"Linear Equation Solver using CMOS Technology"

by

Burak Erbağcı

Project supervisors:

Y. Leblebici
A. Tajalli

Lausanne, 2010
Acknowledgements

I would like to thank Professor Yusuf Leblebici who gave me the opportunity to make this work, Armin Tajalli, Alain Vachoux for their everyday support, their help with the tools and guidance through all the stages of my work.

I wish to thank Rabia Tuğce Yazıcıgil, Emrah Taş, Eynen Kurdoğlu, Ali Galip Bayrak, Gözen Köklü, Sevil Zeynep Temel, Hasene Gürperi Özsema for all their friendship, support, and encouragement.

Finally, I would also like to thank my family for their endless support and interest in my career.
Abstract

In this project, a systematic way of designing digital linear equation solvers with output feedbacks is provided. First the methodology is implemented in small scale (system of equations with 4 unknowns) both in VHDL and transistor level. It is verified by both VHDL logic simulations and transistor level simulations. Next, the scale is extended to the system of equations with 8 unknowns. This time, it is implemented only in transistor level, and various simulations are performed. The correct operation of the corresponding solver is verified.

In addition, the drawbacks and challenges regarding the implementation in large scales are indicated. The necessity of further research focusing on improvements in order to facilitate the implementation (e.g. the clocking scheme and the allocation of clocks) is elaborated.
Contents

Acknowledgements ............................................................................................................ 2
Abstract ............................................................................................................................. 3
Contents ............................................................................................................................. 4
List of Figures .................................................................................................................... 6
1 Introduction ..................................................................................................................... 8
2 Overview of the Digital Linear Equation Solver Hardware ........................................... 10
  2.1 Digital Linear Equation Solver in Small Scale (4 unknowns) .................................... 11
  2.1.1 Initial Design ....................................................................................................... 11
  2.1.2 Revised Design .................................................................................................. 13
  2.1.3 Final Design ...................................................................................................... 15
  2.2 Digital Linear Equation Solver with 8 unknowns ...................................................... 18
  2.2.1 Final Design ...................................................................................................... 18
3 Digital Linear Equation Solver Hardware Implementation ........................................... 20
  3.1 Digital Linear Equation Solver with 4 unknowns ....................................................... 20
  3.1.1 Initial Design ....................................................................................................... 20
  3.1.2 Revised Design .................................................................................................. 21
  3.1.3 Final Design ...................................................................................................... 24
  3.2 Digital Linear Equation Solver with 8 unknowns ....................................................... 24
  3.2.1 Final Design ...................................................................................................... 24
4 Implementation Results ................................................................................................. 27
  4.1 Digital Linear Equation Solver with 4 unknowns ....................................................... 27
  4.1.1 Initial Design ....................................................................................................... 27
  4.1.2 Revised Design .................................................................................................. 28
  4.1.3 Final Design ...................................................................................................... 31
  4.2 Digital Linear Equation Solver with 8 unknowns ....................................................... 35
  4.2.1 Final Design ...................................................................................................... 35
5 Further Improvements and Comments ......................................................................... 43
6 Conclusions .................................................................................................................... 44
7 References ...................................................................................................................... 45
Appendices ......................................................................................................................... 46
A1 Basic Building Blocks of the Solver ............................................................................ 46
A1.1 Inverter Cell ............................................................................................................. 46
A1.2 AND Cell ................................................................................................................. 47
A1.3. XOR Cell .................................................................................................................. 48
A1.4. DLATCH Cell ........................................................................................................... 49
A1.5. DFF Cell .................................................................................................................. 50
A1.6. Input Buffers ........................................................................................................... 51
A1.7. Simulation Setups .................................................................................................... 52
  A1.7.1. Simulation Setup for the Initial Solver (4 unknowns) ............................................ 52
  A1.7.2. Simulation Setup for the Revised Solver (4 unknowns) ...................................... 53
  A1.7.3. Simulation Setup for the Final Solver (8 unknowns) .......................................... 54
A2 VHDL Codes .................................................................................................................. 55
  A2.1. Initial Solver (4 unknowns) ...................................................................................... 55
     A2.1.1. GenSolver4_noDFF.vhd .................................................................................. 55
  A2.2. Revised Solver (4 unknowns) ................................................................................ 56
     A2.2.1. GenSolver4.vhd ............................................................................................ 56
     A2.2.2. GenSolver4_tb.vhd ...................................................................................... 59
  A2.3. Final Solver (4 unknowns) .................................................................................... 59
     A2.3.1. CK_prioritizer.vhd ......................................................................................... 59
A3 MATLAB codes ............................................................................................................. 59
  A3.1. All_input_matrices_4x4.m ..................................................................................... 59
  A3.2. Rand_GenTestcase.m ......................................................................................... 61
List of Figures

