IT 19 013
Examensarbete 30 hp
Maj 2019

Design and Acceleration of Linear Integer System Solver on Programmable SoC

Jagadeep Ram Gandhi

Institutionen för informationsteknologi
Department of Information Technology

Abstract

Design and Acceleration of Linear Integer System Solver on Programmable SoC

Jagadeep Ram Gandhi

A simple solver for linear integer systems is designed and accelerated on a CycloneV SoC chip that contains Cortex-A based MCU, programmable FPGA, and inter-connect bridges. The solver is designed based on the Gaussian Elimination method, where a system coefficient matrix is converted to a Row-Echelon matrix and performing Back-Substitution to solve system variables. The matrix conversion is implemented in the FPGA with serial and parallel architectures, where the processing of two equations is performed using single and multiple reducer modules. In comparison with the software-based solver, the solvers with hardware-based matrix conversion modules are faster by at least 75% despite very high MCU clock and data transfer overhead between the subsystems.

Handledare: Philipp Rümmer
Ämnesgranskare: Stefanos Kaxiras
Examinator: Philipp Rümmer
IT 19 013
Tryckt av: Reprocentralen ITC

Acknowledgments

This master thesis report is submitted as a part of the requirement for the degree of Master of Science in Embedded Systems at the Department of Information Technology, Uppsala University, Uppsala, Sweden.

I sincerely thank the thesis supervisor Dr. Philipp Rümmer, Associate Professor at the Department of Information Technology and Programme Director for Master's programme in Embedded System at Uppsala University for guiding throughout the thesis time frame. A special thanks for suggesting this thesis that may lead to various exciting work in the future.

I want to thank Dr. Stefanos Kaxiras, Professor at the Department of Information Technology, Division of Computer Systems at Uppsala University to be the reviewer and his valuable guidance.

I would also like to thank the thesis coordinator Dr. Justin Pearson, Senior lecturer at the Department of Information Technology, Division of Computing Science at Uppsala University for organizing the thesis presentation.

Finally, a big thanks to all my friends and family members for their support during the tenure of the master program.

Contents

List of Figures **iv**

List of Tables **vi**

Listings **vii**

1 Introduction **1**
- 1.1 Thesis Description . . . 2
- 1.2 Thesis Tasks and Goal . . . 3
- 1.3 Structure of the report . . . 3

2 Background **4**
- 2.1 Linear Arithmetic Theories and Solvers . . . 4
  - 2.1.1 Simplex Algorithm for SMT Solvers . . . 4
  - 2.1.2 Gaussian Elimination . . . 8
  - 2.1.3 Divisibility Test . . . 9
- 2.2 DE1-SoC Board . . . 10
  - 2.2.1 FPGA and MCU . . . 11
  - 2.2.2 Interconnect Bridge . . . 12
- 2.3 Similar Work . . . 13
  - 2.3.1 FPGA-Based Acceleration for Boolean Satisfiability . . . 13
  - 2.3.2 FPGA-Based Acceleration of a Linear System Solver . . . 13

3 Algorithm Design **15**
- 3.1 Simplification of Linear Arithmetic Problem . . . 15
- 3.2 Design of the Algorithm for Linear Integer Equations . . . 16
- 3.3 Conversion of Coefficient Matrix to Row-Echelon Form . . . 18
  - 3.3.1 Lead Variable Reduction (LVR) . . . 18
  - 3.3.2 Equation Swap . . . 21
  - 3.3.3 Creation of Upper Triangular Matrix . . . 22
- 3.4 Solving Example Problems . . . 24

4 Algorithm Implementation on SoC **27**
- 4.1 Designing the Interconnect Bridges . . . 27
  - 4.1.1 Data transfer from MCU to FPGA . . . 28
  - 4.1.2 Data transfer from FPGA to MCU . . . 29
- 4.2 Implementation of FPGA components . . . 32
  - 4.2.1 FIFO Read Module . . . 32
  - 4.2.2 Variable Reducer Module . . . 34
  - 4.2.3 Equation Swap Module . . . 35
  - 4.2.4 Matrix Converter Module . . . 36
  - 4.2.5 Serial Converter . . . 37
  - 4.2.6 Parallel Converter . . . 38
  - 4.2.7 SRAM Write Module . . . 39

4.3 Top Level Module 41
4.4 SoC Solver Implementation 42

5 Results and Verification 44
5.1 FPGA Compilation and Synthesis Results 44
5.2 SoC and Software Solvers 45
5.3 Design Verification 46
5.4 Measurement Application 52
5.5 Comparison of Execution Time 55
5.5.1 Comparison of Solvers 55
5.5.2 Comparison of Row Lengths 57
5.5.3 Comparison of Column Lengths 58
5.5.4 Comparison of Integer Ranges 59

6 Conclusion and Future Work 61
6.1 Conclusion 61
6.2 Future Work 61

Bibliography 62

7 Appendix A: Instructions to Run the Solvers on a DE1-SoC Board 64
7.1 Hardware Required 64
7.2 Software Installations 64
7.3 Compilation of MCU software 65
7.4 Setting up Connectivity and Files Transfer 65
7.5 Synthesis and Program FPGA designs 66
7.6 Running Solver Applications 67
7.7 Analyzing Result Files 68

List of Figures

1 SoC FPGA Architecture [2] . . . 2
2 DE1-SoC development board copied from [6] . . . 10
3 Cyclone V SoC architecture copied from [6] . . . 11
4 AXI Bridge copied from [3] . . . 12
5 hps_0 HPS component in the Qsys system . . . 13
6 PLL and HPS components in the bridge . . . 27
7 Bridge clock IP . . . 28
8 PIO IP properties . . . 28
9 FIFO IP properties . . . 29
10 FIFO, PIO and PLL module in the bridge . . . 29
11 SRAM IP properties . . . 30
12 SRAM and PLL module in the bridge . . . 30
13 The complete bridge design . . . 31
14 Bridge architecture . . . 31
15 Top level FPGA design . . . 32
16 FIFO read entity diagram . . . 33
17 Data Transfer from HPS to FPGA . . . 33
18 FIFO read module state diagram . . . 34
19 Variable reducer entity diagram . . . 34
20 Variable reducer state diagram . . . 35
21 Row swap entity diagram . . . 35
22 Row swap state diagram . . . 36
23 Matrix converter entity diagram . . . 36
24 Serial converter state diagram . . . 37
25 Serial converter logic . . . 37
26 Parallel converter logic . . . 38
27 Parallel converter state diagram . . . 39
28 SRAM write entity diagram . . . 40
29 Data Transfer from FPGA to HPS . . . 40
30 SRAM write state diagram . . . 41
31 FPGA top level module . . . 42
32 Generated shared address . . . 44
33 First example problem . . . 46
34 Simple LIA Solver: Solution1 . . . 47
35 Z3 SMT Solver: Solution1 . . . 48
36 Simple LIA Solver: Solution2 . . . 49
37 Z3 SMT Solver: Solution2 . . . 50
38 Simple LIA Solver: Solution3 . . . 51
39 Z3 SMT Solver: Solution3 . . . 52
40 Compare solvers: row length = 6, column length = 4, integer range = 50 . . . 55
41 Compare solvers: row length = 6, column length = 7, integer range = 10 . . . 56
42 Compare solvers: row length = 7, column length = 3, integer range = 20 . . . 56
43 Compare solvers: row length = 7, column length = 4, integer range = 50 . . . 57

44 Execution time by varying row length (r): column length=7, integer range=10 57
45 Speedup by varying row length (r): column length=7, integer range=10 . . . 58
46 Execution time by varying column length (c): row length=7, integer range=5 58
47 Speedup by varying column length (c): row length=7, integer range=5 . . . 59
48 Execution time by varying maximum integer range (int): row length=5, column length=5 59
49 Speedup by varying maximum integer range (int): row length=5, column length=5 60
50 Data set operations 61
51 Make commands 65
52 Command to transfer binary files to the target 66
53 Command to transfer input files to the target 66
54 List of files transferred to the target 66
55 Hardware design properties in soc_eq_solver_pack.vhd 67
56 Run solver applications with an input text file 67
57 Run measurement application 68
58 Transfer results from the target to the host 68
59 Create timing graphs 68

List of Tables

1 Few Theories and their examples [17] 1
2 A formula for Linear Arithmetic Theory[17] 4
3 The tableau of the problem in general form with its constraints. 5
4 Steps in General Simplex algorithm (copied from [17]) 6
5 The tableau after the pivot operation between x and s_1 7
6 The tableau after the pivot operation between y and s_3 8
7 List of FPGA and MCU resources [3] 11
8 Simplified formula for linear integer arithmetic problem 15
9 An example of a linear integer system 16
10 FPGA Resource Utilization 45

Listings

1 Divisibility Test 9
2 Simple LIA Software-Solver: C code snippet 24
3 Main function of the SoC solver 43
4 An Output JASON File 53
5 Measurement Application 54

1 Introduction

A decision problem arises in everyday life where we decide to spend time, money, and energy wisely for optimal returns. In the field of mathematics and computer science, an algorithm that returns True or False for a given decision problem is called a decision procedure. It always terminates successfully and returns SAT (or True) if the problem is satisfiable or UNSAT otherwise. Its applications may range from solving a simple procedure, for example, to check prime numbers (known as Primality Test [25]) to a decision taken in complex systems. The Driver Drowsiness Detection [1] is one such complex system, where a decision made to alert a driver based on steering pattern, vehicle position, and eye/face monitoring. The decision procedures are applicable for many decidable first-order theories that have quantified variables such as propositional logic, equality, difference logic, linear arithmetic, and others as in table 1. A decision problem based on these theories is called a Satisfiability Modulo Theories (SMT) problem and their corresponding solver as SMT solver, which are typical canonical NP-complete problems [16]. Among these, the solver for propositional or Boolean logic (aka SAT solver) is a pioneer among the SMT solvers. The problem contains a formula with clauses having Boolean literals in Conjunctive Normal Form (CNF), verified for True under a combination of values for all variables. In general, the SMT solvers are widely used in hardware and software verification, testing tool, static code analysis, theorem proving, compiler optimization, scheduling, malware detection, and in many other applications [9]. Due to their wide range of applications and practical importance, the solvers are always expected to execute faster and accurate as input data increases.

Theory Name	Example Problem
Propositional logic	$x_1 \wedge (\neg x_2 \vee x_3)$
Equality	$x_1 = x_2 \wedge \neg(x_1 = x_3) \Rightarrow \neg(x_2 = x_3)$
Difference logic	$(x_1 - x_2 > 10) \wedge (x_2 - x_3 \leq 5)$
Linear arithmetic	$(2x_1 + 3x_2 \geq 10) \wedge (4x_2 - 6x_3 \leq 5)$
Bit vectors	$((a \gg b)\&c) < c$

Table 1: Few Theories and their examples [17]

In recent years, the programmable System-On-Chip (SoC) hardware attracts several researchers and algorithm developers with its unique ability to design a custom co-processor. The co-processor is implemented with required digital modules, clock, and data bus to perform a dedicated data processing and works in parallel with a master Micro-Controller Unit (MCU), where a well-defined algorithm can be accelerated to several folds. The SoC hardware contains MCU, Field-Programmable Gate Array (FPGA), interconnect bridges, and other hardware resources (such as storage and digital IPs) packaged in a single chip (as in figure 1), where until then they were available only as discrete modules and externally connected. Due to the advancement in the recent SoC hardware, the FPGAs are large enough to accommodate a moderate size algorithm with intense computational parts, and the MCU performs sequential operations such as data acquisition, buffering, external communication, and algorithm execution. The optimal utilization of the SoC is achieved when the MCU is used for data marshalling and transferring it to the FPGA, where the data is processed concurrently using parallel modules. In the meantime, the software may work on sequential parts and handle the processed data based on an interrupt or a polling event from the FPGA. In this way, the algorithm can be accelerated using the two sub-systems executing in parallel. In a few cases, algorithms are best suited for hardware implementation to achieve a desired speed and robustness; for example, an FFT signal processing algorithm [18].

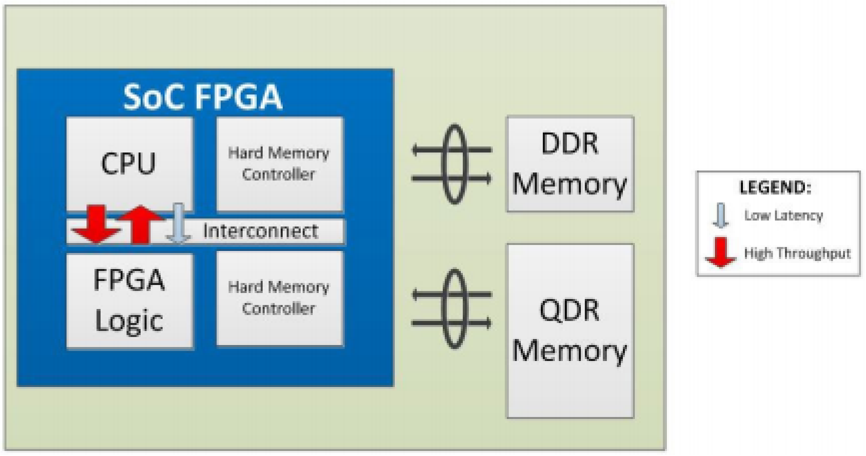


Figure 1: SoC FPGA Architecture [2]

1.1 Thesis Description

This thesis focuses on implementing a decision algorithm on a DE1-SoC development board that contains CycloneV programmable SoC chip from Intel Altera. The Linear Integer Arithmetic (LIA) problems are aimed here, as integer arithmetic operations are performed efficiently in the FPGA compare to real numbers. Since this is an experimental work, certain simplifications are made to the input problem and to its solver to reduce complexity in the initial phase and shall consider a complete solution for the future work based on the result. A simple solver for the simplified theory is designed and implemented on the SoC in two architectures to analyze the behavior of the FPGA co-processor. This SoC contains ARM A9 dual-core processor based MCU running at 950MHz, which is operated using a console based Linux, 1GB DDR3 SDRAM and other hardware (refer section 2.2 for more information). It has a powerful FPGA with 85K programmable logic elements and 4,450 embedded memory. There are two interconnect bridges, AXI bridge for a high-speed transfer which is suitable for data sets and lightweight bridge for a low data rate transfer as in Peripheral Input Out (PIO) pins.

The software in the MCU is written in C programming language using a GCC cross compiler for the ARM Linux architecture and compiled using a custom build system. The hardware design is done in Quartus Prime Lite software using VHDL and the data bridge is designed using Qsys tool from Altera. The Python language is used to extract result, test, verify, and plot timing graphs.

1.2 Thesis Tasks and Goal

The following are identified as the main tasks:

- Identify a class of the LIA decision problem suitable for hardware acceleration.
- Design an algorithm to solve the identified problem and implement in the MCU.
- Analyze the algorithm to identify suitable parts for hardware implementation.
- Construct the interconnect bridge for data transfer.
- Implement two variants of the identified parts in the FPGA with single and multiple instances of components for serial and parallel data processing.
- Verify the correctness of the algorithm and results of its three different implementations (software, hardware-serial, and hardware-parallel) in the SoC.
- Compare their execution time and conclude.

The primary goal of this thesis is to compare the performance of integer arithmetic operations in a matrix, which is executed with a same data flow in the two subsystems in the chosen programmable SoC hardware.

1.3 Structure of the report

The report is continued with the background for LA theories, existing algorithms, DE1-SoC board and similar work in the second chapter. The detailed design of an algorithm for the simplified LIA theory is covered in the third chapter followed by its SoC implementations in chapter four. The fifth chapter contains compilation results, verification, and comparison of execution time between different solutions, followed by the conclusion and future work in chapter six. Appendix A section contains instructions to compile and run the design files on the target board, which are created during the thesis time frame.

2 Background

2.1 Linear Arithmetic Theories and Solvers

A formula for the Linear Arithmetic (LA) problem is defined by the following rules.

formula	*formula* $\wedge$ *formula* | *(formula)* | *atom*
atom	*sum op sum*
op	$=$ | $\leq$ | $<$
sum	*term* | *sum+term*
term	*identifier* | *constant* | *identifier constant*

Table 2: A formula for Linear Arithmetic Theory[17]

$$\begin{gathered} 3x + 6y \geq 5z \\ 2x - 3y = 0 \\ 5y - 2z \geq 0 \end{gathered}$$

Example of LA theory.

The formula mentioned in table 2 represents a set of linear inequalities in general form, where the subtraction operations are considered as the addition of negative numbers, and $\geq$, $>$ symbols are converted to $\leq$ or $<$ by swapping the Right Hand Side (RHS) and the Left Hand Side (LHS). The LA problems may contain coefficients and solutions with real numbers, whereas the problems containing only integers are known as Linear Integer Arithmetic (LIA) problems. The goal of an LA decision algorithm is to find a solution that satisfies the input constraints. For example, in the above formula, a solver shall find a solution that satisfies all the three equations and would return $x = 1.5$, $y = 0.6667$, and $z = 1$. It reports UNSAT when a solution can not be found.

The following are the few methods to solve the LA formula mainly focusing on integer solutions.

2.1.1 Simplex Algorithm for SMT Solvers

This algorithm processes a set of linear constraints with real numbers and bounded variables for inequalities as listed below and solve the problem for satisfiability [17][10]. Even though this algorithm is not used in this thesis, similar matrix operations are performed in the thesis work and it could be used as a reference for accelerating the simplex algorithm in future.

1. Equalities of the form $a_1x_1 + \cdots + \mathrm{a}_nx_n = 0$
2. Lower and upper limits on variables: $l_i \leq x_i \leq u_i$, where l_i and u_i are constants representing upper and lower limits of x_i variable.

As a first step, reduce the formulas with non-equality logic to equality logic and introduce new formulas to track the non-equalities (represented by $\leq, \geq, <$, or $>$). Consider the following example used to demonstrate the simplex algorithm [17].

$$x + y \geq 2 \tag{1a}$$
$$2x - y \geq 0 \tag{1b}$$
$$-x + 2y \geq 1 \tag{1c}$$

The problem is rewritten by introducing new variables to convert the given formulas to equality logic.

$$x + y - s_1 = 0 \tag{2a}$$
$$2x - y - s_2 = 0 \tag{2b}$$
$$-x + 2y - s_3 = 0 \tag{2c}$$
$$s_1 \geq 2 \tag{2d}$$
$$s_2 \geq 0 \tag{2e}$$
$$s_3 \geq 1 \tag{2f}$$

Here the $s_1, \cdots, s_m$ are called basic or additional variables represented by $\mathcal{B}$. The variables $x_1, \cdots, x_n$ (or x and y) are called dependent or problem variables represented by $\mathcal{N}$.

The problem is also represented in the matrix form.

$$Ax = 0 \ \& \ \bigwedge_{i=1}^{m} l_i \leq s_i \leq u_i \tag{3}$$

The matrix A for the equation set (2) is written as,

$$\begin{bmatrix} 1 & 1 & -1 & 0 & 0 \\ 2 & -1 & 0 & -1 & 0 \\ -1 & 2 & 0 & 0 & -1 \end{bmatrix} \tag{4}$$

To represent the processing of the matrix A, it is convenient to have a table without the diagonal matrix formed due to the additional or the basic variables that have -1 along its diagonal. This form of a matrix is called **tableau** as in table 3.

	x	y
s$_1$	1	1
s$_2$	2	-1
s$_3$	-1	2

$2 \leq s_1$
$0 \leq s_2$
$1 \leq s_3$

Table 3: The tableau of the problem in general form with its constraints.

The equation form of the tableau:

$$\bigwedge_{x_i \in \mathcal{B}} \left(x_i = \sum_{x_j \in \mathcal{N}} a_{ij} x_j \right) \tag{5}$$

```
Algorithm: General-Simplex

Input: A linear system of constraints S
Output: "SAT" if the system is satisfiable or "UNSAT" otherwise

1. Transform the system into the general form as in equation (3).
2. Set $\mathcal{B}$ to be the set of additional variables $s_1, \cdots, s_m$.
3. Construct a tableau for the matrix $A$.
4. Determine a fixed order on the variables.
5. If there are no basic variables that violates its bounds, return "SAT".
   Otherwise, let $x_i$ be the first basic variable in the order that violates its bounds.
6. Search for the first suitable nonbasic variable $x_j$ in the order for pivoting with $x_i$. If
   there is no such variable, return "UNSAT"
7. Perform the pivot operation on $x_i$ and $x_j$.
8. Go to step 5.
```

Table 4: Steps in General Simplex algorithm (copied from [17])

The algorithm maintains two properties:

- **In-1.** $Ax = 0$
- **In-2.** The values of the dependent variables are within their bounds:

$$\forall x_j \in \mathcal{N}, l_j \leq \alpha(x_j) \leq u_j \tag{6}$$

In the beginning, x in **In-1**, i.e., all variables (including basic and dependent) are set to 0, and dependent variables have no bounds. During the first iteration, if there are no basic variables that violates its bounds, then the current assignment (i.e., zero) for all variables satisfies the constraints and declared as SAT. The bound check is done by replacing all variables to zero in equation (3) and checked if it satisfies all formulas. If there is a basic variable that violates its bound, consider x_i is lower than its lower bound, then the value for the x_i is increased by increasing the dependent variable x_j so that the x_i is greater or equal to the lower bound when $a_{ij}>0$. In case the violation occurred in the upper bound, then the x_j is decreased when $a_{ij}<0$. An adjusted value of the variable x_j that fulfills the boundary conditions is called Suitable. If there are no suitable variables, then the algorithm returns UNSAT. Let θ represents the change $\alpha(x_j)$ to meet the boundary conditions.

$$\theta = \frac{(u_i - \alpha(x_i))}{a_{ij}} \tag{7}$$

By modifying the x_j value with the θ to make the basic variable x_i confine to its limits, sometimes x_j may not satisfy its own boundary conditions. Therefore, the basic and dependent values are swapped in the tableau until all dependent variables satisfies the formulas and its limits. This swap operation is also known as Pivot operation. If there are no such x_j values, then the algorithm returns UNSAT. The processing of the example using the algorithm (as in table 4) is explain below.

In the beginning, x and y (the dependent variables) are assigned to zero and it is checked if it satisfies the equation set (3). Observe that $s_1 \geq 2$, but $s_1 = 0$ when all variables are zero, which does not satisfy the input problem. To satisfy s_1 (a basic variable), increase its value by 2, which also increase x by 2. The algorithm picks x in the first iteration based on the fixed order of variables from the leftmost and swaps it with s_1. The equations can be rewritten so that x is moved to the LHS as shown in the equation set (9).

$$\theta = \frac{(2 - 0)}{1} = 2, \tag{8a}$$

where the low limit of $s_1 = 2$, $\alpha(s_1) = 0$ and $a_{xs_1} = 1$

$$x = s_1 - y \tag{9a}$$
$$s_2 = 2s_1 - 3y \tag{9b}$$
$$s_3 = -s_1 + 3y \tag{9c}$$

As the result of the pivot operation between x and s_1, the tableau is written as in table 5. Since $\theta = 2$, that increases x by 2 ($\alpha(x) = 2$), s_1 by 2 (($\alpha(s_1) = 2$)), and other values are deduced from the equation set (9).

	s_1	y
x	1	-1
s_2	2	-3
s_3	-1	3

α(x) = 2
α(y) = 0
α(s$_1$) = 2
α(s$_2$) = 4
α(s$_3$) = -2

Table 5: The tableau after the pivot operation between x and s_1

Here $\alpha(s_3)$ is violating its lower limit, where it should be ≥ 1. Since x is processed in the last iteration, and based on the fixed variable order, y is considered in this iteration against s_3. Rewrite the equation (9c) to bring y in the LHS and other equations are modified as follows.

$$x = \frac{2}{3}s_1 - \frac{1}{3}s_3 \tag{10a}$$
$$s_2 = s_1 - s_3 \tag{10b}$$
$$y = \frac{1}{3}s_1 + \frac{1}{3}s_3 \tag{10c}$$

The new θ value is,

$$\theta = \frac{(1-(-2))}{3} = 1 \tag{11a}$$

where the low limit of $s_3 = 1$, $\alpha(s_3) =$ -2 and $a_{ys_3} = 3$

As the result of the pivot operation between y and s_3, the tableau is rewritten as below. Since $\theta = 1$, that increases y by 1 ($\alpha(y) = 1$) and s_1 by 2 ($\alpha(s_1) = 2$) from the previous iteration. Other values are deduced from the equation set (10).

	s_1	s_3
x	2/3	-1/3
s_2	1	-1
y	1/3	1/3

$\alpha(x) = 1$
$\alpha(y) = 1$
$\alpha(s_1) = 2$
$\alpha(s_2) = 1$
$\alpha(s_3) = 1$

Table 6: The tableau after the pivot operation between y and s_3

The assignments $x = 1$ and $y = 1$ is now satisfies the given problem. Selection of the pivot operation based on the fixed basic and the dependent variables ensures that the algorithm always terminates and this way of selecting the pivot elements is called Bland's rule [11].

2.1.2 Gaussian Elimination

This method is applicable only for a problem with the equality logic. It is also known as **Row Reduction** algorithm, where a sequence of reduction operations performed on a coefficient matrix of a linear integer system.

Consider the two equations with m variables that have integer coefficients as below.

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1m}x_m = b_1 \tag{12a}$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2m}x_m = b_2 \tag{12b}$$

The Gaussian Elimination algorithm mainly consists of two steps [15]:

1. Variable Elimination: A left most non-zero variable (also called pivot variable) is eliminated from an equation with reference to another.

2. Back-Substitution: A value for all variables is found by substituting values from the last to the first equation.

To eliminate variable x_1 in equation (12)(b), select (12)(a) as a pivot equation. It indicates that the variable x_1 will stay only in (12)(a) and will disappear or their corresponding coefficients are reduced to zero in equation (12)(b). To do so, multiply equation (a) with (a_{21}/a_{11}) and subtract it with equation (b). Replace the resultant equation with equation (b) that does not contain x_1 variable as in equation (13).

$$\left(a_{22} - \frac{a_{21}}{a_{11}}a_{12}\right)x_2 + \cdots + \left(a_{2m} - \frac{a_{21}}{a_{11}}a_{1m}\right)x_m = \left(b_2 - \frac{a_{21}}{a_{11}}b_1\right) \tag{13}$$

By repeating this variable elimination on all rows in a coefficient matrix will convert it to an upper triangular matrix, or also called **Row-Echelon Form** [12] and followed by the Back-Substitution method to solve a given system.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \qquad \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ 0 & b_{22} & b_{23} \\ 0 & 0 & c_{33} \end{bmatrix}$$

Coefficient and its row-echelon matrix of size 3x3

2.1.3 Divisibility Test

This method is applicable only for the LIA theory with the equality logic. Consider an equation having integer coefficients in a system (equation (14)), and the system can be declared as UNSAT if the constant b is not divisible by the GCD of all coefficients or maybe SAT otherwise [20]. This way of determining satisfiability is called Divisibility test.

$$\sum_{i=1}^{n} a_i x_i = b \tag{14}$$

Listing 1: Divisibility Test

```
GCD = 0;
for i = 1 to n:
    GCD = gcd(a[i], GCD);
if ( b % GCD != 0):
    print("UNSAT")
```

For example, consider the below equation.

$$3x + 6y + 3z = 6 \tag{15}$$

Here the GCD of all coefficients (x, y, and z) is 3, and it divides 6 without a reminder. Thus it is SAT if this is the only equation in the system. If a system has more than one equations and all satisfies the Divisibility test, it does not guarantee that the system is SAT until all the equations are processed with each other. However, if an equation is not satisfying the test, then it can be declared UNSAT without further processing. For example, in the below equation where 8 is not divisible by 3, and the system that contains this equation is declared UNSAT.

$$3x + 6y + 3z = 8 \tag{16}$$

2.2 DE1-SoC Board

DE1-SoC is a development board (figure 2) from Terasic Inc with a programmable SoC hardware. The board contains Cyclone V SoC 5CSEMA5F31C6 device (figure 3) that has an MCU with the dual-core ARM processor, FPGA and interconnect bridge components [6].

