ECE532 - Final Design Report

Spam Detection Using Parallel Hash Table

James Jiang

Winston Sun

Youngjo Kim

Date: Apr 10, 2022

1.0 Overview **3**
1.1. Background and Motivation 3
1.2. Goals 3
1.3. System Block Diagram 4
1.3.1 System Overview 4
1.3.2 Server FPGA 5
1.3.3 Client FPGA 6
1.4. Brief Description of IP 7
1.4.1 Server System Components 7
1.4.2 Client System Components 9

2.0 Outcome **11**
2.1. Results 11
2.2. Future Work 13

3.0 Project Schedule **14**

4.0 Description of Custom IP **15**
4.1 Hashing Algorithm IP 15
4.1.1 Overview 15
4.1.2 Interface Signals 17
4.1.3 Register Map 18
4.1.4 Verification 19
4.2 Hash Table Processing & Synchronization IP 19
4.2.1 Overview 19
4.2.2 Interface Signals 20
4.2.3 Register Map 22
4.2.3 Verification 23
4.3 SD Card Controller IP 23
4.3.1 Block Overview 23
4.3.2 Interface Signals 24
4.3.3 Parameters 25
4.3.4 Bare Metal Code 26
4.4 Spam Detection Inference IP 26
4.4.1 Block Overview 26
4.4.2 Interface Signals 27
4.4.3 Register Map 28
4.4.4 Bare Metal Code 29

5.0 Design Tree **31**

Tips and Tricks **32**

Reference **33**

Appendix A: Project Results **34**

1.0 Overview

1.1. Background and Motivation

Traditionally, email filtering is running in the SaaS application software to constantly detect spam emails and block malicious emails. However, in an enterprise setting, this may not be the most efficient way since spam emails occupy users' limited disk space and processing power of computers, tablets, and smartphones [1]. Moreover, the "bring your own device" revolution where employees can work using their personal devices increases the complexity of email filtering [1]. Email filtering in software has significant overhead and requires a system administrator to constantly reconfigure and maintain the software [1]. To improve the efficiency of email filtering, the team designed a hardware-based spam detection system on FPGAs to showcase the advantages of the hardware accelerator in terms of distributed processing and performance. The implementation of a hardware-based spam detection system requires two major components: the pre-trained spam detection probabilistic model and the inference algorithm. Hash table is used to design and store the pre-trained model on an FPGA since it is one of the most efficient data structures, being able to read, insert, and delete entries in average constant (O(1)) time [2]. The appeal of parallelizing hash tables is twofold: first, multiple clients can make parallel access to the pre-trained model stored in the hash table to improve the throughputs and performance; second, a parallel hash table ensures data access is synchronized so that all clients will receive the most up-to-date probabilistic model parameters to be used for the spam detection inference algorithm.

1.2. Goals

The overall goal of this project is to create a spam filter that is able to run independently and concurrently on multiple clients. To accomplish this, the following goals must be met:

- Create a custom hashing algorithm IP (Section 4.1) for the client
- Create a custom parallel hash table IP (Section 4.2) for the server
- Create a custom spam detection inference IP (Section 4.4) for the client
- Enable client-server network communication

Additionally, the following goals serve to improve usability of the design:

- Enable hash table initialization using an SD card (Section 4.3)
- Create a GUI for client, server, and network performance monitoring (Appendix A)

The client accepts an email from the user and parses it into individual words. Each word is hashed and sent over the network to the server, where a hash table lookup is performed. The hash table contains a mapping of words to a set of pretrained probabilities that the word is associated with spam or ham. The server sends these probabilities back to the client, which uses these values to calculate the overall probability that the email is spam. Once all the words have been processed, the client reports whether or not the email is spam to the user.

1.3. System Block Diagram

1.3.1 System Overview

Figure 1 shows the integrated system over the FPGANet. The system uses one Server FPGA and two Client FPGAs for parallel access. The Server FPGA contains the pre-trained probabilistic model in the hash table and is responsible for different operations of the hash table. The Client FPGAs first receive emails to be filtered and run the hashing algorithm to generate access keys to fetch probabilistic data from the Server FPGA. After receiving back the data from the Server FPGA, the Client FPGA will process the data and run the spam detection inference algorithm. All FPGAs have been connected over the network to a computer for performance monitoring and analysis.