Figure 1. The Specified Shift Registers for A5/1 [3]......................................................................................... 8
Figure 2. Schematic of three LSFRs in A5/1 [3] .................................................................................................. 9
Figure 3. Input Matrices, A, B, the Output Matrix, x, and the Corresponding Logic Operations .......... 11
Figure 4. Schematic of the Solver (4 unknowns) .............................................................................................. 12
Figure 5. Schematic of the Solver (4 unknowns) with DFFs ........................................................................ 14
Figure 6. Clock Timing Analysis for the Solver (4 unknowns) .................................................................... 15
Figure 7. Schematic of the Final Solver (4 unknowns) .................................................................................... 17
Figure 8. Schematic of the Final Solver with 8 unknowns .......................................................................... 18
Figure 9. Cadence Schematic of Initial Solver Design (4 unknowns) ............................................................. 20
Figure 10. Cadence Schematic of Revised Solver Design (4 unknowns) ...................................................... 22
Figure 11. CADENCE Schematic of Revised Solver Design with Input Buffers (4 unknowns) .......... 23
Figure 12. CADENCE Schematic of the Final Solver Design (8 unknowns) ................................................ 25
Figure 13. CADENCE Schematic of the Final Solver Design with Input Buffers (8 unknowns) ............. 26
Figure 14. CADENCE Output waveforms for the Initial Solver Design (4 unknowns) ............................. 27
Figure 15. MODELSIM Output waveforms for the Revised Solver Design (4 unknowns) ...................... 29
Figure 16. MODELSIM Assertions for the Revised Solver Design (4 unknowns) ......................................... 29
Figure 17. CADENCE Output waveforms_1 for the Revised Solver Design (4 unknowns) - 7 inputs ....... 30
Figure 18. CADENCE Output waveforms_2 for the Revised Solver Design (4 unknowns) - 5 inputs .... 30
Figure 19. CADENCE Output waveforms_3 for the Revised Solver Design (4 unknowns) - 8 inputs ....... 31
Figure 20. MODELSIM Output waveforms for the Final Solver Design (4 unknowns) ............................. 32
Figure 21. MODELSIM Assertions for the Final Solver Design (4 unknowns) ............................................. 32
Figure 22. CADENCE Output waveforms_1 for the Final Solver Design (4 unknowns) - 7 inputs ........ 33
Figure 23. CADENCE Output waveforms_2 for the Final Solver Design (4 unknowns) - 5 inputs ........ 33
Figure 24. CADENCE Output waveforms_3 for the Final Solver Design (4 unknowns) - 8 inputs ......... 34
Figure 25. CADENCE Output waveforms_4 for the Final Solver Design (4 unknowns) - 25 inputs ....... 34
Figure 26. CADENCE Output waveforms_5 for the Final Solver Design (4 unknowns) - 30 inputs ....... 35
Figure 27. CADENCE Output waveforms_1 for the Final Solver Design (8 unknowns) - 5 inputs .......... 36
Figure 28. CADENCE Output waveforms_2 for the Final Solver Design (8 unknowns) - 5 inputs ....... 37
Figure 29. CADENCE Output waveforms_3 for the Final Solver Design (8 unknowns) - 5 inputs ....... 37
Figure 30. CADENCE Output waveforms_4 for the Final Solver Design (8 unknowns) - 5 inputs ....... 38
Figure 31. Output waveforms_5 for the Final Solver Design (8 unknowns) - 9 inputs .......................... 38
Figure 32. Output waveforms_6 for the Final Solver Design (8 unknowns) - 6 inputs ........................ 39
Figure 33. Output waveforms_7 for the Final Solver Design (8 unknowns) - 1 input .............................. 39
Figure 34. Output waveforms_7 for the Final Solver Design (8 unknowns) - 1 input - Closer view ..... 40
Figure 35. Output waveforms_8 for the Final Solver Design (8 unknowns) - 1 input ............................. 40
Figure 36. Output waveforms_8 for the Final Solver Design (8 unknowns) - 1 input - Closer view ..... 41
Figure 37. Output waveforms_9 for the Final Solver Design (8 unknowns) - 1 input ............................. 41
Figure 38. Output waveforms_9 for the Final Solver Design (8 unknowns) - 1 input - Closer view ..... 42
Figure 39. Schematic of INV gate ................................................................................................................. 46
Figure 40. Schematic of AND gate ................................................................................................................. 47
Figure 41. Schematic of NAND gate ............................................................................................................. 47
Figure 42. Schematic of XOR gate ................................................................................................................ 48
Figure 43. Schematic of DLATCH gate ............................................................................................................... 49
Figure 44. Schematic of DFF gate .......................................................................................... 50
Figure 45. Schematic of 81b Input Buffer ............................................................................. 51
Figure 46. Simulation setup for the Initial Solver design (4 unknowns) ............................. 52
Figure 47. Simulation setup for the Revised Solver (4 unknowns) ................................. 53
Figure 48. 4 Different Clock Domains for the Revised Solver - 6, 10, 14, 18 ns periods .......... 53
Figure 49. Simulation setup for the Final Solver (8 unknowns) .......................................... 54
Figure 50. 8 Different Clock Domains for the Final Solver - 6, 8, 10, 12, 14, 16, 18, 20 ns periods ..... 54
1 Introduction

In cryptography, encryption is the process of transforming information, plaintext, using a specific algorithm, cipher, to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key [1]. A reverse process, decryption, has to be applied in order to make the encrypted information, ciphertext, readable again. The reliability of a particular Encryption/Decryption algorithm, its algorithmic strength, is defined as its resistance to the mathematical attacks, the process of mathematically cracking the algorithm. It is important to note that even if a cipher cannot be cracked mathematically, such as Advanced Encryption Standard (AES), it may still be cracked via side channel attacks. These cracking methods are based on information gained from the physical implementation of such algorithms, rather than brute force or theoretical weakness in the ciphers (cryptanalysis). For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information which can be exploited to break the system[2]. However, these attacks require some technical knowledge regarding the internal operation of the hardware on which the cipher is implemented.

Encryption/Decryption process is commonly used in various applications in order to protect information. For instance, militaries and governments rely on this process to facilitate secret communication; many kinds of civilian systems (internet, Bluetooth, wireless systems, ATMs, etc...) utilize this process to ensure privacy via the protection of the data at rest, as well as the data in transit over the networks [1]. However, successfully ensuring the data security may be a challenging problem.

For instance, though initially kept secret, A5/1 Cipher, which is the standard encryption algorithm for GSM in Europe and the US, was obtained by reverse engineering.

A5/1 cipher basically produces a 114-bit sequence of keystream which is XORed with line bits prior to modulation, for each burst in a GSM transmission. It is initialized using a 64-bit key together with a publicly-known 22-bit frame number [3]. The cipher utilizes a combination of three linear feedback shift registers (LFSRs) with irregular clocking, specified in Figure 1.