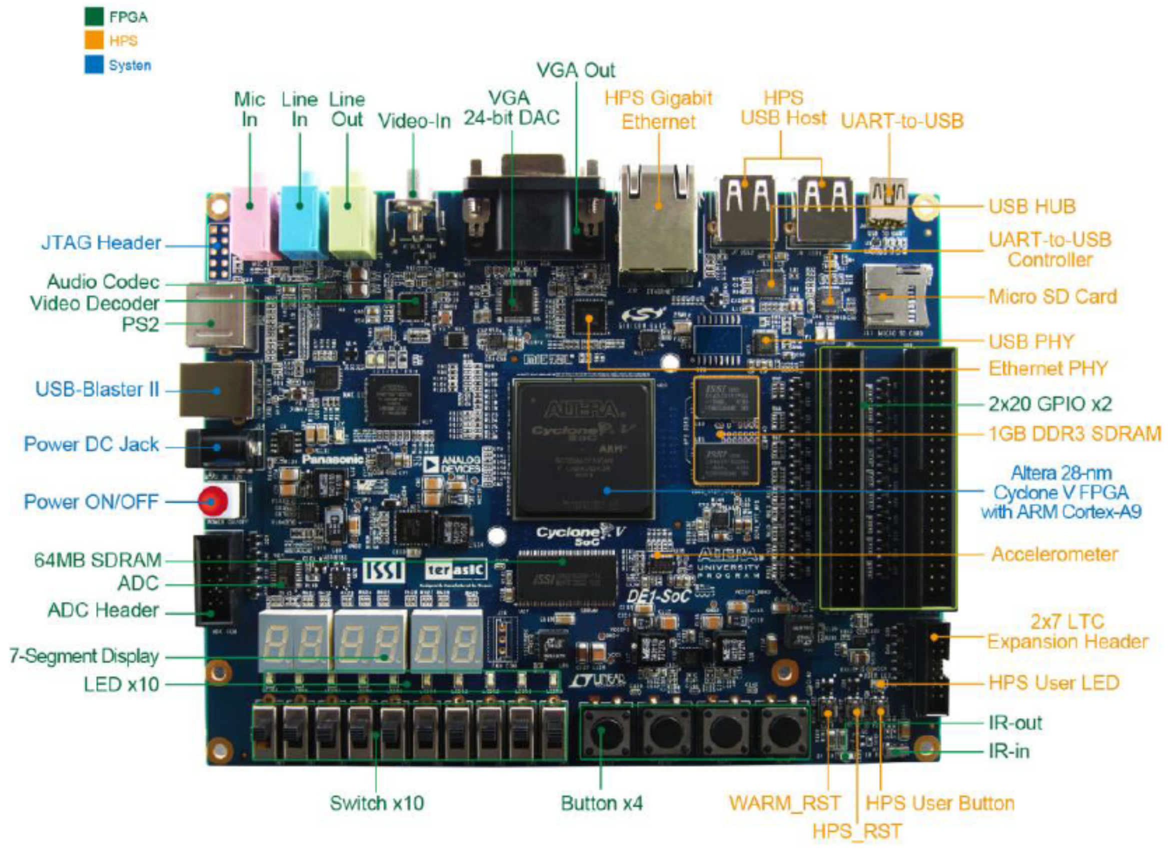


Figure 2: DE1-SoC development board copied from [6]

2.2.1 FPGA and MCU

The FPGA is designed on Quartus Prime tool using hardware description languages (such as VHDL and Verilog) or schematic/block diagram. The same software is used to compile, synthesis, and to program the device. The Hard Processor System (HPS) (also referred as CPU or MCU in this report) contains Dual-core ARM Cortex-A9 Core running at 950MHz. The following table 7 contains the resources that are associated with the MCU and the FPGA.

FPGA	**MCU**
Altera Cyclone V SE 5CSEMA5F31C6N device	Dual-core ARM Cortex-A9
85K programmable logic elements	1GB DDR3 SDRAM (32-bit data bus)
4,450 Kbits embedded memory	1 Gigabit Ethernet PHY with RJ45 connector
6 fractional PLLs	2-port USB Host, normal Type-A USB connector
64MB SDRAM (16-bit data bus)	UART to USB, USB Mini-B connector
10 slide switches and 10 slide switches	One user button and one user LED
Four 50MHz clock sources from the clock generator	LTC 2x7 expansion header

Table 7: List of FPGA and MCU resources [3]

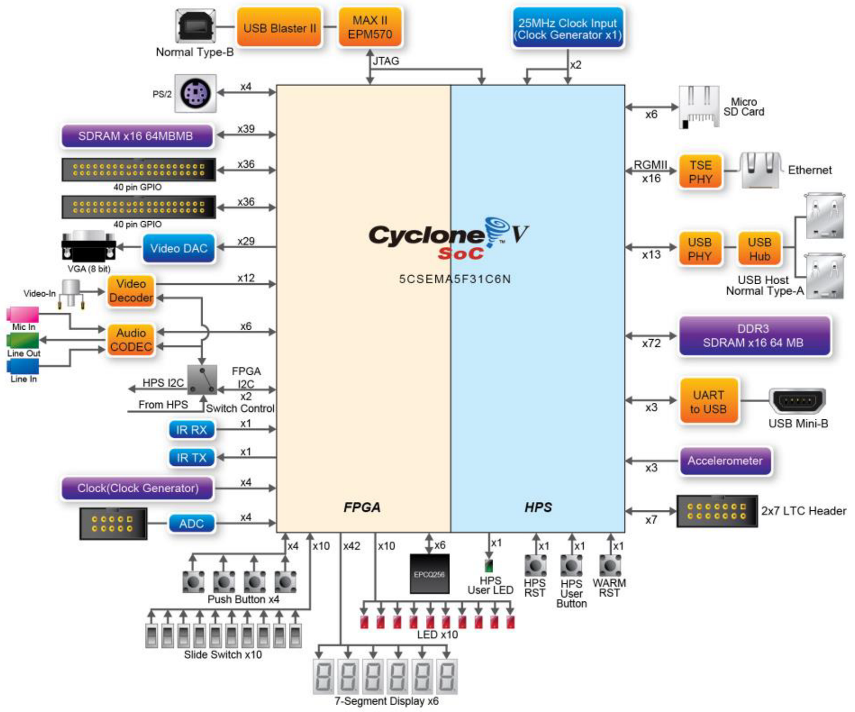


Figure 3: Cyclone V SoC architecture copied from [6]

2.2.2 Interconnect Bridge

Data transfer between the two sub-systems is designed using a tool called Platform Designer (previously known as Qsys). It provides a platform to select components and interconnect them through a drag and drop fashion. These components appear as a VHDL (or Verilog) entity in the top level module, and it is used through port map instruction in the FPGA design. Whereas in the MCU, the components are accessible through shared address pointers and pre-processor directive statements generated by the tool [4].

The data transfer unit contains the main level 3 (L3) interconnect and L4 buses as shown in figure 4. They are based on the ARM Advanced Microcontroller Bus Architecture (AMBA), Advanced eXtensible Interface (AXI), Advanced High-Performance Bus (AHB), and Advanced Peripheral Bus (APB) protocols. The L4 bus contains multiple access control and status registers of peripherals and memory controllers. The system interconnect contain nonblocking and multilayer architecture that supports multiple simultaneous data transfers [5].

The interconnect bridge contains the following three AXI buses:

1. HPS to FPGA: Transfer data set from HPS to FPGA.
2. FPGA to HPS: Transfer data set from FPGA to HPS.
3. Lightweight HPS to FPGA: Peripheral IO transfer from HPS to FPGA.

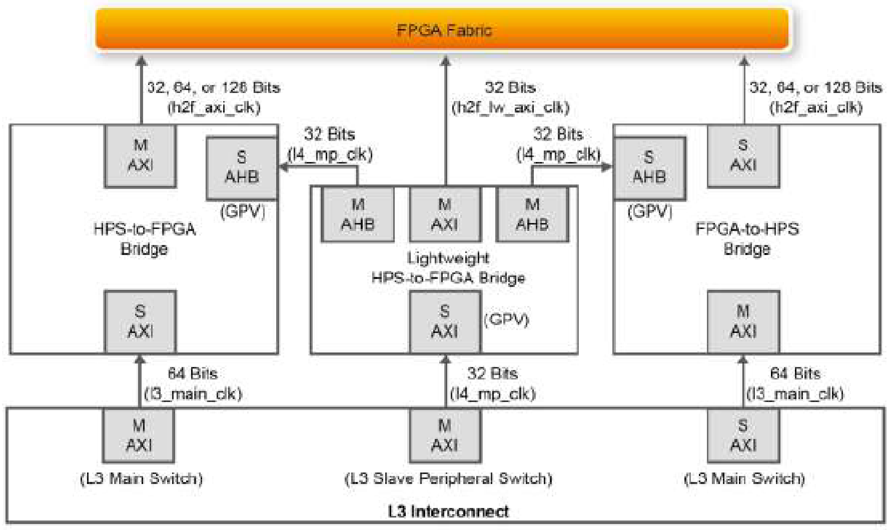
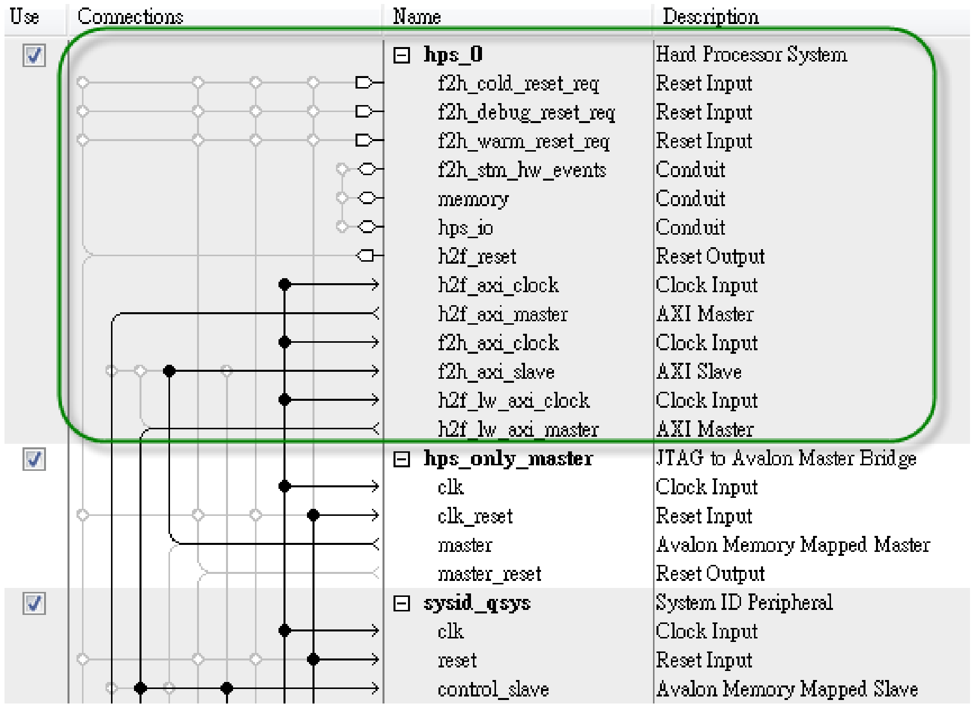


Figure 4: AXI Bridge copied from [3]

The HPS (hps_0) and its clock are the first components to add in the bridge design (figure 5). It is an entity of the processor that acts as an interface to the FPGA space, and other components added in the design appears as ports in this entity. The IP for FIFO, RAM, DMA, GPIO, RTC, timer, and many other digital modules can be included in the design from the component catalog in the platform designer tool.



Use	Connections	Name	Description
✓		**hps_0**	Hard Processor System
		f2h_cold_reset_req	Reset Input
		f2h_debug_reset_req	Reset Input
		f2h_warm_reset_req	Reset Input
		f2h_stm_hw_events	Conduit
		memory	Conduit
		hps_io	Conduit
		h2f_reset	Reset Output
		h2f_axi_clock	Clock Input
		h2f_axi_master	AXI Master
		f2h_axi_clock	Clock Input
		f2h_axi_slave	AXI Slave
		h2f_lw_axi_clock	Clock Input
		h2f_lw_axi_master	AXI Master
✓		**hps_only_master**	JTAG to Avalon Master Bridge
		clk	Clock Input
		clk_reset	Reset Input
		master	Avalon Memory Mapped Master
		master_reset	Reset Output
✓		**sysid_qsys**	System ID Peripheral
		clk	Clock Input
		reset	Reset Input
		control_slave	Avalon Memory Mapped Slave

Figure 5: hps_0 HPS component in the Qsys system

2.3 Similar Work

2.3.1 FPGA-Based Acceleration for Boolean Satisfiability

In the paper [14], a software-based solver for propositional logic is hardware accelerated by executing FPGA modules to process a few of its parts. The parts such as traversal of implication graphs and conflict clause generation are accelerated by multiple of their modules running in parallel. The goal of this work is to create a scalable hardware design on a Virtex-4 FPGA device for MiniSAT (software-based SAT solver) and compare the execution speed with and without using FPGA designs. The design contains a fixed number of clauses and its variables. The group of clauses called bins is reduced through dedicated multiple FPGA modules and executed in parallel. The clauses and variables are stored in the on-chip RAM on the FPGA and processes it as a matrix data. The result shows that by using the FPGA modules, the speed has increased by 17 times on XC4VFX140 hardware, despite of very high clock difference between them. It encourages this thesis as the hardware solver is executed at 450MHz and the software solver on a 3.6GHz Pentium IV machine. It also shows that hardware-based matrix operations are efficient in the FPGA, which is the primary goal of this thesis work.

2.3.2 FPGA-Based Acceleration of a Linear System Solver

In the paper [24], a system of linear equations with real numbers (both single and double precision) is hardware accelerated on FPGA by creating a portable library that can be integrated into various hardware-based solutions. This work simplifies the LU factorization method (a well-known method to solve linear systems) by considering only non-singular matrices (a matrix that has non-zero determinant and contains only one solution). The design is implemented on a Stratix III based FPGA from Xilinx, and it is capable of storing and processing a matrix of size 10000x1000 using on-chip SDRAM and FIFO for data marshalling. In comparison, the hardware solution is 2.2 times faster than a software solution for single precision and 1.7 times for double precision values for 140MHz FPGA clock and 2.67GHz Core2 Duo computer. This thesis aims to solve a set of linear integer equations using the Gaussian Elimination method (as explained in chapter 3), and the implementation and results of this paper indicates possibilities of acceleration although using a different algorithm.

3 Algorithm Design

As mentioned in the introduction, the goal of this thesis is to investigate if a decision procedure can be accelerated in the DE1-SoC development board and to study arithmetic operations on integer matrix that most of the linear arithmetic solvers contain (including the General-Simplex algorithm). While executing a well-defined procedure in FPGA and CPU with a same data flow to compare their performance, usually the FPGA implementation would be a winner due to its ability to process data set in a single clock cycle (Single Instruction on Multiple Data (SIMD)) and data processing using concurrent multiple hardware instances (Multiple Instruction on Multiple Data (MIMD)). But due to the huge difference in the subsystems clock speed (50MHz for FPGA and 950MHz for HPS) and interconnect bridge's overhead in our development board, a question raised whether the FPGA based implementation would still be the fastest in processing a matrix? To experiment this, the linear arithmetic theory is simplified, a solver is designed and implemented in software and hardware using the two sub-systems.

3.1 Simplification of Linear Arithmetic Problem

Let an LA formula contain only integer coefficients as it is easy to handle in FPGA compared to real values. An important simplification is to consider equality as the only logical operator allowed in the formula. This simplifies the LIA problem to a set of equations that are equal to zero, and converts the problem to a Linear Integer System (LIS). On the other hand, the LIS is an interesting problem on its own, which is used in many applications such as linear programming models, banking and finance procedures, and so on [21]. It is often considered that the LIS is harder to solve than the LA having real numbers. The formula for the simplified LIA is as mentioned in table 8.

formula	*formula* $\wedge$ *formula* | *formula* | *atom*
atom	*sum op sum*
op	=
sum	*term* | *sum+term*
term	*variable* | *constant* | *variable*constant*
constant	*only integer(32 bits)*

Table 8: Simplified formula for linear integer arithmetic problem

The following simplifications are made to the original LA problem.

1. A formula should contain only integer coefficients.
2. A formula should contain only equalities.
3. A coefficient should be in the range of 32-bit integer.
4. A variable value (or a solution) is bounded to the range of 32-bit integer.
5. A non-integer solution is considered as UNSAT.

formula	3x-5y+34z=0 ∧ 4x-2y+8z+9=0 ∧ 6x+9y+8=0 ∧ 3x+7y+5z+2=0
atom	3x-5y+34z=0
op	=
sum	3x-5y+34z
term	3x, -5y, 34z, -423p
constant	3, -5, 0, 34, -423

Table 9: An example of a linear integer system

$$\begin{bmatrix} 3 & -5 & 34 & 0 \\ 4 & -2 & 8 & 9 \\ 6 & 9 & 0 & 8 \\ 3 & 7 & 5 & 2 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The example formula expressed in the matrix form

3.2 Design of the Algorithm for Linear Integer Equations

Input equations of a system is written in a general form where x_i as a variable accepts only integers, a_{ij} as an integer coefficient and c_i as an integer constant. The system contains n equations and m variables.

$$\bigwedge_{i=1}^{n}(\sum_{j=1}^{m} a_{ij}x_j + c_i = 0) \tag{17}$$

The formula written in the equation form.

$$\begin{gathered} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1m}x_m + c_1 = 0 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2m}x_m + c_2 = 0 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3m}x_m + c_3 = 0 \\ \vdots \\ a_{(n-1)1}x_1 + a_{(n-1)2}x_2 + a_{(n-1)3}x_3 + \cdots + a_{(n-1)m}x_m + c_{(n-1)} = 0 \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nm}x_m + c_n = 0 \end{gathered}$$

Representation of the coefficient matrix of size $n.(m+1)$ and the variable matrix of size $(m+1).1$ is as below.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} & c_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} & c_2 \\ a_{31} & a_{32} & a_{33} & \dots & a_{3m} & c_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & a_{(n-1)3} & \dots & a_{(n-1)m} & c_{(n-1)} \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} & c_n \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_m \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

The algorithm mainly contains two steps; convert the coefficient matrix into an upper triangular matrix (or row-echelon form) using the Gaussian Elimination method and decide the satisfiability using the Back-Substitution and the Divisibility test methods.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} & c_1 \\ 0 & \beta_{22} & \beta_{23} & \dots & \beta_{2m} & \gamma_2 \\ 0 & 0 & \beta_{33} & \dots & \beta_{3m} & \gamma_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \beta_{(n-1)m} & \gamma_{(n-1)} \\ 0 & 0 & 0 & \dots & 0 & \gamma_n \end{bmatrix}$$

The Upper Triangular (UT) matrix in the equation form.

$$\begin{gathered} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1m}x_m + c_1 = 0 \\ 0x_1 + \beta_{22}x_2 + \beta_{23}x_3 + \cdots + \beta_{2m}x_m + \gamma_2 = 0 \\ 0x_1 + 0x_2 + \beta_{33}x_3 + \cdots + \beta_{3m}x_m + \gamma_3 = 0 \\ \vdots \\ 0x_1 + 0x_2 + 0x_3 + \cdots + \beta_{(n-1)m}x_m + \gamma_{(n-1)} = 0 \\ 0x_1 + 0x_2 + 0x_3 + \cdots + 0x_m + \gamma_n = 0 \end{gathered}$$

From the UT matrix, the result is analyzed using the following rules called Post-Processing. The post-processing is a combination of the Back-Substitution and the Divisibility test methods to find satisfiability of a system.

Satisfiability rules (or Post-Processing steps):

1. UNSAT: If γ_n in the last equation is a non-zero value when the number of equations is greater or equal to number of variables ($n \geq m$).
2. UNSAT: If the last element (or the equation constant) is not divisible by the GCD of all other elements in a row (Divisibility test), i.e. (γ MOD β) $\neq 0$.
3. UNSAT: If a variable contains a non-integer solution, which is obtained during the Back-Substitution method.

4. SAT: If the number of equations is less than number of variables ($n<m$) and the above three conditions are false.
5. SAT: If the number of equations is greater or equal to number of variables ($n \geq m$). Here, a value for all variables is found if the first three conditions are false.

Definition of the result:

- UNSAT: No solution exists for a given problem.
- SAT: A solution exists when $n<m$
- SAT: All variables have a defined integer value when $n \geq m$.

This covers all possible cases of a system and a result will be generated during no overflow.

Limitations of the algorithm (or solver):

- A general SMT solvers work for a broad range of formulas with real values and inequalities logic, but the solver designed here works only for the simplified inputs as discussed above. Thus, this is also referred as **Simple LIA** solver.
- When $n<m$ and if the system is satisfiable, the solver returns only SAT, whereas the general SMT solvers returns a solution.

3.3 Conversion of Coefficient Matrix to Row-Echelon Form

3.3.1 Lead Variable Reduction (LVR)

Consider two equations having n variables and their corresponding coefficients a_i and b_i. Let x_i ($1 \leq i \leq n$) be a variable that is aimed to be eliminated from the second equation. This variable is termed as a lead variable and an absolute smallest non-zero coefficient value among all equations for the chosen lead variable is called a lead coefficient. The lead variable can be removed from the second equation without losing the generality by performing repeated subtraction of the two equations until the lead coefficient is reduced to zero or absolute value of one.

$$a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n + c_1 = 0$$
$$b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n + c_2 = 0$$

For example, let the first variable x_1 be the lead variable and a_1 be the lead coefficient assuming $a_1<b_1$ and a_1,b_1 $\in$N. A straightforward way would be to multiply the first equation with $(\frac{b_1}{a_1})$ and subtract from the second equation as discussed in the Gaussian Elimination method. The resulting equation will have x_1 coefficient as 0 and replaced it with the second equation.

$$a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n + c_1 = 0$$
$$0x_1 + \beta_2x_2 + \beta_3x_3 + \cdots + \beta_nx_n + c_2 = 0$$

But, the multiplication of two big numbers may cause overflow and should be avoided in a hardware-based solution, especially where a bit length can be configurable. As well as, this way of a variable reduction could result in a non-integer coefficients. To avoid these, the lead variable is reduced through repetitive subtractions, which also eliminates GCD.

Lead variable reduction steps:

1. Identify an equation with the lead coefficient and called it a primary and the other a secondary equation.
2. Update the secondary equation with the difference between the two equations.
3. Continue the above two steps until the lead coefficient is either 0 or ±1.
4. When the lead coefficient is ±1, multiply the equation with the coefficient of the other equation and subtract them.
5. The lead variable is now reduced and considered as the secondary equation.

The following example illustrates this logic to reduce the variable x.

$$\begin{gathered} 7x + 4y + 5z + 6 = 0 \\ 5x + 6y + 8z + 3 = 0 \end{gathered}$$

- The equation with smallest x coefficient ($5x$) is considered as the primary.

$$\begin{gathered} 5x + 6y + 8z + 3 = 0 \Longrightarrow \text{primary} \\ 7x + 4y + 5z + 6 = 0 \Longrightarrow \text{secondary} \end{gathered}$$

- Subtract equations, secondary = secondary - primary

$$\begin{gathered} 5x + 6y + 8z + 3 = 0 \Longrightarrow \text{primary} \\ 2x - 2y - 3z + 3 = 0 \Longrightarrow \text{secondary} \end{gathered}$$

- Swap equations, so the primary has $2x$.

$$\begin{gathered} 2x - 2y - 3z + 3 = 0 \Longrightarrow \text{primary} \\ 5x + 6y + 8z + 3 = 0 \Longrightarrow \text{secondary} \end{gathered}$$

- Subtract equations, secondary = secondary - primary

$$\begin{gathered} 2x - 2y - 3z + 3 = 0 \Longrightarrow \text{primary} \\ 3x + 8y + 11z = 0 \Longrightarrow \text{secondary} \end{gathered}$$

- Identify primary and secondary equations

$$\begin{gathered} 2x - 2y - 3z + 3 = 0 \Longrightarrow \text{primary} \\ 3x + 8y + 11z = 0 \Longrightarrow \text{secondary} \end{gathered}$$

- Subtract equations, secondary = secondary - primary

$$\begin{gathered}2x-2y-3z+3=0 \Longrightarrow \text{primary}\\ 1x+10y+14z-3=0 \Longrightarrow \text{secondary}\end{gathered}$$

- Swap equations, so the primary has $1x$.

$$\begin{gathered}1x+10y+14z-3=0 \Longrightarrow \text{primary}\\ 2x-2y-3z+3=0 \Longrightarrow \text{secondary}\end{gathered}$$

- Multiply primary equation with 2

$$\begin{gathered}2x+20y+28z-6=0 \Longrightarrow \text{primary}\\ 2x-2y-3z+3=0 \Longrightarrow \text{secondary}\end{gathered}$$

- Subtract equations, secondary = secondary - primary

$$\begin{gathered}2x+20y+28y-6=0 \Longrightarrow \text{primary}\\ 0x-22y-31y+9=0 \Longrightarrow \text{secondary}\end{gathered}$$

Consider the primary equation before the reduction process and the secondary equation after the process to form a new equation set that still indicates the given problem:

$$\begin{gathered}5x+6y+8z+3{=}0\\ 0x+22y+31z-9=0\end{gathered}$$

Putting this as a function:

*secondary_row = **lead_variable_reducer**(primary_row, secondary_row, lv_index)*

The function accepts primary and secondary rows, and returns a row with the reduced variable at the position of the lead variable index (lv_index).