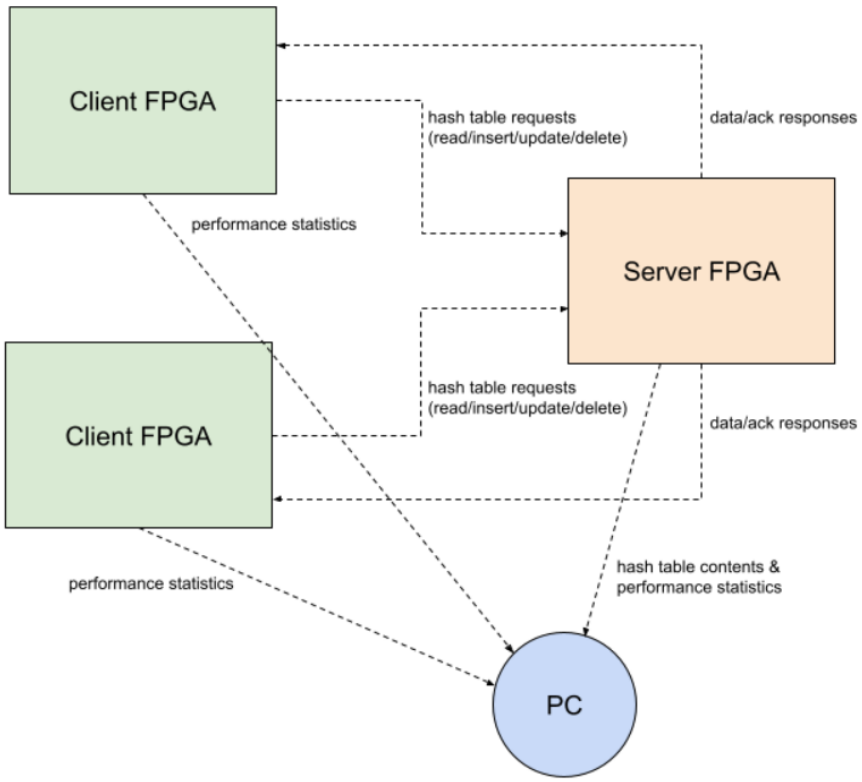


Figure 1. System overview

1.3.2 Server FPGA

The Server FPGA stores the pre-trained spam detection model for multiple clients to access in parallel. The Hash Table Processing & Synchronization IP takes commands (query, insert, delete, and update) from Client FPGAs, executes commands in the DDR3 memory, and returns the result to Client FPGAs through the Ethernet block (TCP protocol). The SD Card Controller IP loads the pre-trained spam detection model binary file from the SD card to the DDR memory (direct access, bypassing the Microblaze) during FPGA startup.

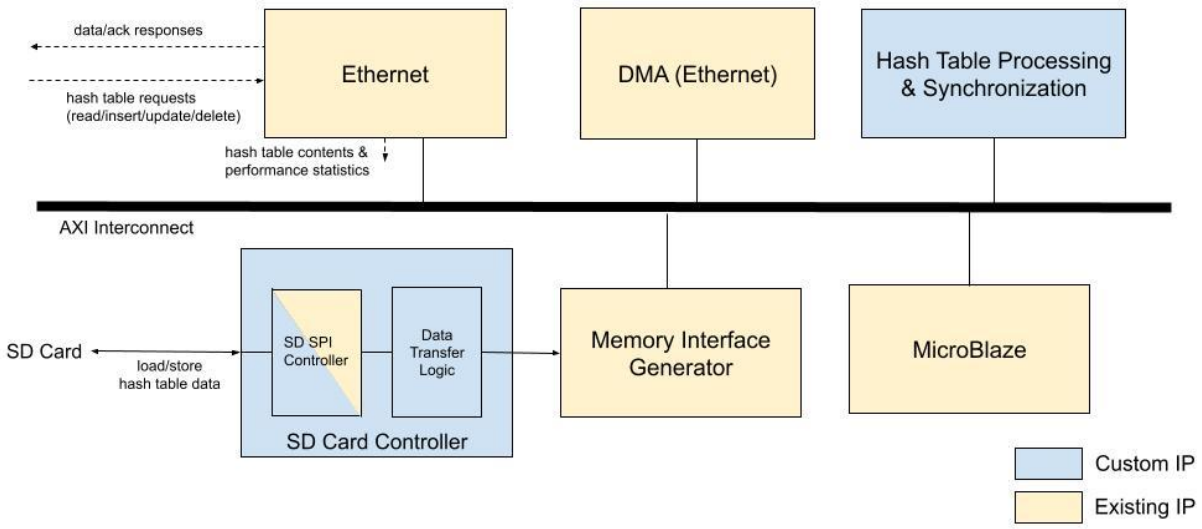


Figure 2. Server FPGA block diagram

1.3.3 Client FPGA

The Client FPGA is responsible for the hashing algorithm and spam detection inference. The Hashing Algorithm IP generates hash keys for each word in the email and generates a list of hash keys to be sent to the Server FPGA through the EthernetLite IP (TCP protocol). After receiving the requested probabilistic data from the Server FPGA, the Client FPGA uses the Spam Detection Inference IP to store the data in a FIFO and run spam detection computation to check if an email is spam or non-spam.

