Project Number: MQP-BS2-0803

SRAM Characteristics as Physical Unclonable Functions

A Major Qualifying Project Report

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Robyn Colopy

Jatin Chopra

Date: March 6, 2009

Sponsored by:

General Dynamic C4 Systems

Approved:

Professor Berk Sunar

Abstract

The purpose of this project is to determine whether the initial contents of SRAM, when treated as a Physical Unclonable Function, are a reasonable choice for generating an encryption key for an FPGA configuration bitstream. The idea of an SRAM PUF was previously proposed by Gerardo et al and we verified his results with a different FPGA family. We characterize the contents of SRAM on restart. Using MATLAB we statistically analyze the data to determine how the contents vary with respect to the multiple dimensions. Once the characterization is complete it will be possible to determine whether a certain bit pattern is likely to have come from the given SRAM.

Acknowledgements

We are extremely grateful for all of the people who have helped us in completing this project. We could not have done it without the on-going support, advice and the guidance of everyone involved.

We would like to thank Professor Berk Sunar, our advisor. We are appreciative of his time and knowledge in helping us achieve our goals and overall success for this project.

We are also grateful for our on-site advisors, Evan Custodio, Brendon Chetwynd and Gerardo Orlando. Their technical and administrative support has helped us in numerous occasions and we are very appreciative of all their efforts.

We would also like to thank General Dynamics C4 Systems and Worcester Polytechnic Institute for making this opportunity available to us and allowing us to take advantage of this program.

Table of Contents

Abstract i
Acknowledgements ii
Table of Contents iii
Table of Figures iv
Problem Statement 1
Literature Review 2
  FPGA 2
  SRAM 2
  PUF 4
  Tools 4
    Virtex-5 Development Board 4
    Xilinx ISE 5
    MATLAB 6
    IP Power 9258 6
Goals 7
Implementation 7
  Reading the Memory 7
  Serial Out 9
  Automating the Reset 11
  Serial In 13
Results 18
  Analysis 19
    Uniformity 19
    Stability 22
    Spatial Correlation 26
    Entropy Estimation 27
    Across Boards 28
Future Considerations 30
Conclusion 31
References 32
Appendix A: MATLAB UART Receiver 33
Appendix B: MATLAB Main Program (per board) 35
Appendix C: MATLAB Single Set Analysis 37
Appendix D: MATLAB Board Uniformity Analysis 40
Appendix E: MATLAB Board Entropy Analysis 42
Appendix F: MATLAB Single Set Entropy Analysis 43
Appendix G: MATLAB Board Stability Analysis (Bitwise) 44
Appendix H: MATLAB Board Stability Analysis (Bytewise) 46
Appendix I: MATLAB Board Uniformity Analysis 47
Appendix J: MATLAB Join Board Samples 49
Appendix K: MATLAB Merge Data Sets 50
Appendix L: Comparing Memories 53
Appendix M: VHDL SRAM & UART 54
Appendix N: UCF File 58

Table of Figures

Figure 1: Cloning Attack 1
Figure 2: Six transistor SRAM cell........ 3
Figure 3: Threshold Voltage and Noise 4
Figure 4: Virtex-5 family members 5
Figure 5: Automated Reset System........ 7
Figure 6: SRAM Read/Write State Machine 8
Figure 7: SRAM Reading State Machine 9
Figure 8: UART Protocol........ 9
Figure 9: UART Transmitter State Machine 10
Figure 10: Final UART State Machine........ 11
Figure 11: Commands to power cycle the power management device 12
Figure 12: Flow-chart outlining MATLAB function to automate the reset........ 13
Figure 13: MATLAB script to automate the reset 13
Figure 14: Simple MATLAB code for serial reception 14
Figure 15: MATLAB's data collecting and storing procedure........ 15
Figure 16: MATLAB commands that initiate the serial port........ 16
Figure 17: MATLAB commands that ensure that COM1 is available 16
Figure 18: MATLAB flow chart with added power-management feature 17
Figure 19: Measurements per Time Interval........ 19
Figure 20: Uniformity vs. Time Off 20
Figure 21: Percent Ones Distribution 21
Figure 22: Memory Map........ 22
Figure 23: Stability vs. Time Off........ 23
Figure 24: Stability vs. Samples 23
Figure 25: Distribution of Stable Bits........ 24
Figure 26: Distribution of Unstable Bits........ 25
Figure 27: Stability Histogram........ 25
Figure 28: Stable Bytes........ 26
Figure 29: Byte Occurrence........ 26
Figure 30: Zeros per byte........ 27
Figure 31: Entropy Equation........ 27
Figure 32: Entropy vs Time Off........ 28
Figure 33: Difference Between Boards 1 and 2........ 29

Problem Statement

Many Intellectual Property (IP) vendors sell Field Programmable Gate Array (FPGA) hardware designs to external parties. There is a concern that attackers may steal and clone the designs without paying the original designer. This results in a loss of revenue to the vendor.

A configuration bitstream, which represents an FPGA design, is stored in external, non-volatile memory such as Programmable Read-Only Memory (PROM). When an FPGA is powered up, the bitstream is used to automatically load the design. Because the bitstream is stored external to the FPGA, it is possible to tap the line connecting them. This attack allows copying and cloning, as seen in Figure 1 below. An attacker can illegally program other FPGAs without paying licensing fees to the IP vendor.

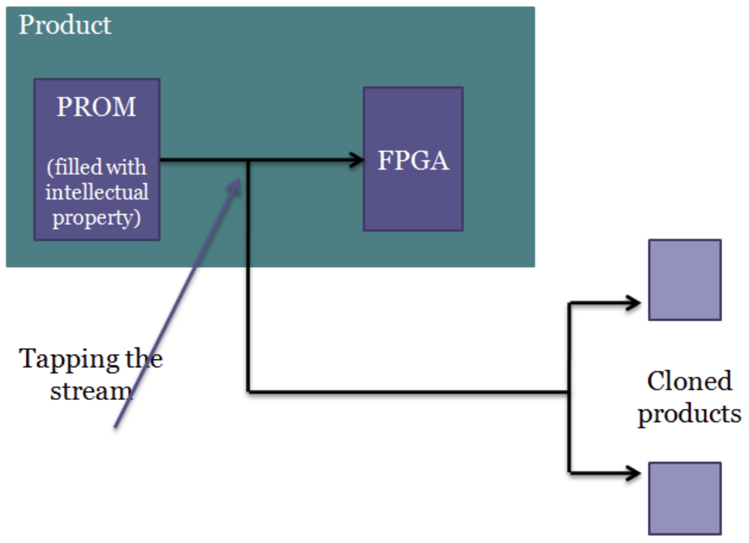


Figure 1: Cloning Attack

One proposed solution to this problem is to encrypt the bitstream. This would prevent an attacker from gaining meaningful information through tapping. But where is the encryption key to be stored? One proposed method is to place the key on a non-volatile ROM. Another is to use a volatile ROM with a backup battery. However, these solutions have additional costs associated with them.

An alternative solution is to use an intrinsic Physical Unclonable Function (PUF) to create a key without overhead. Static Random Access Memory (SRAM), which is internal to many FPGAs, has been proposed as a source of this PUF [3]. We characterize the startup state of two external, independent SRAMs in order to analyze their suitability as the source of a PUF.

Literature Review

FPGA

An FPGA is a semiconductor device that can be programmed and configured after manufacturing. FPGAs can be re-programmed many times after deployment, which is an advantage over ASICs, whose designs must be completed prior to manufacture. FPGAs are programmed through a source code which describes the logical function to be implemented. One language commonly used in FPGA design is VHDL.

The typical FPGA architecture consists of an array of configurable logic blocks (CLBs), rather than solely transistors. These logic blocks contain memory used to implement logic functions and can be a composition of transistor pairs, multiplexers, basic small gates such as two-input NAND gates, exclusive-OR gates or look-up tables (LUTs). These logic blocks are then connected together with wire segments of varying length interconnecting each other by electronically programmable switches which are configured during the synthesis process via a computer. After the design is downloaded onto the device, a routing architecture is created and implemented on the board [8].

SRAM

SRAM is a type of semiconductor memory consisting of CMOS transistors. Unlike dynamic RAM (DRAM) which must be periodically refreshed, SRAM is based on a bi-stable latch which will retain its value as long as the circuit is powered.

Each bit is made of 6 transistors, arranged as two cross-coupled inverters and two access switches, as show in Figure 3 below. This bit has two stable states to represent either a logic zero or a logic one. There are two additional transistors, labeled as ‘M5’ and ‘M6’ in the figure below, which are known as access transistors since they control the access to the storage cell during write and read operations.

In order to read or write to the cell, the word line labeled as ‘WL’ needs to be enabled. This in-turn connects the two access transistors, the bit lines, labeled as BL and BLC, in Figure 2. The presence of two access lines aids in reducing noise. [3]

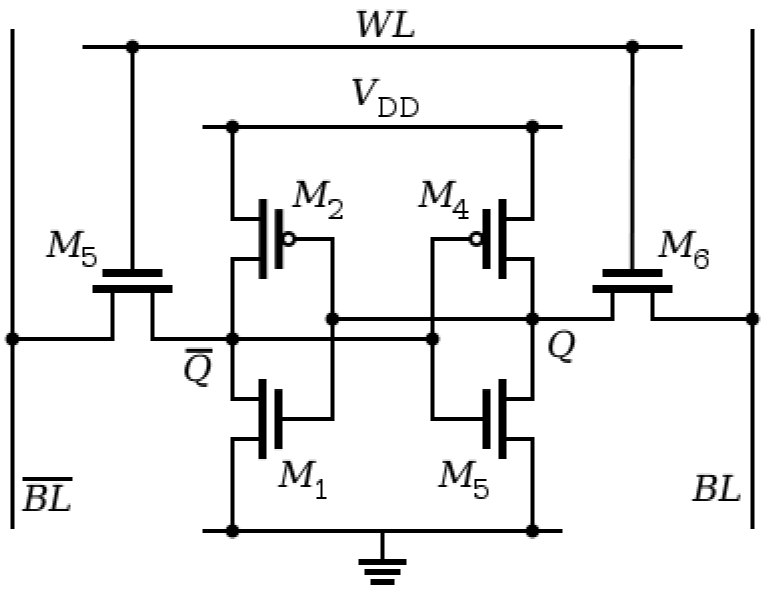


Figure 2: Six transistor SRAM cell

Source: [10] (Creative Commons)

When the SRAM is off, the input to each inverter is 0. However, due to the function of an inverter this is an unstable state which must change once the SRAM is turned on. The two inputs must become either 01 or 10. Which state is taken on is dependent on the characteristics of the transistors making up each cell [4].

Although ideally each transistor would be identical, in actuality they vary slightly due to uncontrollable factors in the manufacturing process such as dopant concentrations. Variations in the relative threshold voltages of the transistors cause each cell to tend toward a 1 or a 0. The more closely matched the threshold voltages are, the more influence noise will have over the initial state, possibly causing the bit to flip. These variations can be seen in Figure 3.

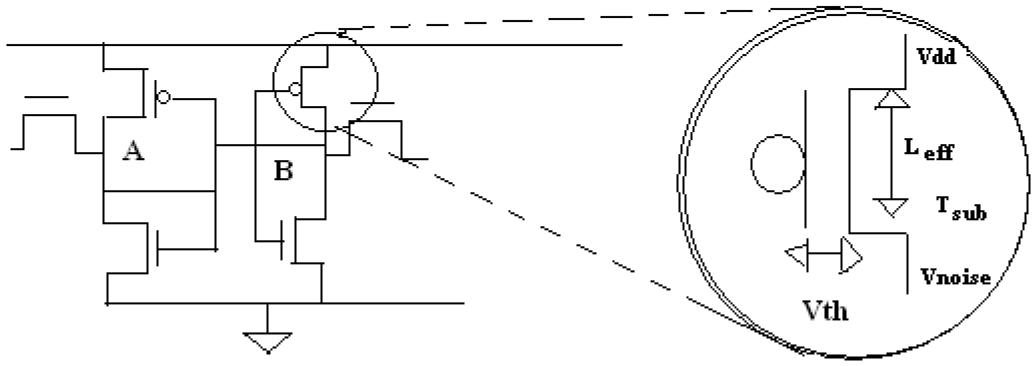


Figure 3: Threshold Voltage and Noise

Source: Reproduced from [4] (Holcomb)

PUF

Physical Unclonable Functions are the result of random uncontrollable variables in the manufacturing process. They are measurable but meaningless aspects of a physical system, and are unique in each instance. A PUF can be used as a source of random but reliable data for applications such as generating encryption keys.

An intrinsic PUF is one that is the result of a preexisting manufacturing process, and does not require any additions, such as a coating, to be used. Since there is no overhead, an intrinsic PUF is cost effective.

If a good PUF is considered as a challenge and a response, (i.e., if section A is measured, result B is found), then one pair of challenges and responses should have no bearing on another pair. Additionally, if one tries to "take apart" or tamper with a PUF, the PUF will no longer function [3].

Tools

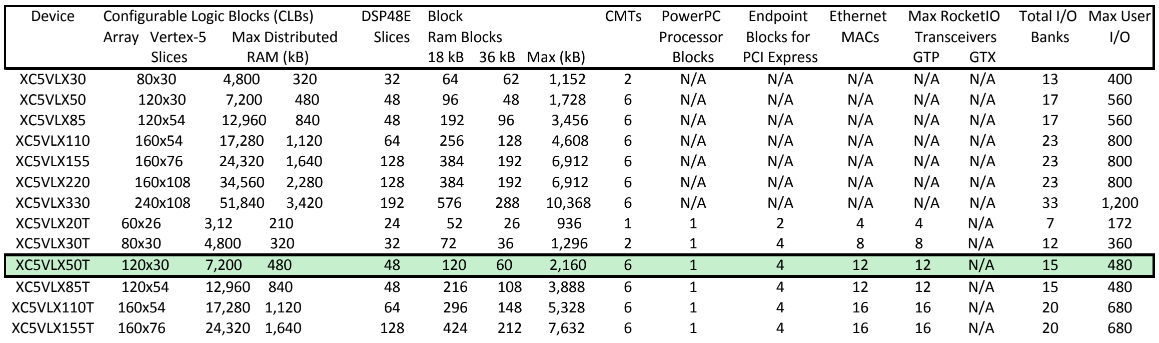
Virtex-5 Development Board

The Virtex-5 family consists of high end Xilinx FPGAs which contain up to 330,000 logic cells, 207,360 internal fabric flip-flops as well as 207,360 six-input look-up tables. The Configurable Logic Block (CLB), which is the basic logic elements for FPGAs, provides synchronous and combinatorial logic. The CLBs on the Virtex-5 differ from previous generations because they are based on six-input look-up table technology which provides a better performance [9].

The Virtex-5 LXT ML505 is the general purpose FPGA development board created by Xilinx which we were using throughout this project.