Using the function definition, the above example equations are presented as,

prim_row $\Longrightarrow$ 5 6 8 3
sec_row $\Longrightarrow$ 7 4 5 6
lv_index $\Longrightarrow$ 0 (Index of x)
sec_row = lead_variable_reducer(prim_row, sec_row, lv_index)
Result: sec_row $\Longrightarrow$ 0 22 31 -9

3.3.2 Equation Swap

In a set of equations, an equation with an absolute smallest non-zero lead coefficient and an equation that is currently identified as primary are swapped. This can also be defined as an identification of primary equation (or primary row in a matrix) at a given instance of the reduction process. This is performed before the variable reduction step to retain equations with smaller coefficients in the resultant triangular matrix. While swapping, zero is considered as the maximum number and should always be swapped while comparing with a non-zero value. The absolute value of an integer is considered for comparisons as in the variable reduction logic.

$$\textit{equation\_set} = \boldsymbol{equation\_swap}(\textit{equation\_set, prim\_row\_index, lead\_col\_index})$$

The variable *equation_set* is a coefficient matrix, the second argument indicates the primary row index and the third is a column index that represents the lead variable. The function returns the coefficient matrix with the swapped primary and the smallest row. An important note to observe is that the row swapped with the primary always has an index greater than the variable *prim_row_index*.

For example, consider the following equations and swap them at the location of the first equation and the first variable.

$$\begin{array}{c} 9x - 5y + 34z = 0 \\ 4x - 9y + 8z + 9 = 0 \\ 6x + 9y + 8 = 0 \\ 3x + 7y + 5z + 2 = 0 \end{array} \implies \begin{array}{c} 3x + 7y + 5z + 2 = 0 \\ 4x - 9y + 8z + 9 = 0 \\ 6x + 9y + 8 = 0 \\ 9x - 5y + 34z = 0 \end{array}$$

$$\text{equation\_set} = \textit{equation}\text{\_swap(equation\_set, 0, 0)}$$

$$\begin{bmatrix} \mathbf{9} & \mathbf{-5} & \mathbf{34} & \mathbf{0} \\ \mathbf{4} & \mathbf{-9} & \mathbf{8} & \mathbf{9} \\ \mathbf{6} & \mathbf{9} & \mathbf{0} & \mathbf{8} \\ \mathbf{3} & \mathbf{7} & \mathbf{5} & \mathbf{2} \end{bmatrix} \implies \begin{bmatrix} 3 & 7 & 5 & 2 \\ 4 & -9 & 8 & 9 \\ 6 & 9 & 0 & 8 \\ 9 & -5 & 34 & 0 \end{bmatrix}$$

equation_swap(0, 0): 1st and last rows are swapped

After the function call, the first and the last equations are swapped based on the x's coefficients. The zeros in the arguments indicate the first row as the primary row and x as the lead variable. This also indicates the row and the column indexes where the swap function begins. The following examples illustrate swapping functions for other indexes.

$$\begin{bmatrix} 9 & -5 & 34 & 0 \\ 4 & \mathbf{-9} & \mathbf{8} & \mathbf{9} \\ 6 & \mathbf{9} & \mathbf{0} & \mathbf{8} \\ 3 & \mathbf{7} & \mathbf{5} & \mathbf{2} \end{bmatrix} \Longrightarrow \begin{bmatrix} 9 & -5 & 34 & 0 \\ 3 & 7 & 5 & 2 \\ 6 & 9 & 0 & 8 \\ 4 & -9 & 8 & 9 \end{bmatrix}$$

equation_swap(1, 1): 2nd and the last rows are swapped.

$$\begin{bmatrix} 9 & -5 & 34 & 0 \\ 4 & -9 & 8 & 9 \\ 6 & 9 & \mathbf{0} & \mathbf{8} \\ 3 & 7 & \mathbf{5} & \mathbf{2} \end{bmatrix} \Longrightarrow \begin{bmatrix} 9 & -5 & 34 & 0 \\ 4 & -9 & 8 & 9 \\ 3 & 7 & 5 & 2 \\ 6 & 9 & 0 & 8 \end{bmatrix}$$

equation_swap(2, 2): 3rd and the last rows are swapped.

$$\begin{bmatrix} 9 & -5 & 34 & 0 \\ 4 & -9 & 8 & 9 \\ 6 & 9 & 0 & 8 \\ 3 & \mathbf{7} & \mathbf{5} & \mathbf{2} \end{bmatrix} \Longrightarrow \begin{bmatrix} 9 & -5 & 34 & 0 \\ 4 & -9 & 8 & 9 \\ 6 & 9 & 0 & 8 \\ 3 & 7 & 5 & 2 \end{bmatrix}$$

equation_swap(3, 1): no swap

$$\begin{bmatrix} 9 & -5 & 34 & 0 \\ 4 & -9 & \mathbf{8} & \mathbf{9} \\ 6 & 9 & \mathbf{0} & \mathbf{8} \\ 3 & 7 & \mathbf{5} & \mathbf{2} \end{bmatrix} \Longrightarrow \begin{bmatrix} 9 & -5 & 34 & 0 \\ 3 & 7 & 5 & 2 \\ 6 & 9 & 0 & 8 \\ 4 & -9 & 8 & 9 \end{bmatrix}$$

equation_swap(1, 3): 2nd and the last rows are swapped

3.3.3 Creation of Upper Triangular Matrix

Assume a variable *coef_eqs* contains a coefficient matrix of size $n * (m + 1)$.

$$coef\_eqs = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1m} & c_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2m} & c_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3m} & c_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{(n-1)1} & a_{(n-1)2} & a_{(n-1)3} & \cdots & a_{(n-1)m} & c_{(n-1)} \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} & c_n \end{bmatrix}$$

Perform the equation swap at the position a_{11} (*equation*_swap(0,0)) to convert it as the smallest absolute non-zero value in the first column. It compares the first value in the first row with the first value in all other rows and swaps if necessary. Then pass the first row as the primary, the second row as the secondary and the first column as the lead index to the variable reducer function to convert a_{21} to zero(*lead_variable_reducer(0,1,0)*). At this stage a_{11} is the smallest lead coefficient, a_{21} is zero and other values in the first column are unchanged. Call the variable reducer function again with the third row as the secondary with other parameters remaining the same (*lead_variable_reducer(0,2,0)*) to convert a_{31} to zero. Continue calling this function for all remaining rows as the secondary row (*lead_variable_reducer(0,1...n-1,0)*)) so that the entire column reduces to zero except for a_{11}.

$$coef\_eqs=\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} & c_1 \\ 0 & \mathbf{a}^1_{22} & \mathbf{a}^1_{23} & \dots & \mathbf{a}^1_{2m} & \mathbf{c}^1_2 \\ 0 & \mathbf{a}^1_{32} & \mathbf{a}^1_{33} & \dots & \mathbf{a}^1_{3m} & \mathbf{c}^1_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \mathbf{a}^1_{(n-1)2} & \mathbf{a}^1_{(n-1)3} & \dots & \mathbf{a}^1_{(n-1)m} & \mathbf{c}^1_{(n-1)} \\ 0 & \mathbf{a}^1_{n2} & \mathbf{a}^1_{n3} & \dots & \mathbf{a}^1_{nm} & \mathbf{c}^1_n \end{bmatrix}$$

Perform the equation swap at the position a^1_{22}(*equation_*swap(1,1)) to convert it as a smallest coefficient in the second column. Then consider the second and the third rows for variable reduction at the second column (*lead_variable_reducer(1,2,1)*)to reduce a^1_{32} to zero. Continue preforming the reduction on all other rows with the second row as the primary row and the second column as the lead variable (*lead_variable_reducer(1,2...n-1,1)*)). After this, all values in the second column are reduced to zero other than the first two coefficients.

$$coef\_eqs=\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} & c_1 \\ 0 & a^1_{22} & a^1_{23} & \dots & a^1_{2m} & \mathbf{c}^1_2 \\ 0 & 0 & \mathbf{a}^2_{33} & \dots & \mathbf{a}^2_{3m} & \mathbf{c}^2_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \mathbf{a}^2_{(n-1)3} & \dots & \mathbf{a}^2_{(n-1)m} & \mathbf{c}^2_{(n-1)} \\ 0 & 0 & \mathbf{a}^2_{n3} & \dots & \mathbf{a}^2_{nm} & \mathbf{c}^2_n \end{bmatrix}$$

Repeat these steps till the minimum value of n and m to obtain a upper triangular matrix.

The steps to reduce the matrix is generalized as follows.

- Call *equation_swap()* function with parameter *prim_row_index* = *lead_col_index* starting from zero.
- Call *lead_variable_reducer()* function with *prim_row* = row(*prim_row_index*) and *sec_row* = row(*prim_row_index+1*) and continue till *sec_row* = last row.
- Repeat the above two steps till *prim_row_index* reaches the minimum of n and m.
- Obtain the result/decision from the triangular matrix by performing the post-processing method.

The steps of the solver can be divided mainly into two parts: the matrix conversion and the post-processing. The matrix conversion is an intense computational part where rows are reduced and swapped repeatedly, and the post-processing is a sequential process that starts from the last to the first row to analyze the result. Hence, **the matrix conversion is more suitable for the FPGA implementation and the later part on the MCU in the SoC**.

A C code snippet describing the algorithm as follows.

Listing 2: Simple LIA Software-Solver: C code snippet

```
int i,j;
int coef_eqs[][];
int prim_eq[], sec_eq[];
/* Read n.m size matrix and assign it to coef_eqs */
coef_eqs = read_input_matrix(n, m);
for (i = 0; i < min(n,m); i++) {

    /* Swap equations along the diagonal rows */
    coef_eqs = equation_swap(coef_eqs , i, i);

    /* Reduce lead variable at index i and copy the row
       to prim_eq */
    prim_eq = coef_eqs[i];

    /* Reduce lead variable for all rows below the ith row */
    for (j= i+1; j < n; j++) {
        sec_eq = coef_eqs[j];
        sec_eq = lead_variable_reducer(prim_eq , sec_eq , i);
    }
}
post_processing();
```

3.4 Solving Example Problems

In this section, few example problems will be solved based on the described algorithm.

Example problem 1: Apply the algorithm to find the values of variable x, y and z if exits.

$$\begin{array}{l} 3y + 5z - 14 = 0 \\ x + y + z - 2 = 0 \\ 5x + 4y + 7z - 9 = 0 \\ 6x + 3y + 3 = 0 \end{array} \quad \Longrightarrow \quad \begin{bmatrix} 0 & 3 & 5 & -14 \\ 1 & 1 & 1 & -2 \\ 5 & 4 & 7 & -9 \\ 6 & 3 & 0 & 3 \end{bmatrix}$$

Index	equation_swap() $\implies$	lead_variable_reducer()
0,0	$\begin{bmatrix}\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{-2} \\ \mathbf{0} & \mathbf{3} & \mathbf{5} & \mathbf{-14} \\ \mathbf{5} & \mathbf{4} & \mathbf{7} & \mathbf{-9} \\ \mathbf{6} & \mathbf{3} & \mathbf{0} & \mathbf{3}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1 & -2 \\ 0 & 3 & 5 & -14 \\ 0 & -1 & 2 & 1 \\ 0 & -3 & -6 & 15\end{bmatrix}$
1,1	$\begin{bmatrix}1 & 1 & 1 & -2 \\ 0 & \mathbf{-1} & \mathbf{2} & \mathbf{1} \\ 0 & \mathbf{3} & \mathbf{5} & \mathbf{-14} \\ 0 & \mathbf{-3} & \mathbf{-6} & \mathbf{15}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1 & -2 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & 11 & -11 \\ 0 & 0 & -12 & 12\end{bmatrix}$
2,2	$\begin{bmatrix}1 & 1 & 1 & -2 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & \mathbf{11} & \mathbf{-11} \\ 0 & 0 & \mathbf{-12} & \mathbf{12}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1 & -2 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & 11 & -11 \\ 0 & 0 & 0 & 0\end{bmatrix}$

Convert the reduced matrix to equations and solve for the variables using the post-processing method.

$$\begin{aligned} x+y+z-2&=0 \\ -y+2z+1&=0 \\ 11z-11&=0 \end{aligned} \implies \begin{aligned} x&=-2 \\ y&=3 \\ z&=1 \end{aligned}$$

The problem is **satisfiable (SAT)** with solution.

Example problem 2: Apply the algorithm when $n < m$.

$$\begin{aligned} 3y+5z+p+2q-19&=0 \\ x+y+z-2p+q-2&=0 \\ 5x+4y+7z+p+q-12&=0 \\ 6x+3y+3&=0 \end{aligned} \implies \begin{bmatrix}0 & 3 & 5 & 1 & 2 & -19 \\ 1 & 1 & 1 & -2 & 1 & -2 \\ 5 & 4 & 7 & 1 & 1 & -12 \\ 6 & 3 & 0 & 0 & 0 & 3\end{bmatrix}$$

Index	equation_swap() $\implies$	lead_variable_reducer()
0,0	$\begin{bmatrix}\mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{-2} & \mathbf{1} & \mathbf{-2} \\ \mathbf{0} & \mathbf{3} & \mathbf{5} & \mathbf{1} & \mathbf{2} & \mathbf{-19} \\ \mathbf{5} & \mathbf{4} & \mathbf{7} & \mathbf{1} & \mathbf{1} & \mathbf{-12} \\ \mathbf{6} & \mathbf{3} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{3}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1 & -2 & 1 & -2 \\ 0 & 3 & 5 & 1 & 2 & -19 \\ 0 & -1 & 2 & 11 & -4 & -2 \\ 0 & -3 & -6 & 12 & -6 & 15\end{bmatrix}$
1,1	$\begin{bmatrix}1 & 1 & 1 & -2 & 1 & -2 \\ 0 & \mathbf{-1} & \mathbf{2} & \mathbf{11} & \mathbf{-4} & \mathbf{-2} \\ 0 & \mathbf{3} & \mathbf{5} & \mathbf{1} & \mathbf{2} & \mathbf{-19} \\ 0 & \mathbf{-3} & \mathbf{-6} & \mathbf{12} & \mathbf{-6} & \mathbf{15}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1 & -2 & 1 & -2 \\ 0 & -1 & 2 & 11 & -4 & -2 \\ 0 & 0 & 11 & 34 & -10 & -25 \\ 0 & 0 & -12 & 21 & 6 & 21\end{bmatrix}$
2,2	$\begin{bmatrix}1 & 1 & 1 & -2 & 1 & -2 \\ 0 & -1 & 2 & 11 & -4 & -2 \\ 0 & 0 & \mathbf{11} & \mathbf{34} & \mathbf{-10} & \mathbf{-25} \\ 0 & 0 & \mathbf{-12} & \mathbf{21} & \mathbf{6} & \mathbf{21}\end{bmatrix}$	$\begin{bmatrix}1 & 1 & 1 & -2 & 1 & -2 \\ 0 & -1 & 2 & 11 & -4 & -2 \\ 0 & 0 & 11 & 34 & -10 & -25 \\ 0 & 0 & 0 & 177 & -54 & -69\end{bmatrix}$

The matrix succeeds the Divisibility test, hence the problem is **satisfiable (SAT)**.

Example problem 3: Similar to the example 1.

$$\begin{array}{l} 3y + 5z - 14 = 0 \\ x + y + z - 2 = 0 \\ 5x + 4y + 7z - 9 = 0 \\ 6x + 3y + 7 = 0 \end{array} \quad \Longrightarrow \quad \begin{bmatrix} 0 & 3 & 5 & -14 \\ 1 & 1 & 1 & -2 \\ 5 & 4 & 7 & -9 \\ 6 & 3 & 0 & 7 \end{bmatrix}$$

$$\begin{array}{lcc}
\text{Index} & \text{equation\_swap()} \Longrightarrow & \text{lead\_variable\_reducer()} \\
0,0 & \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{-2} \\ \mathbf{0} & \mathbf{3} & \mathbf{5} & \mathbf{-14} \\ \mathbf{5} & \mathbf{4} & \mathbf{7} & \mathbf{-9} \\ \mathbf{6} & \mathbf{3} & \mathbf{0} & \mathbf{7} \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 & -2 \\ 0 & 3 & 5 & -14 \\ 0 & -1 & 2 & 1 \\ 0 & -3 & -6 & 19 \end{bmatrix} \\
1,1 & \begin{bmatrix} 1 & 1 & 1 & -2 \\ 0 & \mathbf{-1} & \mathbf{2} & \mathbf{1} \\ 0 & \mathbf{3} & \mathbf{5} & \mathbf{-14} \\ 0 & \mathbf{-3} & \mathbf{-6} & \mathbf{19} \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 & -2 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & 11 & -11 \\ 0 & 0 & -12 & 16 \end{bmatrix} \\
2,2 & \begin{bmatrix} 1 & 1 & 1 & -2 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & \mathbf{11} & \mathbf{-11} \\ 0 & 0 & \mathbf{-12} & \mathbf{16} \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 & -2 \\ 0 & -1 & 2 & 1 \\ 0 & 0 & 11 & -11 \\ 0 & 0 & 0 & 44 \end{bmatrix}
\end{array}$$

The last element of the last row is not zero, hence the problem is **not satisfiable (UN-SAT))**. The last row should have either more than one non-zero or all zero elements in a satisfiable UT matrix.

4 Algorithm Implementation on SoC

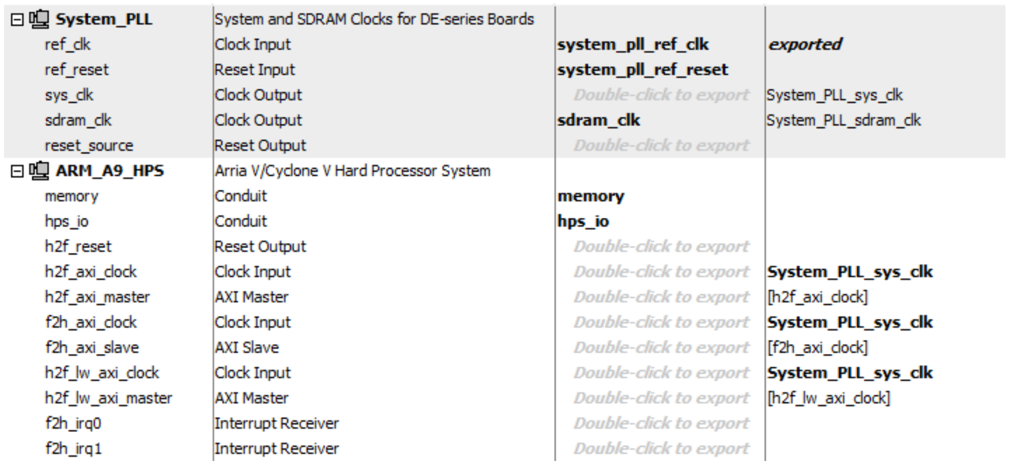
As mentioned in the previous chapter, the matrix conversion is suitable for the FPGA implementation and the post-processing on the MCU, the implementation of these functions in the SoC are covered here in detail.

The implementation can be broadly divided into the following parts.

1. Design the interconnect bridges to transfer matrix.
2. Design and implement data transfer modules in the FPGA.
3. Implement the matrix conversion in the FPGA.
4. Implement the post-processing in the MCU.

4.1 Designing the Interconnect Bridges

The aim of the bridge design is to transfer a coefficient matrix from the MCU to the FPGA and the converted UT matrix from the FPGA back to the MCU. The bridge components must be selected so that there should not be a data loss or duplication as the two sub-systems work on different principles. The component for the HPS and the system PLL are selected before the data components (figure 6), where the system PLL clock source is set to 100MHz, twice that of the bridge clock (50MHz) for an efficient data transfer.



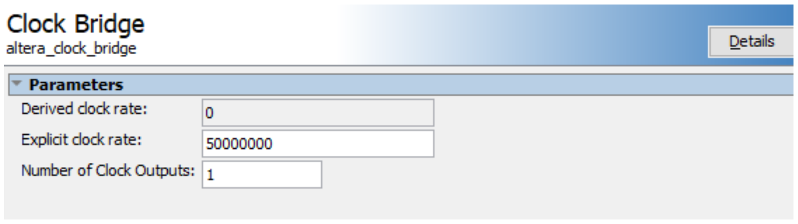
⊟ **System_PLL**	System and SDRAM Clocks for DE-series Boards		
ref_clk	Clock Input	**system_pll_ref_clk**	***exported***
ref_reset	Reset Input	**system_pll_ref_reset**	
sys_clk	Clock Output	*Double-click to export*	System_PLL_sys_clk
sdram_clk	Clock Output	**sdram_clk**	System_PLL_sdram_clk
reset_source	Reset Output	*Double-click to export*	
⊟ **ARM_A9_HPS**	Arria V/Cyclone V Hard Processor System		
memory	Conduit	**memory**	
hps_io	Conduit	**hps_io**	
h2f_reset	Reset Output	*Double-click to export*	
h2f_axi_clock	Clock Input	*Double-click to export*	**System_PLL_sys_clk**
h2f_axi_master	AXI Master	*Double-click to export*	[h2f_axi_clock]
f2h_axi_clock	Clock Input	*Double-click to export*	**System_PLL_sys_clk**
f2h_axi_slave	AXI Slave	*Double-click to export*	[f2h_axi_clock]
h2f_lw_axi_clock	Clock Input	*Double-click to export*	**System_PLL_sys_clk**
h2f_lw_axi_master	AXI Master	*Double-click to export*	[h2f_lw_axi_clock]
f2h_irq0	Interrupt Receiver	*Double-click to export*	
f2h_irq1	Interrupt Receiver	*Double-click to export*	

Figure 6: PLL and HPS components in the bridge

Along with a data array component, a one-bit data line is required to act as an *enable* signal from the MCU that controls the FPGA modules. A 1-bit PIO component is used for this purpose.

4.1.1 Data transfer from MCU to FPGA

This is done using an FIFO IP along with the *enable* signal. The 1-bit PIO based *enable* signal (figure 8) is used by the MCU to activate the FPGA to read the FIFO data. The matrix row and column sizes are pushed as the first two data followed by the coefficients from the MCU. A *FIFO_Read* state machine in the FPGA reads the row and the column sizes and continue to read the row length $*$ column length number of data. The FIFO contains depth of 512 with 32-bit data width (figure 9) allowing a coefficient to be a 4-byte integer. The data is pushed at 100MHz using the system PLL (*System_PLL*) and popped at 50MHz using the clock bridge module (*clock_bridge_0* in figure 7). The bridge integration of all the components use in data transfer from the MCU to the FPGA is as shown in figure 10.

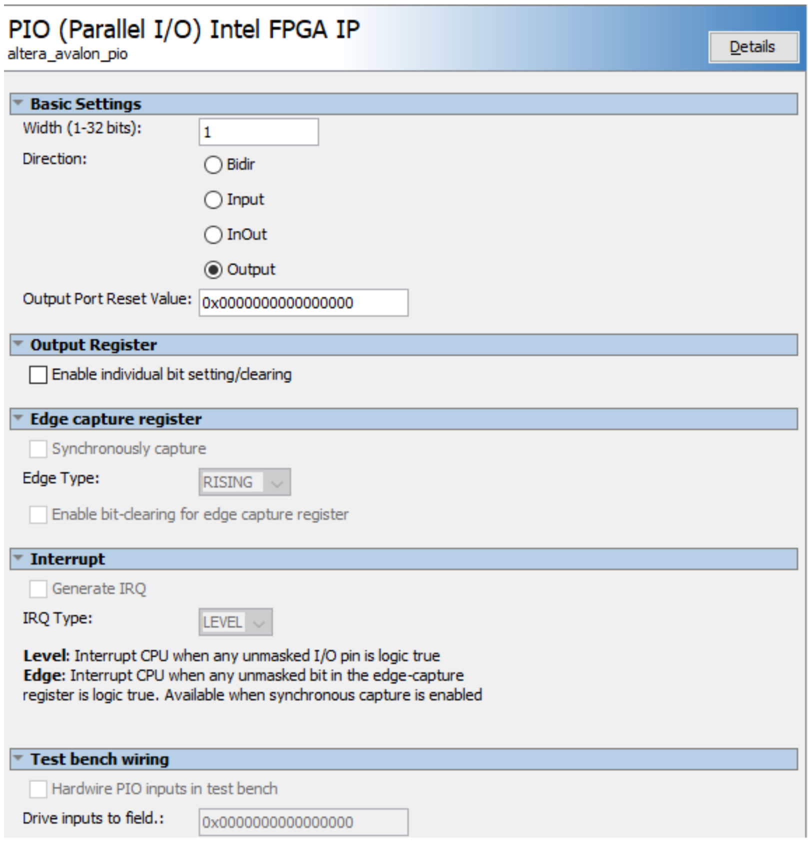

Clock Bridge
altera_clock_bridge

Details

Parameters

Derived clock rate: 0
Explicit clock rate: 50000000
Number of Clock Outputs: 1

Figure 7: Bridge clock IP


PIO (Parallel I/O) Intel FPGA IP
altera_avalon_pio

Details

Basic Settings

Width (1-32 bits): 1
Direction: Bidir Input InOut [✓ Output]
Output Port Reset Value: 0x0000000000000000

Output Register

☐ Enable individual bit setting/clearing

Edge capture register

☐ Synchronously capture
Edge Type: RISING
☐ Enable bit-clearing for edge capture register

Interrupt

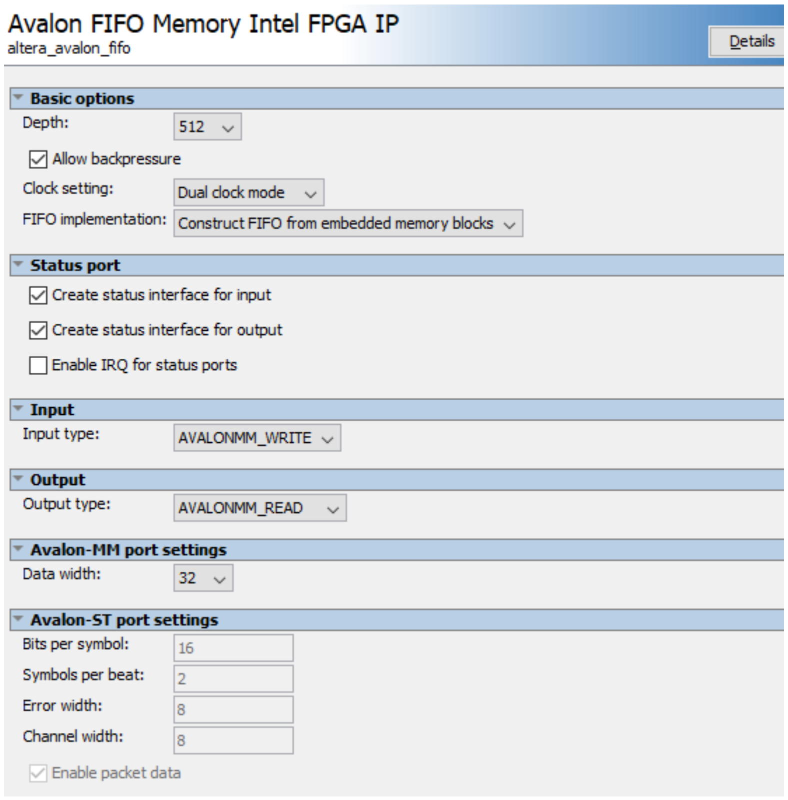
☐ Generate IRQ
IRQ Type: LEVEL

Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled

Test bench wiring

☐ Hardwire PIO inputs in test bench
Drive inputs to field.: 0x0000000000000000

Figure 8: PIO IP properties


Avalon FIFO Memory Intel FPGA IP
altera_avalon_fifo

Basic options
- Depth: 512
- ✓ Allow backpressure
- Clock setting: Dual clock mode
- FIFO implementation: Construct FIFO from embedded memory blocks