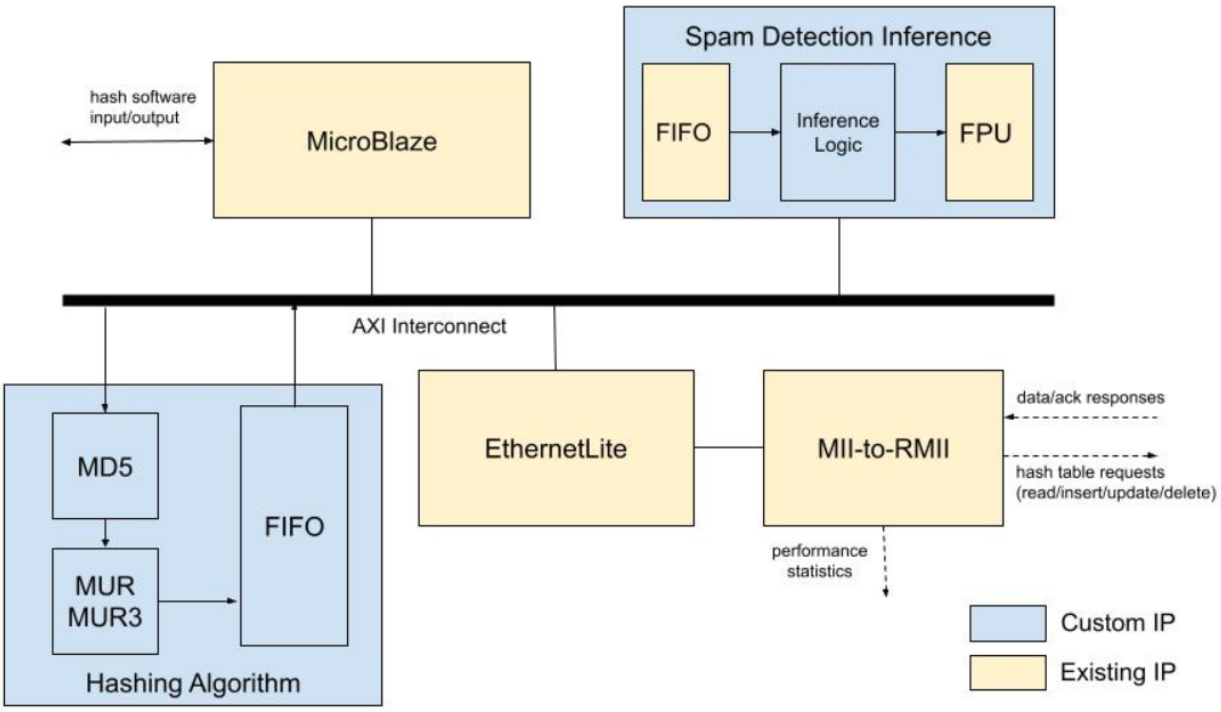


Figure 3. Client FPGA block diagram

1.4. Brief Description of IP

1.4.1 Server System Components

The Server system is implemented on the Nexys4 Video board. The table lists all the IP components used for the Server FPGA system.

Table 1: Server system IP components and their respective descriptions

IP	Function/Description	Origin
Microblaze (11.0)	A soft microprocessor core designed for Xilinx FPGA and used to run software code through Xilinx SDK.	Xilinx
Processor System Reset (5.0)	A block that handles the reset conditions for a given system and generates appropriate resets	Xilinx

	at the output.	
AXI Interrupt Controller (4.1)	A block that handles interrupt signals from other IPs	Xilinx
AXI TIMER (2.0)	A timer module with AXI interface.	Xilinx
Clocking Wizard (6.0)	A block that generates different clock frequencies from the input clock.	Xilinx
Microblaze_local_memory_ block - BRAM	BRAM memory block. Used as cache memory for MicroBlaze.	Xilinx
Memory Interface Generator (MIG 7 series) (4.2)	A block that provides an interface to access the DDR3 memory.	Xilinx
Microblaze Debug Module (MDM) (3.2)	A core that enables JTAG-based debugging of the MicroBlaze microprocessor.	Xilinx
AXI Interconnect (2.1)	Bus interface of the processor and the peripherals. Connects AXI memory-mapped masters with AXI memory-mapped slaves.	Xilinx
AXI SmartConnect (1.0)	A more compact and simpler bus interface that connects AXI memory-mapped masters with AXI memory-mapped slaves.	Xilinx
AXI GPIO (2.0)	General-purpose input/output interface to the AXI interface. The pins are controllable by users at run time through the AXI interface.	Xilinx
AXI 1G/2.5G Ethernet Subsystem (7.1)	Ethernet block that transmits and receives packets through a network connection (1G and 2.5G).	Xilinx

AXI Direct Memory Access (7.1)	A block that provides high bandwidth data transfer between AXI Stream interfaces.	Xilinx
AXI Uartlite (2.0)	Universal Asynchronous Receiver Transmitter. A controller interface for asynchronous serial data transfer. Allow users to use the terminal on SDK.	Xilinx
Constant (1.1)	A block that outputs a constant digital signal.	Xilinx
hash_table_mgr_mig_0	Handles hash table requests.	Custom
sd_card_controller_0	A block that directly transfers data from the SD card to the DDR memory.	Custom + Other Source need reference [3]

1.4.2 Client System Components

The Client system is implemented on the Nexys4 DDR board. The table lists all the IP components used for the Client FPGA system.

Table 2: Client system IP and their respective descriptions

IP	Function/Description	Origin
Microblaze (11.0)	A soft microprocessor core designed for Xilinx FPGA and used to run software code through Xilinx SDK.	Xilinx
Processor System Reset (5.0)	A block that handles the reset conditions for a given system and generates appropriate resets at the output.	Xilinx
AXI Interrupt Controller	A block that handles interrupt signals from	Xilinx

(4.1)	other IPs.	
Concat (2.1)	A block that concatenates multiple signals into a wider bus.	Xilinx
AXI TIMER (2.0)	A timer module with AXI interface.	Xilinx
Clocking Wizard (6.0)	A block that generates different clock frequencies from the input clock.	Xilinx
Microblaze_local_memory_ block - BRAM	BRAM memory block. Used as cache memory for MicroBlaze.	Xilinx
Memory Interface Generator (MIG 7 series) (4.2)	A block that provides an interface to access the DDR2 memory.	Xilinx
Microblaze Debug Module (MDM) (3.2)	A core that enables JTAG-based debugging of the MicroBlaze microprocessor.	Xilinx
AXI Interconnect (2.1)	Bus interface of the processor and the peripherals. Connects AXI memory-mapped masters with AXI memory-mapped slaves.	Xilinx
AXI SmartConnect (1.0)	A more compact and simpler bus interface that connects AXI memory-mapped masters with AXI memory-mapped slaves.	Xilinx
AXI EthernetLite	Ethernet block that transmits and receives packets through a network connection (10 Mbps and 100 Mbps). The block provides minimal network functionalities for the least amount of resources.	Xilinx
Ethernet PHY MII to Reduce MII	A block that follows the MII standard to connect the Ethernet PHY layer with the MAC	Xilinx

	layer.	
AXI Uartlite (2.0)	Universal Asynchronous Receiver Transmitter. A controller interface for asynchronous serial data transfer. Allow users to use the terminal on SDK.	Xilinx
hash_func_top_v1.0	A hash key generation IP that uses MD5, murmur3, and xor to generate a 16 bit hash key from a word (with a max word size of 288 bits)	Custom
spam_detection_infer_0	A block that takes probabilities data and runs spam detection inference to compute if the email is spam or non-spam	Custom + Other Source need reference [4]