This board, along with an on-board memory, has many capabilities such as industry standard connectivity interfaces. The board has a 9 Mb Zero-Bus Turnaround (ZBT) synchronous SRAM which communicates by using a 32-bit data bus with four parity bits. Other features on this board include a JTAG configuration port and SPI FLASH. It also has eight general purpose DIP switches, LEDs, pushbuttons and a rotary encoder. Communications can be made to transmit and receive data from the board via an RS-232 serial port using Universal Asynchronous Receiver/Transmitter (UART) communication. A USB interface chip with host is also available on the board [9].

Figure 4 shows a number of Virtex-5 family members comparing different options such as the number of the CLBs, block RAM, I/O Banks and Ethernet Media Access Control (MAC) on different devices available from Xilinx. The one that we are using is the Xilinx XC5VLX50T which is highlighted and has an adequate 120 x 30 array of CLBs and 2,160 Kb of total block RAM [9].



Device	Configurable Logic Blocks (CLBs)			DSP48E	Block			CMTs	PowerPC	Endpoint	Ethernet	Max RocketIO		Total I/O	Max User
	Array	Vertex-5	Max Distributed	Slices	Ram Blocks				Processor	Blocks for	MACs	Transceivers		Banks	I/O
		Slices	RAM (kB)		18 kB	36 kB	Max (kB)		Blocks	PCI Express		GTP	GTX		
XC5VLX30	80x30	4,800	320	32	64	62	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120x30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120x54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160x54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160x76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160x108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240x108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60x26	3,12	210	24	52	26	936	1	1	2	4	4	N/A	7	172
XC5VLX30T	80x30	4,800	320	32	72	36	1,296	2	1	4	8	8	N/A	12	360
XC5VLX50T	120x30	7,200	480	48	120	60	2,160	6	1	4	12	12	N/A	15	480
XC5VLX85T	120x54	12,960	840	48	216	108	3,888	6	1	4	12	12	N/A	15	480
XC5VLX110T	160x54	17,280	1,120	64	296	148	5,328	6	1	4	16	16	N/A	20	680
XC5VLX155T	160x76	24,320	1,640	128	424	212	7,632	6	1	4	16	16	N/A	20	680

Figure 4: Virtex-5 family members

Source: Reproduced from [9] (Xilinx)

The development board we are using has a 9 Mb high-speed, low-power synchronous SRAM designed by Integrated Silicon Solution Incorporated (ISSI). Designed to provide a continuous or ‘burstable’ read, it is a high-performance device for networking and communications applications. The architecture consists of 256K words by 36 bits with advanced CMOS technology. The device also incorporates a ‘no-wait’ state, where wait cycles are eliminated when switching between modes, such as read to write or write to read operation [6].

Xilinx ISE

Xilinx Integrated Software Environment (ISE) is the software we used to implement our design on our FPGA. It is a design software suite created by Xilinx which includes a design environment for creating a top-level design using a Hardware Description

Language (HDL). We used VHDL to implement our design; however other languages such as Verilog, ABEL and schematic layout are also supported by the software.

Using the software, code is written that describes the design or function to be performed on the board. After the coding process, the software takes the created design and optimizes it during synthesis. It also verifies functionality and timing restrictions through its own simulations. The language design is then verified if it complies with Xilinx's rules of syntax and if successful, it is converted into a corresponding netlist file.

This file is then converted into a physical file format known as a bitmap and is ready to be downloaded onto the target device. After loading the design onto the FPGA, there are a number of ways to debug the design if it does not function properly. One method is by using an internal logic analyzer created by Xilinx known as ChipScope that can view internal signals or nodes on the board for debugging purposes [11].

MATLAB

MATLAB is a high-performance language used for technical computing with many applications including: data acquisition, math computation, data analysis, scientific and engineering graphics. It is a high-level language that can perform computing problems relatively faster compared with other languages because of its build-in functions and commands [7].

MATLAB also has input/output communication capabilities to a serial port and can also read and write to text-files or MATLAB's data format files with a .MAT extension. It also performs data analysis, such as matrix manipulation, data plotting and build-in functions that can perform common statistics [7].

IP Power 9258

Aviosys International Inc., developed IP Power 9258 which is an Ethernet-controlled device used to manage power on a time schedule. With this device, a user can control or query the power supply of up to four 120 V electrical outlets using pre-configured commands. Communication is through an RJ-45 Ethernet cable via the PC. Other capabilities of this device include an RS-232 port for debugging purposes and a manual on-off switch. It also has LEDs to indicate power usage of the device connected to it [5]. The automated reset system is shown below in Figure 5:

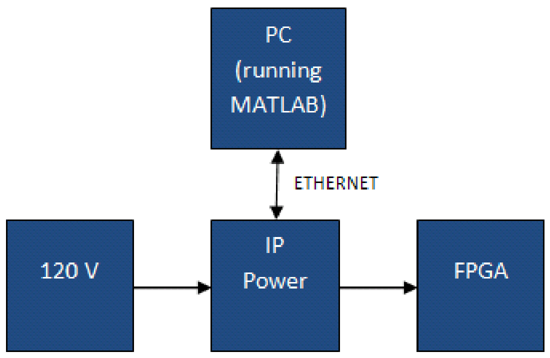


Figure 5: Automated Reset System

Goals

Our overall technical goal was to determine whether the initial state of SRAM creates a reasonable PUF for use as a cryptographic key. In order to meet this goal, we established several milestones. These are outlined below:

- Read entire memory, initiated by power-on
- Import entire memory to PC
- Continuous automated full reads, full imports
- Batch process samples
- Interpret results

Implementation

Reading the Memory

Our first goal was to read an entire memory block on the Virtex-5 development board. Initially, we intended to use the Block Random-Access Memory (BRAM), which is inside the Virtex-5 FPGA chip itself. However after research and further investigation, we discovered that in order to read values from the BRAM, we would first have to instantiate them. This instantiation process clears the BRAM and sets all of its bits to zero. This deletes the data which we are interested in. Therefore we instead decided to read values from the external SRAM which had no such instantiation process. Our development board has an external on-board SRAM which we used.

This on-board SRAM that we used is manufactured by Integrated Silicon Solution Incorporated (ISSI) and is 9 Mb arranged as 256K x 36. After careful analysis of its data sheet, we decided to implement a simple state machine which could both read from and write to a user defined address. Both the write and read could be triggered by the user, and the lower bits of the read data would be viewable on the board's LEDs. By both reading and writing, we could verify that our SRAM interface was working correctly. To accomplish this, we implemented the following state machine as shown in Figure 6:

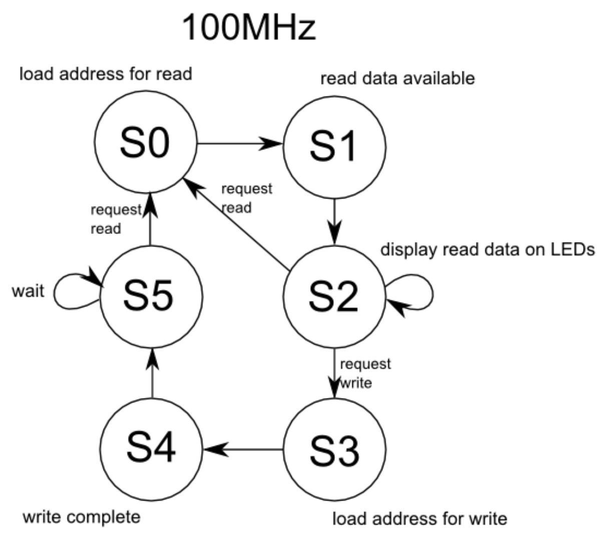


Figure 6: SRAM Read/Write State Machine

Once we verified that our reading interface was correct, we removed the "write" portion, and changed the address from user-selectable to a slow counter as shown in Figure 7. With these changes, we could see that that data changed with the addresses.

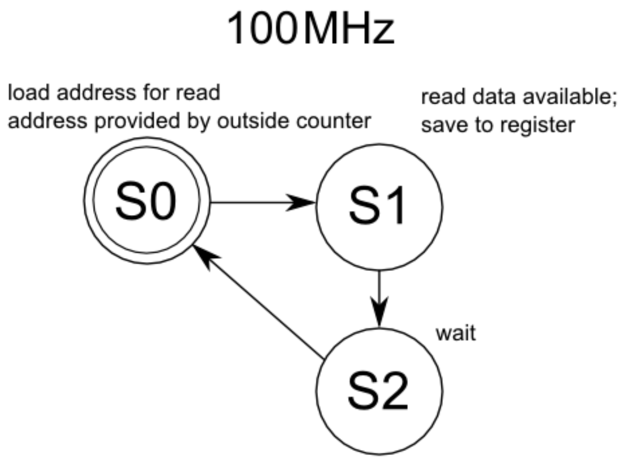


Figure 7: SRAM Reading State Machine

In this configuration, this state machine is always reading from the SRAM. However, the address that is being read will only change based on an external counter. As we began to work with the serial communications, we determined that the best time to change the address would be immediately after the "send" of the each address' data completed.

Serial Out

In order to save our reads of the SRAM, it was necessary to communicate with a PC. After analyzing several options, we determined that an RS-232 UART connection would be the easiest to implement.

A UART connection consists of one start bit, a set number of data bits, an optional parity bit, and one stop bit, sent at a baud rate know to both the transmitter and receiver. We chose to send 8 data bits at a time, at 115200 bits per second (bps), which is the fastest common baud. As a result, we wanted our transmit line to appear as in the Figure 8 below:

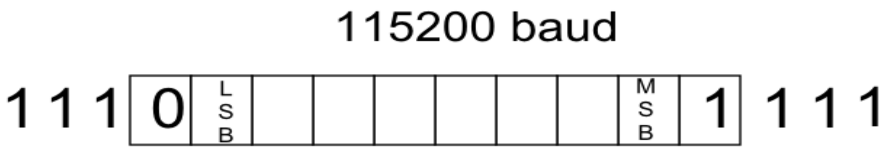


Figure 8: UART Protocol

Additionally, we determined that it would not be necessary for our FPGA to receive any data and only to transmit. Therefore we decided to implement our UART as a simple state machine which controlled the TX line of our development board's RS-232 port. The most basic form of this state machine is seen in Figure 9:

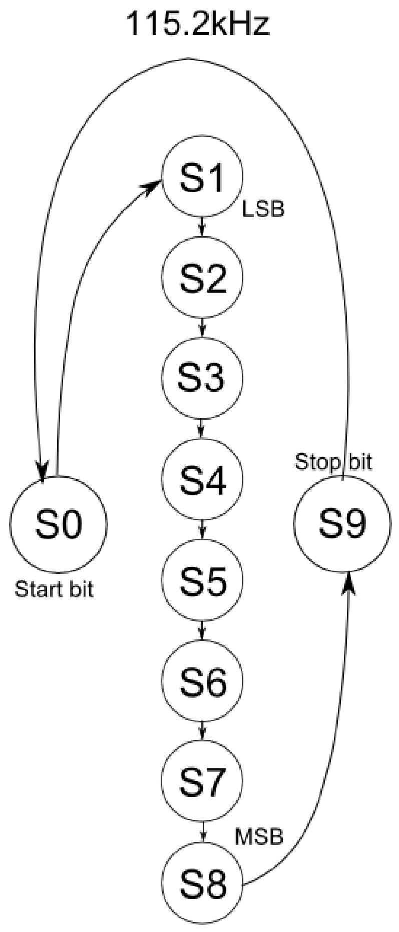


Figure 9: UART Transmitter State Machine

In order to tailor UART transmitter to our purposes, we made a few modifications. First, since we needed to transmit 36 bits for each address, we added a loop to send 5 packets at a time. Next, because the 100 MHz clock on our board does not divide down to exactly 115.2 kHz, we added an idle state after each word in which the clocks could resynchronize. This was also the state in which the address incremented. Finally, because our application requires the transmission to start immediately and accurately on power up and cease transmission once the entire addresses have been sent, we added start and stop states as shown in Figure 10.

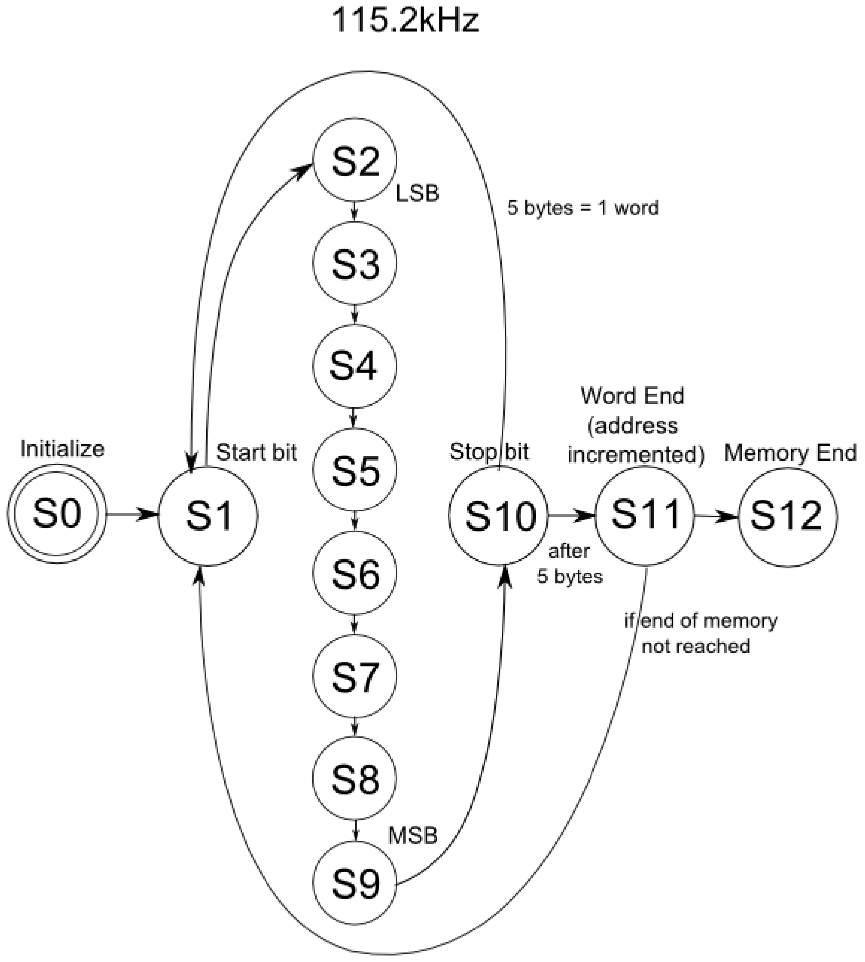


Figure 10: Final UART State Machine

Since need this design to be immediately available on start up, we used Xilinx to create a *.MCS file to be stored on the non-volatile PROM. This file automatically loads our design onto the FPGA on power up.

