing with matrices of size $(N, N) \geq (16 \times 16)$. Pipelining all column loops produces second best results, we show that implementation can compute a GEMM operation on a matrix of size $64 \times 64$ while not using all the available resources. Array partitioning has shown to be effective, since it decreased the total latency of *gemm* by a factor of 3.3. GEMM operation could be further improved using block matrix multiplication. We could also perform some function entirely in parallel (e.g $\alpha A \times B$ and $\beta C$ operations can be done in parallel).

## References

[KMN18] Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer. Parallel programming for fpgas. *arXiv preprint arXiv:1805.03648*, 2018.

# APPENDIX A

## Performance Estimates

- **Timing (ns)**
  - **Summary**

| Clock | Target | Estimated | Uncertainty |
|---|---|---|---|
| ap_clk | 20.00 | 14.984 | 2.50 |

- **Latency (clock cycles)**
  - **Summary**

| Latency | | Interval | | Type |
|---|---|---|---|---|
| min | max | min | max | |
| 9832 | 9832 | 9832 | 9832 | none |

  - **Detail**
    - **Instance**

| Instance | Module | Latency | | Interval | | Type |
|---|---|---|---|---|---|---|
| | | min | max | min | max | |
| grp_matrix_mult_fu_42 | matrix_mult | 8737 | 8737 | 8737 | 8737 | none |
| grp_matrix_add_fu_52 | matrix_add | 545 | 545 | 545 | 545 | none |
| grp_scalar_mult_fu_59 | scalar_mult | 545 | 545 | 545 | 545 | none |

    - **Loop**

N/A

## Utilization Estimates

- **Summary**

| Name | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|
| DSP | - | - | - | - |
| Expression | - | - | 0 | 2 |
| FIFO | - | - | - | - |
| Instance | - | 6 | 159 | 576 |
| Memory | 1 | - | 0 | 0 |
| Multiplexer | - | - | - | 200 |
| Register | - | - | 9 | - |
| Total | 1 | 6 | 168 | 778 |
| Available | 270 | 240 | 126800 | 63400 |
| Utilization (%) | ~0 | 2 | ~0 | 1 |

- **Detail**
  - **Instance**

| Instance | Module | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|---|
| grp_matrix_add_fu_52 | matrix_add | 0 | 0 | 37 | 168 |
| grp_matrix_mult_fu_42 | matrix_mult | 0 | 3 | 85 | 258 |
| grp_scalar_mult_fu_59 | scalar_mult | 0 | 3 | 37 | 150 |
| Total | | 3 | 0 | 6 | 159 | 576 |

Fig. 3: Baseline results.

## Performance Estimates

- **Timing (ns)**
  - **Summary**

| Clock | Target | Estimated | Uncertainty |
|---|---|---|---|
| ap_clk | 20.00 | 16.232 | 2.50 |

- **Latency (clock cycles)**
  - **Summary**

| Latency | | Interval | | Type |
|---|---|---|---|---|
| min | max | min | max | |
| 2572 | 2572 | 2572 | 2572 | none |

  - **Detail**
    - **Instance**

| Instance | Module | Latency | | Interval | | Type |
|---|---|---|---|---|---|---|
| | | min | max | min | max | |
| grp_matrix_mult_fu_42 | matrix_mult | 2051 | 2051 | 2051 | 2051 | none |
| grp_matrix_add_fu_52 | matrix_add | 258 | 258 | 258 | 258 | none |
| grp_scalar_mult_fu_59 | scalar_mult | 258 | 258 | 258 | 258 | none |

    - **Loop**

N/A

## Utilization Estimates

- **Summary**

| Name | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|
| DSP | - | - | - | - |
| Expression | - | - | 0 | 2 |
| FIFO | - | - | - | - |
| Instance | - | 9 | 389 | 1387 |
| Memory | 2 | - | 0 | 0 |
| Multiplexer | - | - | - | 227 |
| Register | - | - | 9 | - |
| Total | 2 | 9 | 398 | 1616 |
| Available | 270 | 240 | 126800 | 63400 |
| Utilization (%) | ~0 | 3 | ~0 | 2 |

- **Detail**
  - **Instance**

| Instance | Module | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|---|
| grp_matrix_add_fu_52 | matrix_add | 0 | 0 | 38 | 212 |
| grp_matrix_mult_fu_42 | matrix_mult | 0 | 6 | 313 | 981 |
| grp_scalar_mult_fu_59 | scalar_mult | 0 | 3 | 38 | 194 |
| Total | | 3 | 0 | 9 | 389 | 1387 |

Fig. 4: Pipeline all column loops.