2.0 Outcome

2.1. Results

The final design has met all the functional requirements and features proposed in the proposal. Two clients were able to concurrently access the hash table located at the server, and they were able to detect spam emails using the probabilities from the table. The clients and server could also communicate with performance GUI to display performance metrics.

Table 3 shows the original features proposed and their respective results. Although the original proposal has stated that the spam email detection will be performed at the software layer, the final design performs the algorithm at the hardware layer using a spam email detection inference IP. This is an additional feature that the team has accomplished.

Table 3: The original features from proposal and results

Functional requirements and features	**Acceptance criteria**	**Results/Status**
Hash table implementation	- Access a specific entry using key, and perform read and write - Mechanism to resolve conflicts	**PASS**, details of the hash table ip are found in section 4.2 Fig A-4, Appendix A shows that the server can access the hash table for probability.
Hashing algorithm	IP block can generate a hash key from a word	**PASS**, details of the hashing algorithm ip are found in section 4.1. See Fig A-6, Appendix A for result
Load data to the hash table through SD card	SD card contents are properly written to DDR memory	**PASS**, details of the SD card IP are found in section 4.3
Network communication	PCs and FPGAs can communicate using TCP/UDP	**PASS**, see Fig A-1,A-2 and A-3, Appendix A
Pretrained probabilistic models for spam detection	Generate a list of words with probability	**PASS**, see Fig A-8, Appendix A
Spam detection algorithm	The design can detect SPAM email	**PASS**, see Fig A-5, Appendix A Stretch goal: implementing the algorithm in hardware is completed
Performance test and visualization	The performance of hashing algorithm, hash table access time and network are displayed on GUI	**PASS**, see Fig A-7, Appendix A

2.2. Future Work

Although the current project has met all the proposed goals and features, there are rooms to improve the system. The first improvement that can be made is to develop an IP to directly process the email content from the memory. The current design splits the email into a list of words at the software layer before sending the list to the hash function IP. Instead of depending on MicroBlaze to split the string, the improvement idea will use an IP to directly access DDR, read words, generate hash keys, and save the results (hash keys) to DDR memory allocated for the hash function IP. This idea will simplify the software coding and improve the latency of hash key generation.

The second improvement idea is to upgrade the SD card IP to support FAT and file system, and integrate the IP into the client system. This will allow the design to save multiple email contents on an SD card and process multiple emails rather than processing one email in the memory.

The last improvement to mention is to duplicate the hash table, store the table in every client system (ie. remove the server), and keep the multiple hash tables synchronized. In other words, updating a hash table at client A will trigger an update to occur at the hash table at client B. This improvement has to make sure the tables located at different clients have the same content all the time. This will allow more clients to access the hash table in parallel and reduce the hash table access latency.

If the project could start over, we would not design the atomic version of hash function IP as an initial design. To be more specific, we would directly design the pipelined version of the hash function IP, and add a feature mentioned in the first improvement above. This would reduce the hassle of using the string library at the software layer and improve the efficiency of the hash function IP.

To take over the project, the next recommended step is to implement the first and second improvement ideas mentioned above. This will improve the system to be more efficient and fast.

3.0 Project Schedule

Table 4 compares the original milestone goals from the project proposal with the team's actual accomplishments for each milestone.

Table 4: Original milestone goals and actual accomplishments

	Original Goals	**Actual Accomplishments**
1	• Research hash functions, hash table implementations, spam word matching algorithms, and SD card and flash memory	• Researched hash functions • Researched hash table implementation • Researched SD card and serial flash • Researched spam detection algorithms
2	• Write RTL for custom hash function and hash table IPs (no verification required)	• Implemented hash table with no hash collision functionality • Implemented pretrained probabilistic model on spam detection • Implemented hash function and verified a submodule
3	• Verify custom IPs and integrate into client and server MicroBlaze systems	• Implemented hash collision functionality • Implemented SD card controller and verified basic read/write operations • Verified and fixed timing issues in hash function IP, and integrated into client system
4	• Implement network connection using software • Integrate SD card or flash memory functionality with hash table	• Verified hash table IP and integrated into server system • Implemented SD card controller interface with MIG • Fixed issues in hash function IP • Implemented network connection

5	• Catch up on previous milestones	• Added hash table functionality for multiple clients and invalid hash key handling • Verified SD card data transfer to DDR3 • Integrated SD card controller with hash table IP • Wrote spam detection software
6	• Write spam detection software • Implement performance monitoring	• Verified SD card controller with hash table IP • Implemented performance monitoring GUI • Implemented spam detection IP

Despite being behind schedule with verifying the hash table IP and with integrating the SD card controller with the hash table, the team finished the project on time and accomplished the stretch goal of implementing the spam detection in hardware instead of software. Verification of the hash table IP was planned to be done before Milestone 3, but was not completed until Milestone 4. Similarly, verification of the SD card controller operating in tandem with the hash table was planned to be done before Milestone 4, but was not completed until Milestone 6. Since the team did not commit to anything for Milestone 5, we used this milestone to implement missing functionality and begin working on goals for the next milestone, as well as stretch goals. The spam detection software was originally planned to be done for Milestone 6, but was completed ahead of schedule for Milestone 5. By Milestone 6, the spam detection hardware IP was code-complete but not yet verified, although it was verified in time for the final demo.