Automating the Reset

In order to perform analysis on the data from the SRAM, it would be beneficial to get as many multiple readings as possible. We initially performed a manual reset using the standard power-switch on the board. However, in order to obtain numerous samples with a time-precise rest period between readings, we researched a method to automate this process of resetting the board.

After considering several options, we ordered the IP Power9258 which would serve as our power management tool. Our motivations for this choice were a tradeoff between price and ease of use. We communicated to it via an Ethernet cable by sending commands through its IP address. The following commands shown in Figure 11 correspond to turning one outlet "on" and "off."

```
http://admin:12345678@192.168.1.207/Set.cmd?CMD=SetPower+P60=1


http://admin:12345678@192.168.1.207/Set.cmd?CMD=SetPower+P60=0
```

Figure 11: Commands to power cycle the power management device

Our next task was to find a way to send these two commands automatically, time-precisely and repeatedly. We also had to allow enough time for the board to boot-up properly from PROM, extract the data and finally increment a variable rest period between reads.

We decided to implement a simple MATLAB script. The command to turn the device on was linked to a browser shortcut labeled as 'on.url' and a similar command to turn the device off was linked to another labeled, 'off.url.' Once these shortcuts were created, MATLAB was able to execute these shortcuts within its development environment by opening a web-browser. We also configured it to close this web browser using the system 'task-kill' command. Between turning the board on and off, we also initiated a pause, using the 'pause' command. This pause was about 2 minutes, the time it takes to cycle through the entire SRAM. We also added another pause as the "off time". These commands were placed within a for-loop so for multiple repetitions. The program successfully automated the process of resetting the board. The flow-chart outlining this procedure is as follows in Figure 12:

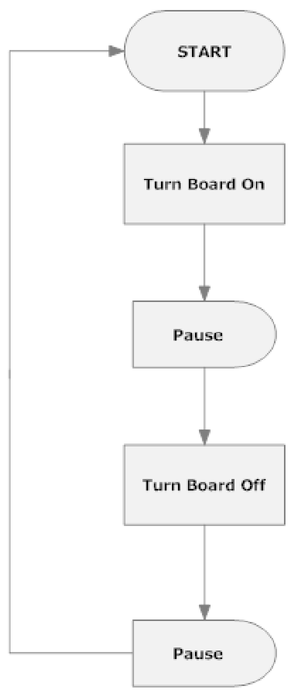


Figure 12: Flow-chart outlining MATLAB function to automate the reset

The script that was used to implement this is shown below in Figure 13:

```
for i=1:1000
    !d:/on.url                                 %Opens up a shortcut that turns
    sprintf('Webpage is on')                   %%power monitoring device on
     pause(1)
    system('taskkill /IM iexplore.exe')        %Closes the shortcut
    pause(126)                                 %Waits until board is done
    !d:/off.url                                %Opens up a shortcut that turns
    sprintf('Webpage is off')                  %%power monitoring device off
     pause(1)
    system('taskkill /IM iexplore.exe')        %Closes the shortcut
    pause(300)                                 %Provides a rest of 5 minutes
end
```

Figure 13: MATLAB script to automate the reset

Serial In

Next, we dealt with receiving information from the board. In order to verify the functionality of our UART design, we first used HyperTerminal, a communications program included in Windows XP. Although HyperTerminal was helpful in determining if data was being received, it was not capable of storing our data or displaying it as individual bits.

Once we verified that our design worked and communication was possible from the board, our next task was to read, save and separate the data from the SRAM. After researching several options, we found that MATLAB can complete all of these tasks and since we were planning on using it for analysis, it was perfect for our application.

After reading several help and example files related to MATLAB’s serial communications capabilities, we some simple commands to read some values from the serial port, one that simply defines the serial port and another that opens the communication to it. This script is shown below in Figure 14:

```
s = serial('COM1');     %setups the serial port to open COM1
fopen(s);   %opens the serial port connection
```

Figure 14: Simple MATLAB code for serial reception

After verifying that MATLAB could successfully communicate with the serial port, we modified our code to detect the end of a send, save the data with a timestamp and to maintain a list of all of the saved files. A block diagram of MATLAB’s data collecting and storing procedure is as follows in Figure 15:

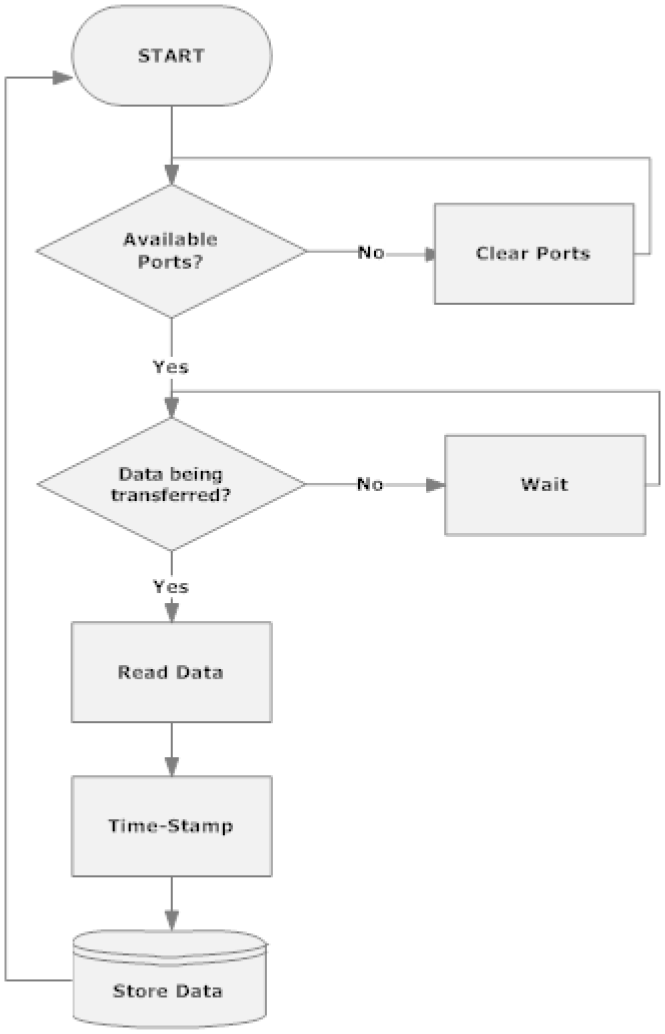


Figure 15: MATLAB's data collecting and storing procedure

MATLAB is able to communicate through the serial port at various baud rates. The maximum baud rate that MATLAB can read data is 115,200 baud. In order to initiate communication, we used MATLAB's 'serial' command to define the baud rate, the input buffer size, timeout length, port number, and whether to read asynchronously or synchronously. We chose 115,200 bits per second since this allowed for the fastest data transfer possible and an asynchronous read operation since we were not sending out any data through the UART or writing to the SRAM. The input buffer size was calculated based on the available space within the SRAM on the FPGA, which was 262144 words *40 bits = 1,310,720 bits. The timeout was defined as 500,000 seconds which allowed MATLAB to continuously have the serial port open and gather data for several days. Once these initial values were defined, MATLAB's 'fopen' command permitted MATLAB to access this port. The MATLAB script which brings all of these tools together is as follows in Figure 16:

```
                   %setup the serial port
s = serial('COM1', 'InputBufferSize', 1310720, ...
      'BaudRate', 115200, ...
      'Timeout', 500000, 'ReadAsyncMode', 'continuous');
fopen(s)                %opens the serial port connection
```

Figure 16: MATLAB commands that initiate the serial port

Once the initial communication was established, MATLAB's 'fread' command was used to start reading data and storing it to a temporary variable, labeled 'out'. It read 8 bits at a time to eventually fill an allocated 5 x 262144 space for doubles. Each reading was saved as a .MAT file only after the Input Buffer was filled. The filename contained the date, time, and off-time for the reading. This information was gathered using MATLAB's 'clock' and 'tic toc' commands. Saving the time was helpful during the analysis, ensuring us that the readings were well-timed and that there was no erroneous data.

Along with saving the data from the SRAM to MATLAB's data format files, it was also important to log what files were actually saved to retrieve them for future analysis. We saved the names of all of the files that were generated into another data format file which created a log of everything saved. This allowed us to streamline analysis by loading this log of all of the files generated, extracting each individual time-stamped data and performing analysis on each batch.

We initially had some difficulties getting MATLAB to automatically close the serial port after completing a read. Trying to open a serial port already in use resulted in an error which halted the program. To combat this, we took additional steps to ensure that the port closed before attempting to open a new one as shown in Figure 17.

```
d = instrfind('Port', 'COM1');           %lists used COM1 ports
    if length(d) ~= 0                    %if the port is in use
        fclose(d);                       %closes the port
        delete(d);                       %deletes those devices
    end
```

Figure 17: MATLAB commands that ensure that COM1 is available

This MATLAB code was created to be left overnight and over the weekends in order to gather as much data as possible. Due to this all of the MATLAB code was placed within a forever-while loop which was always running, ready to receive data whenever the board is ready to transmit.

We later combined the two MATLAB scripts that can both perform the automation of resetting the board and data collection and storage script. The finished script's procedure is illustrated below in Figure 18:

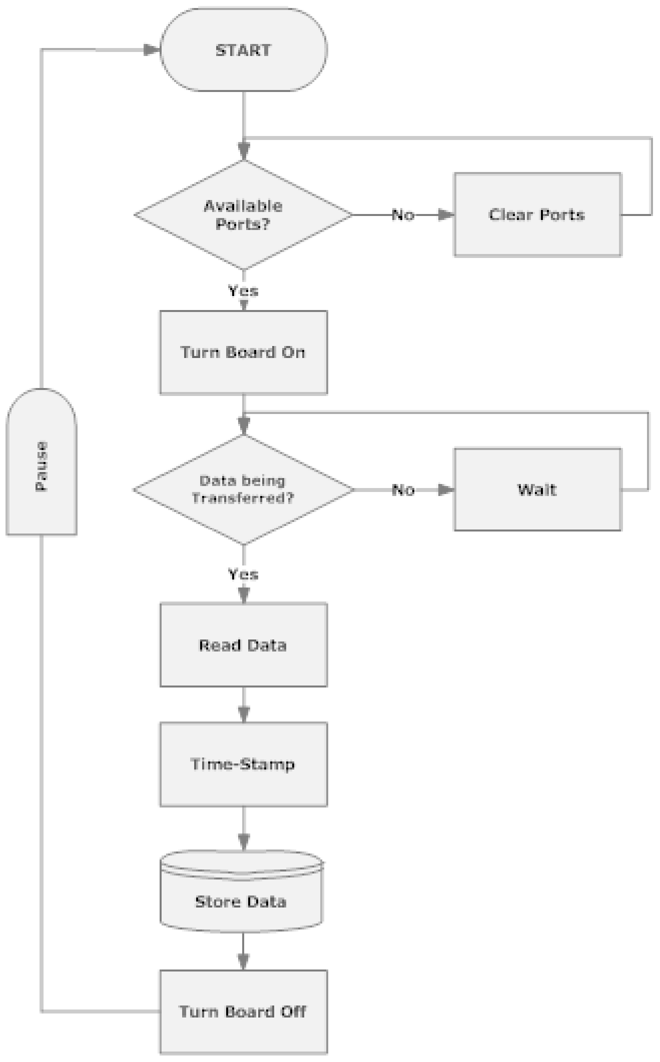


Figure 18: MATLAB flow chart with added power-management feature

Minor modifications can be made to this code to allow for sequential data gathering, i.e., gather 30 samples at 20 seconds off, then 20 samples and 30 seconds off.

Results

We took numerous samples varying the time-interval between resetting the board. We initially read from one board and later switched to a similar but independent, Virtex-5 LXT ML505 development board near the end of the project and compared the results.

For most of the time periods we took 30 samples. However for a few we took slightly less or more. Additionally for two time periods we took a much greater number of samples.

For one of the time-intervals, at a 300-second wait period, we took two batches of samples for the first board. One of these batches totaled 30 readings altogether and another batch totaled 500. We decided to keep both batches of data and perform analysis on them.

At the lowest intervals of 1 and 5 seconds, we encountered some instability in our MATLAB retriveal software and as a result gained fewer samples. We believe the reason for this instability was that the hardware did not have time to fully power down.

After we switched to the second board, we attempted to get as many readings as possible; however we were subject to time constraints. We also later discovered that a reading of a 300 wait period for the second board was erroneous and decided to delete the readings for this particular time interval. We did not get a chance to replace the readings for this particular time-interval of 300 seconds for the second board.

After all of the data was gathered, we created MATLAB scripts that interpreted the data to process the analysis in the most efficient way. The results were plotted using a number of visual graphs such as histogram, logarithmic and semi-logarithmic graphs.

Figure 19 shows how many measurements were taken for each board for a corresponding time-interval:

Time Off (seconds)	**Time Off**	**Board # 1 #Samples**	**Board # 2 #Samples**
1		**18**	**13**
5		**30**	**19**
10		**30**	**30**
20		**30**	**25**
30		**30**	**23**
40		**30**	**15**
50		**20**	**17**
60	**1 minute**	**400**	**30**
75		**12**	**28**
90		**30**	**18**
120	**2 minutes**	**30**	**19**
300	**5 minutes**	**30 and 500**	**0**
900	**15 minutes**	**150**	**18**
1800	**half an hour**	**30**	**13**
10800	**3 hours**	**7**	**0**

Figure 19: Measurements per Time Interval

The tests detailed below were all executed using MATLAB scripts included in the appendices.

Analysis

Uniformity

First, we attempted to find the uniformity of the memory. We wanted to know if the memory was composed of mostly ones, mostly zeros, or if it was about equal. In order to accomplish this, we took the sum of the bitmap and divided it by the total number of bits to obtain what percent of bits were one.

We found the uniformity of every sample we took, and plotted them together against the time the board was off. This is shown below in Figure 20. Red circles are data points from board 1, while the blue are from board 2.