Status port
- ✓ Create status interface for input
- ✓ Create status interface for output
- ☐ Enable IRQ for status ports

Input
- Input type: AVALONMM_WRITE

Output
- Output type: AVALONMM_READ

Avalon-MM port settings
- Data width: 32

Avalon-ST port settings
- Bits per symbol: 16
- Symbols per beat: 2
- Error width: 8
- Channel width: 8
- ✓ Enable packet data

Figure 9: FIFO IP properties

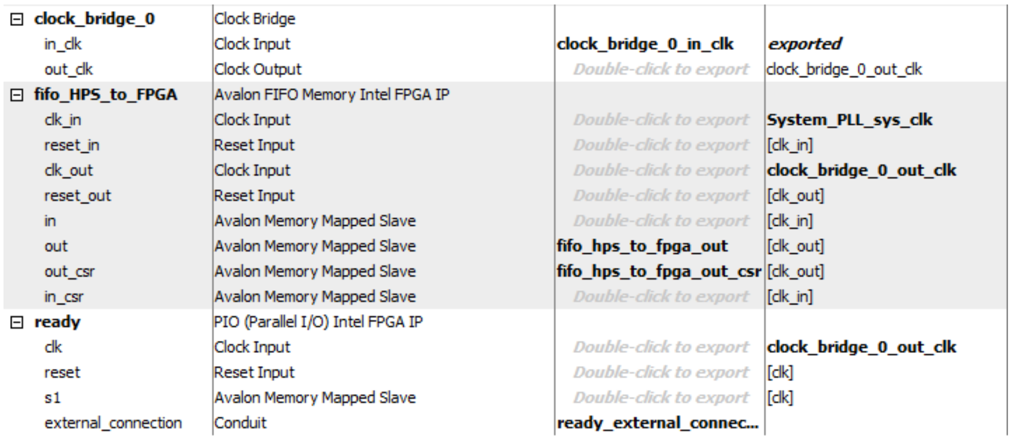
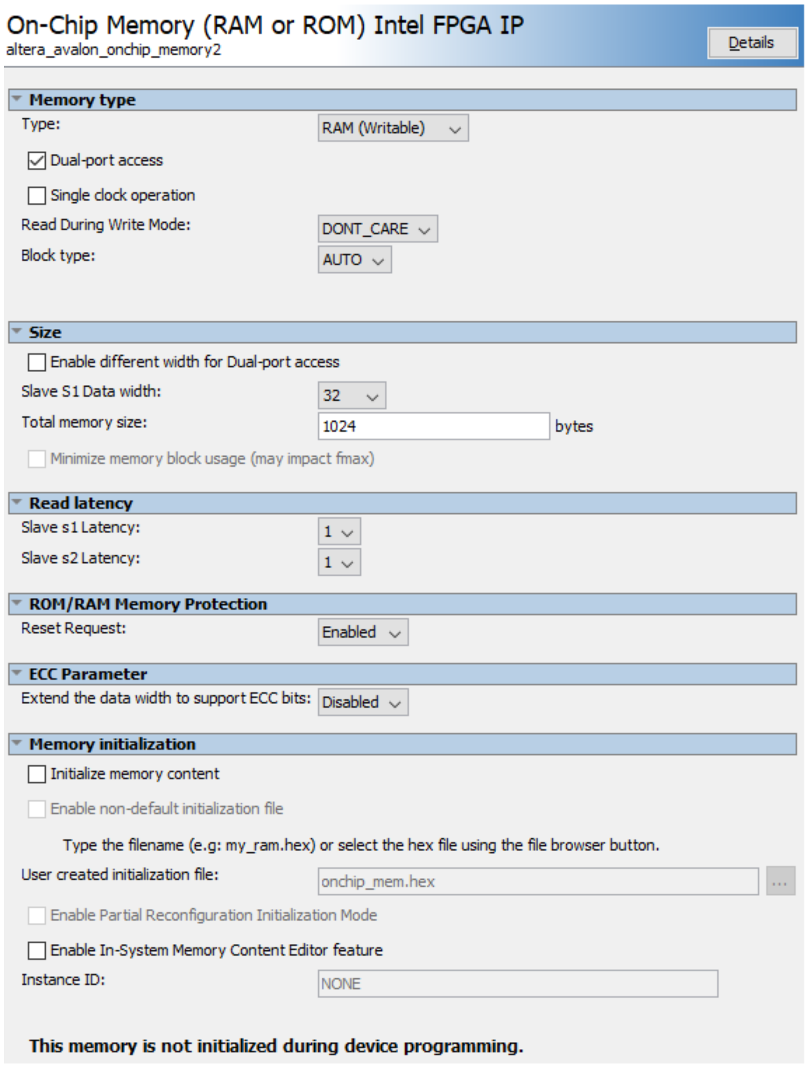

⊟ **clock_bridge_0**	Clock Bridge		
in_clk	Clock Input	**clock_bridge_0_in_clk**	***exported***
out_clk	Clock Output	*Double-click to export*	clock_bridge_0_out_clk
⊟ **fifo_HPS_to_FPGA**	Avalon FIFO Memory Intel FPGA IP		
clk_in	Clock Input	*Double-click to export*	**System_PLL_sys_clk**
reset_in	Reset Input	*Double-click to export*	[clk_in]
clk_out	Clock Input	*Double-click to export*	**clock_bridge_0_out_clk**
reset_out	Reset Input	*Double-click to export*	[clk_out]
in	Avalon Memory Mapped Slave	*Double-click to export*	[clk_in]
out	Avalon Memory Mapped Slave	**fifo_hps_to_fpga_out**	[clk_out]
out_csr	Avalon Memory Mapped Slave	**fifo_hps_to_fpga_out_csr**	[clk_out]
in_csr	Avalon Memory Mapped Slave	*Double-click to export*	[clk_in]
⊟ **ready**	PIO (Parallel I/O) Intel FPGA IP		
clk	Clock Input	*Double-click to export*	**clock_bridge_0_out_clk**
reset	Reset Input	*Double-click to export*	[clk]
s1	Avalon Memory Mapped Slave	*Double-click to export*	[clk]
external_connection	Conduit	**ready_external_connec...**	

Figure 10: FIFO, PIO and PLL module in the bridge

4.1.2 Data transfer from FPGA to MCU

This is done using an On-Chip RAM IP (figure 11) where the converted UT matrix is stored in the RAM from a fixed location, and a special location is updated with a status. The MCU may poll for the status and resume data reading and perform post-processing if there exists a non-zero value. The size of the RAM is 1024 of 32-bit data width and have dual port access that allows writing from the two sub-systems. The data read and write operations clocked at 50MHz using the same bridge clock module (figure 12). The complete bridge setup is as shown in figure 13 and the bridge overview in figure 14.



On-Chip Memory (RAM or ROM) Intel FPGA IP
altera_avalon_onchip_memory2

Memory type

Type: RAM (Writable)

[✓] Dual-port access

[ ] Single clock operation

Read During Write Mode: DONT_CARE

Block type: AUTO

Size

[ ] Enable different width for Dual-port access

Slave S1 Data width: 32

Total memory size: 1024 bytes

[ ] Minimize memory block usage (may impact fmax)

Read latency

Slave s1 Latency: 1

Slave s2 Latency: 1

ROM/RAM Memory Protection

Reset Request: Enabled

ECC Parameter

Extend the data width to support ECC bits: Disabled

Memory initialization

[ ] Initialize memory content

[ ] Enable non-default initialization file

Type the filename (e.g: my_ram.hex) or select the hex file using the file browser button.

User created initialization file: onchip_mem.hex

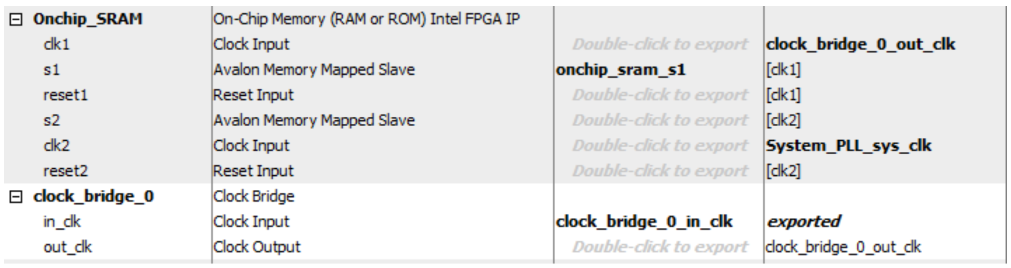
[ ] Enable Partial Reconfiguration Initialization Mode

[ ] Enable In-System Memory Content Editor feature

Instance ID: NONE

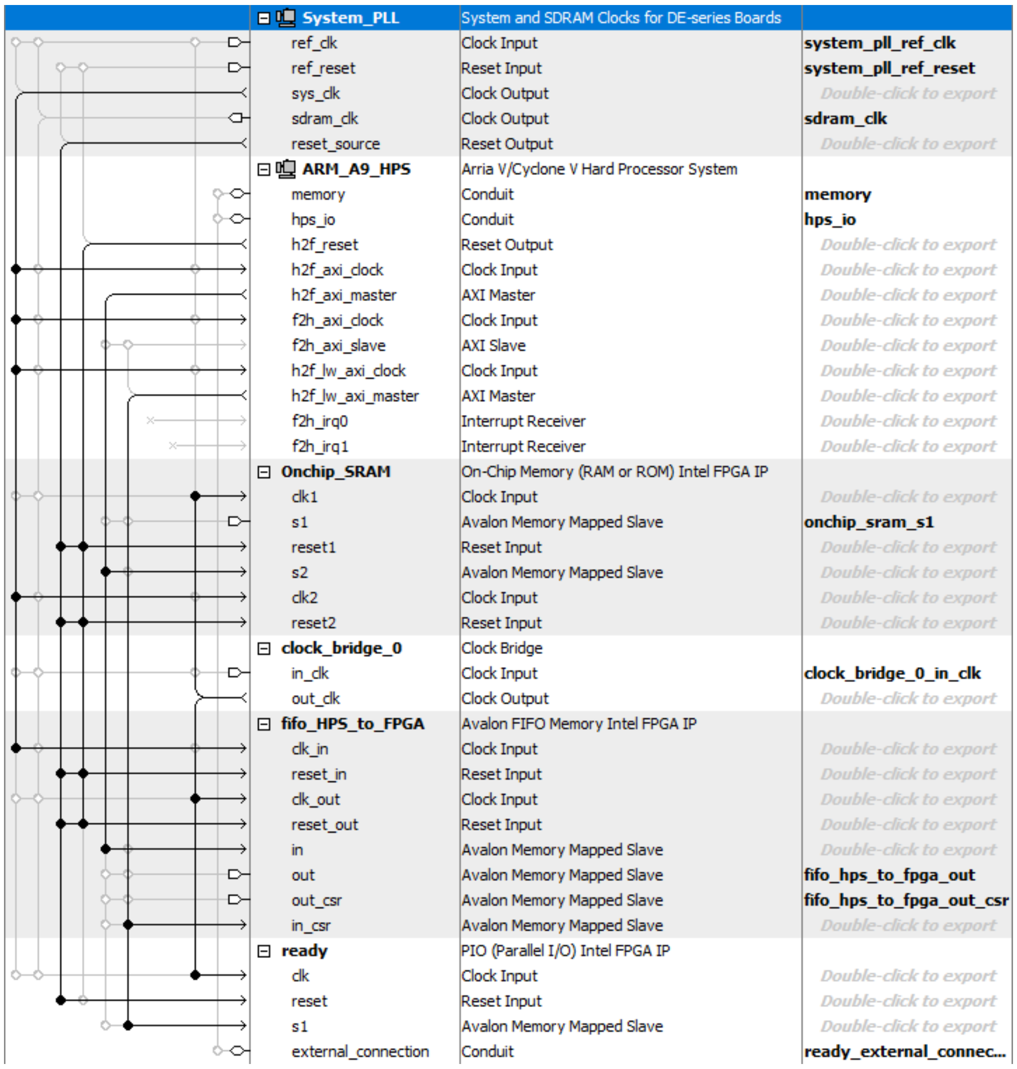
This memory is not initialized during device programming.

Figure 11: SRAM IP properties



⊟ **Onchip_SRAM**	On-Chip Memory (RAM or ROM) Intel FPGA IP		
clk1	Clock Input	Double-click to export	**clock_bridge_0_out_clk**
s1	Avalon Memory Mapped Slave	**onchip_sram_s1**	[clk1]
reset1	Reset Input	Double-click to export	[clk1]
s2	Avalon Memory Mapped Slave	Double-click to export	[clk2]
clk2	Clock Input	Double-click to export	**System_PLL_sys_clk**
reset2	Reset Input	Double-click to export	[clk2]
⊟ **clock_bridge_0**	Clock Bridge		
in_clk	Clock Input	**clock_bridge_0_in_clk**	***exported***
out_clk	Clock Output	Double-click to export	clock_bridge_0_out_clk

Figure 12: SRAM and PLL module in the bridge



System_PLL	System and SDRAM Clocks for DE-series Boards	
ref_clk	Clock Input	**system_pll_ref_clk**
ref_reset	Reset Input	**system_pll_ref_reset**
sys_clk	Clock Output	Double-click to export
sdram_clk	Clock Output	**sdram_clk**
reset_source	Reset Output	Double-click to export
ARM_A9_HPS	Arria V/Cyclone V Hard Processor System	
memory	Conduit	**memory**
hps_io	Conduit	**hps_io**
h2f_reset	Reset Output	Double-click to export
h2f_axi_clock	Clock Input	Double-click to export
h2f_axi_master	AXI Master	Double-click to export
f2h_axi_clock	Clock Input	Double-click to export
f2h_axi_slave	AXI Slave	Double-click to export
h2f_lw_axi_clock	Clock Input	Double-click to export
h2f_lw_axi_master	AXI Master	Double-click to export
f2h_irq0	Interrupt Receiver	Double-click to export
f2h_irq1	Interrupt Receiver	Double-click to export
Onchip_SRAM	On-Chip Memory (RAM or ROM) Intel FPGA IP	
clk1	Clock Input	Double-click to export
s1	Avalon Memory Mapped Slave	**onchip_sram_s1**
reset1	Reset Input	Double-click to export
s2	Avalon Memory Mapped Slave	Double-click to export
clk2	Clock Input	Double-click to export
reset2	Reset Input	Double-click to export
clock_bridge_0	Clock Bridge	
in_clk	Clock Input	**clock_bridge_0_in_clk**
out_clk	Clock Output	Double-click to export
fifo_HPS_to_FPGA	Avalon FIFO Memory Intel FPGA IP	
clk_in	Clock Input	Double-click to export
reset_in	Reset Input	Double-click to export
clk_out	Clock Input	Double-click to export
reset_out	Reset Input	Double-click to export
in	Avalon Memory Mapped Slave	Double-click to export
out	Avalon Memory Mapped Slave	**fifo_hps_to_fpga_out**
out_csr	Avalon Memory Mapped Slave	**fifo_hps_to_fpga_out_csr**
in_csr	Avalon Memory Mapped Slave	Double-click to export
ready	PIO (Parallel I/O) Intel FPGA IP	
clk	Clock Input	Double-click to export
reset	Reset Input	Double-click to export
s1	Avalon Memory Mapped Slave	Double-click to export
external_connection	Conduit	**ready_external_connec...**

Figure 13: The complete bridge design

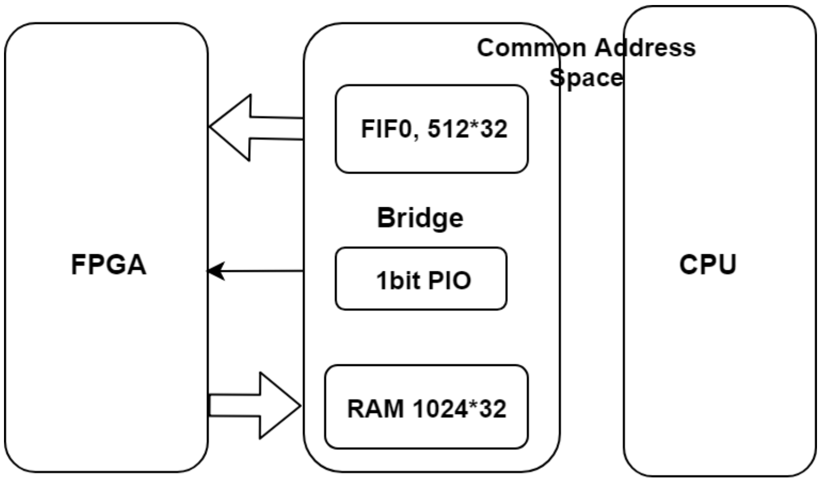


Figure 14: Bridge architecture

4.2 Implementation of FPGA components

The FPGA design mainly consists of three modules to read, process, and write data which are clocked at 50MHz. All the modules have the input signal *enable* which activates their process and sets the output signal *done* when the task is completed. The signal *done* of a module and the signal *enable* of a next module are connected to work in serial and to have a sequential data flow in the design as in figure 15.

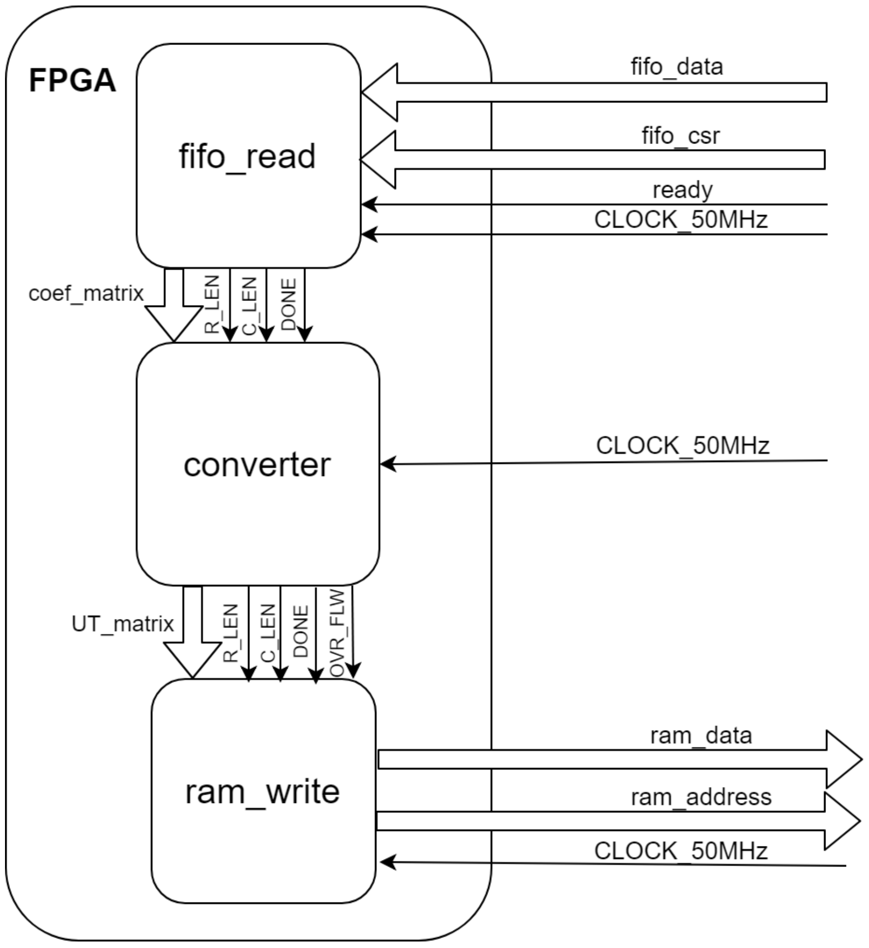


Figure 15: Top level FPGA design

4.2.1 FIFO Read Module

The module has *enable*, *fifo_data*, and *fifo_csr* as the input signals, and *coef_matrix, row_len, col_len,* and *done* as the output signals (figure 16). The *enable* signal is set to '1' by the MCU when its ready to send matrix size and data (as shown in figure 17). This activates the module to query the FIFO buffer (through the input signal *fifo_data*) using the control status register (*fifo_csr*). The first two 32-bit integer data is recognized as row and column lengths, and they are latched to their corresponding output signals. Starting from the third data, the state machine (as in figure 18) collects the coefficient matrix, transfers it to the output signal *coef_matrix* and sets signal *done* to '1'. The signal *done* is set to '0' whenever the signal *enable* is '0'.

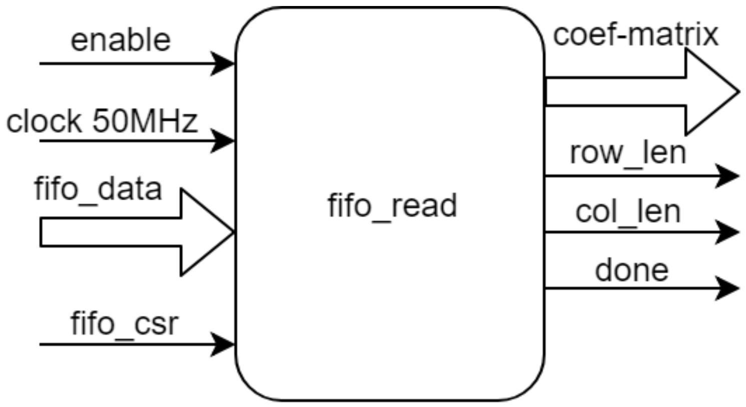
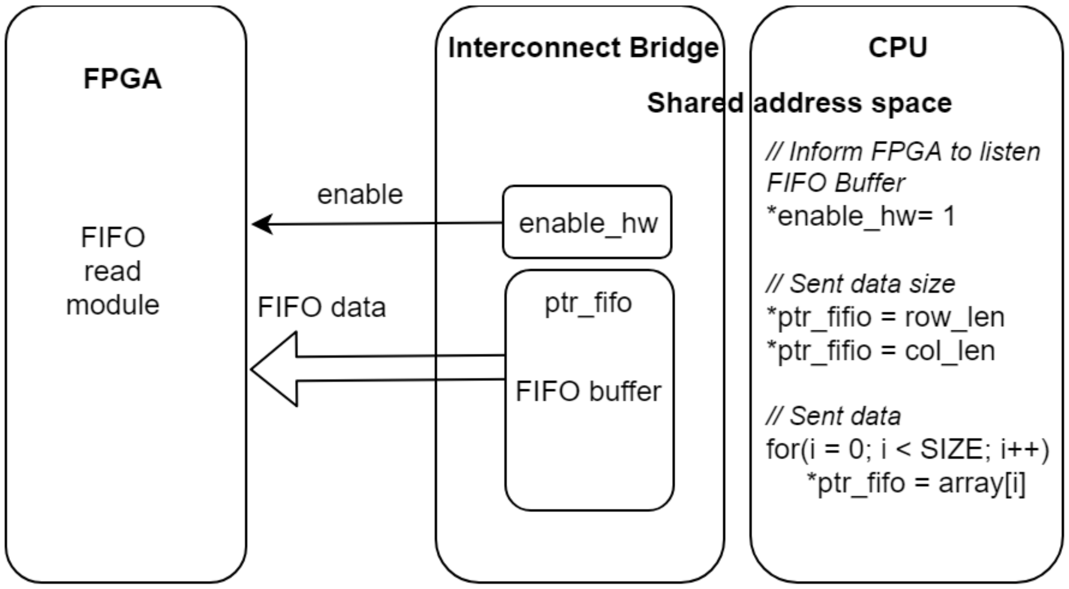


Figure 16: FIFO read entity diagram



```
// Inform FPGA to listen
FIFO Buffer
*enable_hw= 1

// Sent data size
*ptr_fifio = row_len
*ptr_fifio = col_len

// Sent data
for(i = 0; i < SIZE; i++)
    *ptr_fifo = array[i]
```

Figure 17: Data Transfer from HPS to FPGA

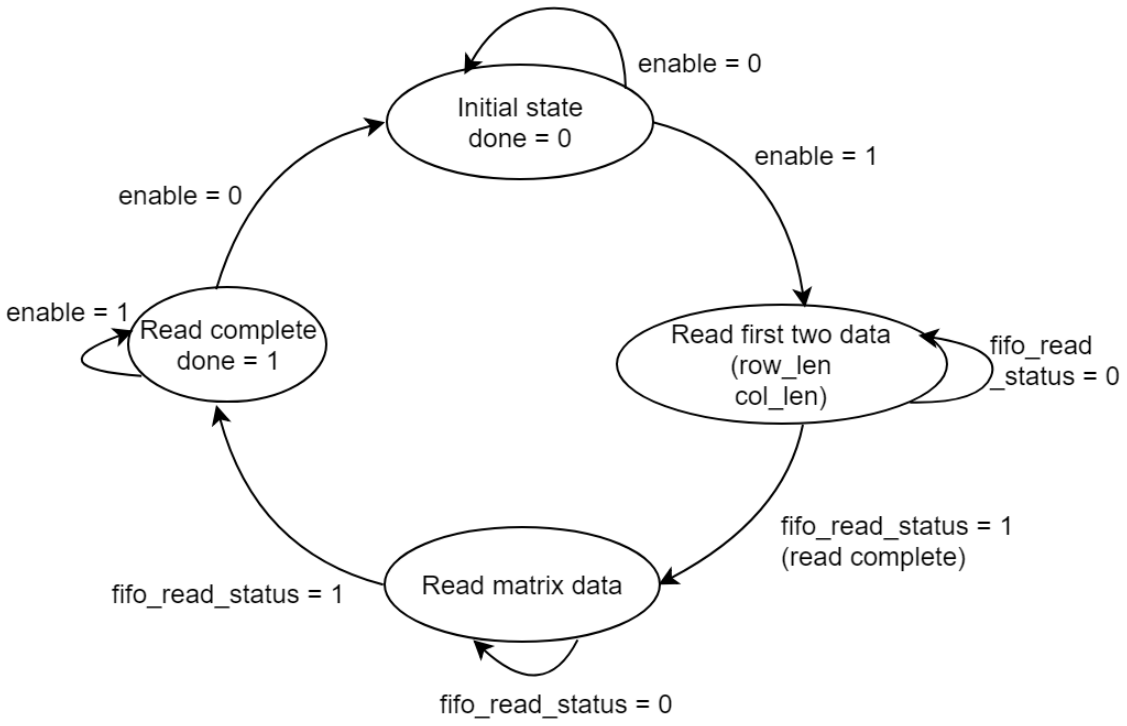


Figure 18: FIFO read module state diagram

4.2.2 Variable Reducer Module

The module has *enable, primary_row, secondary_row,* and *col_index* as the input signals and *over_flow*, *res_secondary_row*, and *done* as the output signals (figure 19). It is implemented as per the logic described in the section 3.3.1, where the primary and the secondary rows are subtracted repeatedly (as in figure 20) at the specified column to create a new row with the reduced lead variable and latched to the output signal *res_secondary_row*. If an overflow [22] is detected while performing arithmetic operations, it aborts the reduction process immediately by setting the signal *over_flow* to '1'. After the successful reduction (or while aborted), the signal *done* is set to '1' or to '0' whenever the signal *enable* is '0'. This module is used inside the converter module which is described below.