4.0 Description of Custom IP

4.1 Hashing Algorithm IP

4.1.1 Overview

Hashing algorithm IP utilizes MD5, murmur3 and XOR gate to generate a 16 bit hash key from a word (with maximum size of 288 bits). The MD5 submodule is designed using [5] and murmur3 submodule is designed using [6,7]. The AXI4 slave interface is built on top of the Xilinx's AXI4

template, and the FIFO is custom designed. Fig 4 shows the block diagram of the hashing algorithm IP, and table 6 shows the AXI4 interface registers and their descriptions.

The general operation of the hashing IP starts with storing an input word across Reg3 to Reg11, the size of the input word in bytes at Reg12, and the total number of words to process at Reg2 (the Reg2 setup can be ignored if the user has a separate mechanism to count the number of words to process). If the input data is less than 288 bits, the data has to be extended to 288 bits by adding 0s. The hashing operation starts when a user sets the Reg0 to 1. This will start the hashing process, and about 100 cycles later the hash key result will be stored in the FIFO. When a valid result exists in the FIFO, the FIFO sets Reg14 to the hash key data and sets the LSb of Reg13 to 1 to indicate the valid result is available.

The data flows as follow:

1. AXI4 slave registers, input data reg, are set
2. The 288 bits of data is extended to 416 bits of data by adding 0s at the end
3. MD5 process 416 bits of data to produce 128 bits of data
4. Murmur3 uses the 128 bits of data to produce 32 bit data
5. XOR gate is used to reduce the data to 16 bits by XORing the leftmost 16 bits and the rightmost 16 bits.
6. 16 bit hash key is stored in FIFO until the data is read from AXI4 interface

Current design has the FIFO size set to 150 words, so if the maximum limit is reached, the FIFO sends out a stall signal to both MD5 and murmur3 submodules to stall the entire pipeline. The pipelines will be stalled until a user reads a result value from the AXI4 slave register.

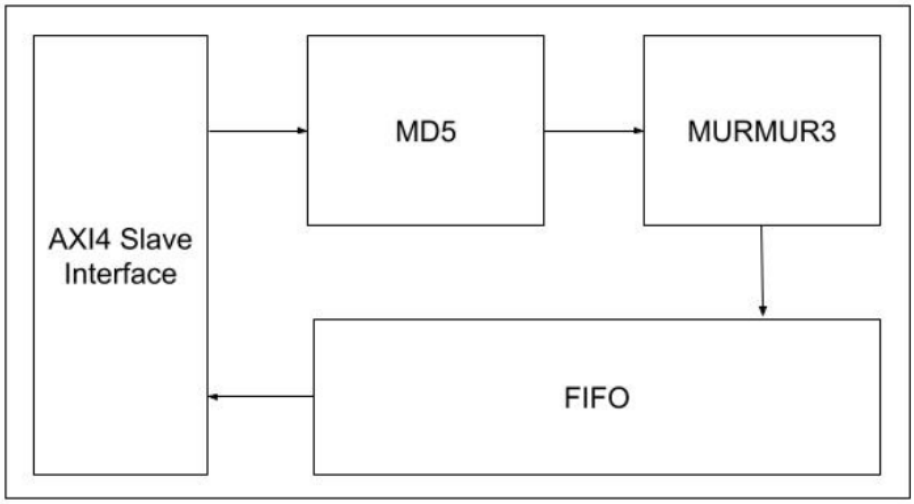


Figure 4. Block diagram of Hashing Algorithm IP

4.1.2 Interface Signals

AXI4 slave interface has been used for the hash function ip. The table 5 describes the main interface signals such as clk, and resetn. The control signals have been grouped together. The details about the AXI4 slave interface should be refered to Xilinx's AXI4 document.

Table 5: AXI4 interface signals for hash function IP

Signal Name	**Direction**	**Width**	**Description**
s01_axi_$CTRL	in/out	-	$CTRL is used to describe multiple AXI4 control signals such as awready, wready, arready, rready and etc.
s01_axi_aclk	Input	1	100MHz clock
s01_axi_aresetn	Input	1	Active-low synchronous reset
s01_axi_$DATA	In/Out	32	$DATA can be either rdata or wdata. The details about the data content is available in table 6

s01_axi_$ADDR	In/Out	32	$ADDR can be either araddr or awaddr. The details about the address mapping of the registers is available in table 6

4.1.3 Register Map

The table 6 shows the AXI4 slave address and data interface of the hashing algorithm IP. Reg1 is not included since it is not used for the final design of the IP.