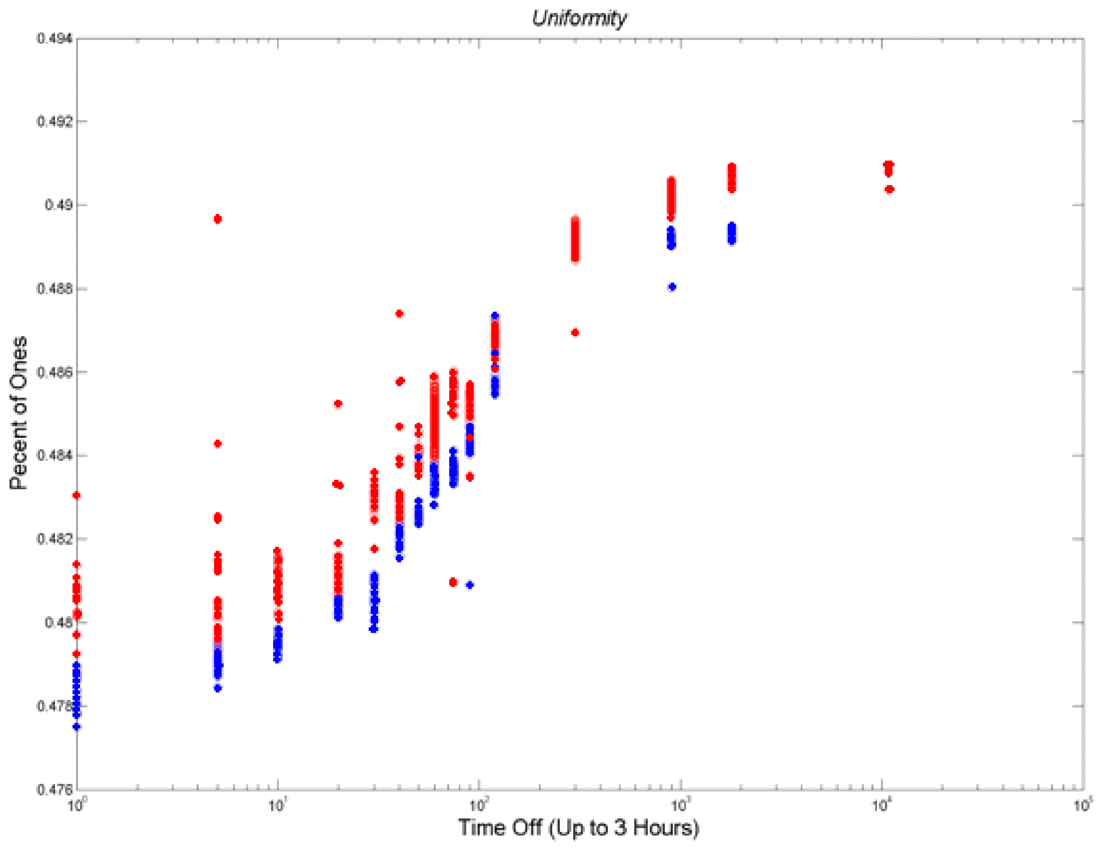


Figure 20: Uniformity vs. Time Off

All of the data points range from 47.5% to 49.5%, so there are slightly more zero bits than one bits. One percent represents about 100,000 bits.

A number of other trends are also visible in the figure. Both boards followed the same basic shape, but the second board (blue) tends to have lower values than the first.

The memory becomes more uniform as the time off increases, although there are a number of outliers. After examining our data, we found that these outliers occurred at the beginning of each data set, which implies there may be a residual affect from the previous reading. Since it is known that outside temperature can affect initial SRAM values, perhaps this is an artifact of the heat generated by constantly reading the memory at different rates. However, this would need further tests to verify.

It also seems like the range of values may decrease at longer intervals. It would be necessary to gather more samples to verify this, since we have very few samples at longer intervals.

Figure 21 shows the distribution of the uniformity for one of the larger datasets. It has a Gaussian distribution with a range of about .01% (10,000 bits).

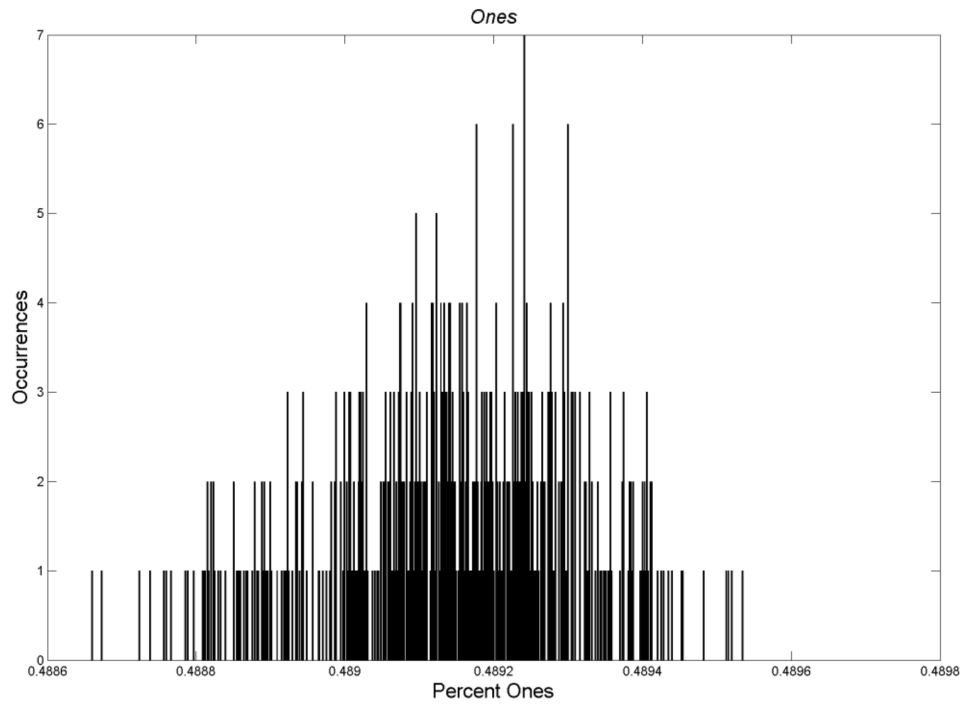


Figure 21: Percent Ones Distribution

Memory Map

Our next goal was to produce a map of the memory to see if any clear visual patterns appeared. For example, were there any large sections of all ones? To do this, we first translated our data, which came in as 8 bit integers arranged as 5x266144, into single bits arranged as 36x262144. We then used MATLAB's 'imagesc' command to create a simple visual representation. We did not see any noticeable patterns in this image.

By stacking several of these bitmaps on top of each other and taking the average of each bit, we were able to determine which bits changed or stayed the same over time. Figure 22 below shows the first hundred addresses of data. Those bits which were always the same are pure white or black. Those that showed a tendency toward a certain value are gray. The bits that changed most often are in red.

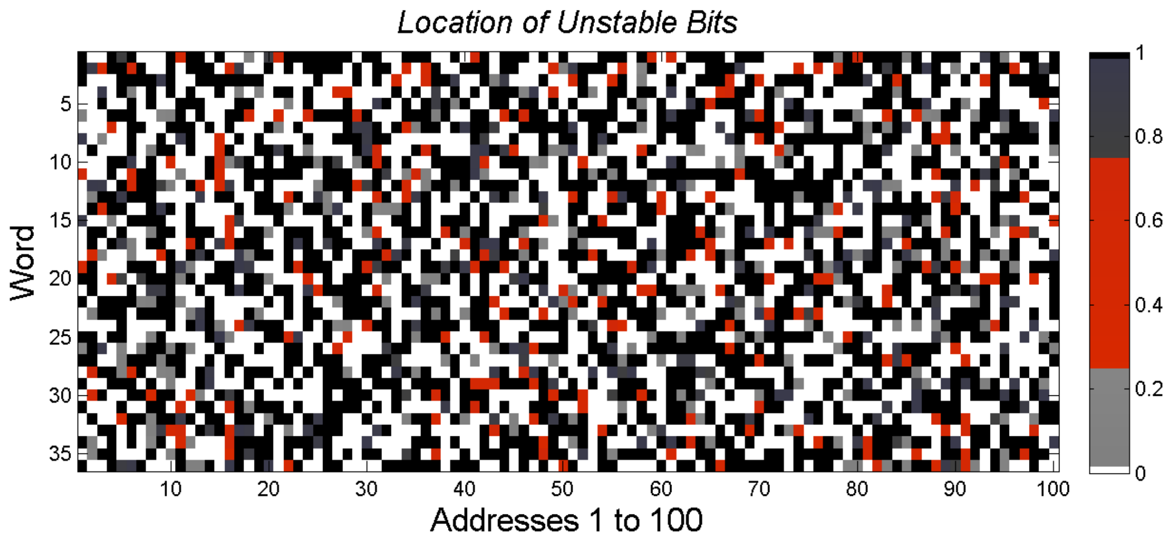


Figure 22: Memory Map

(Black = Always 1, White = Always 0, Red = Unstable)

This map shows that most bits remain constant, while relatively few are constantly changing. We also see that the proportion of ones to zeros appears about equal. We see that some blocks of values occur, but there are no tendencies over of the entire memory: for example, there are no rows or columns that are all ones.

Stability

Next, we wondered how many of the bits were stable, and whether the uniformity data we found still held when it was broken down further. For example, were stable bits more likely to be ones? Did bits that flipped take on a certain value more often?

We define 'stable' as a bit which does not change in any trial.

To accomplish this, we found the stability of each data set and plotted it against a number of variables. In Figure 23, we see that there does not appear to be a relationship between the number of stable bits and the length of time the board is off.

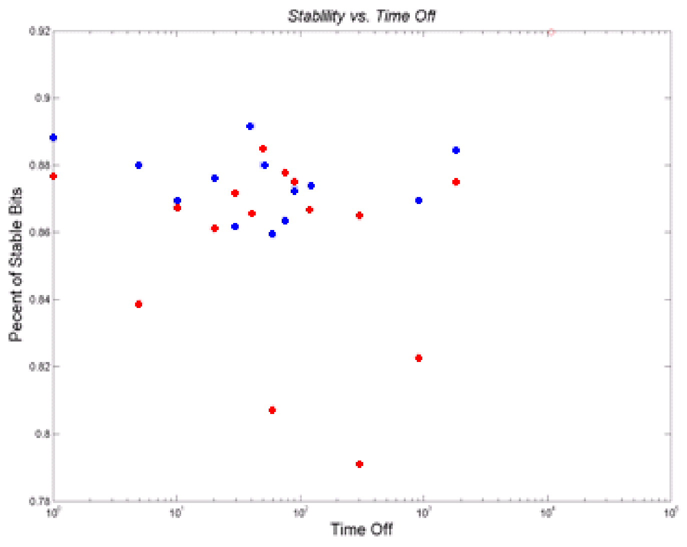


Figure 23: Stability vs. Time Off

In Figure 24 however, we see that those data sets with more samples had much greater instability. We do not mean to imply that taking the samples affects the stability of the bits; rather, many bits have the potential to flip and the more samples that are taken, the more likely it is that this flip will occur.

With only a few samples, we see 92% stability, however when the number of samples is much greater, 500, there is only 79% stability. It would be useful to gather data over a larger range of samples to see if there is a lower bound on the stability, and to see if the data truly follows the implied curve.

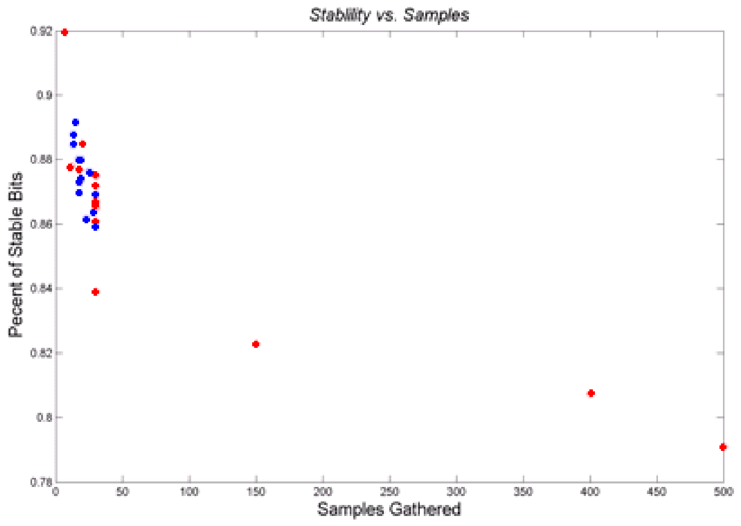


Figure 24: Stability vs. Samples

Next we wondered how the stable bits were distributed. Figure 25 shows that the stable bits follow that same uniformity trend as the overall data. Stable bits are not more likely to be a certain value.

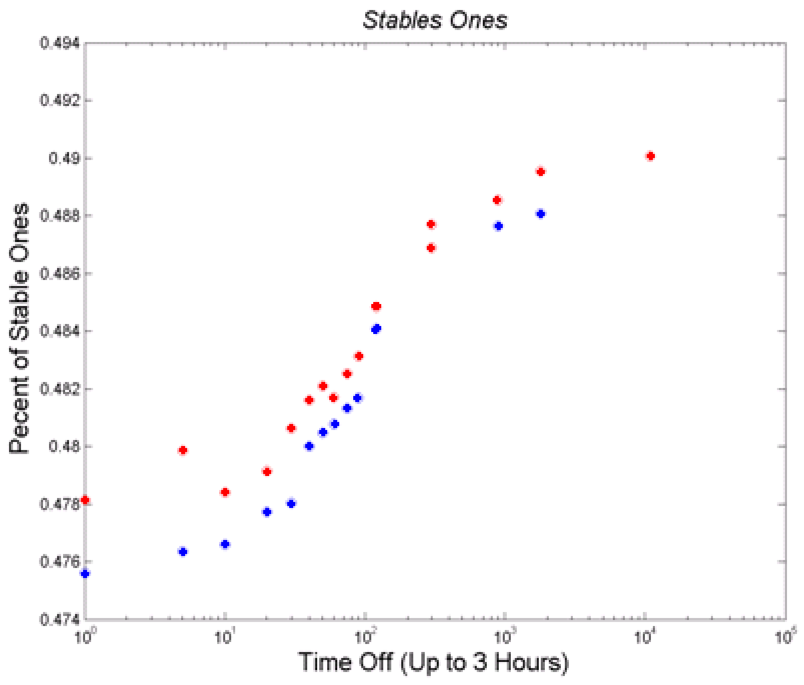


Figure 25: Distribution of Stable Bits

Next we wondered how the unstable bits were distributed. Figure 26 below shows that of those bits that are classified as unstable, about 37% tend to 1 (in ¾ of cases), 37% tend to 0, and 25% are unclassifiable. However, this data would benefit from a greater number of samples.