<table>
<thead>
<tr>
<th>LFSR number</th>
<th>Length in bits</th>
<th>Characteristic polynomial</th>
<th>Clocking bit</th>
<th>Tapped bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>$x^{18} + x^{17} + x^{16} + x^{13} + 1$</td>
<td>8</td>
<td>13, 16, 17, 18</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>$x^{21} + x^{20} + 1$</td>
<td>10</td>
<td>20, 21</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>$x^{22} + x^{21} + x^{20} + x^{7} + 1$</td>
<td>10</td>
<td>7, 20, 21, 22</td>
</tr>
</tbody>
</table>

*Figure 1. The Specified Shift Registers for A5/1 [3]*
Each register has an associated clocking bit (orange) as shown in Figure 2. A register is clocked if it agrees with one or both of the clocking bits of the other two registers. Initially, all the registers are set to zero. For the next 64 cycles, the 64-bit secret key is masked according to the following scheme:

- for each clock cycle $i$, $0 \leq i < 64$, the $i^{th}$ key bit is added to the least significant bit (LSB) of each LSFRs using the equation, $R[0] = R[0] \oplus K[i]$.

Then each register is clocked, and the 22-bits of the frame number are added in 22 cycles. It takes 100 cycles for the output to be discarded. After the completion of this step, cipher produces two 114 bit sequences of output keystream, first 114 for downlink and last 114 for uplink [3].

Several design flaws of A5/1 allow for a complete recovery of the keystream by solving $\sim 2^{40}$ linear equation systems in $\mathbb{Z}_2$ with 64 unknowns [4]. Therefore, solving a linear equation system with many unknowns (64 in this case) in a reasonable time is significant.

The aim of this project is to implement a considerably fast digital linear equation solver which is capable of solving the systems with $\leq 64$ unknowns in $\mathbb{Z}_2$ to perform a live A5/1 attack. However, unlike the usual methods used for implementing such solvers, which solve the system by performing some operations step-by-step in an FPGA or a processor, this solver uses output feedbacks, or loops to settle down to the solution in a considerably short time, provided there exists a non-trivial solution to the linear system of equations. Namely, circuit will "instantly" solve the system of equations and the final stable state of the system will provide the outputs. It is also important to note that in addition to the expected speedup for solving a particular system, the hardware with feedback loops is expected to show more resistance to the digital power attacks. These attacks are based on the power analysis of a circuit in consideration in order to non-invasively extract the keys and the other secret information, depending on its iterative nature during the operation.
Overview of the Digital Linear Equation Solver Hardware

The proposed digital solver hardware is capable of solving the linear systems of equations with the following properties:

- Given a matrix $A$ and a vector $b$, determine $x$ such that $Ax = b$ with $a_{ij}$, $b_i$, $x_i$ in $Z_2$
- $A$ is a quadratic $n \times n$ matrix ($n \leq 64$), $b$ and $x$ have $n$ coefficients
- The linear systems of equations with $Ax = b$ have a non-trivial solution
- All diagonal elements of $A$ are 1.

It utilizes feedback loops to settle down to the solution in a considerably short time. Therefore, unlike the usual solvers that first compute the inverse of the matrix $A^{-1}$ and multiply by $b$, the solution $x$ is determined in an iterative fashion. Since input matrix $A$ is invertible, feedback of the outputs together with the input matrices $A$ and $b$ force the circuit to a stable state which is in fact the solution $x$. It will "instantly" solve for $x$ iteratively depending on the input matrices without computing the intermediate values like $A^{-1}$ or the algorithm specific variables for solving the matrices. This provides a considerable speedup for solving a particular system.

Since the proposed solver hardware is intended to be used for a practical application in cryptanalysis, a live A5/1 attack, it is supposed to solve the systems with $\leq 64$ unknowns in $Z_2$. Hence, the goal is to devise a generic way of designing such solvers with feedback that can operate on the quadratic $n \times n$ matrices ($n \leq 64$). In this context, first the linear solver is implemented in small scale (for the system with 4 unknowns) to characterize the overall performance and performing the stability analysis. As further simulations reveal, the systematic way devised for the small scale system can be extended to larger scales with a reasonable complexity.

Another important remark regarding the design process is that the small scale hardware is first implemented in VHDL and the correct operation for all cases is verified by the logic simulation. Later, it is implemented in Cadence environment, and transistor level simulations are performed for more precise results.
2.1. Digital Linear Equation Solver in Small Scale (4 unknowns)

2.1.1. Initial Design

The digital solver operates on a square 4x4 matrix \( A \) with all diagonal elements 1 (A1, A6, A11, A16) and matrix \( b \) with 4 coefficients, and produces the output matrix \( x \) with 4 coefficients.

\[
\begin{align*}
A1 &= A2.x2 \oplus A3.x3 \oplus A4.x4 \oplus B1 \\
A2 &= A5.x1 \oplus A7.x3 \oplus A8.x4 \oplus B2 \\
A3 &= A9.x1 \oplus A10.x2 \oplus A12.x4 \oplus B3 \\
A4 &= A13.x1 \oplus A4.x2 \oplus A15.x3 \oplus B4
\end{align*}
\]

As seen from Figure 3, in \( \mathbb{Z}_2 \), addition is realized by logical XOR operation, and multiplication is realized by logical AND operation. It is also important to note that XOR operation can both implement addition and subtraction.

As seen by the equations in Figure 3 and by the schematic in Figure 4, combinational feedback loops exist in the hardware. One-to-one mapping between the equations in Figure 3 and the schematic in Figure 4 can easily be observed.