Table 6: Hashing algorithm IP AXI4 Slave interface registers and description

Register Num	**Offset**	**Register Name**	**Description**
Reg0	0x0	Start Ctrl	LSb is used to indicate the input data is valid, and start the hash key generation process. It is automatically reset to 0 after a clock cycle.
Reg2	0x8	Num Words	Used to store the total number of words to process
Reg3 - Reg11	0xC - 0x2C	Input Data	Registers to store a word, If the word is less than 288 bits the rest should be padded with 0s
Reg12	0x30	Length of input word in byte	Stores the number of bytes of the current word
Reg13	0x34	Done Ctrl	Used to indicate the result data is valid or not. Byte3 is set to 1 if the total number of words to process matches the number of words processed. Byte2 indicates the number of words left to process. Byte0 indicates the validity of the result data. The byte0 should be set to 0 after reading the result data to read

			the next available hash key from FIFO
Reg14	0x38	Result Data	Hash key result data

4.1.4 Verification

The IP block has been verified using testbenches and a software model. The results from the testbenches and the software model are compared to verify the correctness of the design. The testbenches and the software model can be found at the Github repo mentioned in section 5.

4.2 Hash Table Processing & Synchronization IP

4.2.1 Overview

The hash table IP interacts with the MIG and DDR3 memory to implement a parallel hash table. The entire hash table is partitioned into 32 sections, with a lock associated with each section to enforce mutual exclusion. Both the hash key (the word) and data (the probabilities associated with the word being spam or ham) are stored in the hash table to enable hash collision detection. Hash collisions are resolved linearly (i.e. by checking the memory addresses immediately above the given hash table address).

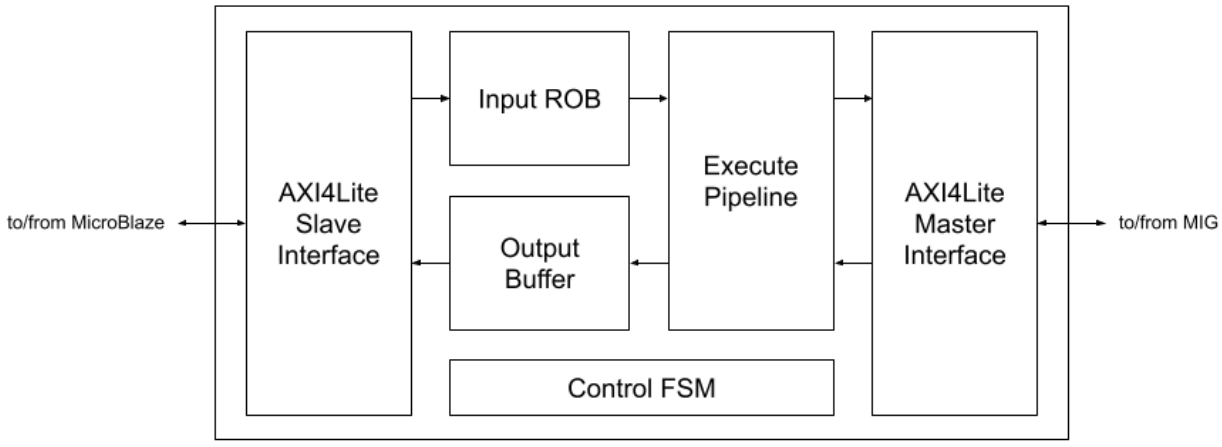


Figure 5. Block diagram of Hash Table Processing & Synchronization IP

The block diagram of the custom hash table IP is shown in Figure 5. The MicroBlaze processor handles hash table requests by writing to slave registers via an AXI4Lite slave interface. These slave registers store the hash key, hash table address, data, and request type (read, insert, update, or delete) for the incoming request, and write these values to the input reorder buffer (ROB). A request in the input ROB is issued to the execute pipeline if the lock associated with the hash table address is not set; upon issuing, the lock is set. Importantly, the requests in the ROB may be issued in a different order than that in which they were received.

In the execute pipeline, first memory reads are issued in order to find the address with the correct hash key (in the case of read, update, or delete requests) or an empty hash key (in the case of insert requests). If a read, update, or delete request specifies a key that does not exist in the hash table, the request is immediately removed from the pipeline. After finding the correct address, memory reads are issued for read requests, and memory writes are issued for insert, update, and delete requests. All memory operations are handled by an AXI4Lite master interface to the MIG.

For read requests, the data from memory reads is written to the output buffer. For insert, update, and delete requests, an acknowledgement is written to the output buffer to indicate that the request has been served. Data in the output buffer is written to slave registers, then are read by the MicroBlaze processor, again through the AXI4Lite slave interface.

4.2.2 Interface Signals

AXI4Lite slave and master interfaces have been used for the hash table IP. Table 7 describes the main interface signals such as the clocks and resets. The control signals have been grouped together. The details about the AXI4 interfaces can be found in Xilinx's AXI4 document.

Table 7: AXI4Lite interface signals for hash table IP

Signal Name	**Direction**	**Width**	**Description**
s01_axi_$CTRL	in/out	1	$CTRL is used to describe multiple AXI4 control signals such as

			awready, wready, arready, rready, etc.
s01_axi_aclk	in	1	50 MHz clock
s01_axi_aresetn	in	1	Active-low synchronous reset
s01_axi_$DATA	in/out	32	$DATA can be either rdata or wdata. Details about the data content are available in Table 8.
s01_axi_$ADDR	in/out	32	$ADDR can be either araddr or awaddr. Details about the address mapping of the registers are available in Table 8.
m01_axi_$CTRL	in/out	1	$CTRL is used to describe multiple AXI4 control signals such as awready, wready, arready, rready, etc.
m01_axi_aclk	in	1	50 MHz clock
m01_axi_aresetn	in	1	Active-low synchronous reset
m01_axi_$DATA	in/out	32	$DATA can be either rdata or wdata. Details about the data content are available in Table 8.
m01_axi_$ADDR	In/Out	32	$ADDR can be either araddr or awaddr. Details about the address

			mapping of the registers are available in Table 8.