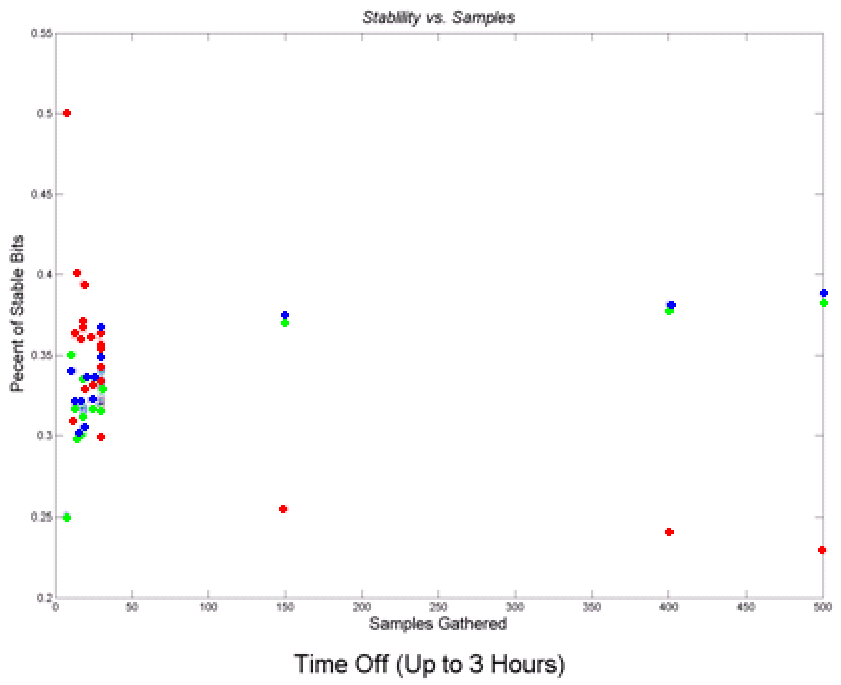


Figure 26: Distribution of Unstable Bits

Finally, Figure 27 shows a histogram of the values of each bit. We see that a large number of bits are completely stable. A small amount has a marked tendency one way or the other, and bits which have less than a 90% tendency toward one side or the other occur with equal low frequency. This is an inverse Gaussian distribution with very steep sides.

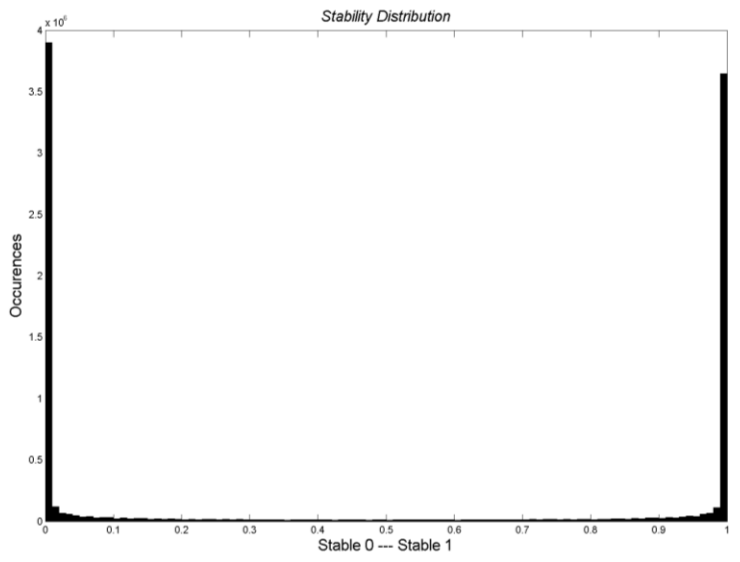


Figure 27: Stability Histogram

Bytewise stability can also be considered. Figure 28 below shows the location in memory of all bytes (red) which were completely stable for all 1,400 or so reads of the first board. Completely stable bytes account for 7.96% of all bytes on the first board.

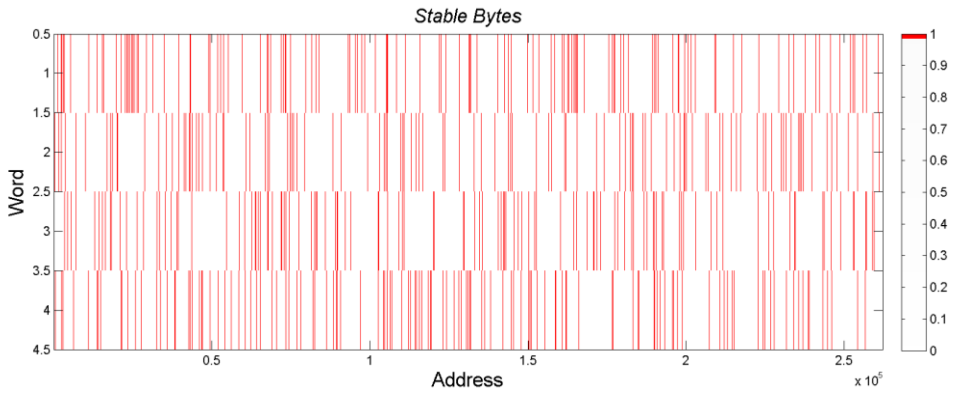


Figure 28: Stable Bytes

Spatial Correlation

We also looked for spatial correlation. We found that there was no linkage between adjacent bits of different addresses. These were found to be completely random. However we did find that the words had some dependencies. Figure 29 below is a histogram of the integer values of each byte over one dataset. While there is a base line, there are also significant peaks in several locations.

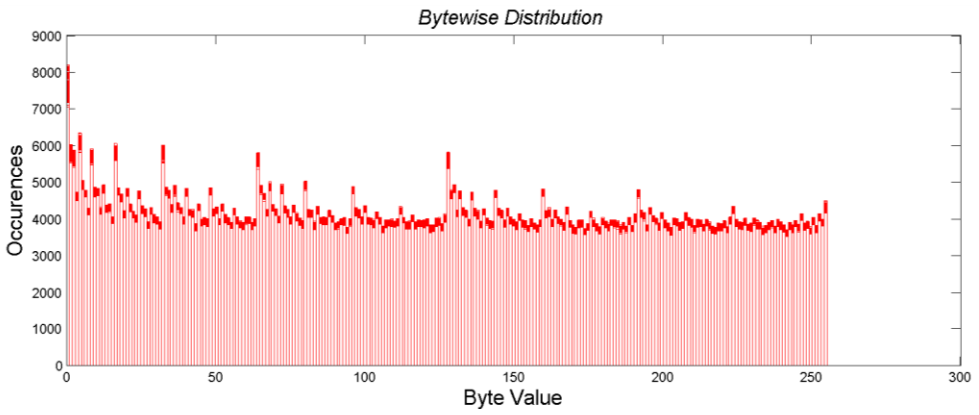


Figure 29: Byte Occurrence

Compare Figure 29 to Figure 30 below. Figure 30 shows how many zeros appear in the binary representation of each 8 bit integer. The peaks in Figure 31 correspond to those values with the most zeros. Consecutive zeros are more likely to appear than consecutive ones. For example in the extreme case, a byte with a value of zero (all zeros) occurs twice as often as a byte with the value of 255 (all ones).

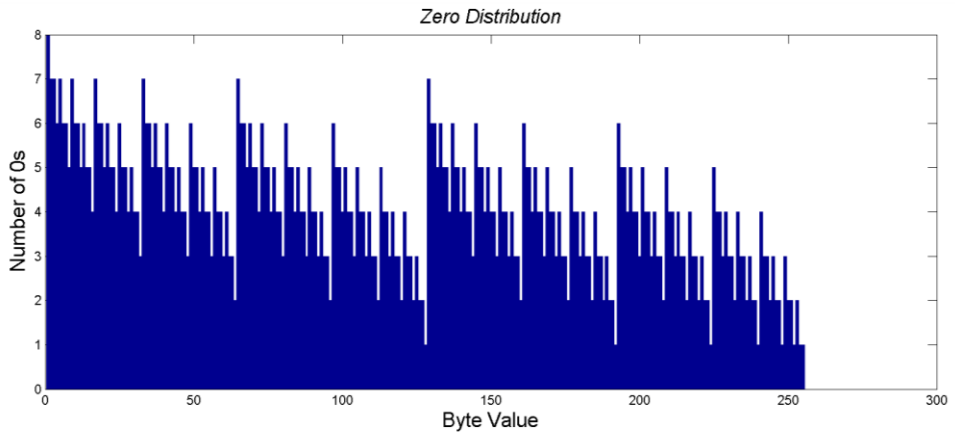


Figure 30: Zeros per byte

Entropy Estimation

Shannon entropy is a measure of the unpredictability of a message *x* taken from the distribution *X*. By taking a large number of samples from the distribution, we can form a probablity distribution function from which we can estimate the Shannon Entropy as follows in Figure 31:

$$\boldsymbol{H(X) = -K \sum_{i=1}^{n} p_i \, log \, p_i}$$

Figure 31: Entropy Equation

In this case, our K was equal to 1. 'N' is 256, since that is the number of values which can be represented with 8 bits. P_i is percent occurrences of each byte value. Since a byte is 8 bits, an entropy value of 8 indicates that the distribution is completely uniform.

Entropy was plotted with respect to time in Figure 32. The entropy follows the same trend as the uniformity graph seen in Figure 20, with values ranging from 7.97 to 7.99.

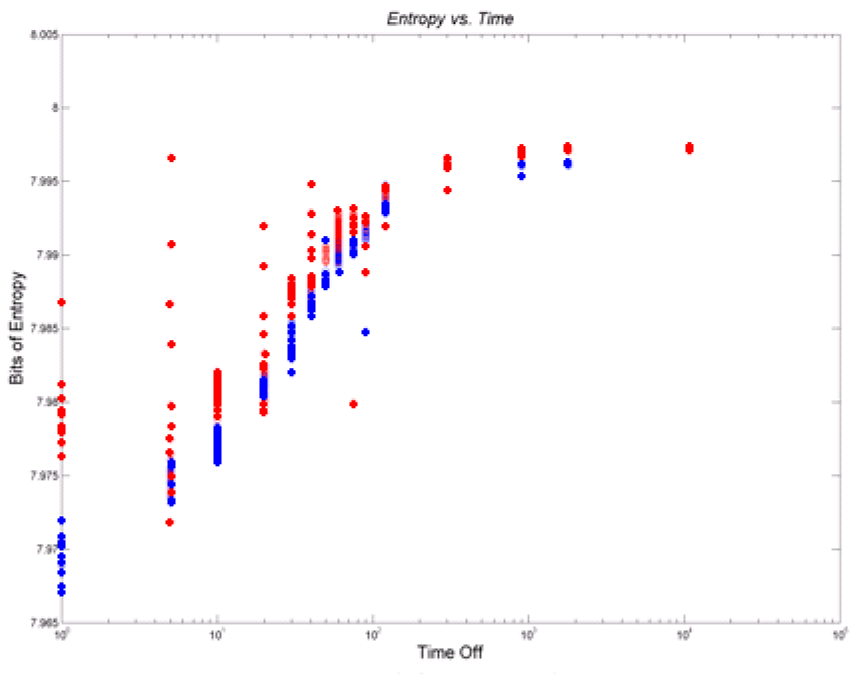


Figure 32: Entropy vs Time Off

Across Boards

Finally, it was necessary to determine whether there was any correlation between the two boards. We used the average reading of each bit over all readings to create a "correct" memory map for each board. We then XORed the two maps to find how many of the corresponding bits were different. We found that 49.42% of bits were different. This suggests that there is no noticable correlation between the two memories. However, two memories is very small sample size and this test should be repeated with more data across many boards.

Figure 33 below shows a portion of the XORed memory map. The red bits are those which are different across the two memories.

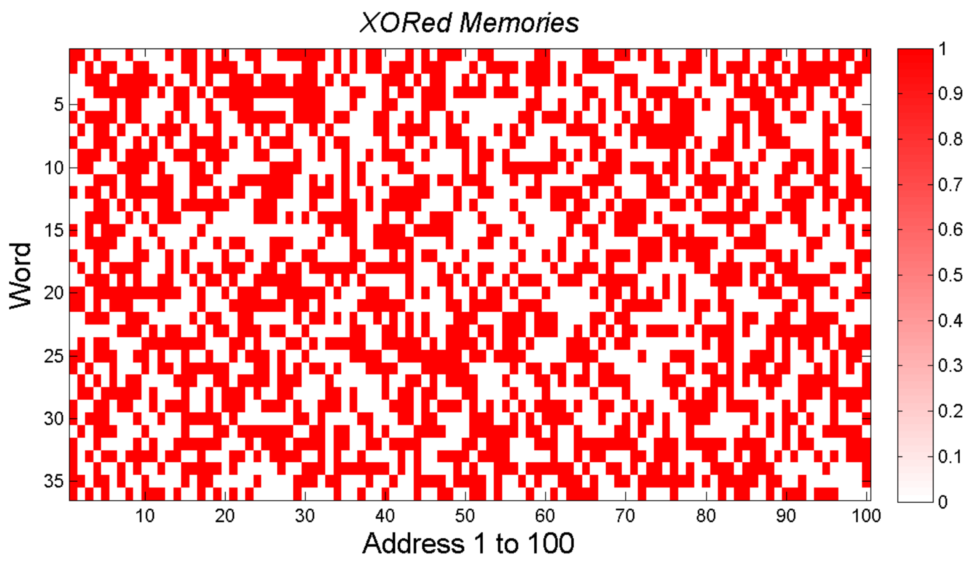


Figure 33: Difference Between Boards 1 and 2

Future Considerations

Due to the noisy nature of PUF responses and the fact that the responses are not uniformly distributed, a fuzzy extractor or helper data algorithm is typically implemented to extract the secure keys from the PUF responses. For instance we observed a noise level of between 10% and 20%. Within this fuzzy extractor, error correction is implemented to compensate for noisy measurements. Privacy amplification is also implemented which guarantees a uniform distribution of the final secret. We were interpreting the results from the SRAM without the use of this block and for future considerations; this fuzzy extractor or helper data algorithm can be implemented to extract a secure key [3].

We attempted to get as many reads from the SRAM as possible. However, when we increased the time duration between reads, we could only obtain a certain number of samples within a restricted time period. We did leave the board on over-night and over the weekend to try to get as many samples. For future considerations, more readings from the boards would be benefit the analysis and we would get more accurate plots.

With this particular FPGA, Virtex-5 LXT ML505, reading from the initial or start-up values of the Block RAM (BRAM) was not possible. Although originally we were attempting to read and perform analysis on the BRAM, we soon realized that in order to read the contents, for this particular chip, we first had to initialize the contents. This initialization process outlined by Xilinx set all of the bits of the BRAM equal to zero. This would destroy the original contents and defeat our purposes of characterizing a chip that can be used to authenticate a particular board based on a key. However for future considerations, with an FPGA chip whose capabilities allow the reading of the BRAM without destroying the original contents, an implementation of this similar project can be done to characterize the BRAM. Similar analysis can be performed to conclude if an encryption key can then be generated from this characterization.

We only had two boards to work with, but for future consideration more boards would benefit the analysis. For example, it can support the fact that a PUF circuit is unique and prove that it can be distinguishable among all boards.

It seems from some of our analysis that temperature has a significant effect. It would be useful to gather quantitative data regarding this effect, perhaps through the use of a temperature chamber or onboard monitoring device.

Conclusion

We successfully implemented a design to read the initial state of SRAM. We gathered and characterized data in hope of showing that SRAM's initial state is a Physical Unclonable Function which may be used to create an intrinsic cryptographic key for identification and IP protection.