The major drawback of this implementation is that there exists many possible oscillation paths which are difficult to visualize from the Figure 4. Due to large number of cascaded gain stages, gates, the circuit is most likely to oscillate. Hence, the design has to be revised. The implementation results will be elaborated in the next chapter.
Figure 4. Schematic of the Solver (4 unknowns)
2.1.2. Revised Design

In order to prevent the possible oscillations due to combinational feedbacks, the corresponding loops have to be broken. This can be achieved inserting either extra capacitances into the circuit, which has a similar effect as physically breaking the loops, or Flip Flops (FFs). However, since the capacitances requires more silicon area, and they are not effective in terms of controlling of the circuit operation, the second method is implemented. It is important to note that keeping the number of inserted FFs minimal is essential. In order not to the increase the silicon area as well as the clocking complexity of the circuit, FFs are only inserted at the outputs of the gates providing the solutions. As shown in Figure 5, D-type FFs (DFFs) are preferred since they are easy to implement, and yet effective for the implementation purposes.

Since each output calculation requires the values of the other outputs, it is necessary to solve them one-by-one in an iterative fashion. Hence, all DFFs are clocked differently, which introduces 4 different clock domains, from the fastest to the slowest. Proper timing of DFFs is required for circuit to operate properly.

For instance, assuming that first output X1 is to be determined, then the second, and the third, and finally X4, then the fastest clock has to be applied to the first DFF, and slowest to the fourth DFF and the rest has be distributed with respect to outputs’ order of being solved, with respect to their predetermined sampling order. The implementation results will be elaborated in the next chapter.
Figure 5. Schematic of the Solver (4 unknowns) with DFFs
2.1.3. Final Design

The timing of the clocks is a major design issue for the solver shown in Figure 5. Since the input matrices directly affect the allocation of clocks to the DFFs, an input dependent clocking scheme has to be introduced.

Figure 6. Clock Timing Analysis for the Solver (4 unknowns)
Figure 6 illustrates the input dependence of the clocking scheme. Assuming that input matrix $A$ is applied to the circuit, the matrix coefficients with 0-value will mask the corresponding outputs because of the 0-controlling value property of the AND gate. Hence, the coefficients of $A$ that have 0 value will effectively break the loop corresponding to the specific output. For instance as shown in Figure 6, the coefficient $A_{10}$ will mask $X2$ by forcing 0 at the output of the corresponding AND gate, and similarly $A_2$ and $A_3$ will mask $X2$ and $X3$, respectively. Therefore, the equalities shown at the output of the XOR gates placed just before the outputs have to be hold. The equation for $X4$ depends on $X1$, $X2$ and $X3$ (here $X1/X1'$ or $X4/X4'$ denotes the dependence on $B2$ or $B3$, since input matrix $b$ does not affect the clocking scheme, the ordering of the clocks) and the corresponding DFF has to receive the slowest clock (to be sampled last). By the same logic, the next slowest clock should be allocated to the one corresponding to $X3$. However, for the cases in which the dependence on the number of outputs is the same, as for $X1$ and $X2$, a prioritization has to be adopted. For this solver, the default priority is determined as the highest for $X1$, then $X2$, then $X3$, and the lowest for $X4$. Hence, $X1$ and $X2$ should receive the fastest and the next fastest clock, respectively.

As shown in Figure 7, the clock allocation based on the input matrix $A$ is performed by the CK_distributor block. It basically counts the number one 1's in the corresponding lines of $A$. The corresponding output on the line with minimum number of coefficients having 1-value (maximum number of coefficients having 0-value) receives the fastest clock, and vice versa. For the cases in which two or more lines have the same number of coefficients with 1-value, predetermined priority scheme is applied, and allocation is performed accordingly. The implementation results will be elaborated in the next chapter.
Figure 7. Schematic of the Final Solver (4 unknowns)
2.2. Digital Linear Equation Solver with 8 unknowns

2.2.1. Final Design

Figure 8. Schematic of the Final Solver with 8 unknowns
After verifying the correct operation of the final digital solver with 4 unknowns, the scale is extended to system of equations with 8 unknowns. Therefore, the digital solver with 8 unknowns is designed by applying exactly the same topology and principles as in the case with 4 unknowns (Figure 8). However it is important to note that as the scale is extended, the CK_distribution block gets complicated. Therefore, it is not actually designed, the clock allocation is rearranged for each simulated input in VHDL test bench instead. The implementation results will be elaborated in the next chapter.
3  Digital Linear Equation Solver Hardware Implementation

3.1. Digital Linear Equation Solver with 4 unknowns

3.1.1. Initial Design

As mentioned before, solver hardware is implemented in both VHDL (Appendix A2.1) and Cadence environment. As shown in Figure 9, the basic building blocks are INV (Appendix A1.1), AND (Appendix A1.2), XOR (Appendix A1.3) cells.

Figure 9. Cadence Schematic of Initial Solver Design (4 unknowns)
3.1.2. Revised Design

As mentioned before, solver hardware is implemented in both VHDL (Appendix A2.2) and Cadence environment. As shown in Figure 10, the basic building blocks are INV (Appendix A1.1), NAND (Appendix A1.2), XOR (Appendix A1.3), DLATCH (Appendix A1.4) and DFF (Appendix A1.5) cells. Here DFFs are inserted in order to break the feedback loops and prevent possible oscillations. The input buffers (Appendix A1.6) in Figure 11 are inserted for simulation purposes. Since test benches are coded in VHDL (Appendix 2.2.2) and then applied to the Spectre, they are used to stimulate real input signal behavior with finite rise and fall times unlike the ideal ones.
Figure 10. Cadence Schematic of Revised Solver Design (4 unknowns)
Figure 11. CADENCE Schematic of Revised Solver Design with Input Buffers (4 unknowns)
3.1.3.  
Final Design

As mentioned before, solver hardware is implemented in both VHDL (Appendix A2.2.1 and A2.3.1) and Cadence environment. It is important to note that CK_distributor block specified in chapter 2.1.3 is not implemented in Cadence. VHDL design is used for simulation purposes and clocks are allocated accordingly (see Appendix 2.3.1).