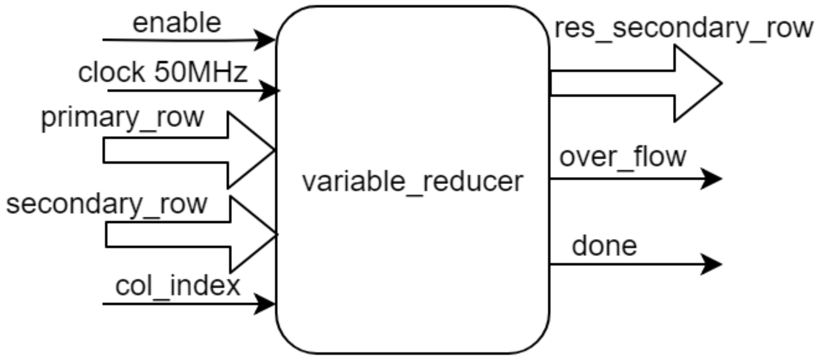


Figure 19: Variable reducer entity diagram

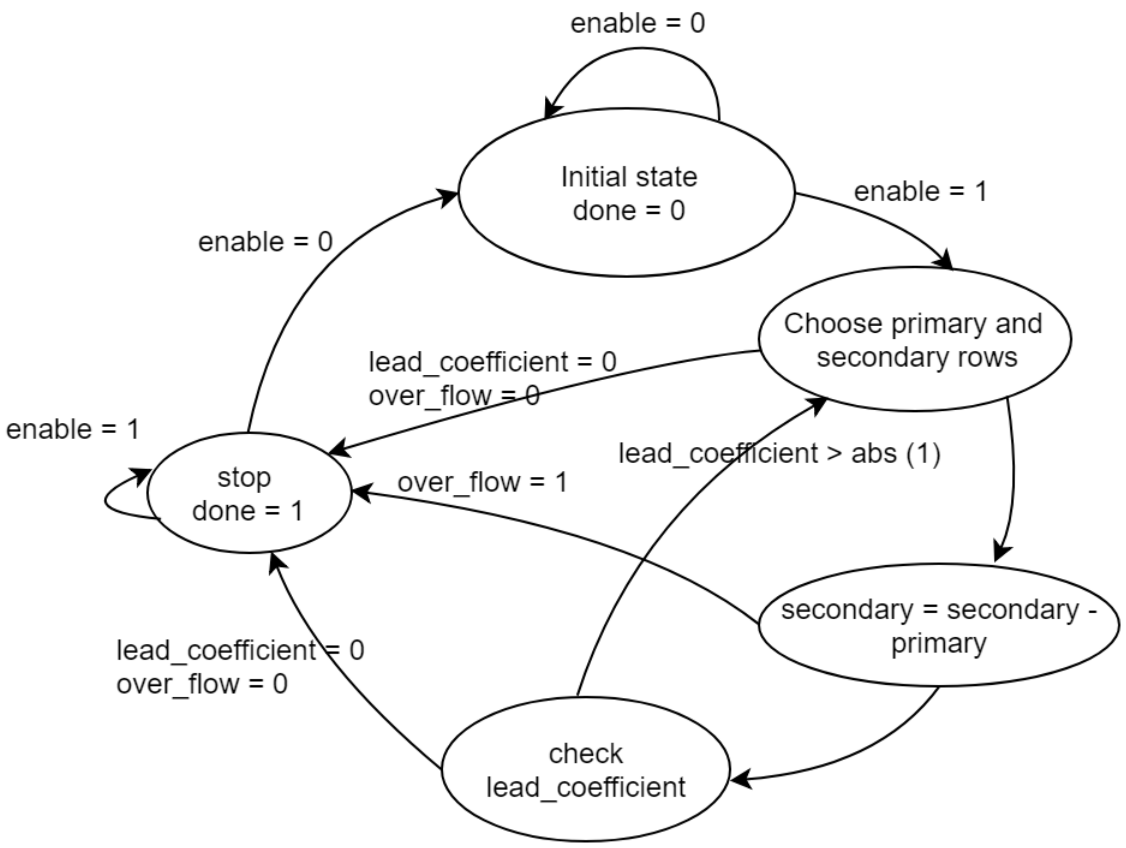


Figure 20: Variable reducer state diagram

4.2.3 Equation Swap Module

The module has *enable, matrix, row_index,* and *col_index* as the input signals and *matrix,* and *done* as the output signals (figure 21). It is implemented as per the logic described in the section 3.3.2, where the current primary row (*row_index*) and the row with the least absolute non-zero coefficient for a given column (*col_index*) are swapped (as in figure 22). After swapping the rows successfully, the signal *done* is set to '1' or to '0' whenever the signal *enable* is '0'. This module is also used inside the converter module.

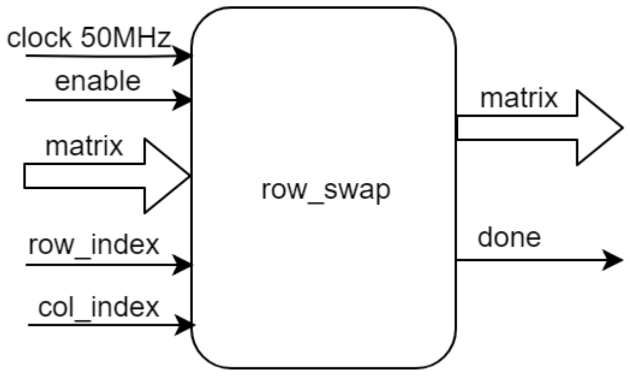


Figure 21: Row swap entity diagram

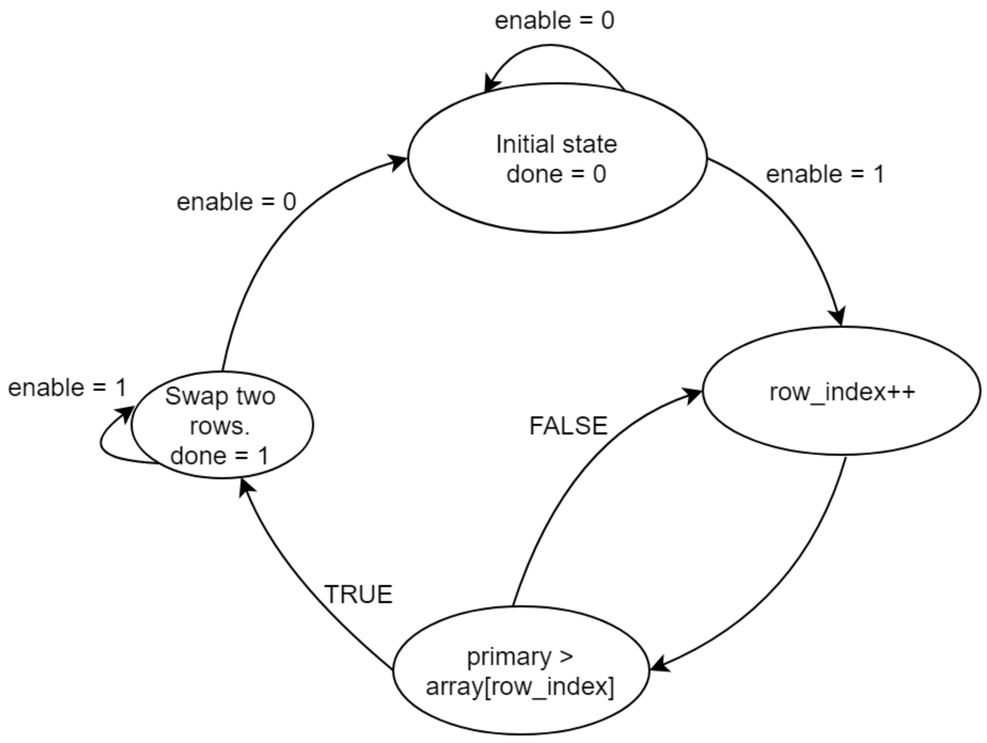


Figure 22: Row swap state diagram

4.2.4 Matrix Converter Module

The module has *enable, coef_matrix, row_len,* and *col_len* as the input signals and *UT_matrix*, *done*, and *over_flow* as the output signals (figure 23). It is implemented as per the logic described in the section 3.3.3, where the variable reducer and the swap modules are used to convert the given coefficient matrix to an upper triangular matrix. After the successful matrix conversion, the signal *done* is set to '1' or to '0' whenever the signal *enable* is '0'. The data processing is stopped immediately by writing the signal *done* to '1', when an overflow is detected (*over_flow = '1'*) by the reducer module and latched to its output signal (*over_flow*). The converter is implemented in the two variants based on the number of reducer modules used.

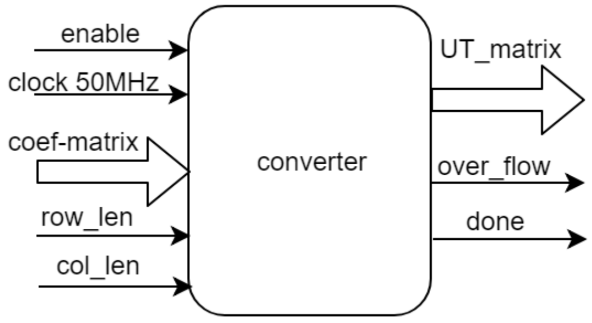


Figure 23: Matrix converter entity diagram

4.2.5 Serial Converter

A single instance of a reducer module is used to eliminate a variable (or a given *col_index*) in the serial converter (figure 25). By having the same primary row, the secondary row is incremented for every iteration until reaching the last row, and all the coefficients in the *col_index* column below the primary row are reduced to zero. The swap module is used to identify a new primary row and continue to eliminate the next variable using the same reducer module. An upper triangular matrix will be created by continuing this process until the last row (as mentioned in figure 24).

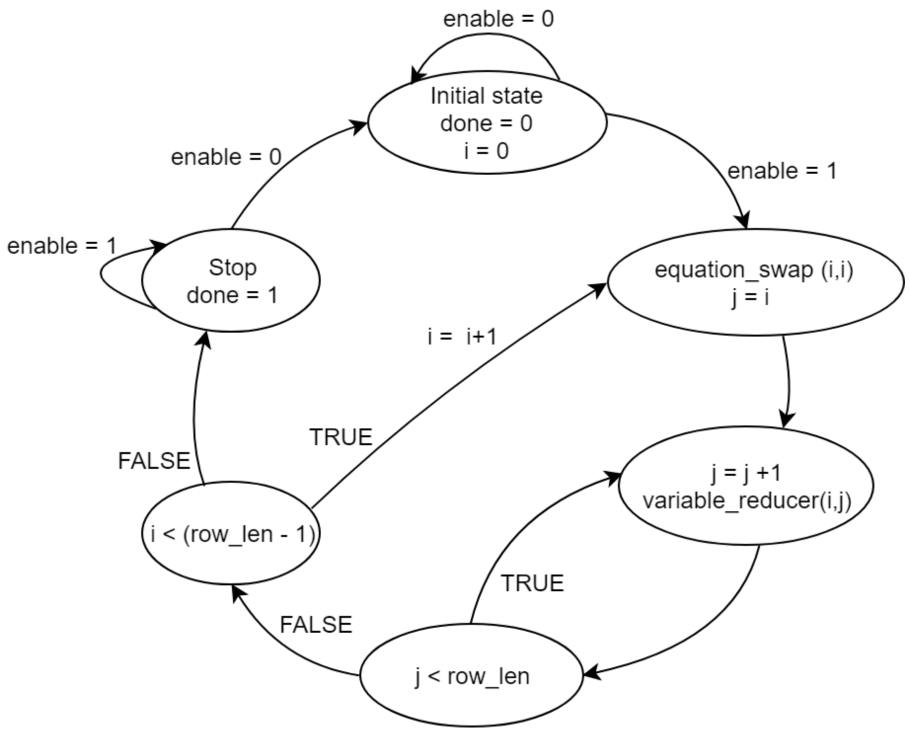


Figure 24: Serial converter state diagram

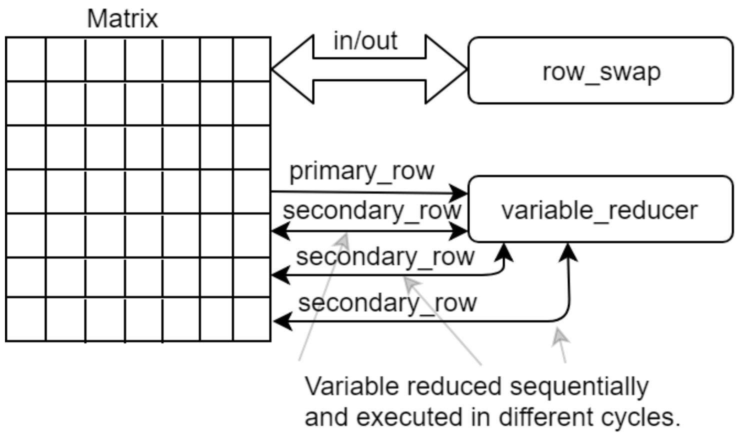


Figure 25: Serial converter logic

4.2.6 Parallel Converter

Here the *row_len* - 1 number of reducer components are executed in parallel to reduce a given variable (figure 26). For a given column, all the reducer components shall contain the same primary row and different secondary rows. All the rows below the primary row are loaded to their respective reducer module and processed simultaneously to eliminate a variable in a single iteration (as shown in figure 27). The total time taken for a column reduction is the time taken by the slowest reducer module, whereas in the serial converter, it is the summation of time taken to process all the secondary rows individually. Thus the parallel converter is expected to be faster than the serial converter.

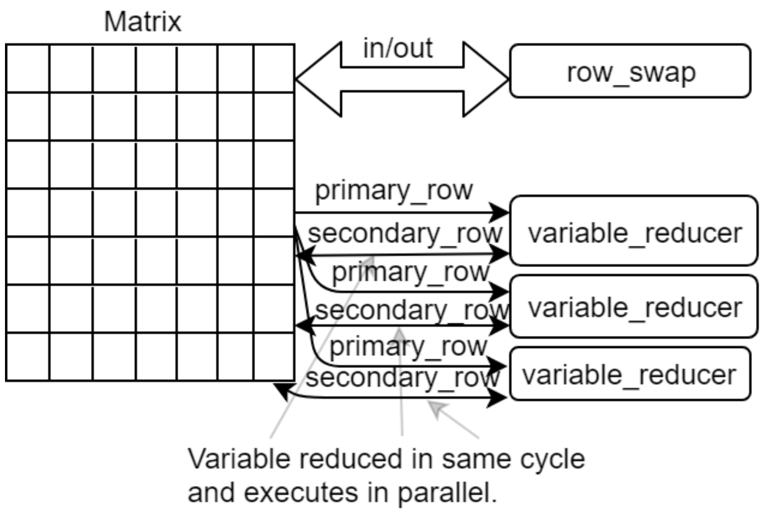


Figure 26: Parallel converter logic

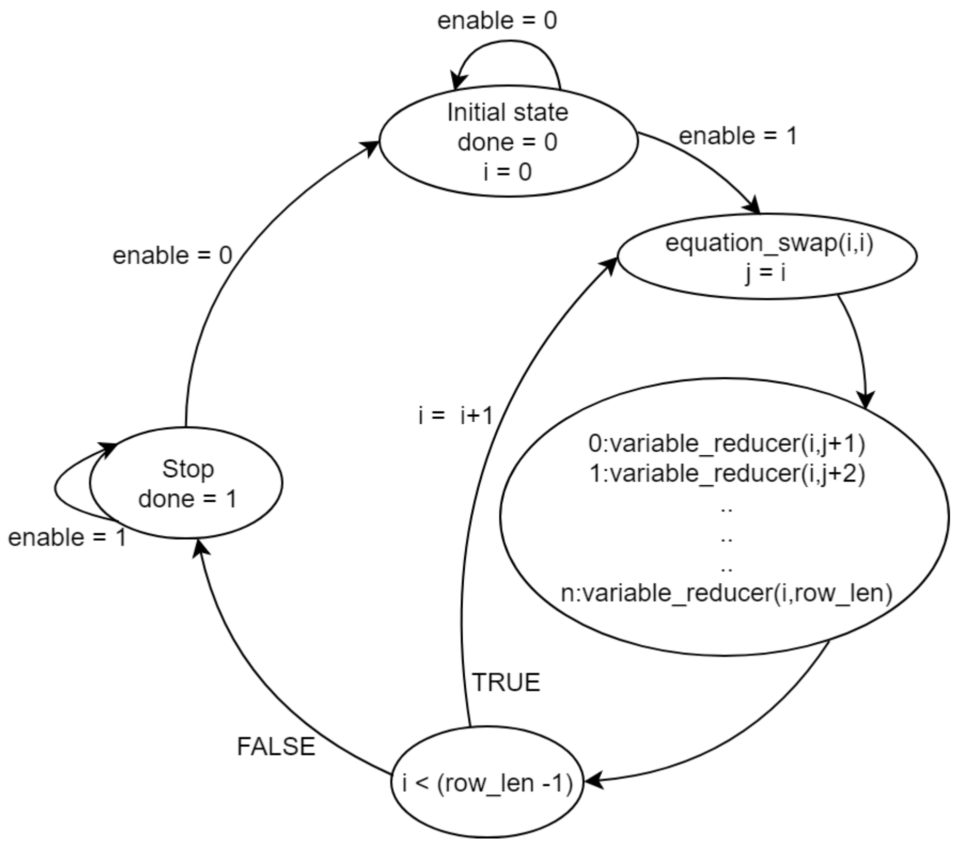


Figure 27: Parallel converter state diagram

4.2.7 SRAM Write Module

The module has *enable, over_flow, upper_triangular_matrix, row_len,* and *col_len* as the input signals and *ram_data,* and *done* as the output signals (figure 28). By detecting '1' in the signal *enable*, the converted matrix is copied to the SRAM (*ram_data*) from the second location (address 0x1). During the detection of overflow, the address 0x0 (also called status location) is updated with -1, and when no overflow has occurred and the matrix is copied successfully to the SRAM, the status is updated by the transferred data size (figure 30). The MCU in the other end will reset the status before asserting the signal *enable_hw* and poll the status for a non-zero value after transferring the matrix (figure 29). Based on the return status, the software shall read the converted matrix or report overflow. The signal *done* is set to '1' after updating the status, but this signal is not used in the design as this is the last module. Similar to other modules, it also sets the signal *done* to '0' whenever the signal *enable* is '0'.

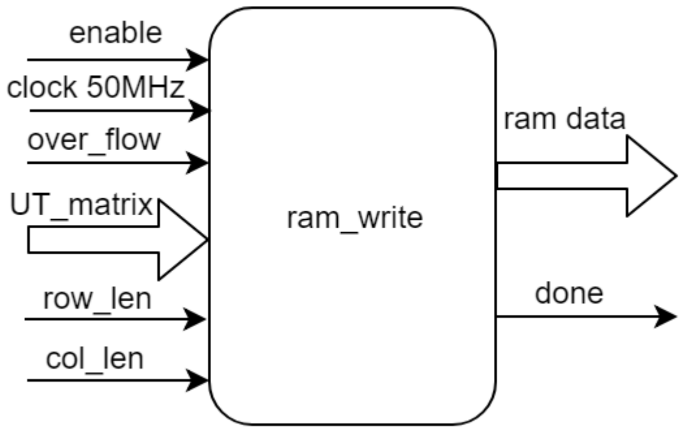
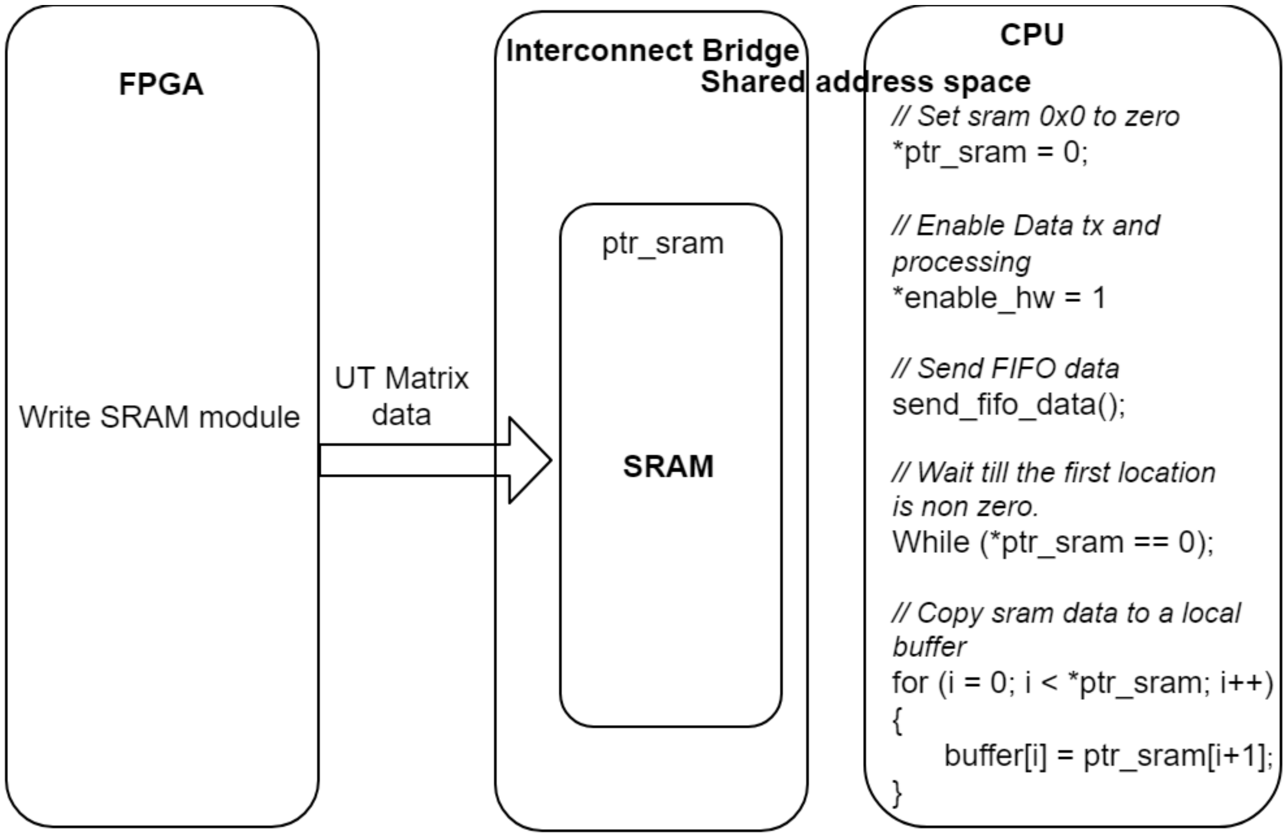


Figure 28: SRAM write entity diagram



```
// Set sram 0x0 to zero
*ptr_sram = 0;

// Enable Data tx and
processing
*enable_hw = 1

// Send FIFO data
send_fifo_data();

// Wait till the first location
is non zero.
While (*ptr_sram == 0);

// Copy sram data to a local
buffer
for (i = 0; i < *ptr_sram; i++)
{
    buffer[i] = ptr_sram[i+1];
}
```

Figure 29: Data Transfer from FPGA to HPS

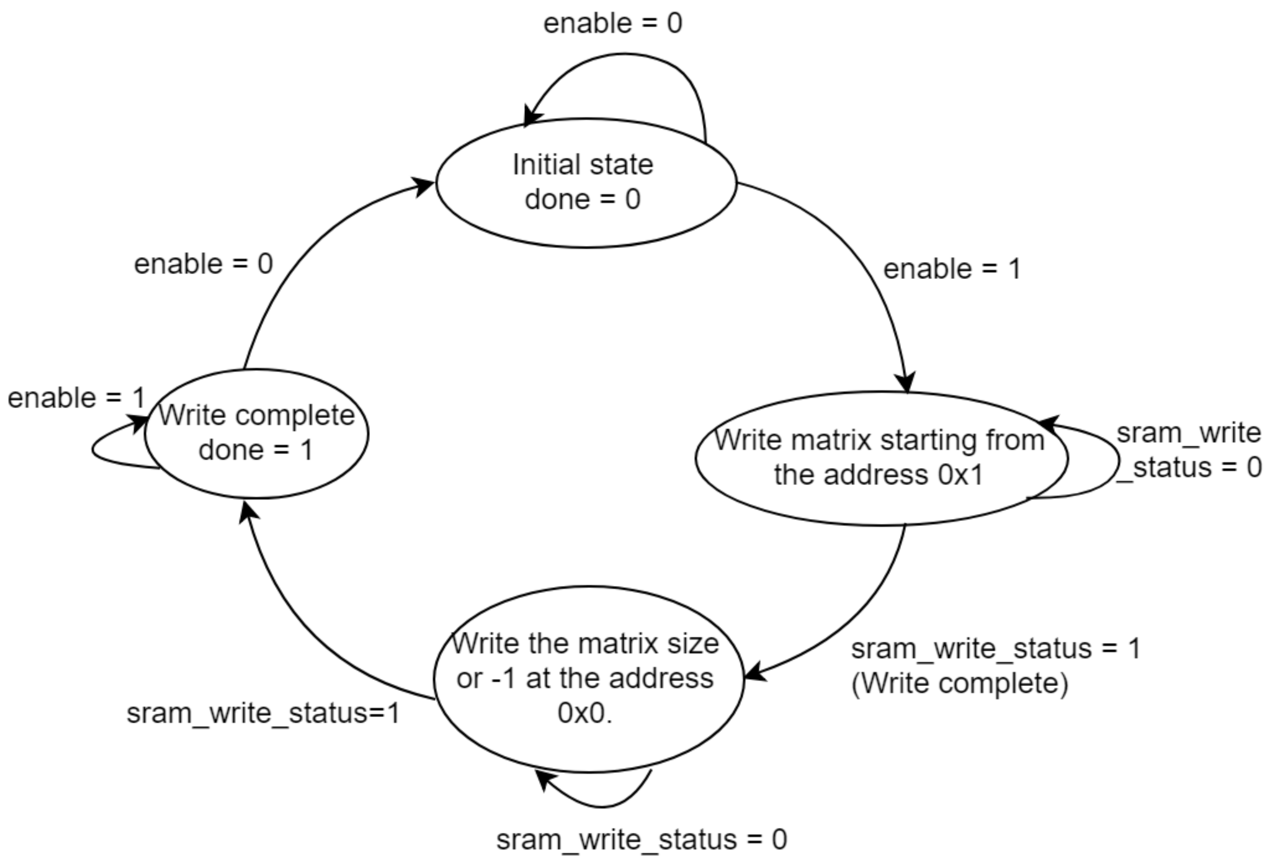


Figure 30: SRAM write state diagram

4.3 Top Level Module

The top-level module (figure 31) contains all the modules connected serially using the *port map* instruction in VHDL [19]. The design contains a package file where the following properties can be altered. These are compile time values and can not be changed after the hardware synthesis.