All of the goals set forth at the beginning of the project were met. The complete development environment and production flow used to achieve the final PUF are fully documented to allow future designs to implement a similar SRAM characterization and analysis. We also include considerations for future expandability and reusability to assist in any future implementations based on this research and work performed in this project.

The first goal was to implement a read of the entire memory which was initiated as soon as the board powered on. We first started by reading from one address and then setting up a counter to cycle through all of the addresses. We were able to successfully verify that all of the addresses were being read by using the LEDs on the board to represent the hexadecimal equivalent of the address.

Following the completion of reading the entire memory, the next main goal was to import the contents of the memory to the PC for analysis. This was successfully achieved via an RS-232 connection. We were able to view the data from HyperTerminal confirming that bits were being sent and then turned to MATLAB to process the data in batch-form more efficiently. We were successfully able to import the data, store it and place a time-stamp on it for analysis.

The final portion of the project involved implementing a continuous automated reads and imports. This included automating the reset of the board and allowing MATLAB to wait until data was sent from the board. We used MATLAB's build-in pause command and incremented the duration of time between reads to notice any interesting changes to the analysis. We were able to leave the board over-night and over the weekend to continuously read and extract data from the SRAM.

After all of the data was gathered, we used MATLAB's analysis tools to plot our data and interpret the results. These results showed that as time-interval increases so does the number of one's in the data. Also, the amount of time the SRAM is unpowered does not have an effect on the stability of the bits. About 80% of bits are stable, and 50% change between different memory chips of the same type. Using these guidelines, it should be possible to generate a key.

References

[1] ChipDesignMag. http://www.chipdesignmag.com/display.php?articleId=434&issueId=16. Accessed February 20, 2009.

[2] ChipDesignMag. http://www.chipdesignmag.com/display.php?articleId=2899. Accessed March 7, 2009.

[3] Guajardo, Jorge, Kumar Sandeep, Schrijen, Geert-Jan, Tuyls. FPGA Intrinsic PUFs and Their Use for IP Protection. In *Information and System Security Group*, pages 63-80, 2000.

[4] Holcomb, David E., Wayne P. Burleson, and Kevin Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," Proceedings of the Conference on RFID Security, July 2007. http://prisms.cs.umass.edu/~kevinfu/papers/holcomb-FERNS-RFIDSec07.pdf

[5] "IP Power 9258 User Manual," Aviosys International Inc., February, 2002, http://www.jeffcosoho.com/docs/ippower9258.pdf.

[6] "IS61NLP2563A," Integrated Silicon Solution, Inc., May, 2005, http://www.xilinx.com/products/boards/ml505/datasheets/61NLP_NVP25636A_51218A.pdf .

[7] "Introduction to MATLAB," Mathworks, February, 2008. http://www.mathworks.com/moler/intro.pdf.

[8] Rose, Jonathan, Abbas El Gamal, Alberto Sangiovanni-Vincentelli, "Architecture of Field-Programmable Gate Arrays," Proceedings of the IEEE, vol. 81, no. 7, pp. 1013-1029, July 1993.

[9] "Virtex-5 Family Overview," Xilinx, February, 2009, http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

[10] Creative Commons . http://en.wikipedia.org/wiki/File:SRAM_Cell_(6_Transistors).svg. Accessed March 17, 2009.

[11] Xilinx. http://www.xilinx.com/itp/xilinx10/isehelp/isehelp_start.htm. Accessed February 18, 2009.

Appendix A: MATLAB UART Receiver

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%filename: my_serial_simple.m
%
%This code opens the communication with the serial port and reads in %data

clear                                                %Clears all local variables
tic                                                  %Initiate the start of clock
while(1)                                             %Always run

    a = datestr(clock, 'mm_dd__HH_MM_SS'); %defines the format stamp
    fprintf(1,'Starting now. %s\n', a)     %Display on command prompt

    load('file_list');                               %load previously logged
                                                    %files

d = instrfind('Port', 'COM1');              %lists all available ports
    if length(d) ~= 0                       %if the port is not available
        fclose(d);                          %closes those devices that
                                         %are using COM1
        delete(d);                          %deletes those devices
    end
                                                    %setups the serial port

    s = serial('COM1', 'InputBufferSize', 1310720, 'BaudRate', 115200, ...
        'Timeout', 500000, 'ReadAsyncMode', 'continuous');

    fopen(s)                            %opens the serial port connection

    a = datestr(clock, 'mm_dd__HH_MM_SS');  %defines the format stamp
    fprintf(1,'Waiting for input. %s\n', a) %displays the time
    readasync(s)                                      %asynchronous read mode

                                                      %Turn board on
    !d:/on.url
    a = datestr(clock, 'mm_dd__HH_MM_SS__');
    fprintf(1,'Turning ON. %s\n', a)
    pause(1)
    system('taskkill /IM iexplore.exe')


    while s.BytesAvailable == 0                     %waits  within  this  loop  until  %data
                                                    is available to be read
    end

    a = datestr(clock, 'mm_dd__HH_MM_SS__');
    filename = ['data/', a];
    fprintf(1,'Data received. %s\n', a)



    out = fread(s,[5, 262144],'uint8');    %data is read and placed in
                                                     %'out'

    b = toc;                                  %initiates the time stamp
    c = num2str(b, '%04.0f');              %time stamp is converted to
                                                     %a string
    filename = [filename, c];              %filename gets the value of
                                           %the timestamp
    tic                                    %calculates the difference
```

```
                                                       %of time

   save(filename, 'out');                       %saves 'out' as the time
                                             %stamp

                                                      %generates a log of all of
                                                      %the filenames

   all_my_files = [all_my_files; {filename}];
   save('file_list', 'all_my_files');     %logs all of the files
                                                       %generated

   file_count = length(all_my_files);
   fprintf(1,'File count: %1.0f\n', file_count)

                                               %Turn board off
   !d:/off.url
   a = datestr(clock, 'mm_dd__HH_MM_SS__');
   fprintf(1,'Turning OFF. %s\n', a)
   pause(1)
   system('taskkill /IM iexplore.exe')
   fprintf(1,'.\n')
   pause(10);
   fprintf(1,'.\n')
   pause(900)                                  %Wait 900 seconds before next
                                            %read


end
```

Appendix B: MATLAB Main Program (per board)

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: main_prog.m
%
%
%This program opens up the saved data which is stored in folders and %performs all of the
analysis. It saves the figures

clear

tic

mkdir('..\fig');

status = 1
join_files

files = length(new_my_files);
out = zeros(5, 262144);
no_parity = zeros(4, 262144);
all_bits = 36*262144;


status = 2
%file_data
for j = 1:files
    file_name = new_my_files(j);
    b = char(file_name);
    c = b(10:14);
    time(j) = str2num(c);
    c = b(1);
    board(j) = str2num(c);
    c = b(6:8);
    trials(j) = str2num(c);
    c = b(3:4);
    order(j) = str2num(c);
end

clear b c file_name j out

status = 3
all_ent

status = 4
all_uniform

status = 5
all_stable_bits

status = 6
all_stable_bytes

status = 7

folders = max(order);
start = zeros(folders,1);
stop = start;
i = 1;
start(i) = order(i);
for j = 1:files-1
```

```
    a = order(j);
    b = order(j+1);
    if a ~= b
        stop(i) = j;
        i = i + 1;
        start(i) = j+1;
    end
end
stop(folders) = files;

stats = zeros(folders, 17);

status = 8
for CURR_FOLDER = 1:folders
    hold off;
    status = 100+CURR_FOLDER
    some_files = new_my_files(start(CURR_FOLDER):stop(CURR_FOLDER));
    save('folder_files', 'some_files');
    %programs which run on one folder only
    entropy_graph
    uniform_stable_corr
end

save('all_stats', 'stats');

status = 9
for CURR_FOLDER = 1:folders
    some_files = new_my_files(start(CURR_FOLDER):stop(CURR_FOLDER));
    file_name = some_files(1);
    b = char(file_name);
    c = b(10:14);
    time_set(CURR_FOLDER) = str2num(c);
    c = b(1);
    board_set(CURR_FOLDER) = str2num(c);
    c = b(6:8);
    trials_set(CURR_FOLDER) = str2num(c);
    c = b(3:4);
    order_set(CURR_FOLDER) = str2num(c);
end

hold off;

x = time_set;
y = stats(:,3);
save('..\fig\st_ones', 'x', 'y');

x = time_set;
y = stats(:,12);
y2 = stats(:,13);
save('..\fig\row', 'x', 'y', 'y2');

x = trials_set;
y = stats(:, 5);
y2 = stats(:,6);
y3 = stats(:, 7);
save('..\fig\unstable_dist', 'x', 'y', 'y2', 'y3');

x = trials_set;
y = stats(:, 2);
save('..\fig\unstable_tot', 'x', 'y');

toc
```

Appendix C: MATLAB Single Set Analysis

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: uniform_stable_corr.m
%
%This codes generates a graph of total ones, finds stability, does %spatial correlation
%both column-wise and row-wise to see if three %bits are the same

load('folder_files');
files = length(some_files);

total_mem_map = zeros(36, 262144);
total_mem_map_sort = zeros(36, 262144);
total_mem_map_percent = zeros(36, 262144);
mem_map = zeros(40, 262144);
mem_map_short = zeros(36, 262144);
one_byte = zeros(8,1);
out = zeros(5, 262144);
all_bits = 36*262144;
totals_one = zeros(files,1);
totals_zero = zeros(files,1);

for j = 1:files
    file = char(some_files(j));
    load(file);
    for addr = 1:262144
        for byte = 1:5
            choose_byte = (8*(byte-1));
            temp = out(byte,addr);
                if temp >= 128
                    one_byte(1) = 1; temp = temp - 128;
                else one_byte(1) = 0; end
                if temp >= 64
                    one_byte(2) = 1; temp = temp - 64;
                else one_byte(2) = 0; end
                if temp >= 32
                    one_byte(3) = 1; temp = temp - 32;
                else one_byte(3) = 0; end
                if temp >= 16
                    one_byte(4) = 1; temp = temp - 16;
                else one_byte(4) = 0; end
                if temp >= 8
                    one_byte(5) = 1; temp = temp - 8;
                else one_byte(5) = 0; end
                if temp >= 4
                    one_byte(6) = 1; temp = temp - 4;
                else one_byte(6) = 0; end
                if temp >= 2
                    one_byte(7) = 1; temp = temp - 2;
                else one_byte(7) = 0; end
                if temp >= 1
                    one_byte(8) = 1; temp = temp - 1;
                else one_byte(8) = 0; end
            for i = 1:8
                bit_loc = choose_byte + i;
                mem_map(bit_loc, addr) = one_byte(i);
            end
        end
    end
    mem_map_short = mem_map(5:40,:);
    total_mem_map = total_mem_map + mem_map_short;
    col_sum = sum(mem_map_short);
    total_sum = sum(col_sum);
    totals_one(j,1) = total_sum/all_bits;
end

tot_ones = mean(totals_one);
stats(CURR_FOLDER, 1) = tot_ones;

hist(totals_one,files);
name = ['..\fig\ones', num2str(CURR_FOLDER)];
saveas(h, name, 'fig')

%stability
B = reshape(total_mem_map,[],1);
B = B/files;
figure(2); hist(B,50);

total_mem_map_percent = total_mem_map/files;

stable_o = 0;
stable_z = 0;
high = 0;
low = 0;
mid = 0;
for bit = 1:36
    for addr2 = 1:262144
        if total_mem_map_percent(bit, addr2) == 1
            stable_o = stable_o + 1;
            total_mem_map_sort(bit, addr2) = 5;

        elseif total_mem_map_percent(bit, addr2) >=.75
            high = high + 1;
            total_mem_map_sort(bit, addr2) = 4;

        elseif total_mem_map_percent(bit, addr2) == 0
            stable_z = stable_z + 1;
            total_mem_map_sort(bit, addr2) = 1;

        elseif total_mem_map_percent(bit, addr2) <=.25
            low = low + 1;
            total_mem_map_sort(bit, addr2) = 2;

        else
            mid = mid + 1;
            total_mem_map_sort(bit, addr2) = 3;
        end
    end
end



%uniformity
stable_tot = stable_o + stable_z;
st_ones = stable_o/stable_tot;
st_zeros = stable_z/stable_tot;
unstable = (all_bits - stable_tot);



a_stables_tot = stable_tot/all_bits;
a_unstables_tot = unstable/all_bits;
un_highs = high/unstable;
un_lows = low/unstable;
un_mids = mid/unstable;

stats(CURR_FOLDER, 2) = a_stables_tot;

stats(CURR_FOLDER, 3) = st_ones;
stats(CURR_FOLDER, 4) = st_zeros;
stats(CURR_FOLDER, 5) = un_highs;
stats(CURR_FOLDER, 6) = un_lows;
stats(CURR_FOLDER, 7) = un_mids;

%spatial correlation

col = zeros(262144,1);
row = zeros(36,1);
for m=2:35
    for addr = 2:length(mem_map_short)-1
        if     (mem_map_short(m,addr)==0     &&     (mem_map_short(m,addr+1)     ==0     ||
mem_map_short(m,addr-1)==0))
            x1=0;
        elseif    (mem_map_short(m,addr)==1    &&    (mem_map_short(m,addr+1)    ==1    ||
mem_map_short(m,addr-1)==1))
            x1=0;
        else
            x1=1;
        end
        if x1==0
            col(addr)=col(addr)+1;
        end
    end
end
perc_col = col(2:262143)/36;
cmin_col=min(perc_col);
cmax_col=max(perc_col);
cmean_col=mean(perc_col);
cstd_col=std(perc_col);

stats(CURR_FOLDER, 8) = cmin_col;
stats(CURR_FOLDER, 9) = cmax_col;
stats(CURR_FOLDER, 10) = cmean_col;
stats(CURR_FOLDER, 11) = cstd_col;