As shown in Figure 10, the basic building blocks are INV (Appendix A1.1), NAND (Appendix A1.2), XOR (Appendix A1.3), DLATCH (Appendix A1.4) and DFF (Appendix A1.5) cells. Here DFFs are inserted in order to break the feedback loops and prevent possible oscillations. The input buffers (Appendix A1.6) in Figure 11 are inserted for simulation purposes. Since test benches are coded in VHDL (Appendix 2.2.2) and then applied to the Spectre, they are used to stimulate real input signal behavior with finite rise and fall times unlike the ideal ones.

3.2.  
Digital Linear Equation Solver with 8 unknowns

3.2.1.  
Final Design

The final solver hardware with 8 unknowns is implemented in transistor level in CADENCE environment (Figure 12). However it is important to note that as the scale is extended, clock distribution gets complicated. Therefore, it is not designed, the clock allocation is rearranged for each simulated input in VHDL test bench instead. The input buffers (Appendix A1.6) in Figure 13 are inserted for simulation purposes. Since test benches are coded in VHDL and then applied to the Spectre, they are used to stimulate real input signal behavior with finite rise and fall times unlike the ideal ones.
Figure 12. CADENCE Schematic of the Final Solver Design (8 unknowns)
Figure 13. CADENCE Schematic of the Final Solver Design with Input Buffers (8 unknowns)
4 Implementation Results

4.1 Digital Linear Equation Solver with 4 unknowns

4.1.1 Initial Design

The simulation setup for the initial solver design is provided in Appendix A1.7.1. A few simulations are performed, since the circuit oscillates, as expected. Figure 1 shows the oscillating behavior of the circuit. X1 and X2 are constant because of specific the input matrix A, they are basically masked by the matrix coefficients. In addition, as seen in the simulation setup, diagonal elements are always assumed to be 1, hence, they are not included in the pin list.

VHDL initial solver design is also simulated. However, the outputs could not be computed (logic U-value is assumed) because of the oscillations.

Figure 14. CADENCE Output waveforms for the Initial Solver Design (4 unknowns)
4.1.2. Revised Design

All possible inputs that lead to a unique solution are generated in MATLAB (see Appendix 3.1). In order to verify the correct functionality of the solver design, the inputs are first simulated in MODELSIM. The circuit has only 138 assertion misses out of 27008 (1688 A matrices and 16 b matrices for each) applied inputs (Figure 16). Despite such a high coverage ratio, the circuit fails for some of the inputs. It is important to note that clock periods of 6 ns, 10 ns, 14 ns, 18 ns with each having half of their period pulse widths are applied in the test bench (Appendix A1.7.2). Actually, the only requirement regarding the clocks for the correct operation of the circuit is not to choose clock periods which are the same (obvious), and integer multiples of each others, such as 2 ns, 4 ns, 6 ns, 8 ns, or similarly. The simulation results are provided in Figure 15.

Next, the transistor level simulations are performed for a detailed analysis. The simulation setup for the revised solver design is provided in Appendix A1.7.2. A binary2decimal converter is added for facilitating the analysis of the outputs. 20 (out of 138) different problematic inputs observed in logic simulations are simulated with a transient simulation time of 10 us for each.

The simulation results are shown in Figure 17, 18 and 19. Unlike the MODELSIM simulations, just for some input cases, the circuit fails and the numerical oscillations are observed. The difference originates from the fact that the simulation methods are different for the logic and transistor level simulations. However, the latter provides more precise and accurate results. Therefore, each successfully solvable input in MODELSIM simulations is also expected to be solved in transistor level simulations, but the opposite is not correct as observed in the simulations.
Figure 15. MODELSIM Output waveforms for the Revised Solver Design (4 unknowns)

Figure 16. MODELSIM Assertions for the Revised Solver Design (4 unknowns)
Figure 17. CADENCE Output waveforms_1 for the Revised Solver Design (4 unknowns) - 7 inputs

Figure 18. CADENCE Output waveforms_2 for the Revised Solver Design (4 unknowns) - 5 inputs
4.1.3. Final Design

Exactly the same simulations are performed on the final solver design. All possible inputs are simulated in MODELSIM and the correct functionality of the design is verified. The circuit has no assertion misses out of 27008 applied inputs (Figure 21). In addition, exactly the same clocks are used as in the revised design. The simulation results are provided in Figure 20.

Next, the transistor level simulations are performed for a detailed analysis. The simulation setup for the final solver design is provided in Appendix A.1.7.2. A binary2decimal converter is added for facilitating the analysis of the outputs. 20 (out of 138) different problematic inputs observed in logic simulations for the revised design are simulated with a transient simulation time of 10 us for each. The simulation results are shown in Figure 22, 23, 24. In addition, 55 random inputs are generated in MATLAB (Appendix 3.2) and simulated (Figure 25, 26).

As shown in the transistor level simulation results, no numerical oscillations are observed.
Figure 20. MODELSIM Output waveforms for the Final Solver Design (4 unknowns)

Figure 21. MODELSIM Assertions for the Final Solver Design (4 unknowns)
Figure 22. CADENCE Output waveforms_1 for the Final Solver Design (4 unknowns) - 7 inputs

Figure 23. CADENCE Output waveforms_2 for the Final Solver Design (4 unknowns) - 5 inputs
Figure 24. CADENCE Output waveforms_3 for the Final Solver Design (4 unknowns) - 8 inputs

Figure 25. CADENCE Output waveforms_4 for the Final Solver Design (4 unknowns) - 25 inputs
4.2. Digital Linear Equation Solver with 8 unknowns

4.2.1. Final Design

First the circuit is simulated with the random inputs generated in MATLAB (Appendix 3.2) and no clock allocation is performed. Exactly the same logic is applied to the clocks as in chapter 4.1.2 and 4.1.3. They are generated in such a way that no clock is the integer multiple of the others. A default ordering is selected and clocks are defined as 6 ns, 8 ns, 10 ns, 12 ns, 14 ns, 16 ns, 18 ns, 20 ns each having their half of the period pulse width (Appendix A1.7.3), for the first DFF, second DFF, ... , for the last DFF, respectively. The simulation setup is provided in Appendix A1.7.3.