4.2.3 Register Map

Table 8 shows the AXI4 slave registers of the hash table IP. Register numbers not mentioned in the table are unused.

Table 8: Hash table IP AXI4 slave interface registers and description

Register Num	**Offset**	**Register Name**	**Description**
Reg0	0x0	Input valid	Used to indicate that registers 1-12 are valid.
Reg1	0x4	Input address	Base address of hash table lookup to perform.
Reg2	0x8	Opcode	Hash table operation to perform: 0 - read, 1 - insert, 2 - update, 3 - delete.
Reg3 - Reg4	0xC - 0x10	Input data	Input data to hash table. Used only for opcodes 1 and 2.
Reg5 - Reg12	0x14 - 0x30	Input key	Input hash key. Used for collision detection.
Reg16	0x40	Output valid	Used to indicate that registers 17-28 are valid.
Reg17	0x44	Output address	Base address of hash table lookup performed.
Reg18	0x48	Output type	Output type: 0 - data, 1 - acknowledgement.
Reg19 - Reg20	0x4C - 0x50	Output data	Output data. Used only for output type 0.

Reg21 - Reg28	0x54 - 0x70	Output key	Output hash key.

4.2.3 Verification

This IP block has been verified using a testbench, which can be found in the GitHub repo mentioned in Section 5.

4.3 SD Card Controller IP

4.3.1 Block Overview

The SD Card Controller IP is responsible for data transfer directly from the SD card to the DDR memory on the FPGA. The IP block bypasses the Microblaze during data transfer to achieve better performance.

As a basis for the SD card access portion of the design, we used a reference design by Jonathan Matthews posted on GitHub [3]. The reference design includes an SD SPI controller to communicate with the SD card. On top of it, the team added the initialization sequence FSM to target the specific SD card (Gigabyte Camera Plus 32 GB MicroSD card) in the lab and modified the SD card read operation, customizing it to our specific use case. The team also designed the data transfer hardware block to move the data from the SD card to the DDR.

The SPI clock runs at 25MHz and loading the 4MB hash table data takes less than 3 seconds.

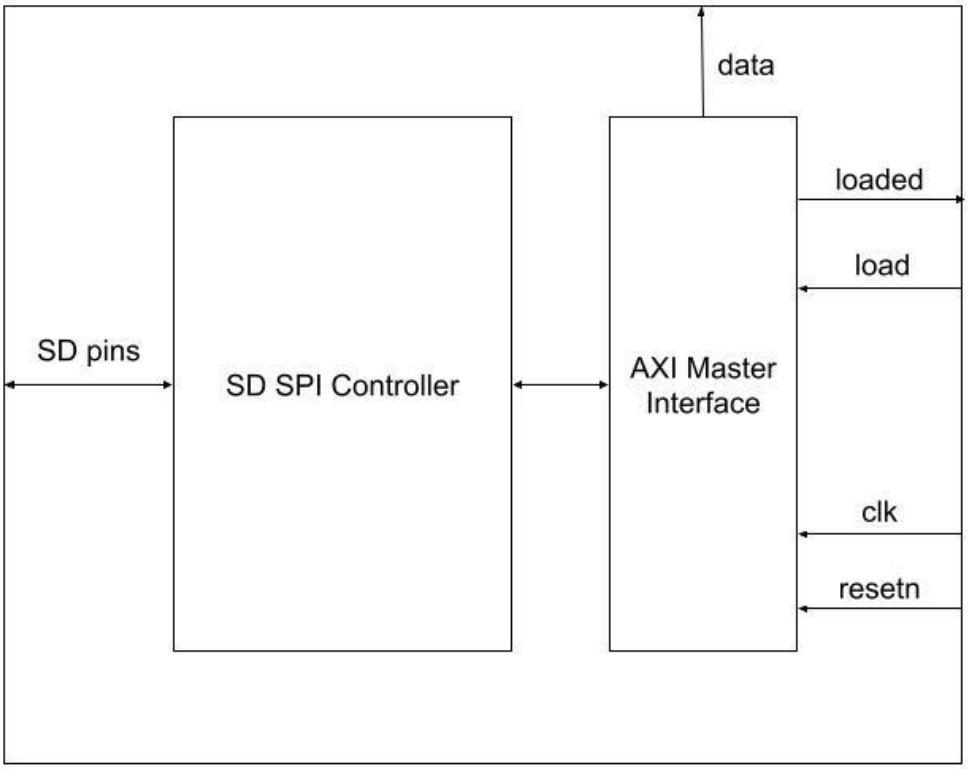


Figure 6. Block diagram of SD Card Controller IP

4.3.2 Interface Signals

The interface signals are listed in Table 9. The SD-related signals should be assigned to external pins in the constraint file (find the pin assignments in the Nexys board schematics).

Table 9: SD Card Controller IP interface signals

Signal Name	**Direction**	**Width**	**Description**
clk	Input	1	100 MHz Clock
resetn	Input	1	Active-low synchronous reset.
SD_CD	Input	1	External pin

SD_RESET	Output	1	External pin
SD_SCK	Output	1	External pin, SPI SCLK
SD_CMD	Output	1	External pin, SPI DI
SD_DAT	Inout	4	External pin, SPI DO, SPI CS
load	Input	1	When load signal is high, the SD card controller will start the data transfer
loaded	Output	1	When data transfer (load) is finished, the SD card controller will be set high

4.3.3 Parameters

The IP has three parameters: SD_START_ADDR, SD_DATA_SIZE, and SD_DDR_BASE_ADDR (Table 10). The user can decide the address to load from the SD card, the address to store on the DDR memory, and the size of the data transfer.