%row

for m=2:35
    for addr = 2:length(mem_map_short)-1
        if  (mem_map_short(m,addr)==0  &&  (mem_map_short(m+1,addr)==0  ||  mem_map_short(m-
1,addr)==0))
            x1=0;
        elseif    (mem_map_short(m,addr)==1    &&    (mem_map_short(m+1,addr)==1        ||
mem_map_short(m-1,addr)==1))
            x1=0;
        else
            x1=1;
        end
        if x1==0
            row(m)=row(m)+1;
        end
    end
end
perc_row = row(2:35)/262144;
rmin_row=min(perc_row);
rmax_row=max(perc_row);
rmean_row=mean(perc_row);
rstd_row=std(perc_row);

stats(CURR_FOLDER, 12) = rmin_row;
stats(CURR_FOLDER, 13) = rmax_row;
```

Appendix D: MATLAB Board Uniformity Analysis

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: all_uniform.m
%
%
%This code converts the mem_map for all the samples and converts it to %bytes. It then
calculates and generates a plot of how many ones are %present in all of the samples from
one particular board. It saves this %figure.
%
%It also generates a plot of an inverse Gaussian and saves this figure.
%as well

total_mem_map = zeros(36, 262144);
mem_map = zeros(40, 262144);
mem_map_short = zeros(36, 262144);
one_byte = zeros(8,1);
totals_one = zeros(files,1);

for j = 1:files
    file = char(new_my_files(j));
    load(file);
    for addr = 1:262144
        for byte = 1:5
            choose_byte = (8*(byte-1));
            temp = out(byte,addr);
                if temp >= 128
                    one_byte(1) = 1; temp = temp - 128;
                else one_byte(1) = 0; end
                if temp >= 64
                    one_byte(2) = 1; temp = temp - 64;
                else one_byte(2) = 0; end
                if temp >= 32
                    one_byte(3) = 1; temp = temp - 32;
                else one_byte(3) = 0; end
                if temp >= 16
                    one_byte(4) = 1; temp = temp - 16;
                else one_byte(4) = 0; end
                if temp >= 8
                    one_byte(5) = 1; temp = temp - 8;
                else one_byte(5) = 0; end
                if temp >= 4
                    one_byte(6) = 1; temp = temp - 4;
                else one_byte(6) = 0; end
                if temp >= 2
                    one_byte(7) = 1; temp = temp - 2;
                else one_byte(7) = 0; end
                if temp >= 1
                    one_byte(8) = 1; temp = temp - 1;
                else one_byte(8) = 0; end
            for i = 1:8
                bit_loc = choose_byte + i;
                mem_map(bit_loc, addr) = one_byte(i);
            end
        end
    end
    mem_map_short = mem_map(5:40,:);
    total_mem_map = total_mem_map + mem_map_short;

    col_sum = sum(mem_map_short);
    total_sum = sum(col_sum);
    totals_one(j,1) = total_sum/all_bits;
end

save('..\fig\all_ones', 'time', 'totals_one');
save('all_the_data', 'total_mem_map');

B = reshape(total_mem_map,[],1);
B = B/files;
save('..\fig\inverse_gauss', 'B', 'files');

clear B addr bit_loc byte choose_byte col_sum file h i j ...
    mem_map mem_map_short one_byte out temp total_sum totals_one
```

Appendix E: MATLAB Board Entropy Analysis

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: all_ent.m
%
%
%This code performs the entropy of all of the samples from one %particular board and
plots them together on a log graph
%(entropy vs. time)

B = zeros(1048576, 1);
a = zeros(1, 256);
ent = zeros(1, files);
                                      %iterates through all of the time- %intervals from
                                      one board

for j = 1:files
    file = char(new_my_files(j));
    load(file);
    no_parity = out(2:5, 1:262144);
    B = reshape(no_parity,[],1);
    a = hist(B,256);
    ent(j) = 0;
                                      %Performs the entropy
    for i = 1:256
        p(i) = a(i)/sum(a);
        ent(j) = ent(j) + (p(i)*log2(p(i)));
    end
    ent(j) = -ent(j);
end
                                      %Saves the figure

save('..\fig\all_ent', 'time', 'ent');

                                      %Clears variables
clear B a ent file h i j out p
```

Appendix F: MATLAB Single Set Entropy Analysis

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: entropy_graph.m
%
%
%This code generates a histogram for each time-interval that represents %the bits in the
SRAM from 0 to 255

load('folder_files');
files = length(some_files);

out = zeros(5, 262144);
no_parity = zeros(4, 262144);
B = zeros(1048576, 1);
C = zeros(1048576, 1);
a = zeros(1, 256);
                                       %iterates through all of the time-
                                       %intervals from one board

for j = 1:files
    file = char(some_files(j));
    load(file);
    no_parity = out(2:5, 1:262144);
    B = reshape(no_parity,[],1);
                                              %Generates the histogram

    a = hist(B,256);
    figure(1);hist(B,256)

      hold on;

                                              %Performs the entropy

    ent(j) = 0;
    for i = 1:256
        p(i) = a(i)/sum(a);
        ent(j) = ent(j) + (p(i)*log2(p(i)));
    end
    ent(j) = -ent(j);
end

stats(CURR_FOLDER, 14) = min(ent);
stats(CURR_FOLDER, 15) = max(ent);
stats(CURR_FOLDER, 16) = mean(ent);
stats(CURR_FOLDER, 17) = std(ent);

h = findobj(gca,'Type','patch');
set(h,'FaceColor','w','EdgeColor','r')
h = gcf;
                                              %Saves the figure for each
                                              %individual time-interval

name = ['..\fig\ent', num2str(CURR_FOLDER)];
saveas(h, name, 'fig')
hold off;
```

Appendix G: MATLAB Board Stability Analysis (Bitwise)

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: all_stable_bits.m
%
%
%This code goes through all the bit_map and calculates the stables ones %and the stables
zeros in the data. It then plots a sorted map where it %shows the stable ones and the
stable zeros and the distribution in %between with colors

load('colors');
load('colors2');

total_mem_map_percent = zeros(36, 262144);
total_mem_map_sort = zeros(36, 262144);

%stability

total_mem_map_percent = total_mem_map/length(new_my_files);

stable_o = 0;
stable_z = 0;
high = 0;
low = 0;
mid = 0;
for bit = 1:36
    for addr2 = 1:262144
        if total_mem_map_percent(bit, addr2) == 1
            stable_o = stable_o + 1;
            total_mem_map_sort(bit, addr2) = 5;

        elseif total_mem_map_percent(bit, addr2) >=.75
            high = high + 1;
            total_mem_map_sort(bit, addr2) = 4;

        elseif total_mem_map_percent(bit, addr2) == 0
            stable_z = stable_z + 1;
            total_mem_map_sort(bit, addr2) = 1;

        elseif total_mem_map_percent(bit, addr2) <=.25
            low = low + 1;
            total_mem_map_sort(bit, addr2) = 2;

        else
            mid = mid + 1;
            total_mem_map_sort(bit, addr2) = 3;
        end
    end
end

%uniformity
stable_tot = stable_o + stable_z;
st_ones = stable_o/stable_tot;
st_zeros = stable_z/stable_tot;
unstable = (all_bits - stable_tot);

a_stables_tot = stable_tot/all_bits;
a_unstables_tot = unstable/all_bits;
un_highs = high/unstable;
un_lows = low/unstable;
un_mids = mid/unstable;
```

```
total_mem_map_percent2 = total_mem_map_percent(1:36,1:100);
total_mem_map_sort2 = total_mem_map_sort(1:36,1:100);
imagesc(total_mem_map_sort2, [1 5]); colormap(my_color); colorbar;
h = gcf;
saveas(h, '..\fig\sorted_map', 'fig')
imagesc(total_mem_map_percent2, [0 1]); colormap(grayscale); colorbar;
h = gcf;
saveas(h, '..\fig\gradiant_map', 'fig')

clear a_* addr2 bit h high low mid my_color st_* stab* total_mem_map_* un*
```

Appendix H: MATLAB Board Stability Analysis (Bytewise)

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: all_stable_bytes.m
%
%
%This code goes through all the bit_map and calculates the stables and %unstable bytes

stable_b = zeros(4, 262144);

for j = 1:files
    file = char(new_my_files(j));
    load(file);
    no_parity = out(2:5, 1:262144);
    if j == 1
        stable_b = no_parity;
    else
        for addr = 1:262144
            for byte = 1:4
                if ( no_parity(byte, addr) ~= stable_b(byte, addr))
                    stable_b(byte, addr) = 300;
                end
            end
        end
    end
end

stable_list = zeros(1048576,3);

total = 0;
unstables = 0;
m = 1;
for byte = 1:4
    for addr = 1:262144
        total = total + 1;
        if stable_b(byte, addr) == 300
            unstables = unstables + 1;
            stable_b(byte, addr) = 0;
        else
            stable_list(m, 1) = byte;
            stable_list(m, 2) = addr;
            stable_list(m, 3) = stable_b(byte, addr);
            m = m + 1;
            stable_b(byte, addr) = 1;
        end
    end
end

stable = total - unstables;
stable_list = stable_list(1:stable);
save('list_stables', 'stable_list');
hold off;
imagesc(stable_b, [0 1]); colormap(grayscale); colorbar;
h = gcf;
saveas(h, '..\fig\all_stable', 'fig')
perc_stables = (total-unstables)/total;

clear addr ans byte file graysacle h h1 h2 j k m n one_byte...
    ones_array out stable stable_b stable_list...
    temp total total_mem_map unstables xout
```

Appendix I: MATLAB Board Uniformity Analysis

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: all_uniform.m
%
%
%This program generates a plot of total ones vs time

total_mem_map = zeros(36, 262144);
mem_map = zeros(40, 262144);
mem_map_short = zeros(36, 262144);
one_byte = zeros(8,1);
totals_one = zeros(files,1);

for j = 1:files
    file = char(new_my_files(j));
    load(file);
    for addr = 1:262144
        for byte = 1:5
            choose_byte = (8*(byte-1));
            temp = out(byte,addr);
                if temp >= 128
                    one_byte(1) = 1; temp = temp - 128;
                else one_byte(1) = 0; end
                if temp >= 64
                    one_byte(2) = 1; temp = temp - 64;
                else one_byte(2) = 0; end
                if temp >= 32
                    one_byte(3) = 1; temp = temp - 32;
                else one_byte(3) = 0; end
                if temp >= 16
                    one_byte(4) = 1; temp = temp - 16;
                else one_byte(4) = 0; end
                if temp >= 8
                    one_byte(5) = 1; temp = temp - 8;
                else one_byte(5) = 0; end
                if temp >= 4
                    one_byte(6) = 1; temp = temp - 4;
                else one_byte(6) = 0; end
                if temp >= 2
                    one_byte(7) = 1; temp = temp - 2;
                else one_byte(7) = 0; end
                if temp >= 1
                    one_byte(8) = 1; temp = temp - 1;
                else one_byte(8) = 0; end
            for i = 1:8
                bit_loc = choose_byte + i;
                mem_map(bit_loc, addr) = one_byte(i);
            end
        end
    end
    mem_map_short = mem_map(5:40,:);
    total_mem_map = total_mem_map + mem_map_short;

    col_sum = sum(mem_map_short);
    total_sum = sum(col_sum);
    totals_one(j,1) = total_sum/all_bits;
end

save('..\fig\all_ones', 'time', 'totals_one');
save('all_the_data', 'total_mem_map');
```

```
B = reshape(total_mem_map,[],1);
B = B/files;
save('..\fig\inverse_gauss', 'B', 'files');

clear B addr bit_loc byte choose_byte col_sum file h i j ...
    mem_map mem_map_short one_byte out temp total_sum totals_one
```

Appendix J: MATLAB Join Board Samples

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: join_files.m
%
%
%This program joins the files so that analysis can be performed for %each time interval


new_my_files = {};

folder_list = dir;
a = ls;
a = cellstr(a);
c = length(a)-1;
b = a(3:length(a)-1);
c = length(b);

for i = 1:c
    if folder_list(i+2).isdir == 1
        folder = char(b(i));
        str_file_list = [char(b(i)), '\file_list'];
        load(str_file_list);
        files = length(all_my_files);


        for j = 1:files
            a = char(all_my_files(j));
            new_name = [folder, '/', a];
            new_my_files = [new_my_files; {new_name}];
        end
    end
end
save('all_files', 'new_my_files');

clear a b c all_my_files files folder folder_list i j new_name str_file_list
```

Appendix K: MATLAB Merge Data Sets

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%file name: merge.m
%
%
%This program both boards to be plotted on the same graph

clear
mkdir('fig');

a = '.\board1\fig\';
b = '.\board2\fig\';

c(1) = {'all_ent.mat'};
c(2) = {'all_ones.mat'};
c(3) = {'inverse_gauss.mat'};
c(4) = {'st_ones.mat'};
c(5) = {'unstable_dist.mat'};
c(6) = {'unstable_tot.mat'};
c(7) = {'row.mat'};

one = c;
two = c;

for j = 1:length(c);
    m = char(c(j));
    p = [a, m];
    n = cellstr(p);
    one(j) = n;
    p = [b, m];
    n = cellstr(p);
    two(j) = n;
end

j = 1;
clear a b;
a = char(one(j));
b = char(two(j));
load(a);
semilogx(time, ent, 'or')
hold on;
load(b);
semilogx(time, ent, 'ob')
hold off;
h = gcf;
saveas(h, '.\fig\all_ent', 'fig')

j = 2;
clear a b;
a = char(one(j));
b = char(two(j));
load(a);
semilogx(time, totals_one, 'or')
hold on;
load(b);
semilogx(time, totals_one, 'ob')
hold off;
h = gcf;
saveas(h, '.\fig\totals_one', 'fig')
```

```
j = 3;
clear a b;
a = char(one(j));
b = char(two(j));
load(a);
hist(B, 100)
hold on;
load(b);
hist(B, 100)
hold off;
h = gcf;
saveas(h, '.\fig\inverse_gauss', 'fig')

j = 4;
clear a b;
a = char(one(j));
b = char(two(j));
load(a);
semilogx(x, y, 'or')
time_set1 = x;
hold on;
load(b);
semilogx(x, y, 'ob')
time_set2 = x;
hold off;
h = gcf;
saveas(h, '.\fig\st_ones', 'fig')

j = 5;
clear a b;
a = char(one(j));
b = char(two(j));
load(a);
plot(x, y, 'og', x, y2, 'ob', x, y3, 'or')
hold on;
load(b);
plot(x, y, 'squareb', x, y2, 'squareb', x, y3, 'squareb')
hold off;
h = gcf;
saveas(h, '.\fig\unstable_dist', 'fig')

j = 6;
clear a b;
a = char(one(j));
b = char(two(j));
load(a);
semilogx(time_set1, y, 'or')
hold on;
load(b);
semilogx(time_set2, y, 'squareb')
hold off;
h = gcf;
saveas(h, '.\fig\unstable_tot1', 'fig')

j = 6;
clear a b;
a = char(one(j));
b = char(two(j));
load(a);
plot(x, y, 'or')
hold on;
load(b);
plot(x, y, 'squareb')
hold off;
saveas(h, '.\fig\unstable_tot2', 'fig')

j = 7;
clear a b;
a = char(one(j));
```

```
b = char(two(j));
load(a);
semilogx(x, y, 'or', x, y2, 'or')
hold on;
load(b);
semilogx(x, y, 'squareb', x, y2, 'squareb')
hold off;
saveas(h, '.\fig\row', 'fig')
```

Appendix L: Comparing Memories