In addition, the random inputs are simulated with a transient simulation time of 5 us for each, since it is expected to be enough according to the simulations for the solver design with 4 unknowns. As the simulation results reveal (Figure 27 - 32), for some input cases the circuit fails, and numerical oscillations are observed (Figure 30 - for the first two inputs). Analyzing the problematic inputs points out the numerical oscillations as the cause of circuit failure, not the insufficient transient simulation time for each input (5 us).
Next, clock allocation is performed, and the benches for the inputs A and b as well as the clocks are generated in VHDL. Again it is important to note, **CK_distributor** block is not actually designed, the clocks are ordered according to the applied inputs in the test bench, instead.

Two problematic inputs (Figure 30 - first two inputs) are simulated as well as an additional input in which the circuit did not fail when no clock allocation was performed. No numerical oscillations for the problematic inputs are observed (Figure 33 - 34 and Figure 35 - 36). In addition, compared to the case in which no clock allocation was performed, now the same output is observed to be solved faster, namely the circuit settles down the solution faster (Figure 37 - 38).

![Figure 27. CADENCE Output waveforms_1 for the Final Solver Design (8 unknowns) - 5 inputs](image-url)
Figure 28. CADENCE Output waveforms_2 for the Final Solver Design (8 unknowns) - 5 inputs

Figure 29. CADENCE Output waveforms_3 for the Final Solver Design (8 unknowns) - 5 inputs
Figure 30. CADENCE Output waveforms_4 for the Final Solver Design (8 unknowns) - 5 inputs

Figure 31. Output waveforms_5 for the Final Solver Design (8 unknowns) - 9 inputs
Figure 32. Output waveforms_6 for the Final Solver Design (8 unknowns) - 6 inputs

Figure 33. Output waveforms_7 for the Final Solver Design (8 unknowns) - 1 input
Figure 34. Output waveforms_7 for the Final Solver Design (8 unknowns) - 1 input - Closer view

Figure 35. Output waveforms_8 for the Final Solver Design (8 unknowns) - 1 input
Figure 36. Output waveforms_8 for the Final Solver Design (8 unknowns) - 1 input - Closer view

Figure 37. Output waveforms_9 for the Final Solver Design (8 unknowns) - 1 input
Figure 38. Output waveforms for the Final Solver Design (8 unknowns) - 1 input - Closer view
5  Further Improvements and Comments

The design methodology for the digital linear equation solvers introduced in this project is verified in small scale. The correct operation of the final solver with 4 unknowns is demonstrated for each possible input. Next, the scale is extended to systems of equations with 8 unknowns. The digital solver is designed by following the same principles, and its correct operation is also demonstrated. Unfortunately, not all of the possible input patterns could be simulated as in the case with 4 unknowns, since it is practically impossible (the number is on the order of hundred thousand). In fact, a mathematical prove for the correct operation of such solvers for the larger scales is of great importance, and this requires further extending the scale for the solver, and performing many simulations. By this way, such a mathematical model can be deduced.

Another issue is the increasing number of different clock domains that have to be introduced as the scale goes up. Actually, for the case with 64 unknowns, 64 different clock domains have to be provided. A useful approach to overcome this problem could be the utilization of a single clock and its delayed versions (shifted versions). However, further analysis is required for such an implementation. Furthermore, clock allocation scheme gets complicated as the scale goes up. Up to this point, a basic LUT is implemented in VHDL for this purpose, however, a more complicated implementation is required for larger matrix sizes. A useful approach regarding this problem could be the implementation of CK_distribution block as a FSM, and again further analysis is required.

Last but most important point is the timing estimation for larger scales. Namely, a mathematical model for the estimation of timing complexity of such large scale solvers is of great importance.

As a final remark, regarding with the implementation of the large scale solvers, the solver hardware itself can be designed in full custom way, and the performances of the gates can be optimized, however, the clock allocation hardware needs to be designed in semi custom way.
6 Conclusions

In this project, a systematic way of designing digital linear equation solvers with output feedbacks is provided. First the methodology is implemented in small scale (system of equations with 4 unknowns) both in VHDL and transistor level. It is verified by both VHDL logic simulations and transistor level simulations. Next, the scale is extended to the system of equations with 8 unknowns. This time, it is implemented only in transistor level, and many simulations are performed. The correct operation of the corresponding solver is verified.

As a final remark, despite the proposed methodology demonstrated to be functional, further research and improvements in order to facilitate the implementation (e.g. the clocking scheme and the allocation of clocks) are required in larger scales.
7 References

1 - "Encryption", Data retrieved on 4 June 2010 from

2 - "Side Channel Attack", Data retrieved on 4 June 2010 from

3 - "A5/1", Data retrieved on 4 June 2010 from
http://en.wikipedia.org/wiki/A5/1

4 - B. Driessen, "Towards Solving the Linear Equations in $Z_2$ in the analog domain", PowerPoint presentation, EPFL, 8 March 2010.