1. ROW_LEN & COL_LEN: Row and column lengths of the coefficient matrix. The default values are based on the converter type and the hardware utilization.

2. INT_WIDTH: Width of an integer. The default value is set to 32-bits.

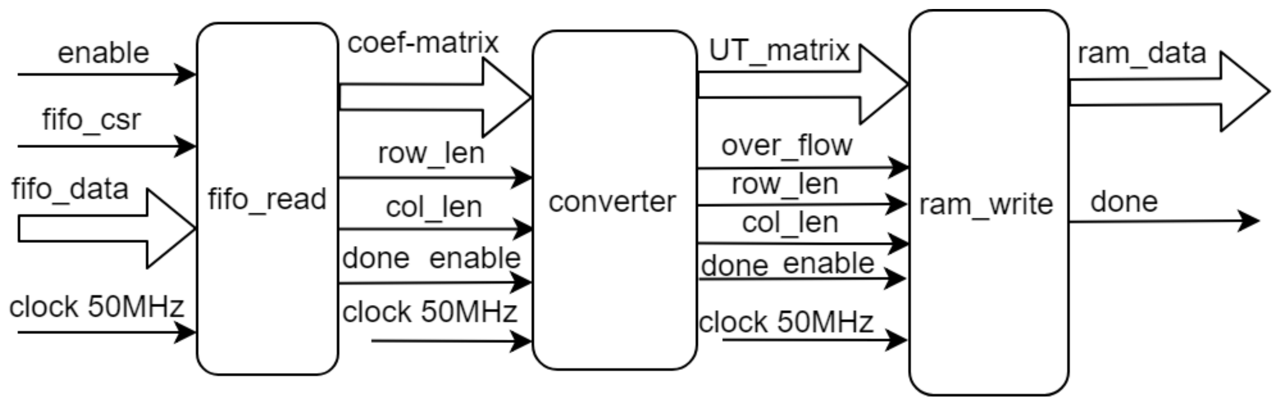


Figure 31: FPGA top level module

4.4 SoC Solver Implementation

The solver using the FPGA matrix converter is called SoC solver. The addresses of the bridge components are retrieved through the memory device (/dev/mem) using the memory-map C function (mmap()), and the shared addresses are generated in the Platform Designer tool. Through this, the base address of the FIFO, the SRAM, and the PIO components are derived and assigned to a pointer variables *sram*, *fifo*, and *enable_hw* respectively (refer the listing 4). Even though the default value of the PIO signal is zero, its safe to reset the signal *enable_hw* and disable all the FPGA modules at the beginning. After reading a coefficient matrix from an external file, set the signal *enable_hw* and copy the row and column lengths to the *fifo* pointer followed by the entire matrix. This will transfer the data to the FIFO buffer. After this, the software waits until the SRAM status location contains a non-zero value, which is set to zero before enabling the hardware. If the status contains a positive integer, it represents the number of 32-bit data available from the address 0x1. If it is -1, terminate the solver without further processing due to overflow detection. The signal *enable_hw* is reset in the end irrespective of the conversion status to reset all the FPGA modules so that they can process the next data set. After reading the converter matrix, the post-processing is performed in the end.

Listing 3: Main function of the SoC solver

```c
int *sram, *fifo, *enable_hw;
int *matrix, row_len, col_len, data_len;

/* Load shared addresses of bridge components */
load_base_address(sram, fifo, enable_hw);
/* Disable hardware */
*enable_hw = 0;
/* Read matrix */
int *matrix = read_matrix(&row_len, &col_len);

data_len = row_len * col_len;

/* Set status location in SRAM to zero */
*sram = 0;

/* Enable hardware to receive FIFO data.
*enable_hw = 1;

/* Send the first & second data */
*fifo = row_len; *fifo = col_len;

/* Transfer entire matrix */
for (int i = 0; i < data_len; i++) {
    *fifo = matrix[i];
}

/* Wait for matrix conversion */
while (*sram == 0);
/* Solve the triangular matrix during no overflow */
if (*sram > 0) {
    /* Copy the triangular matrix from SRAM */
    for (int i = 0; i < *sram; i++) {
         matrix[i]  = *(sram+1+i);
    }
    post_processing(matrix);
} else {
    printf("Overflow");
}
/* Reset all FPGA modules to process next matrix */
*enable_hw = 0;
```

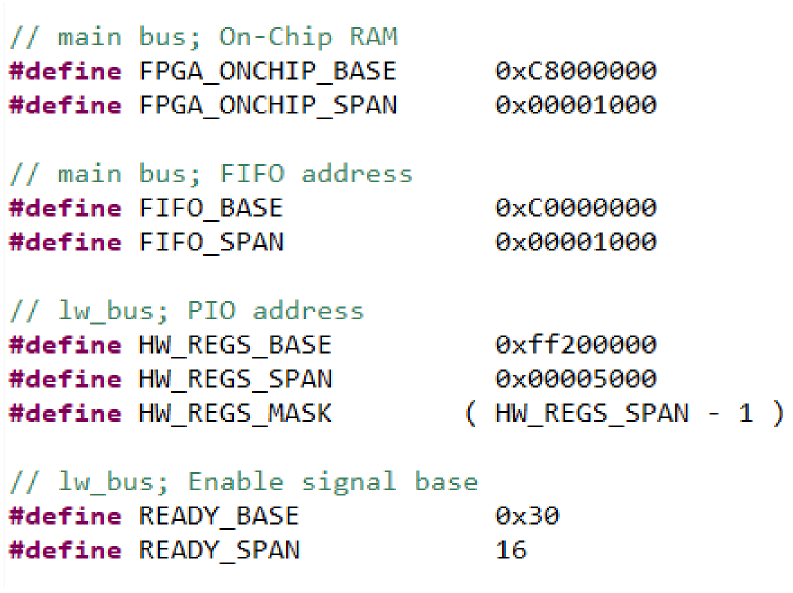
5 Results and Verification

5.1 FPGA Compilation and Synthesis Results

The serial and the parallel matrix converters are compiled and synthesized successfully. The parallel design is compiled for 7x7 matrix due to the limitations in DSP blocks. It contains six variable reducer modules where each has 14 (2 x column length) adder/subtracter and multiplier hardware that are used in the row reduction and the overflow detection modules.

The serial converter is compiled for 15x8 matrix due to the limitations in the logic utilization of Adaptive Logic Modules (ALM). The ALMs are used to latch signals, registers, and to store matrix data at a particular clock event [7], especially while activating the *enable* and *done* signals at intermediate stages. This design is also compiled for the same number of rows and columns supported by the parallel design for speed comparison.

The interconnect bridge is compiled before the synthesis process. It generates the *hps* entity containing IO ports for all components required for data transfer and included in the top-level design. It also generates a list of base and span addresses of all the components to be accessed from the MCU software. The generated address for the design is as mentioned in below figure 32.



```
// main bus; On-Chip RAM
#define FPGA_ONCHIP_BASE      0xC8000000
#define FPGA_ONCHIP_SPAN      0x00001000

// main bus; FIFO address
#define FIFO_BASE             0xC0000000
#define FIFO_SPAN             0x00001000

// lw_bus; PIO address
#define HW_REGS_BASE          0xff200000
#define HW_REGS_SPAN          0x00005000
#define HW_REGS_MASK        ( HW_REGS_SPAN - 1 )

// lw_bus; Enable signal base
#define READY_BASE            0x30
#define READY_SPAN            16
```

Figure 32: Generated shared address

The compilation and synthesize took approximately 30 minutes for a single design on a machine having Intel i7 processor with 32GB RAM, and Quartus Prime Lite, version 18.0 software (a cost free edition). The compilation would have been faster by using a licensed edition of the Quartus software. The detailed resource utilization is as mentioned in table 10. The solutions are designed to work on an input matrix which is less than or equal to the synthesized matrix size and able to process many inputs through re-enabling the signal *enable_hw*, which resets the internal states and signals of all the FPGA modules. The complete design and source code files are available at the following link.
Source Code: **https://github.com/Jagadeepram/linear_equation_solver_DE1_SoC**

Resources	**Serial (R=15, C=8)**	**Serial (R=7, C=7)**	**Parallel (R=7, C=7)**
Family	Cyclone V	Cyclone V	Cyclone V
Device	5CSEMA5F31C6	5CSEMA5F31C6	5CSEMA5F31C6
Timing Models	Final	Final	Final
Logic utilization (in ALMs)	**27,889/32,070 (87%)**	9,745/32,070 (30%)	17,459/32,070 (54 %)
Total registers	20525	9840	12263
Total pins	323/457 (71%)	323/457 (71%)	323/457 (71%)
Total virtual pins	0	0	0
Total block memory bits	24,576/4,065,280 (<1%)	24,576/4,065,280 (<1%)	24,576/4,065,280 (<1%)
Total RAM Blocks	4/397 (1%)	4/397 (1%)	4/397 (1%)
Total DSP Blocks	16/87 (18%)	14/87 (16 %)	**84/87 (97 %)**
Total HSSI RX PCSs	0	0	0
Total HSSI PMA RX Deserializers	0	0	0
Total HSSI TX PCSs	0	0	0
Total HSSI PMA TX Serializers	0	0	0
Total PLLs	1/6 (17%)	1/6 (17%)	1/6 (17%)
Total DLLs	1/4 (25%)	1/4 (25%)	1/4 (25%)

Table 10: FPGA Resource Utilization

5.2 SoC and Software Solvers

The solver applications are compiled successfully and their binary names are as follows:

1. **soc_solver**: A simple LIA solver contains either the serial or the parallel matrix converter and the software-based post-processing. It is also called serial solver and parallel solver respectively.

2. **soft_solver**: A simple LIA solver contains both the converter and the post-processing implemented in the MCU.

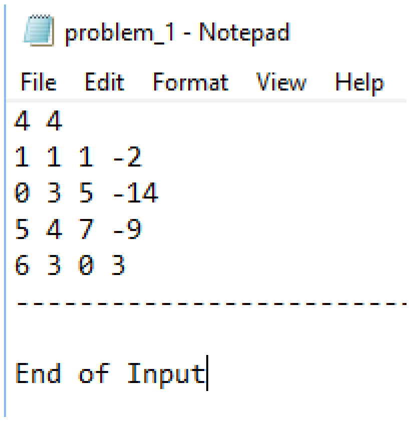
They are compiled using a build system that builds a specific application if an argument matches a binary name. For example, the following command builds the soc_solver.

```
$ make soc_solver
```

To compile all the applications, run the make command without an argument.

```
$ make
```

The solvers accepts a string argument that contains name of a text file. The file contains row and column lengths in the first line followed by the matrix as shown in figure 33. It represents coefficients of a linear integer system where all equations are equal to zero and having column-1 number of variables.



```
4 4
1 1 1 -2
0 3 5 -14
5 4 7 -9
6 3 0 3
------------------------

End of Input
```

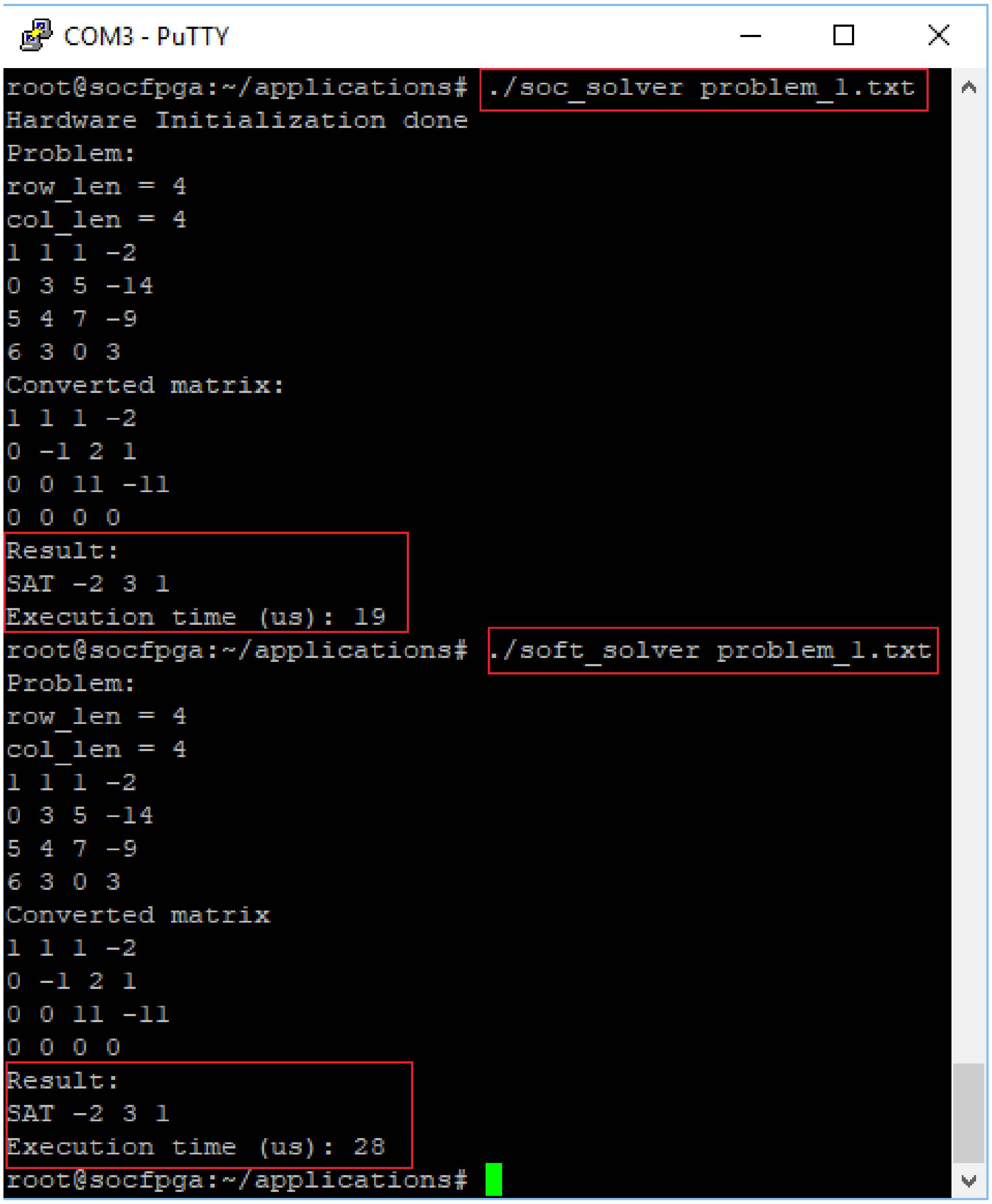
Figure 33: First example problem

After reading the input file, the solver prints the input coefficient matrix, the upper triangular matrix, and followed by the result as shown in figure 38.

Note: Either the serial or the parallel converter should be programmed to the FPGA before executing the soc_solver applications.

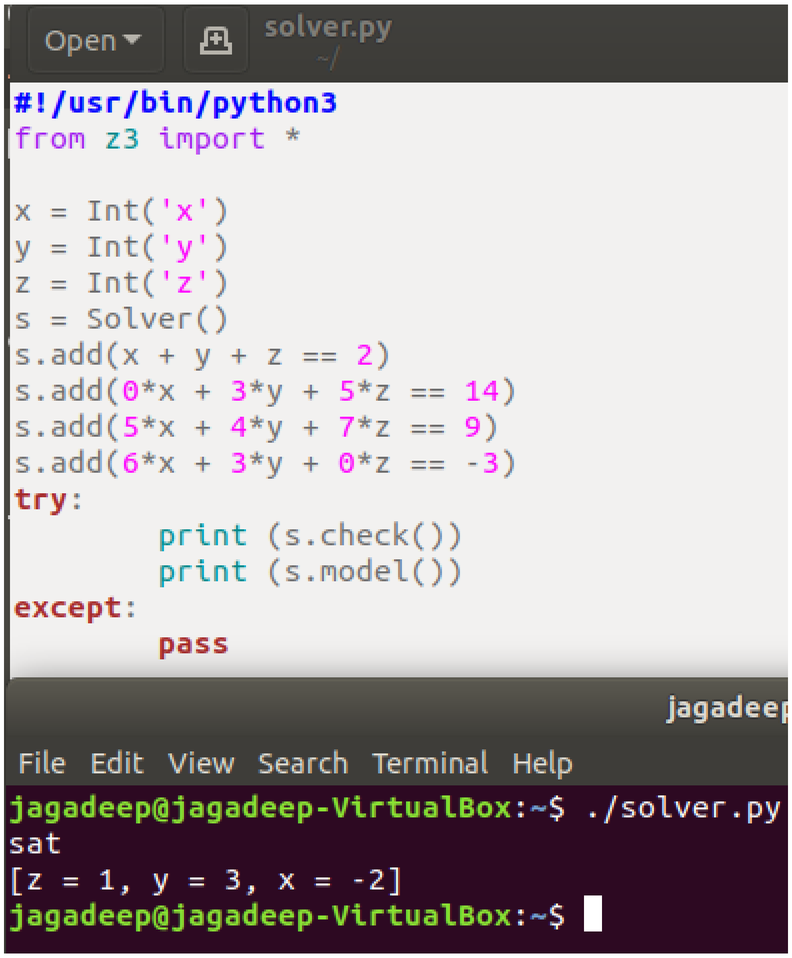
5.3 Design Verification

The examples discussed in chapter 3 are executed using the serial and the software solvers and verified with Z3, an existing SMT solver [26]. It is observed that in all the cases, the serial solver is faster than the software solver and results match with the Z3 solver. Note that for the second example, the designed solvers return SAT, whereas the Z3 return a solution. This is an important limitation considered while designing the solver, where a solution is not possible when the number of equations is less than the number of variables. Several linear integer systems with a predefined solution are created manually and successfully verified for the correctness against the Z3 solver. The following figures (from figure 34 to figure 39) shows the execution of the designed solvers on the target Linux and the Z3 solver on the host machine for the examples.



```
root@socfpga:~/applications# ./soc_solver problem_1.txt
Hardware Initialization done
Problem:
row_len = 4
col_len = 4
1 1 1 -2
0 3 5 -14
5 4 7 -9
6 3 0 3
Converted matrix:
1 1 1 -2
0 -1 2 1
0 0 11 -11
0 0 0 0
Result:
SAT -2 3 1
Execution time (us): 19
root@socfpga:~/applications# ./soft_solver problem_1.txt
Problem:
row_len = 4
col_len = 4
1 1 1 -2
0 3 5 -14
5 4 7 -9
6 3 0 3
Converted matrix
1 1 1 -2
0 -1 2 1
0 0 11 -11
0 0 0 0
Result:
SAT -2 3 1
Execution time (us): 28
root@socfpga:~/applications#
```

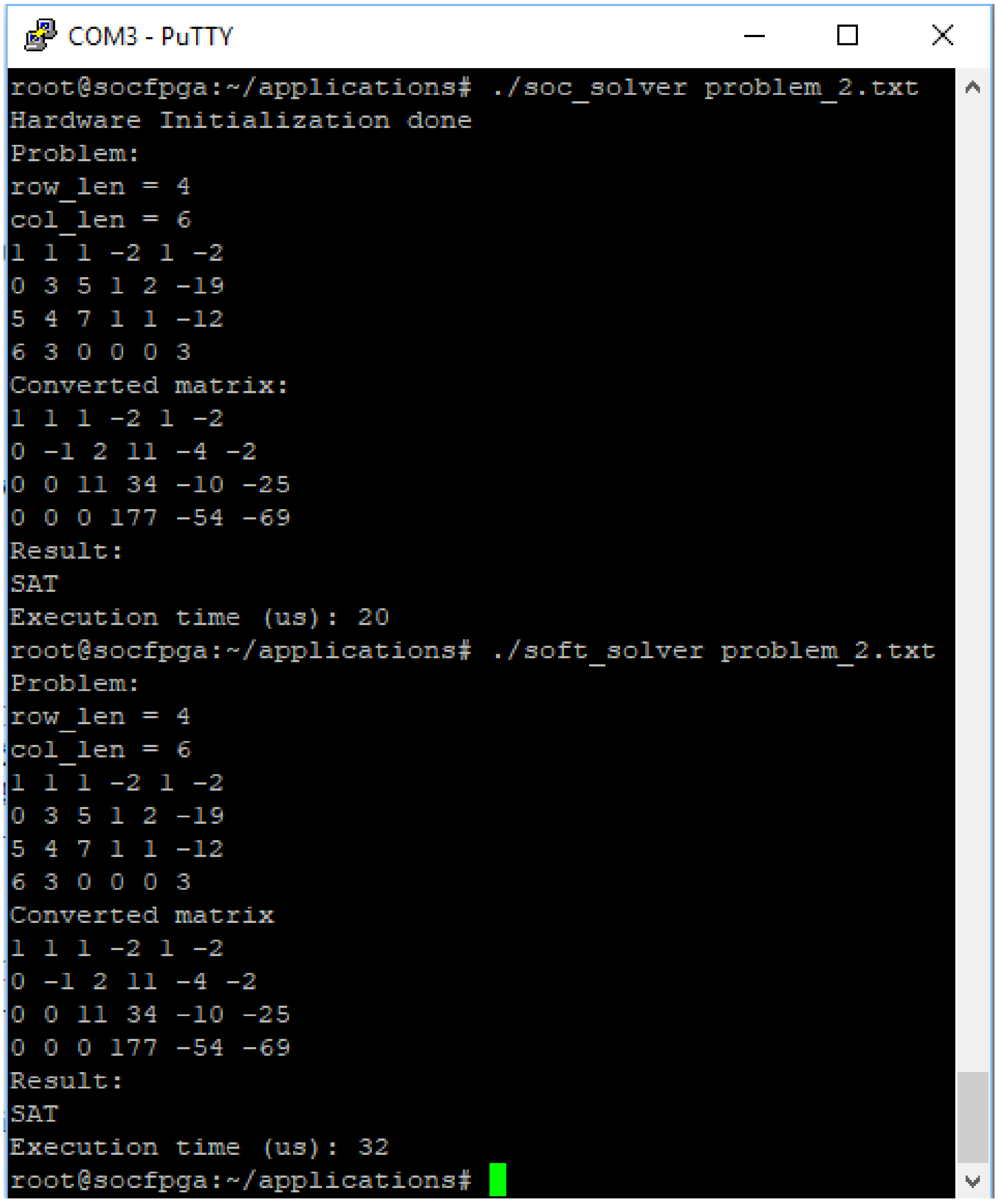
Figure 34: Simple LIA Solver: Solution1



```
#!/usr/bin/python3
from z3 import *

x = Int('x')
y = Int('y')
z = Int('z')
s = Solver()
s.add(x + y + z == 2)
s.add(0*x + 3*y + 5*z == 14)
s.add(5*x + 4*y + 7*z == 9)
s.add(6*x + 3*y + 0*z == -3)
try:
        print (s.check())
        print (s.model())
except:
        pass
```

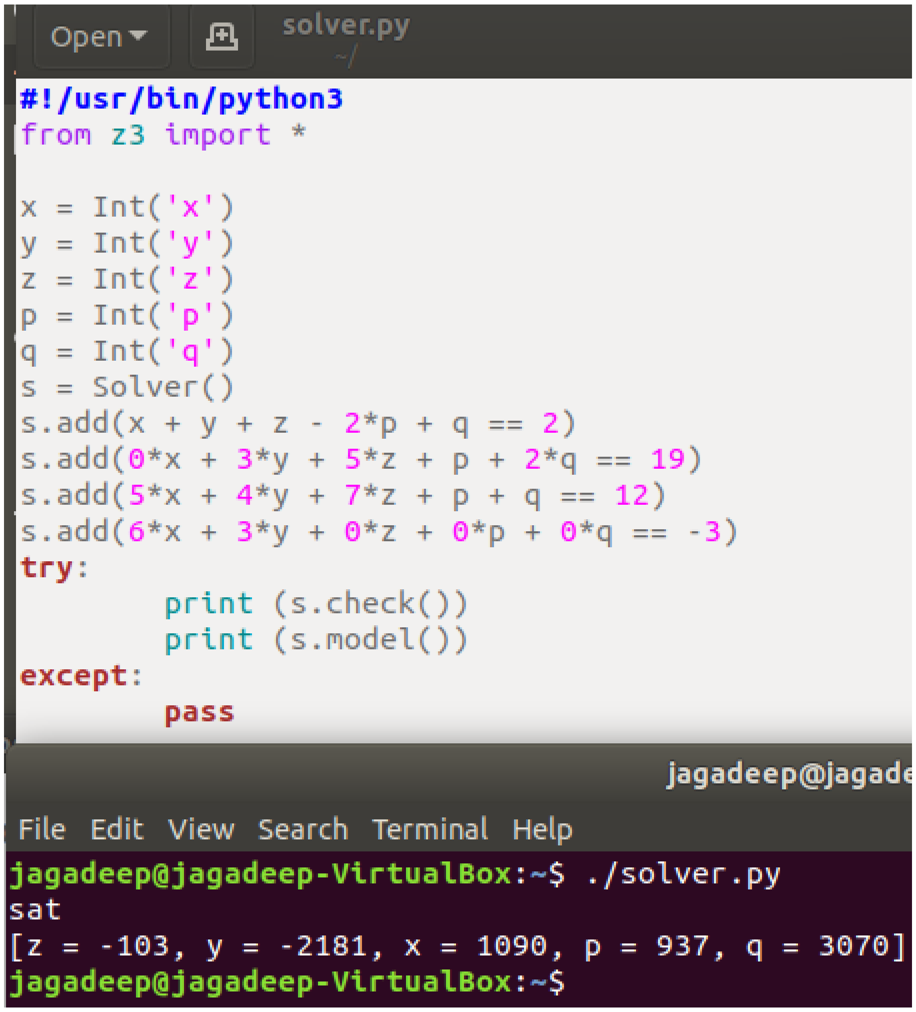
```
jagadeep@jagadeep-VirtualBox:~$ ./solver.py
sat
[z = 1, y = 3, x = -2]
jagadeep@jagadeep-VirtualBox:~$
```