```
% Worcester Polytechnic Institute
% General Dynamics C4 Systems MQP
% Robyn Colopy
% Jatin Chopra
% C Term 2009
% SRAM PUF
%
%This program creates two “average” maps and compares them

load('.\board1\old_data\all_the_data')
load('.\board1\old_data\file_num')
b_one = total_mem_map;
f_one = files;

load('.\board2\old_data\all_the_data')
load('.\board2\old_data\file_num')
b_two = total_mem_map;
f_two = files;

clear total_mem_map

b_one_p = b_one/f_one;
b_two_p = b_two/f_two;

a = round(b_one_p);
b = round(b_two_p);

c = xor(a,b);

d = sum(c);
e = sum(d);

total_bits = 36*262144;

perc_dif = e/total_bits;

c = c(1:36,1:100);
imagesc(c, [0 1]); colorbar;
h = gcf;
xlabel('Address 1 to 100','FontSize',16)
ylabel('Word','FontSize',16)
title('\it{XORed Memories}','FontSize',16)
set(gcf,'PaperPositionMode','auto')

print -dpng C:\MQP_final\jpg\changes
```

Appendix M: VHDL SRAM & UART

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-- Worcester Polytechnic Institute
-- General Dynamics C4 Systems MQP
-- Robyn Colopy
-- Jatin Chopra
-- C Term 2009
-- SRAM PUF
-- Filename: sram_top.vhd
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--This file is designed to read from the SRAM
-- It cycles through all of the address, and sends the contents
--of the SRAM through the UART port
--The main clock is converted down to the clock frequency
--of the UART which is 115200 Hz
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sram_top is
Port (
                      GPIO_LED : out std_logic_vector(7 downto 0);
                      GPIO_LED_C : out std_logic;
                      GPIO_LED_E : out std_logic;
                      GPIO_LED_N : out std_logic;
                      GPIO_LED_S : out std_logic;
                      GPIO_LED_W : out std_logic;
                      GPIO_SW_N : in std_logic;
                      USER_CLK : in std_logic;
                      SRAM_A : out std_logic_vector(21 downto 0);
                      SRAM_D : inout std_logic_vector(31 downto 0);
                      SRAM_DQP : inout std_logic_vector(3 downto 0);
                      SRAM_BW : out std_logic_vector(3 downto 0);
                      SRAM_MODE : out std_logic;
                      SRAM_OE_B : out std_logic;
                      SRAM_WE_B : out std_logic;
                      SRAM_ADV_LD_B : out std_logic;
                      SRAM_CLK : out std_logic;
                      SRAM_CS_B : out std_logic;
                      uart_out : out std_logic
       );
end sram_top;

architecture Behavioral of sram_top is

       type STATE_TYPE is (S0, S1, S2);
       type UART_STATE_TYPE is (U0, U1, U2, U3, U4, U5, U6, U7, U8, U9, U10, U11, U12);
       signal UCURR_STATE : UART_STATE_TYPE;
       signal UNEXT_STATE : UART_STATE_TYPE;
       signal CURR_STATE : STATE_TYPE;
       signal NEXT_STATE : STATE_TYPE;
       signal addr : std_logic_vector(21 downto 0);
       signal data : std_logic_vector(31 downto 0);
       signal buff : std_logic_vector(39 downto 0);
       signal data_p : std_logic_vector(3 downto 0);
       signal bwx : std_logic_vector(3 downto 0);
       signal oe : std_logic;
       signal we : std_logic;
       signal adv : std_logic;
       signal ce : std_logic;
```

```
        signal count_addr : std_logic_vector(19 downto 1);
        signal UART_CLK : std_logic;
        signal uart_data : std_logic;
        signal send_five : integer range 0 to 4;
        signal choose_bit : integer range 0 to 7;
        signal buff_index : integer range 0 to 39;
        signal inc_sends, rst_sends, inc_addr, RESET, inc_buff : std_logic;

begin

uart_out <= uart_data;
SRAM_CLK <= USER_CLK;

SRAM_D <= data;
SRAM_DQP <= data_p;
SRAM_BW <= bwx;
SRAM_MODE <= '0';
SRAM_OE_B <= oe;
SRAM_WE_B <= we;
SRAM_ADV_LD_B <= adv;
SRAM_CS_B <= ce;

SRAM_A <= addr;

bwx <= X"0";
adv <= '0';

RESET <= GPIO_SW_N;
--high impedance
data <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
data_p <= "ZZZZ";
we <= '1';
addr <= "00" & count_addr & '0';

counter: process(USER_CLK)
variable COUNTER : integer range 0 to  434;
begin
        if rising_edge(USER_CLK) then
                COUNTER := COUNTER + 1;
                if COUNTER =  434 then
                        UART_CLK <= not UART_CLK;
                        COUNTER := 0;
                end if;
        end if;
end process counter;


buff(39 downto 36) <= "1000";
state_memory: process(USER_CLK)
begin

        if rising_edge(USER_CLK) then
                if RESET = '1' then
                        CURR_STATE <= S0;
                else
                        if inc_buff = '1' then buff(35 downto 0) <= data_p & data;
                        else buff(35 downto 0) <= buff(35 downto 0);
                        end if;
                        CURR_STATE <= NEXT_STATE;
                end if;
        end if;
end process state_memory;

next_state_logic: process(CURR_STATE)
begin
        case CURR_STATE is
                when S0 =>  NEXT_STATE <= S1;         ---load address for read
                when S1 =>  NEXT_STATE <= S2;         ---data available
                when S2 =>  NEXT_STATE <= S0;         ---show state
        end case;
end process next_state_logic;

        with CURR_STATE select
                oe <=  '1' when S0,
                                        '1' when S1,
                                        '0' when S2,
                                        '1' when others;

        with CURR_STATE select
                ce <=  '0' when S0,
                                        '1' when S1,
                                        '1' when S2,
                                        '1' when others;

        with CURR_STATE select
                inc_buff <= '1' when S1,
                                        '0' when others;

        GPIO_LED <=  count_addr(19 downto 12);

uart_state_memory: process(UART_CLK, inc_sends, rst_sends)
begin

        if rising_edge(UART_CLK) then
                if RESET = '1' then
                        send_five <= 0;
                        count_addr <= "00000000000000000000";
                        UCURR_STATE <= U0;
                else
                        if inc_sends = '1' then
                                send_five <= send_five + 1;
                        elsif rst_sends = '1' then
                                send_five <= 0;
                        else send_five <= send_five;
                        end if;

                        if inc_addr = '1' then
                                count_addr <= count_addr + '1';
                        else count_addr <= count_addr;
                        end if;

                        UCURR_STATE <= UNEXT_STATE;
                end if;
        end if;

end process uart_state_memory;


uart_next_state_logic: process(UCURR_STATE, send_five, count_addr)
begin
        case UCURR_STATE is
                when U0 =>                   --idle = 1   --- initializing state
                                UNEXT_STATE <= U1;
                when U1 =>  UNEXT_STATE <= U2;               --start = 0
                when U2 =>  UNEXT_STATE <= U3;               --lsb
                when U3 =>  UNEXT_STATE <= U4;
                when U4 =>  UNEXT_STATE <= U5;
                when U5 =>  UNEXT_STATE <= U6;
                when U6 =>  UNEXT_STATE <= U7;
                when U7 =>  UNEXT_STATE <= U8;
                when U8 =>  UNEXT_STATE <= U9;
                when U9 =>  UNEXT_STATE <= U10;              --msb
                when U10 =>                                  --stop = 1
                        if send_five = 4 then UNEXT_STATE <= U11;
                                              else UNEXT_STATE <= U1;
                                              end if;
                when U11 =>                                  --idle = 1
```

```
               if count_addr = "1000000000000000000" then  UNEXT_STATE <= U12;
                                              else UNEXT_STATE <= U1;
                                              end if;

               when U12 => UNEXT_STATE <= UCURR_STATE;
        end case;
end process uart_next_state_logic;

        with UCURR_STATE select
               inc_addr  <=   '1' when U11,
                                                     '0' when others;

        with UCURR_STATE select
               inc_sends <=   '1' when U10,
                                                     '0' when others;

        with UCURR_STATE select
               rst_sends <=   '1' when U0,
                                                     '1' when U11,
                                                     '0' when others;

        with UCURR_STATE select
               choose_bit <=  0 when U2,
                                                     1 when U3,
                                                     2 when U4,
                                                     3 when U5,
                                                     4 when U6,
                                                     5 when U7,
                                                     6 when U8,
                                                     7 when U9,
                                                     0 when others;

        with send_five select  --- send MSBs first
               buff_index <=  32 + choose_bit when 0,
                                                     24 + choose_bit when 1,
                                                     16 + choose_bit when 2,
                                                      8 + choose_bit when 3,
                                                      0 + choose_bit when 4,
                                                      0 when others;

        with UCURR_STATE select
               uart_data <=   '1' when U0,
                                                     '0' when U1,
                                                     '1' when U10,
                                                     '1' when U11,
                                                     '1' when U12,
                                                     buff(buff_index) when others;

        with UCURR_STATE select
               GPIO_LED_S <=  '1' when U12,
                                                     '0' when others;

        with UCURR_STATE select
               GPIO_LED_E <=  '1' when U12,
                                                     '0' when others;

        with UCURR_STATE select
               GPIO_LED_W <=  '1' when U12,
                                                     '0' when others;

        with UCURR_STATE select
               GPIO_LED_C <=  '1' when U12,
                                                     '0' when others;

        with UCURR_STATE select
               GPIO_LED_N <=  '1' when U12,
                                      '0' when others;

end Behavioral;
```

Appendix N: UCF File

```
# Worcester Polytechnic Institute
# General Dynamics C4 Systems MQP
# Robyn Colopy
# Jatin Chopra
# C Term 2009
# SRAM PUF
# Filename: sram_top.ucf

NET  GPIO_LED(0)           LOC="H18";   # Bank 3, Vcco=2.5V, No DCI
NET  GPIO_LED(1)           LOC="L18";   # Bank 3, Vcco=2.5V, No DCI
NET  GPIO_LED(2)           LOC="G15";   # Bank 3, Vcco=2.5V, No DCI
NET  GPIO_LED(3)          LOC="AD26";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET  GPIO_LED(4)          LOC="G16";   # Bank 3, Vcco=2.5V, No DCI
NET  GPIO_LED(5)          LOC="AD25";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET  GPIO_LED(6)          LOC="AD24";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET  GPIO_LED(7)          LOC="AE24";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET  GPIO_LED_C           LOC="E8";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  GPIO_LED_E           LOC="AG23";  # Bank 2, Vcco=3.3V
NET  GPIO_LED_N           LOC="AF13";  # Bank 2, Vcco=3.3V
NET  GPIO_LED_S           LOC="AG12";  # Bank 2, Vcco=3.3V
NET  GPIO_LED_W           LOC="AF23";  # Bank 2, Vcco=3.3V
NET  GPIO_SW_N            LOC="U8";    # Bank 18, Vcco=3.3V, No DCI
NET  USER_CLK             LOC="AH15";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_A(0)        LOC="K12";   # Bank 1, Vcco=3.3V
NET  SRAM_A(1)        LOC="K13";   # Bank 1, Vcco=3.3V
NET  SRAM_A(2)        LOC="H23";   # Bank 1, Vcco=3.3V
NET  SRAM_A(3)        LOC="G23";   # Bank 1, Vcco=3.3V
NET  SRAM_A(4)        LOC="H12";   # Bank 1, Vcco=3.3V
NET  SRAM_A(5)        LOC="J12";   # Bank 1, Vcco=3.3V
NET  SRAM_A(6)        LOC="K22";   # Bank 1, Vcco=3.3V
NET  SRAM_A(7)        LOC="K23";   # Bank 1, Vcco=3.3V
NET  SRAM_A(8)        LOC="K14";   # Bank 1, Vcco=3.3V
NET  SRAM_A(9)        LOC="L14";   # Bank 1, Vcco=3.3V
NET  SRAM_A(10)        LOC="H22";   # Bank 1, Vcco=3.3V
NET  SRAM_A(11)       LOC="G22";   # Bank 1, Vcco=3.3V
NET  SRAM_A(12)       LOC="J15";   # Bank 1, Vcco=3.3V
NET  SRAM_A(13)       LOC="K16";   # Bank 1, Vcco=3.3V
NET  SRAM_A(14)       LOC="K21";   # Bank 1, Vcco=3.3V
NET  SRAM_A(15)       LOC="J22";   # Bank 1, Vcco=3.3V
NET  SRAM_A(16)       LOC="L16";   # Bank 1, Vcco=3.3V
NET  SRAM_A(17)       LOC="L15";   # Bank 1, Vcco=3.3V
NET  SRAM_A(18)       LOC="L20";   # Bank 1, Vcco=3.3V
NET  SRAM_A(19)       LOC="L21";   # Bank 1, Vcco=3.3V
NET  SRAM_A(20)       LOC="AE23";  # Bank 2, Vcco=3.3V
NET  SRAM_A(21)       LOC="AE22";  # Bank 2, Vcco=3.3V
NET  SRAM_D(0)        LOC="AD19";  # Bank 2, Vcco=3.3V
NET  SRAM_D(1)        LOC="AE19";  # Bank 2, Vcco=3.3V
NET  SRAM_D(2)        LOC="AE17";  # Bank 2, Vcco=3.3V
NET  SRAM_D(3)        LOC="AF16";  # Bank 2, Vcco=3.3V
NET  SRAM_D(4)        LOC="AD20";  # Bank 2, Vcco=3.3V
NET  SRAM_D(5)        LOC="AE21";  # Bank 2, Vcco=3.3V
NET  SRAM_D(6)        LOC="AE16";  # Bank 2, Vcco=3.3V
NET  SRAM_D(7)        LOC="AF15";  # Bank 2, Vcco=3.3V
NET  SRAM_D(8)        LOC="AH13";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(9)        LOC="AH14";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(10)       LOC="AH19";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(11)       LOC="AH20";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(12)       LOC="AG13";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(13)       LOC="AH12";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(14)       LOC="AH22";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(15)       LOC="AG22";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_D(16)       LOC="N10";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(17)       LOC="E13";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(18)       LOC="E12";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(19)       LOC="L9";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(20)       LOC="M10";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(21)       LOC="E11";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(22)       LOC="F11";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(23)       LOC="L8";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(24)       LOC="M8";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(25)       LOC="G12";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(26)       LOC="G11";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(27)       LOC="C13";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(28)       LOC="B13";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(29)       LOC="K9";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(30)       LOC="K8";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_D(31)       LOC="J9";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_DQP(0)      LOC="D12";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_DQP(1)      LOC="C12";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_DQP(2)      LOC="H10";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_DQP(3)      LOC="H9";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_MODE        LOC="A13";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_OE_B        LOC="B12";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_WE_B           LOC="AF20";  # Bank 2, Vcco=3.3V
NET  SRAM_ADV_LD_B    LOC="H8";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_BW(0)         LOC="D10";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_BW(1)         LOC="D11";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_BW(2)         LOC="J11";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_BW(3)         LOC="K11";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET  SRAM_CLK          LOC="AG21";  # Bank 4, Vcco=3.3V, No DCI
NET  SRAM_CS_B        LOC="J10";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET uart_out LOC="AG20";
```