## Performance Estimates

- **Timing (ns)**
  - **Summary**

| Clock | Target | Estimated | Uncertainty |
|---|---|---|---|
| ap_clk | 20.00 | 16.772 | 2.50 |

- **Latency (clock cycles)**
  - **Summary**

| Latency | | Interval | | Type |
|---|---|---|---|---|
| min | max | min | max | |
| 780 | 780 | 780 | 780 | none |

  - **Detail**
    - **Instance**

| Instance | Module | Latency | | Interval | | Type |
|---|---|---|---|---|---|---|
| | | min | max | min | max | |
| grp_matrix_mult_fu_42 | matrix_mult | 259 | 259 | 259 | 259 | none |
| grp_matrix_add_fu_52 | matrix_add | 258 | 258 | 258 | 258 | none |
| grp_scalar_mult_fu_59 | scalar_mult | 258 | 258 | 258 | 258 | none |

    - **Loop**

N/A

## Utilization Estimates

- **Summary**

| Name | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|
| DSP | - | - | - | - |
| Expression | - | - | 0 | 2 |
| FIFO | - | - | - | - |
| Instance | - | 51 | 475 | 1433 |
| Memory | 2 | - | 0 | 0 |
| Multiplexer | - | - | - | 227 |
| Register | - | - | 9 | - |
| Total | 2 | 51 | 484 | 1662 |
| Available | 270 | 240 | 126800 | 63400 |
| Utilization (%) | ~0 | 21 | ~0 | 2 |

- **Detail**
  - **Instance**

| Instance | Module | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|---|
| grp_matrix_add_fu_52 | matrix_add | 0 | 0 | 38 | 212 |
| grp_matrix_mult_fu_42 | matrix_mult | 0 | 48 | 399 | 1027 |
| grp_scalar_mult_fu_59 | scalar_mult | 0 | 3 | 38 | 194 |
| Total | | 3 | 0 | 51 | 475 | 1433 |

Fig. 5: Pipeline all column loops and reshape the arrays.

## Performance Estimates

- **Timing (ns)**
  - **Summary**

| Clock | Target | Estimated | Uncertainty |
|---|---|---|---|
| ap_clk | 20.00 | 16.772 | 2.50 |

- **Latency (clock cycles)**
  - **Summary**

| Latency | | Interval | | Type |
|---|---|---|---|---|
| min | max | min | max | |
| 641 | 641 | 641 | 641 | none |

  - **Detail**
    - **Instance**

| Instance | Module | Latency | | Interval | | Type |
|---|---|---|---|---|---|---|
| | | min | max | min | max | |
| grp_matrix_mult_fu_42 | matrix_mult | 129 | 129 | 128 | 128 | function |
| grp_matrix_add_fu_52 | matrix_add | 255 | 255 | 256 | 256 | function |
| grp_scalar_mult_fu_59 | scalar_mult | 255 | 255 | 256 | 256 | function |
| grp_scalar_mult_fu_66 | scalar_mult | 255 | 255 | 256 | 256 | function |

    - **Loop**

N/A

## Utilization Estimates

- **Summary**

| Name | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|
| DSP | - | - | - | - |
| Expression | - | - | - | - |
| FIFO | - | - | - | - |
| Instance | - | 11670 | 107490 | 226972 |
| Memory | 2 | - | 0 | 0 |
| Multiplexer | - | - | - | 2457 |
| Register | - | - | 645 | - |
| Total | 2 | 11670 | 108135 | 229429 |
| Available | 270 | 240 | 126800 | 63400 |
| Utilization (%) | ~0 | 4862 | 85 | 361 |

- **Detail**
  - **Instance**

| Instance | Module | BRAM_18K | DSP48E | FF | LUT |
|---|---|---|---|---|---|
| grp_matrix_add_fu_52 | matrix_add | 0 | 0 | 8384 | 5849 |
| grp_matrix_mult_fu_42 | matrix_mult | 0 | 11658 | 82210 | 211761 |
| grp_scalar_mult_fu_59 | scalar_mult | 0 | 6 | 8448 | 4681 |
| grp_scalar_mult_fu_66 | scalar_mult | 0 | 6 | 8448 | 4681 |
| Total | | 4 | 0 | 11670 | 107490 | 226972 |

Fig. 6: Pipeline entire function.