Figure 35: Z3 SMT Solver: Solution1


COM3 - PuTTY

```
root@socfpga:~/applications# ./soc_solver problem_2.txt
Hardware Initialization done
Problem:
row_len = 4
col_len = 6
1 1 1 -2 1 -2
0 3 5 1 2 -19
5 4 7 1 1 -12
6 3 0 0 0 3
Converted matrix:
1 1 1 -2 1 -2
0 -1 2 11 -4 -2
0 0 11 34 -10 -25
0 0 0 177 -54 -69
Result:
SAT
Execution time (us): 20
root@socfpga:~/applications# ./soft_solver problem_2.txt
Problem:
row_len = 4
col_len = 6
1 1 1 -2 1 -2
0 3 5 1 2 -19
5 4 7 1 1 -12
6 3 0 0 0 3
Converted matrix
1 1 1 -2 1 -2
0 -1 2 11 -4 -2
0 0 11 34 -10 -25
0 0 0 177 -54 -69
Result:
SAT
Execution time (us): 32
root@socfpga:~/applications#
```

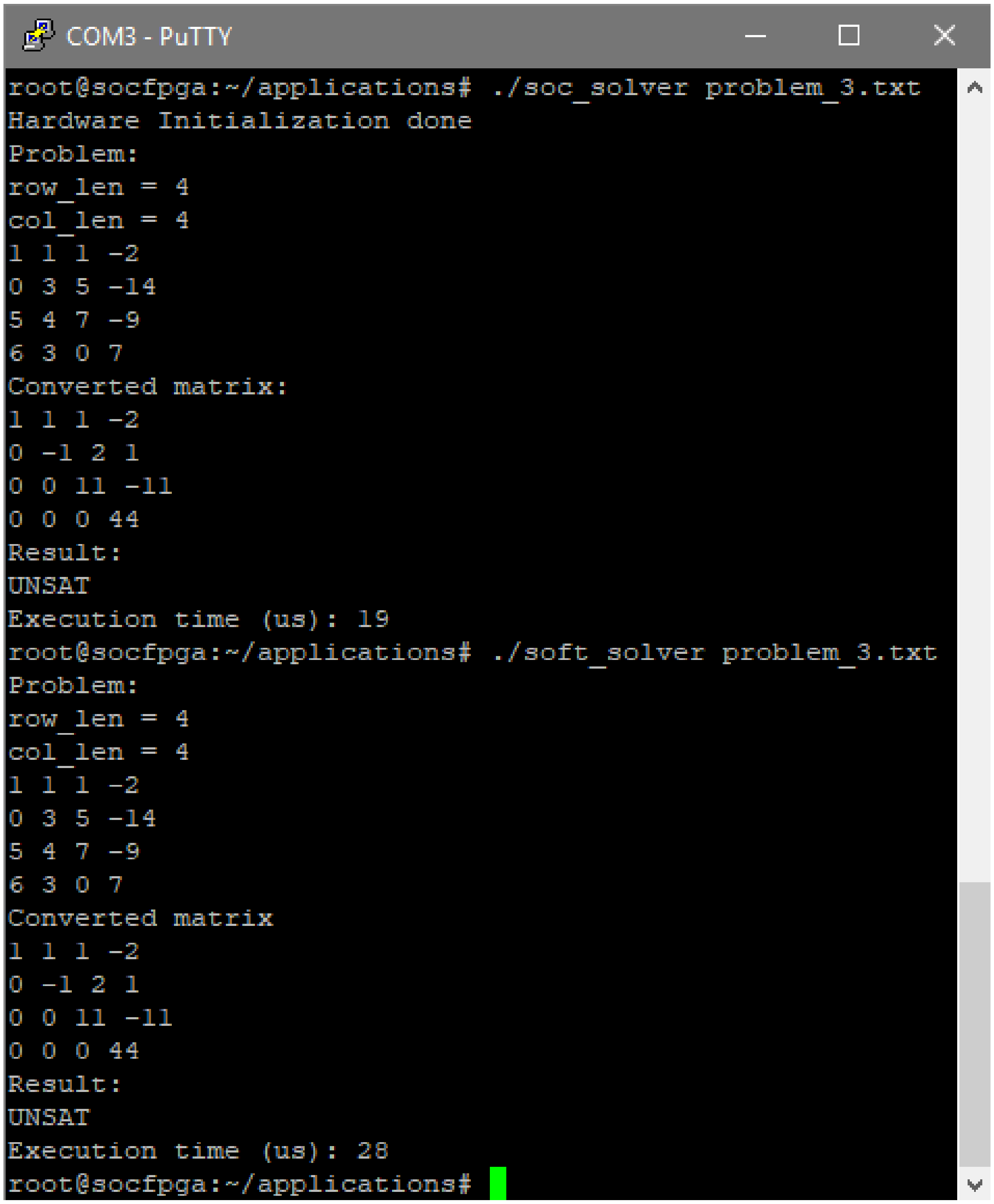
Figure 36: Simple LIA Solver: Solution2



```
#!/usr/bin/python3
from z3 import *

x = Int('x')
y = Int('y')
z = Int('z')
p = Int('p')
q = Int('q')
s = Solver()
s.add(x + y + z - 2*p + q == 2)
s.add(0*x + 3*y + 5*z + p + 2*q == 19)
s.add(5*x + 4*y + 7*z + p + q == 12)
s.add(6*x + 3*y + 0*z + 0*p + 0*q == -3)
try:
        print (s.check())
        print (s.model())
except:
        pass
```

```
jagadeep@jagadeep-VirtualBox:~$ ./solver.py
sat
[z = -103, y = -2181, x = 1090, p = 937, q = 3070]
jagadeep@jagadeep-VirtualBox:~$
```

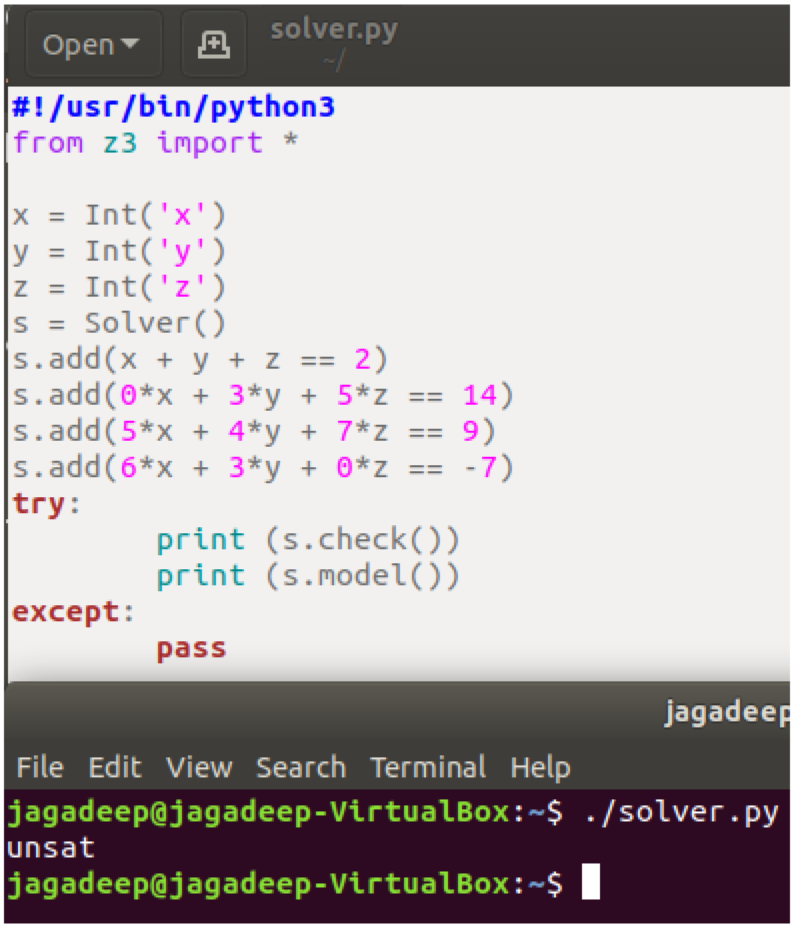
Figure 37: Z3 SMT Solver: Solution2



COM3 - PuTTY

```
root@socfpga:~/applications# ./soc_solver problem_3.txt
Hardware Initialization done
Problem:
row_len = 4
col_len = 4
1 1 1 -2
0 3 5 -14
5 4 7 -9
6 3 0 7
Converted matrix:
1 1 1 -2
0 -1 2 1
0 0 11 -11
0 0 0 44
Result:
UNSAT
Execution time (us): 19
root@socfpga:~/applications# ./soft_solver problem_3.txt
Problem:
row_len = 4
col_len = 4
1 1 1 -2
0 3 5 -14
5 4 7 -9
6 3 0 7
Converted matrix
1 1 1 -2
0 -1 2 1
0 0 11 -11
0 0 0 44
Result:
UNSAT
Execution time (us): 28
root@socfpga:~/applications#
```

Figure 38: Simple LIA Solver: Solution3



```
#!/usr/bin/python3
from z3 import *

x = Int('x')
y = Int('y')
z = Int('z')
s = Solver()
s.add(x + y + z == 2)
s.add(0*x + 3*y + 5*z == 14)
s.add(5*x + 4*y + 7*z == 9)
s.add(6*x + 3*y + 0*z == -7)
try:
        print (s.check())
        print (s.model())
except:
        pass
```

```
jagadeep@jagadeep-VirtualBox:~$ ./solver.py
unsat
jagadeep@jagadeep-VirtualBox:~$
```

Figure 39: Z3 SMT Solver: Solution3

5.4 Measurement Application

This application executes the software solver and the SoC solvers with a same set of randomly generated inputs and records their execution time in an output file. It also compares the converted matrices and asserts an error during mismatch. After programming the FPGA with a 7∗7 matrix converter, execute the application with a character argument ('s' for serial and 'p' for parallel) indicating the type of the converter. The following parameters are changed during run-time for a different set of the input matrix.

1. row_len (r): Indicates number of equations and varied from 3 to 7.
2. col_len (c): Indicates number of variables (col_len -1) and also varied from 3 to 7.
3. int_range (int): An upper limit of a coefficient, tested for the values 5, 10, 15, 20, 50, and 100.

The application stores matrix properties, timing information and other results in a file using JASON format, as in the below listing.

Listing 4: An Output JASON File

```
"test_result": {
        "row_len": 7,
        "col_len": 7,
        "int_range": 10,
        "nbr_test": 3,
        "solver_type": "PARALLEL",
        "test_cases": [
                [{"test_nbr": 0,"soft": 208,"hard": 32,
                  "overflow": "NO","result": "PASS"}],
                [{"test_nbr": 1,"soft": 227,"hard": 33,
                  "overflow": "NO","result": "PASS"}],
                [{"test_nbr": 2,"soft": 262,"hard": 45,
                  "overflow": "NO","result": "PASS"}],
        ],
        "result": "PASS"
}
```

The file contains row, column, upper limit, number of test cases, and solver type, which are the input parameters. Every test cases contain ”soft” and ”hard” values that represent their execution time in microseconds for a single input matrix. Note that the ”hard” value also includes time taken to send and receive the matrix from the FPGA. The value ”PASS” in the ”result” field indicates that the converted matrices are identical and ”NO” in the ”overflow” field indicates that overflow has not occurred during arithmetic operations. The collective result is mentioned at the bottom, and the value ”PASS” indicates that identical matrices were created throughout the test cases.

Listing 5: Measurement Application

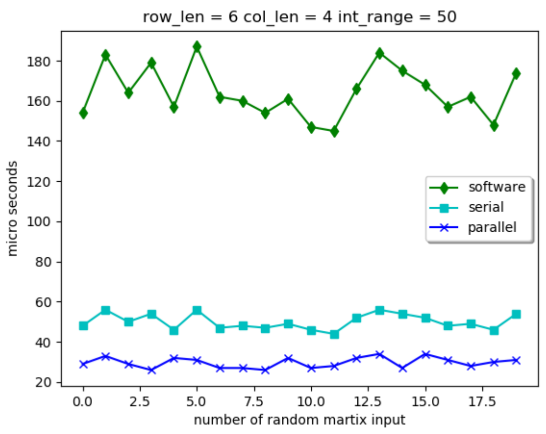
```
#define NBR_TEST_CASE 500

int *coef_matrix;
int *hard_matrix;
int *soft_matrix;
time_t time;

meas_function (int r, int c, int int_range) {
    for (int i = 0; i < NBR_TEST_CASE ; i++) {

        // Fill matrix by random numbers
        // coefficient = random()%int_range
        coef_matrix = fill_random_numbers(r, c, int_range);

        start_time(time);
        soft_matrix = soft_convert_matrix(coef_matrix);
        stop_time(time);
        // Store latency in a JSON file.
        write_time_to_file(time);

        start_time(time);
        hard_matrix = hard_convert_matrix(coef_matrix);
        stop_time(time);
        // Store latency in a JSON file.
        write_time_to_file(time);

        // Compare matrix and declare results
        if (hard_matrix != soft_matrix) {
            fprintf("Result:Fail");
            break;
        } else {
            fprintf("Result:Pass");
        }
    }
    int main (){
        for (int r = 3; r <= 7; r++)
            for (int c = 3; c <= 7; c++)
                for (int i = 5; i <= 20; i=i+5)
                    meas_function(r, c, i);
    }
}
```

5.5 Comparison of Execution Time

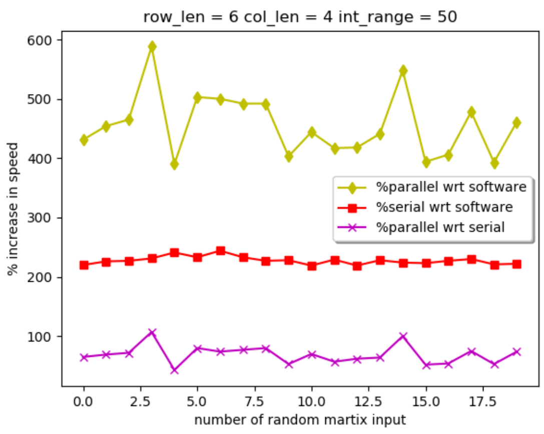
The measurement application is executed with the serial and the parallel matrix converter for different combinations of inputs. Their output files are analyzed to study the timing behavior when a particular parameter is changed (while other remains constant) for the three different solvers. The test cases only with no overflow are considered and the data round-trip time is included for the SoC solvers. Nearly 500 test cases were performed for a single matrix property, which is created through combination of different row, column, integer range, and solver type (refer the above listing of the measurement application) and the collective test case result is always reported as ”PASS” during no overflow.

5.5.1 Comparison of Solvers

Here the solver type is changed without modifying other parameters. Figure (a) plots execution time versus number of test cases for different solvers, and figure (b) plots percentage increase in speed for the serial solver with respect to the software solver, the parallel solver with respect to the software solver and the parallel solver with respect to the serial solver.



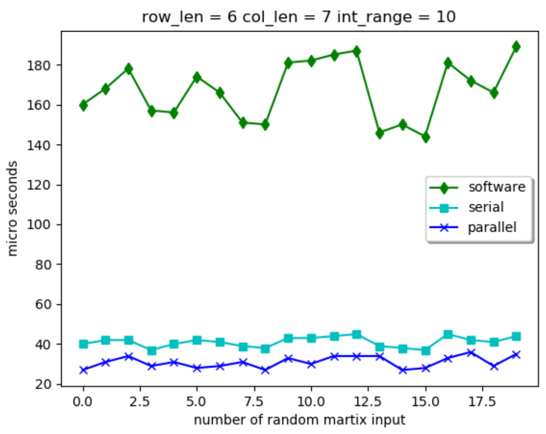
(a) Execution time



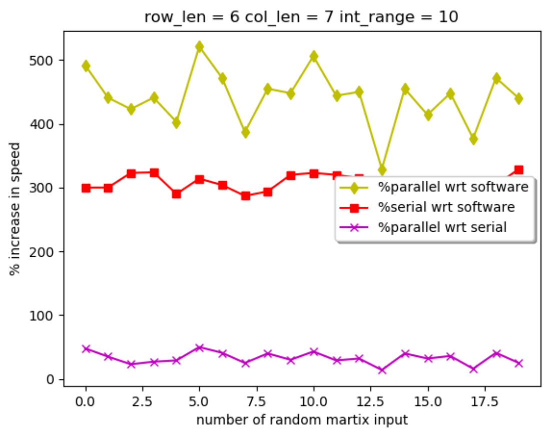
(b) Speedup

Figure 40: Compare solvers: row length = 6, column length = 4, integer range = 50

In all the cases for different row lengths, column lengths, and integer ranges, the parallel solver is the fastest followed by the serial solver (refer figure 40a). This behaviour is expected as the parallel solver performs the same instructions on different data sets using the dedicated reducer modules, whereas the serial solver contains a single module and performs a single instruction on a single data set and the software reducer performs a single instruction on a single data element. Observe their percentage differences in the execution time to understand the efficiency of the SoC solvers in figure 40b. The following figures (from figure 41 to figure 43) shows the execution time and the speedup for the different solvers for various row, column, and integer values.

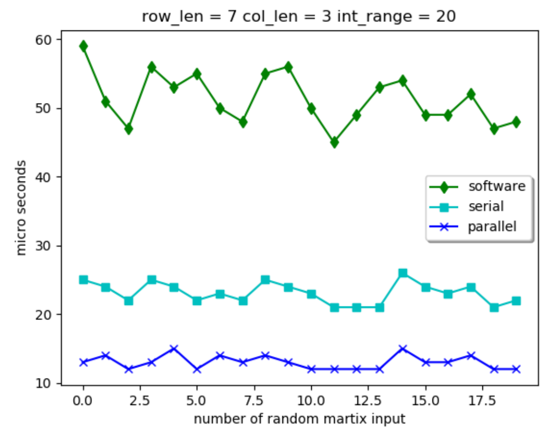


(a) Execution time

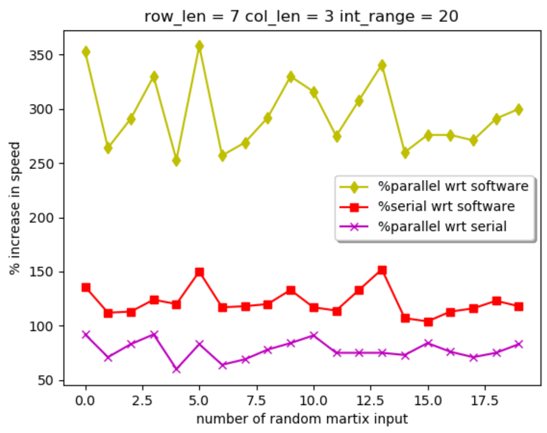


(b) Speedup

Figure 41: Compare solvers: row length = 6, column length = 7, integer range = 10

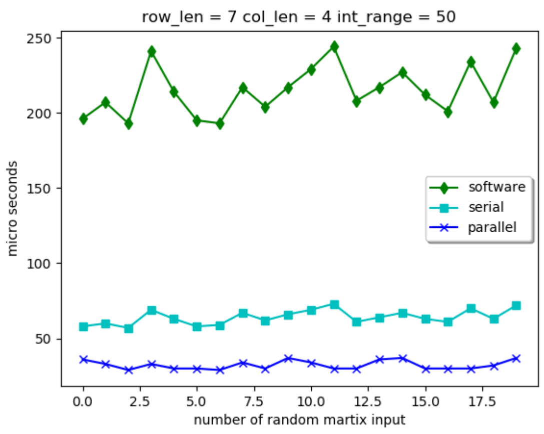


(a) Execution time

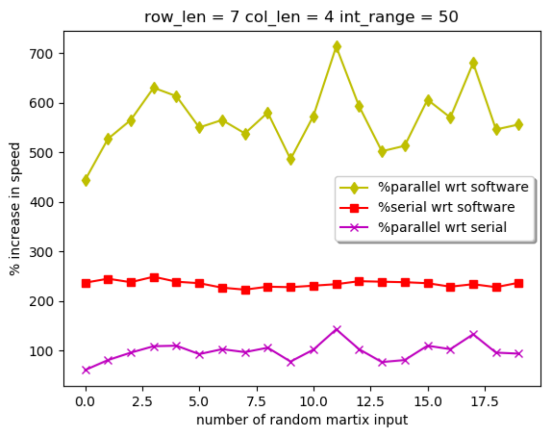


(b) Speedup

Figure 42: Compare solvers: row length = 7, column length = 3, integer range = 20



(a) Execution time



(b) Speedup

Figure 43: Compare solvers: row length = 7, column length = 4, integer range = 50

5.5.2 Comparison of Row Lengths

The efficiency of the parallel solver increases by increasing only the row length from 3 to 7 (refer figure 44). An interesting observation is that the percentage increase of the parallel solver with respect to the serial increases along the row length, which proves that the multiple reducer modules are executed in parallel (MIMD) and increases the speed on comparison with the serial solver that has a single reducer module (SIMD). Refer to figure 45 to notice a clear increase in the percentage of the speed as the row length increases.

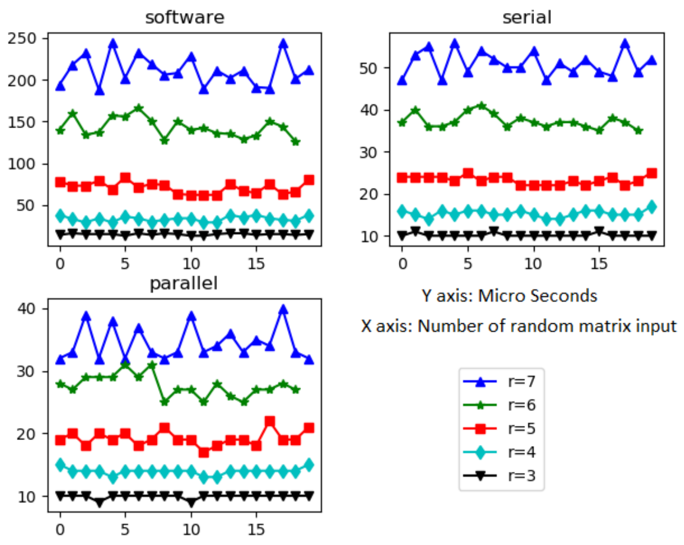


Figure 44: Execution time by varying row length (r): column length=7, integer range=10

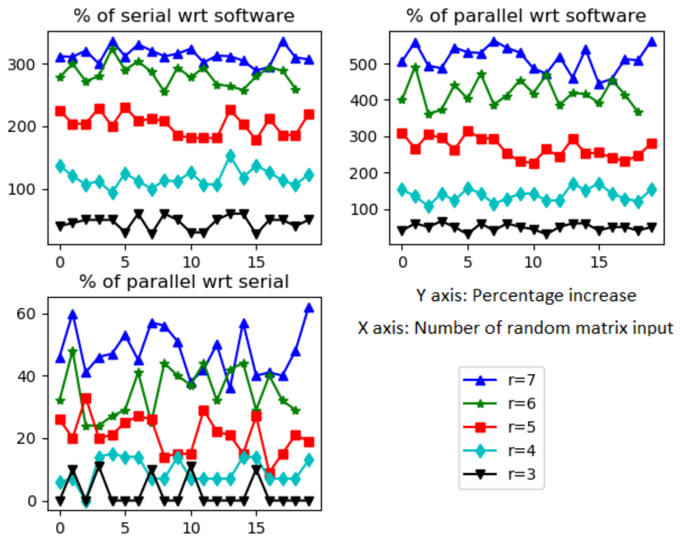


Figure 45: Speedup by varying row length (r): column length=7, integer range=10

5.5.3 Comparison of Column Lengths

By increasing only the column length from 3 to 7 (refer figure 46), the efficiency of the serial solver increases with respect to the software solver as the variable reducer and the swap modules contains rows with the dedicated hardware for 7 column. The arithmetic operations are performed on an entire row for an instruction irrespective of the input column length. Due to this, no improvement is observed in the parallel solver with respect to the serial solver while increasing the data column length (refer figure 47).

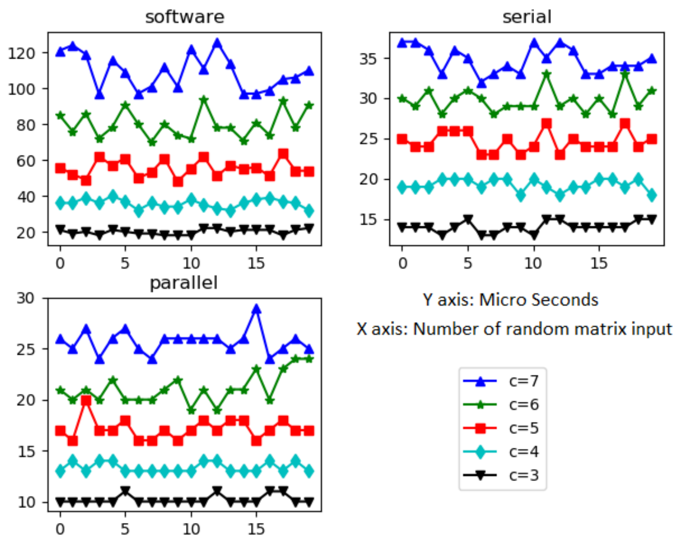


Figure 46: Execution time by varying column length (c): row length=7, integer range=5

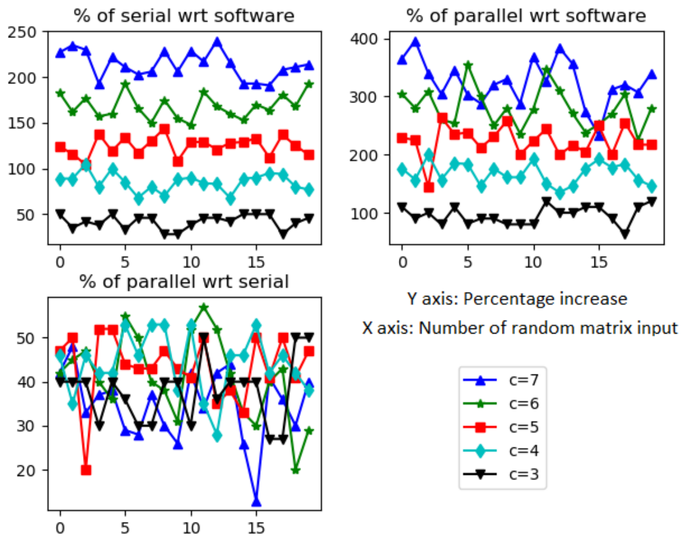


Figure 47: Speedup by varying column length (c): row length=7, integer range=5

5.5.4 Comparison of Integer Ranges

The efficiency of the parallel solver increases by increasing only the upper limit of coefficient from 5 to 20 in 5 steps (refer figure 48). The rise in the upper limit increases the execution time of a reducer module, as it may require to perform more subtractions. However, the SIMD operation in the hardware reducers helps to perform it better than its software counterpart. This also provides an advantage to the parallel solver due to the MIMD operation through multiple reducer components, which is noticed especially when the range of the coefficient increases. Observe the percentage raise of the parallel solver to the serial solver for the integer range 5 and 20 in figure 49.