Appendices

A1 Basic Building Blocks of the Solver

A1.1. Inverter Cell

Figure 39. Schematic of INV gate
A1.2. AND Cell

Figure 40. Schematic of AND gate

Figure 41. Schematic of NAND gate
A1.3. XOR Cell

Figure 42. Schematic of XOR gate
A1.4. DLATCH Cell

Figure 43. Schematic of DLATCH gate
A1.5. DFF Cell

Figure 44. Schematic of DFF gate
A1.6. Input Buffers

Figure 45. Schematic of 81b Input Buffer
A1.7. Simulation Setups

A1.7.1. Simulation Setup for the Initial Solver (4 unknowns)

Figure 46. Simulation setup for the Initial Solver design (4 unknowns)
A1.7.2. Simulation Setup for the Revised Solver (4 unknowns)

Figure 47. Simulation setup for the Revised Solver (4 unknowns)

Figure 48. 4 Different Clock Domains for the Revised Solver - 6 ns, 10 ns, 14 ns, 18 ns periods
A1.7.3. Simulation Setup for the Final Solver (8 unknowns)

Figure 49. Simulation setup for the Final Solver (8 unknowns)

Figure 50. 8 Different Clock Domains for the Final Solver - 6 ns, 8 ns, 10 ns, 12 ns, 14 ns, 16 ns, 18 ns, 20 ns periods
A2  VHDL Codes

A2.1.  Initial Solver (4 unknowns)

A2.1.1.  GenSolver4_noDFF.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
extentity GenSolver4 is
Port ( 
    A_in   : in  STD_LOGIC_VECTOR (1 to 16);
    B_in   : in  STD_LOGIC_VECTOR (1 to 4);
    X_out  : out STD_LOGIC_VECTOR (1 to 4)
    );
end GenSolver4;

architecture Behavioral of GenSolver4 is
signal Xout1, Xout2, Xout3, Xout4: std_logic;
beg
begin
process (A_in, B_in, Xout1, Xout2, Xout3, Xout4)
beg
begin
Xout1 <= B_in (1) xor (A_in (2) and Xout2) xor (A_in (3) and Xout3) xor (A_in (4) and Xout4);
Xout2 <= B_in (2) xor (A_in (5) and Xout1) xor (A_in (7) and Xout3) xor (A_in (8) and Xout4);
Xout3 <= B_in (3) xor (A_in (9) and Xout1) xor (A_in (10) and Xout2) xor (A_in (12) and Xout4);
Xout4 <= B_in (4) xor (A_in (13) and Xout1) xor (A_in (14) and Xout2) xor (A_in (15) and Xout3);
end process;

X_out(1) <= Xout1;
X_out(2) <= Xout2;
X_out(3) <= Xout3;
X_out(4) <= Xout4;
end Behavioral;

A2.2. Revised Solver (4 unknowns)

A2.2.1. GenSolver4.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity GenSolver4 is
Port (
  A_in   : in  STD_LOGIC_VECTOR (1 to 16);
  B_in   : in  STD_LOGIC_VECTOR (1 to 4);
  CK1    : in  STD_LOGIC;
  CK2    : in  STD_LOGIC;
  CK3    : in  STD_LOGIC;
  CK4    : in  STD_LOGIC;
  R      : in  STD_LOGIC;
  X_out  : out STD_LOGIC_VECTOR (1 to 4)
);
end GenSolver4;
architecture Behavioral of GenSolver4 is
signal Xout, rXout1, rXout2, rXout3, rXout4: std_logic;
begin
process (A_in, B_in, rXout1, rXout2, rXout3, rXout4)
begins
Xout (1) <= B_in (1) xor (A_in (2) and rXout2) xor (A_in (3) and rXout3) xor (A_in (4) and rXout4);
Xout (2) <= B_in (2) xor (A_in (5) and rXout1) xor (A_in (7) and rXout3) xor (A_in (8) and rXout4);
Xout (3) <= B_in (3) xor (A_in (9) and rXout1) xor (A_in (10) and rXout2) xor (A_in (12) and rXout4);
Xout (4) <= B_in (4) xor (A_in (13) and rXout1) xor (A_in (14) and rXout2) xor (A_in (15) and rXout3);
end process;
FF1: process (CK1, R)
begin
if(R = '0') then
   rXout1 <= '0';
else
   if(CK1'event and CK1 = '1') then
      rXout1 <= Xout (1);
   end if;
end if;
end process FF1;
FF2: process (CK2, R)
begin
if(R = '0') then
   rXout2 <= '0';
else
   if(CK2'event and CK2 = '1') then
      rXout2 <= Xout (2);
   end if;
end if;
end process FF2;
FF3: process (CK3, R)
begin
if(R = '0') then
   rXout3 <= '0';
else
   if(CK3'event and CK3 = '1') then
      rXout3 <= Xout (3);
   end if;
end if;
end process FF3;
rXout3 <= Xout (3);
end if;
end if;
end process FF3;

FF4: process (CK4, R)
begin
if(R = '0') then
  rXout4 <= '0';
else
  if(CK4'event and CK4 = '1') then
    rXout4 <= Xout (4);
    end if;
  end if;
end process FF4;
X_out(1) <= rXout1;
X_out(2) <= rXout2;
X_out(3) <= rXout3;
X_out(4) <= rXout4;
end Behavioral;
A2.2.2. GenSolver4_tb.vhd

As mentioned before, this test bench includes all possible input matrices, 27008 inputs (1688 A matrices and 16 b matrices for each). It is used to verify the correct operation of the final solver. Since the code is enormously long, around couple hundred thousand lines, it is not included in this report.

A2.3. Final Solver (4 unknowns)

A2.3.1. CK_prioritizer.vhd

CK_prioritizer basically consists of multiple case statements, which acts as CK_distributor block described in the corresponding section. It allocates the clocks based on the number of the coefficients that have 1-value for each specific input matrix A row. Priority issue is also taken into consideration, it assumes a default priority. For testing purposes of the currently adopted clock allocation scheme, it is implemented as a LUT in which for each input case, a corresponding allocation scheme of the clocks is defined. For the extended scales (for higher number of unknowns), it has to be effectively designed as a FSM. Since the code is considerably long, it is not included in the report.