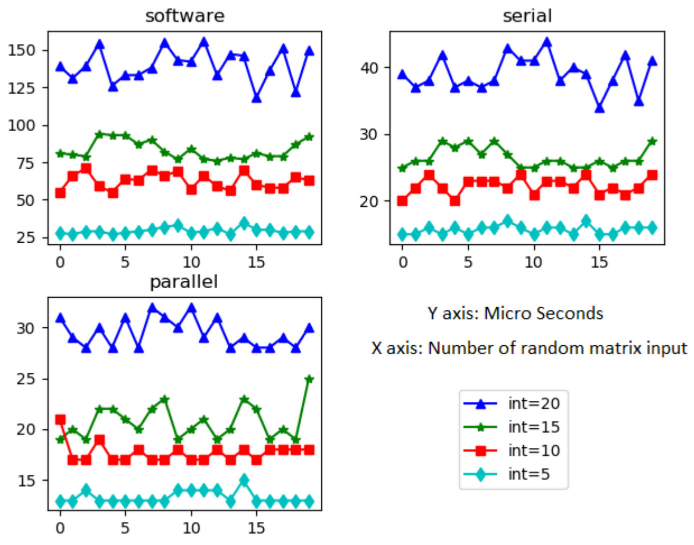


Figure 48: Execution time by varying maximum integer range (int): row length=5, column length=5

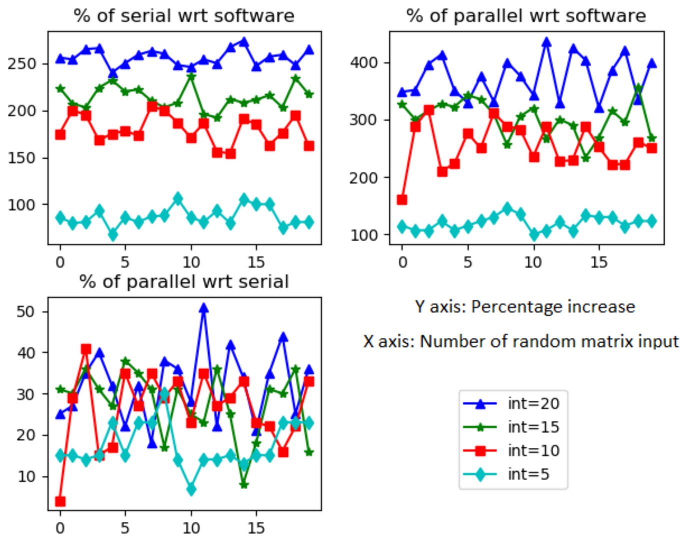


Figure 49: Speedup by varying maximum integer range (int): row length=5, column length=5

6 Conclusion and Future Work

6.1 Conclusion

This thesis shows that the hardware-based solvers are faster than the software solver by several folds even after including the overhead from the interconnect bridges and clock differences. The speed is increased at least by 75% considering the parallel and the software solvers for row and column lengths greater or equal to three. In few best cases, the speed is increased by more than 500%. This is observed in the speedup graphs of the parallel solver with respect to the software solver. It is mainly due to the FPGA modules process a data set in arithmetic operations and uses dedicated hardware to execute the reduction algorithm in parallel. The speedup figures are mainly dependent on the implementation of the software reducer module. Here, it is implemented to have the same data flow as the hardware reducer modules and the speedup values could be different if the data flow is different.

The technique of reducing a matrix to the row-echelon form is used in other applications, such as, computing determinants of a square matrix, and finding an inverse of a matrix [23]. This method of acceleration can also be applied to an algorithm that contains matrix processing, especially arithmetic operations performed on an entire row as in figure 50.

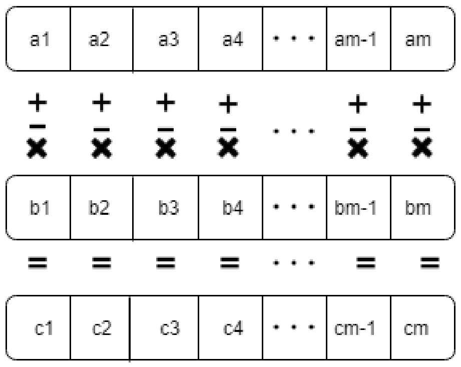


Figure 50: Data set operations

6.2 Future Work

The following items are considered for the future work to improve the simple LIA solver.

1. Optimize the FPGA modules to synthesize bigger matrix using the SRAM IP for internal matrix storage and avoid ALMs (Look-up tables).
2. Expand the LIA problem to include non-equalities logic by implementing the Simplex algorithm (or similar).
3. Expand the LIA problem to include real numbers using floating point IP blocks[13].

The result also motivates to accelerate an existing SMT solver (for example, Princess[8] or Z3) that handles many first-order theories.

Bibliography

[1] *Driver Drowsiness Detection System for Cars.* http://81.47.175.201/compass/index.php, accessed: 2018-11-30.

[2] *SoC FPGA Evaluation.* https://www.datarespons.com/soc-fpga-evaluation-guidelines/, accessed: 2018-12-12.

[3] *DE1-SoC Getting Started Guide.* https://www.terasic.com, accessed: 2018.04.02.

[4] *Platform Designer System Design Tutorial.* https://www.intel.com, accessed: 2018.04.02.

[5] *Cyclone V Hard Processor System Technical Reference Manual.* https://www.intel.com, accessed: 2018.07.17.

[6] *DE1-SoC Terasic home page.* https://www.terasic.com, accessed: 2019-01-12.

[7] *Cyclone V Device Handbook.* https://www.intel.com, accessed: 2019.01.17.

[8] *Princess: The Scala Theorem Prover.* http://www.philipp.ruemmer.org/princess.shtml, accessed: 2019.02.10.

[9] Ashish Sabharwal Carla P. Gomes, Henry Kautz and Bart Selman. *Satisfiability Solvers.* Elsevier B.V., 2008.

[10] Ellis; Wets Roger Cottle, Richard; Johnson. *George B. Dantzig (1914–2005).* American Mathematical Society, March 2007.

[11] Bernd Gärtner and Jiří Matoušek. *Understanding and Using Linear Programming.* Springer, 2006.

[12] Jim Hefferon. *Linear Algebra.* 2017.

[13] Intel. *Floating-Point IP Cores User Guide.* DEC 2016.

[14] Suganth Paul Intel Inc Sunil P Khatri Texas A and M University Srinivas Patil Intel Inc Kanupriya Gulati, TexasA and M University and Inc Abhijit Jas, Intel. *FPGA-Based Hardware Acceleration for Boolean Satisfiability.* ACM, March 2009.

[15] Egwu Kaw, Autar; Kalu. *Numerical Methods with Applications.* University of South Florida., 2010.

[16] Mary Sheeran Koen Claessen, Niklas Een and Niklas Sörensson. *SAT-solving in practice.*

[17] Daniel Kroening and Ofer Strichman. *Decision Procedures An Algorithmic Point of View, Second Edition.* Springer, 2016.

[18] O.J. Morales. *FFT hardware accelerator.* IEEE, 1988.

[19] Douglas L. Perry. *VHDL : Programming By Example 4th Edition.* McGraw-Hill.

[20] William Pugh. *The Omega Test: A fast and practical integer programming algorithm for dependence analysis.* ACM, 1991.

[21] Andy Schmitz. *Elementary Algebra.* Saylor Academy, 2012.

[22] Darek Mihocka; Jens Troeger. *Detection for Array and Bitfield Operations.*

[23] Prof. Anna Vainchtein. *Using row reduction to calculate the inverse and the determinant of a square matrix.*

[24] Vaughn Betz Wei Zhang and Jonathan Rose from University of Toronto. *Portable and Scalable FPGA-Based Acceleration of a Direct Linear System Solver.* ACM, March 2012.

[25] Hugh C. Williams. *Edouard Lucas and Primality Testing.* Wiley-Interscience, 1998.

[26] Dennis Yurichev. *SAT/SMT by Example.* January 25, 2019.

7 Appendix A: Instructions to Run the Solvers on a DE1-SoC Board

This section contains instructions to install necessary software, setup board connectivity, and to compile and run the different solver applications on the DE1-SoC board. The following instructions should guide to compile and run the source code written during the thesis time-frame.

7.1 Hardware Required

1. DE1-SoC development board.
2. Micro SD card (at least 4 GB).
3. Micro USB cable.
4. Ethernet cable.
5. Windows computer with at least 8GB RAM and 20GB disk space.

7.2 Software Installations

The following software are required for a successful development environment setup on the Windows operating system (Tested on Windows 10).

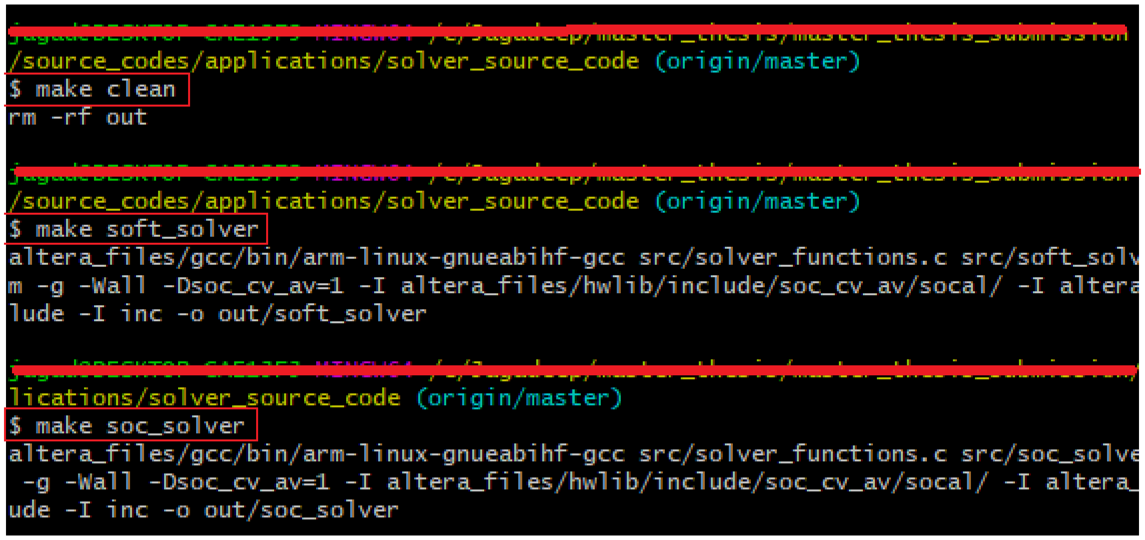
1. Quartus Prime Software: Download it from Intel's homepage and install it to design, synthesis, and program FPGA. The thesis project files are based on version 18.1 and it is recommended to install the same, but it might also work for the higher versions by upgrading the IP tools. The source code is designed using Lite edition which is available for a free of cost and does not require a license. One might choose any edition based on their needs (with subscription or license) and the project files are expected to work regardless of the editions. Download Cyclone V device support files to the same folder as the Quartus installation file in the host machine before installation.
2. Python: Install Python3 software, Matplotlib and Numpy packages. The packages can be installed using PIP tool (*$pip install matplotlib* and *$pip install numpy*) after installing Python3. They are required to verify the result files and plot timing graphs.
3. Putty: Install a terminal software to read/write a serial port. It is used as an command prompt interface towards the target Linux.
4. MinGW: Install Linux based GNU software on Windows OS to use Make tool for building applications. Remember to set its binary directory in the user path under environment variables.
5. Thesis Design Files: Download the source code from the github website.
   https://github.com/Jagadeepram/linear_equation_solver_DE1_SoC

7.3 Compilation of MCU software

In the repository, navigate to the "applications/solver_source_code" folder in a terminal and execute the following commands.

- Clean application files using "make clean" command.
- Compile by typing "make" followed by an application name to compile a specific application and without an argument to compile all the three applications as shown in figure 51.

The ARM cross compiler is included in the "altera_files/gcc" folder which is used by the build system and necessary header files under "altera_files/hwlib" for successful compilation. It is highly recommended not to change these files and folders.



```
[redacted]
/source_codes/applications/solver_source_code (origin/master)
$ make clean
rm -rf out

[redacted]
/source_codes/applications/solver_source_code (origin/master)
$ make soft_solver
altera_files/gcc/bin/arm-linux-gnueabihf-gcc src/solver_functions.c src/soft_solv
m -g -Wall -Dsoc_cv_av=1 -I altera_files/hwlib/include/soc_cv_av/socal/ -I altera
lude -I inc -o out/soft_solver

[redacted]
lications/solver_source_code (origin/master)
$ make soc_solver
altera_files/gcc/bin/arm-linux-gnueabihf-gcc src/solver_functions.c src/soc_solve
 -g -Wall -Dsoc_cv_av=1 -I altera_files/hwlib/include/soc_cv_av/socal/ -I altera_
ude -I inc -o out/soc_solver
```

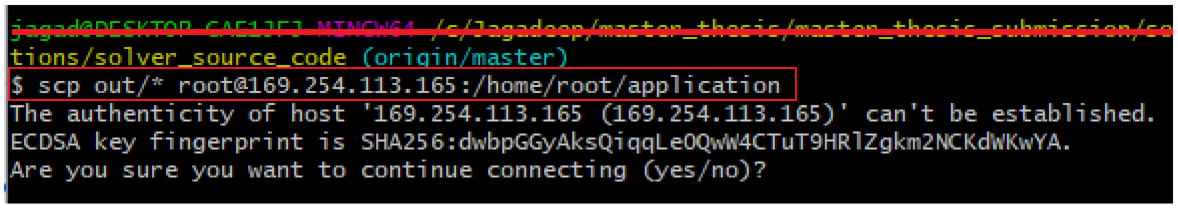
Figure 51: Make commands

7.4 Setting up Connectivity and Files Transfer

Download board manual, user guide, and all attachments from the home page of the DE1-SoC [3]. Go through the user manual and set the dip switches for terminal-based Linux. Connect the USB terminal port from the board to the host computer and start the Putty software. Look for the comport number in device manager tool and open it as a serial port with baud rate 115200. Prepare an SD card containing Linux image as mentioned in the user manual and insert it before turning ON the board. Wait for 5 seconds during the Linux start-up for auto-boot process to complete and enter the login name **root**. This takes to the home directory of the root login.

Connect the host computer and the board through an Ethernet cable. Find the IP address of the host and set the IP address of the board with plus one in the last digit to that of the host's. For example, if the host has 169.254.113.164, then set the target as 169.254.113.165 using the command "**ifconfig eth0 169.254.113.165**". To confirm the connectivity, ping the address from the host.

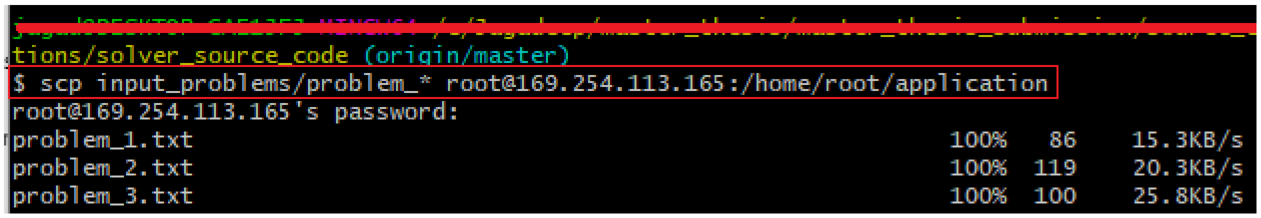
The application binaries created in the folder "out" can be transferred to the target using SCP command as mentioned in figure 52. Create a folder called "application" under the home directory in the target before the file transfer.



```
[redacted]
tions/solver_source_code (origin/master)
$ scp out/* root@169.254.113.165:/home/root/application
The authenticity of host '169.254.113.165 (169.254.113.165)' can't be established.
ECDSA key fingerprint is SHA256:dwbpGGyAksQiqqLeOQwW4CTuT9HRlZgkm2NCKdWKwYA.
Are you sure you want to continue connecting (yes/no)?
```

Figure 52: Command to transfer binary files to the target

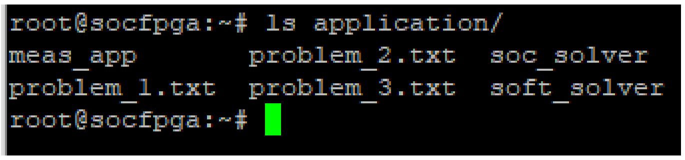
Type "yes" to a security question and enter the password **terasic** (or find it in the board user manual) to complete the file transfer. Also transfer the input problem files in the same way.



```
[redacted]
tions/solver_source_code (origin/master)
$ scp input_problems/problem_* root@169.254.113.165:/home/root/application
root@169.254.113.165's password:
problem_1.txt                                               100%   86    15.3KB/s
problem_2.txt                                               100%  119    20.3KB/s
problem_3.txt                                               100%  100    25.8KB/s
```

Figure 53: Command to transfer input files to the target

Now the "application" folder in the target contains the three applications and the three input files.



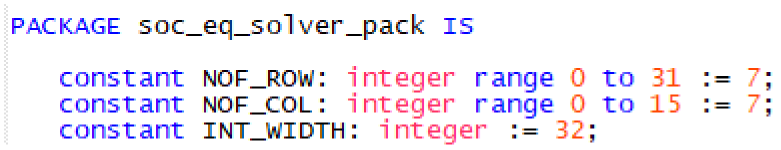
```
root@socfpga:~# ls application/
meas_app       problem_2.txt  soc_solver
problem_1.txt  problem_3.txt  soft_solver
root@socfpga:~#
```

Figure 54: List of files transferred to the target

7.5 Synthesis and Program FPGA designs

Find the folders "soc_eq_solver_parallel" and "soc_eq_solver_serial" under the repository root for the parallel and the serial solver's design files. Open them in the Quartus IDE by double-clicking the "soc_eq_solver.qpf" file under these folders. The folder "vhdl_design_files" contains VHDL design files that are common to the two projects and the individual design files are within the respective project folders. All the design and project files are submitted in the working condition without compilation and run-time errors.

The design properties can be changed in a VHDL package file before synthesis and compilation as in figure 55.



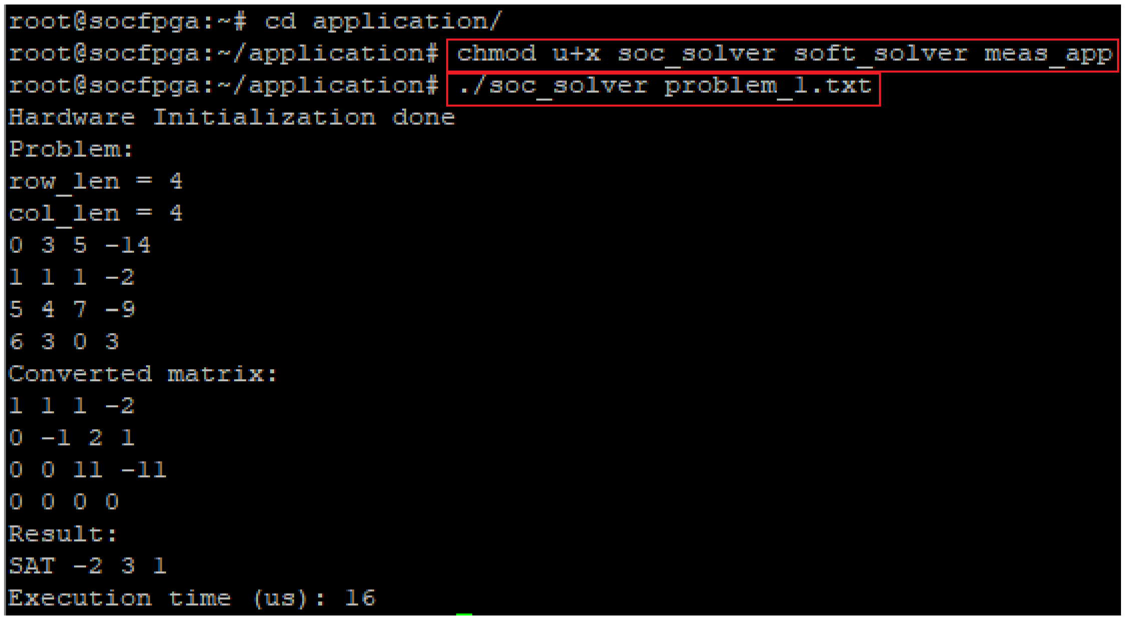
```
PACKAGE soc_eq_solver_pack IS

   constant NOF_ROW: integer range 0 to 31 := 7;
   constant NOF_COL: integer range 0 to 15 := 7;
   constant INT_WIDTH: integer := 32;
```

Figure 55: Hardware design properties in soc_eq_solver_pack.vhd

The package file is common for both the designs and the parameters should be changed before synthesis if they are different. Follow the Quartus user guide to modify, compile, and program the hardware converter design before running the applications.

7.6 Running Solver Applications

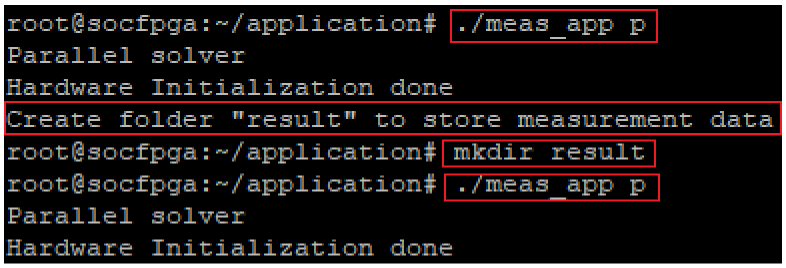
Change access permission of the application files to make them executable and execute the "soc_solver" (after programming FPGA) and the "soft_solver" followed by the name of an input text file as shown in figure 56.



```
root@socfpga:~# cd application/
root@socfpga:~/application# chmod u+x soc_solver soft_solver meas_app
root@socfpga:~/application# ./soc_solver problem_1.txt
Hardware Initialization done
Problem:
row_len = 4
col_len = 4
0 3 5 -14
1 1 1 -2
5 4 7 -9
6 3 0 3
Converted matrix:
1 1 1 -2
0 -1 2 1
0 0 11 -11
0 0 0 0
Result:
SAT -2 3 1
Execution time (us): 16
```

Figure 56: Run solver applications with an input text file

To execute the measurement application, type "meas_app" followed by a character 'p' or 's' based on the type of converter programmed. It is shown in the first LED on the target board. The application would throw an error saying that the "result" folder should be created. Re-run the application after creating the folder for a successful execution. It is designed on purpose that a user should be aware of results stored in this folder. Run this application again with other hardware converter to store all output files under the "result" folder. Note that the last two LEDs will display combinations of values which indicates row and column lengths of a matrix transferred for processing.

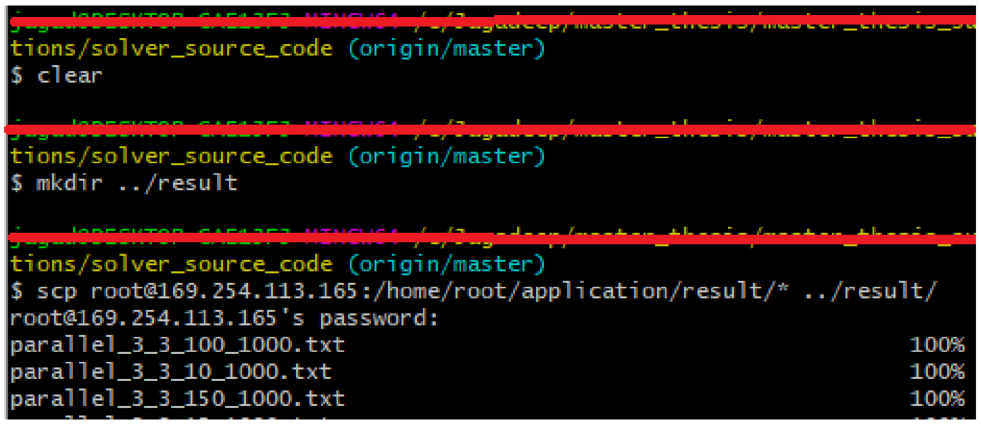


```
root@socfpga:~/application# ./meas_app p
Parallel solver
Hardware Initialization done
Create folder "result" to store measurement data
root@socfpga:~/application# mkdir result
root@socfpga:~/application# ./meas_app p
Parallel solver
Hardware Initialization done
```

Figure 57: Run measurement application

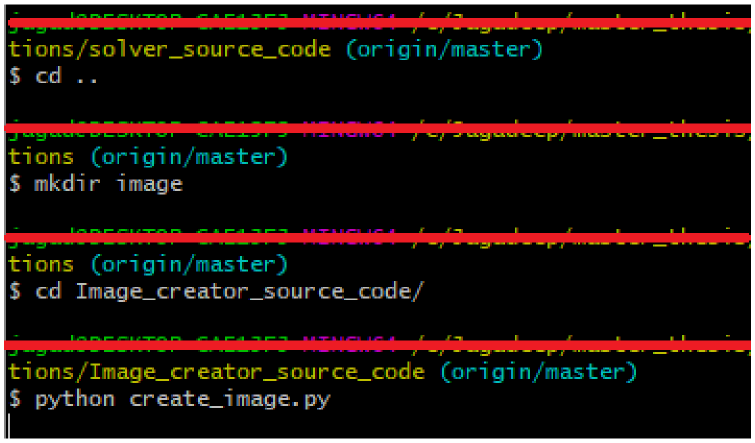
7.7 Analyzing Result Files

Transfer the "result" folder from the target to the host machine to analyze and compare execution time between different solvers. Create a folder called "result" under the "applications" folder and execute the following SCP command to transfer all output files (as in figure 58). The name and location of the "result" folder should not be changed as it is required by the "image_create.py" script to read them. Run all the commands and the python script as shown in figure 59 to store graphs under the "image" folder.



```
[redacted]
tions/solver_source_code (origin/master)
$ clear

[redacted]
tions/solver_source_code (origin/master)
$ mkdir ../result

[redacted]
tions/solver_source_code (origin/master)
$ scp root@169.254.113.165:/home/root/application/result/* ../result/
root@169.254.113.165's password:
parallel_3_3_100_1000.txt                                              100%
parallel_3_3_10_1000.txt                                               100%
parallel_3_3_150_1000.txt                                              100%
```

Figure 58: Transfer results from the target to the host



```
[redacted]
tions/solver_source_code (origin/master)
$ cd ..

[redacted]
tions (origin/master)
$ mkdir image

[redacted]
tions (origin/master)
$ cd Image_creator_source_code/

[redacted]
tions/Image_creator_source_code (origin/master)
$ python create_image.py
```

Figure 59: Create timing graphs