A3 MATLAB codes

A3.1. All_input_matrices_4x4.m

```matlab
function all_input_matrices_4x4()
    count = 0;
    fid = fopen('exp_1.txt', 'w');
    for j = 0:2^12-1
        m(1,1) = 1;
        m(1,2) = bitget(j,1);
        m(1,3) = bitget(j,2);
        m(1,4) = bitget(j,3);
        m(2,1) = bitget(j,4);
        m(2,2) = 1;
        m(2,3) = bitget(j,5);
        m(2,4) = bitget(j,6);
        m(3,1) = bitget(j,7);
        m(3,2) = bitget(j,8);
        m(3,3) = 1;
        m(3,4) = bitget(j,9);
```
m(4,1) = bitget(j,10);
m(4,2) = bitget(j,11);
m(4,3) = bitget(j,12);
m(4,4) = 1;
m = gf(m,1);
if rank(m) == 4
    for t = 0:15
        count = count + 1;
        b = (bitget(t,4:-1:1))';
        b = gf(b,1);
        x = inv(m)*b;
        x = x';
        Temp = 'R <= '''1'' ';  
        Temp(length(Temp) + 1)='; '
        Temp=[Temp char(10) 'wait for 2 ns'];
        Temp(length(Temp) + 1)='; '
        Temp = [Temp char(10) 'R <= ''0'' '];
        Temp(length(Temp) + 1)='; '
        Temp=[Temp char(10) 'wait for 27 ns'];
        Temp(length(Temp) + 1)='; '
        Temp = [Temp char(10) 'R <= ''1'' '];
        Temp(length(Temp) + 1)='; '
        fprintf(fid, '%s\n\n\n',Temp);
        Temp
        Atxt= ['A <= ' ''];
        btxt= ['b <= ' '' ];
        xtxt= ['result := ' '' ];
        for i=1:4
            for k=1:4
                if m(i,k) == 1
                    Atxt=[Atxt ''1'','']
                else
                    Atxt=[Atxt ''0'','']
                end
            end
            if i~=4
                Atxt=[Atxt char(10)];
            end
        end
        Atxt(length(Atxt))=');
        Atxt(length(Atxt) + 1)='; '
        fprintf(fid, '%s\n\n\n',Atxt);
        display(Atxt);
        for q=1:4
            if (b(q)==1)
                btxt=[btxt ''1'' ];
            else
                btxt=[btxt ''0'' ];
            end;
        end
        btxt(length(btxt))='');
        btxt(length(btxt) + 1)='; '
        fprintf(fid, '%s\n\n\n',btxt);
        Temp2='wait for 10000 ns; ';
        fprintf(fid, '%s\n\n\n',Temp2);
        display(btxt);
        Temp2
        for p=1:4
            if (x(p)==1)
                xtxt=[xtxt ''1'' ];
            end;
        end
        xtxt(length(xtxt))='');
else
    xtxt=[xtxt '0', '];
end;
end
xtxt(length(xtxt))=');
xtxt(length(xtxt) + 1)=';
fprintf(fid, '%s


', xtxt); Temp3 = 'assert(X_out = result) report "Incorrect Result!"; '
fprintf(fid, '%s

', Temp3); display(xtxt);
end
end

count
fclose(fid);
end

A3.2. Rand_GenTestcase.m

%% cleanup
clear all;
close all;
clc;

%% GenerateLinEquSystem
%% this part generates the Matrix and the vectors at random

N=8;
umRounds=5;

A=eye(N);
b=round(rand(N, 1));
X=b;

%generate a random matrix by adding rows at random for numRounds rounds.
for R=1:numRounds
    for i = 1:N
        for j=1:N
            if ((randn<0)&&(i~=j))
                A(j,:)=mod(A(j,:)+A(i,:),2);
b(j)=mod((b(j)+b(i)),2);
            end;
        end;
    end;
end;

%make sure that all diagonal elements are 1 (assumption we made so far in %our solvers)
for i=1:N
    if (A(i,i)~=(N))
        A(i,i)=1;
b(i)=mod(b(i)+X(i),2);
    end;
end;
end;
end;

%% Generate txtfile
% here a copy-paste template for VHDL is prepared
Atxt=('{
btxt=('{
xtxt=('{

for i=1:N
    for j=1:N
        if (A(i,j)==1)
            Atxt=[Atxt '''1''',']
        else
            Atxt=[Atxt '''0''',']
        end;
    end
    if (b(i)==1)
        btxt=[btxt '''1''',']
    else
        btxt=[btxt '''0''',']
    end;
    if (x(i)==1)
        xtxt=[xtxt '''1''',']
    else
        xtxt=[xtxt '''0''',']
    end;
    if i~=N
        Atxt=[Atxt char(10)]
    end;
end;
Atxt(length(Atxt))=')'
btxt(length(btxt))=')'
xtxt(length(xtxt))=')'

Atxt
btxt
xtxt

%% solve gaussian style
% Here the equation system is solved once, to check if there are multiple
% solutions
AAA=A;
BBB=b;
for j=1:N %for each row
    if (AAA(j,j)==0) %if the first element of the row is not 1, swap rows
        for K=j+1:N
            if (AAA(K,j)==1)
                Temp=AAA(K,:);
                AAA(K,:)=AAA(j,:);
                AAA(j,:)=Temp
                Temp2=BBB(K);
                BBB(K)=BBB(j);
                BBB(j)=Temp2
                break;
            end
        end
    end
end;
if \( K = N \) \%if you cannot find a 1 in the whole columns, there are multiple solutions
    error('Do not use these values, there is no single unique solution!');
end;

for i = j+1:N \%use gauss to remove all other 1s in the column to move towards triangle shape
    if \( AAA(i,j) == 1 \)
      AAA(i,:) = mod(AAA(j,:) + AAA(i,:),2);
      BBB(i) = mod(BBB(i) + BBB(j),2);
    end;
end;

elseif if the last row is zero only
    if (sum(AAA(end,:))==0)
        error('Do not use these values, there is no single unique solution!');
    end;
end;

\%go from triangular shape to diagonal elements only shape
for j=N:-1:2
    for i = j-1:-1:1
        if \( AAA(i,j) == 1 \)
            AAA(i,:) = mod(AAA(j,:) + AAA(i,:),2);
            BBB(i) = mod(BBB(i) + BBB(j),2);
        end;
    end;
end;

\%BBB holds the solution for X here