Table 10: SD Card Controller IP parameters

Parameter Name	**Description**
SD_START_ADDR	Start address of SD card to read from
SD_DATA_SIZE	Size of the data transfer from the SD card to the DDR
SD_DDR_BASE_ADDR	Start address of DDR memory to write to

4.3.4 Bare Metal Code

Once the reset signal is deserted and the load signal is high, the data transfer from the SD card to DDR will start. Once the loaded signal output is set high by the SD controller, the data transfer is finished. The software pseudo-code is shown in Figure 7.

```
resetn = 0;
resetn = 1;
load = 1;

// Polling mode until SD card data transfer finishes
while (loaded == 0) {

}

// SD card data is loaded onto the DDR (done)
```

Figure 7. Pseudo code for the SD Card Controller IP

4.4 Spam Detection Inference IP

4.4.1 Block Overview

The Spam Detection Inference IP block is designed to receive probabilistic data from the "bag-of-word" model and compute the spam detection result on whether the email is spam or non-spam. This IP instantiates Xilinx FIFO Generator IP and FPU module from OpenCores [4] for floating-point addition and comparison. The team designed the data processing and interconnect logic to handle the data output from the FIFO and then use the FPU to run spam detection inference and compute the probabilistic values on spam and non-spam email based on the Naïve Bayes algorithm.

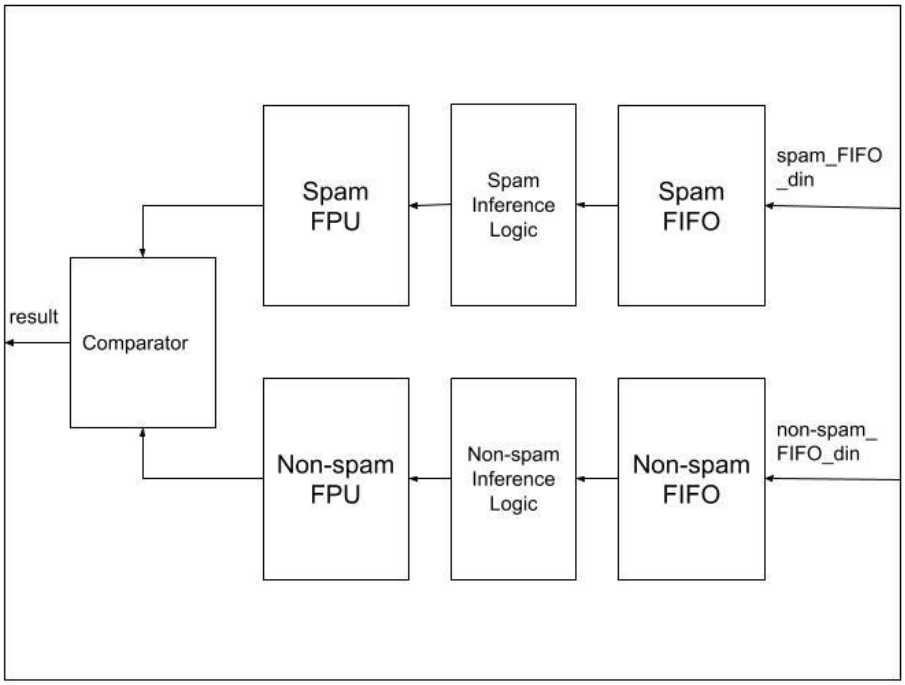


Figure 8. Block diagram of Spam Detection Inference IP

The simulation testbench is located in ip_repo/spam_detection_inference_1.0/src/spam_detection_controller_tb.v and the SDK software test is located in ip_repo/spam_detection_inference_1.0/helloworld.c. Both of the tests input the probabilistic data to the module and check if the output probabilistic value and result (i.e., spam or non-spam) are correct.

4.4.2 Interface Signals

The interface signals are listed in Table 10.

Table 11: Spam Detection Inference IP interface signals

Signal Name	**Direction**	**Width**	**Description**
S00_AXI	-	-	AXI slave that receives probabilistic data and return the result of inference

s00_axi_aclk	Input	1	50MHz clock
S00_axi_aresetn	Input	1	Active-low synchronous reset
result	Output	1	Spam detection result 0: non-spam email 1: spam email

4.4.3 Register Map

The register map is shown in Table 11.

Table 12: Spam Detection Inference IP register map

Offset	**Register Name**	**Type**	**Description**
0x0	spam_fifo_din	RW	Connects to the data input of the spam FIFO.
0x4	spam_fifo_wr_en	WO	Connects to the write enable of the spam FIFO. Once set high, data on spam_fifo_din will be stored to the spam FIFO. Self clear in one cycle.
0x8	spam_prob	RO	Output the spam probability after spam detection inference.
0xC	ham_fifo_din	RW	Connects to the data input of the ham FIFO.
0x10	ham_fifo_wr_en	WO	Connects to the write enable of the ham FIFO. Once set high, data on ham_fifo_din will be stored to the ham FIFO. Self clear in one cycle.

0x14	ham_prob	RO	Output the ham probability after ham detection inference.
0x18	spam_write_done	RW	Set high after all input probabilistic data are fed to the spam FIFO through spam_fifo_din and spam_fifo_wr_en.
0x1C	ham_write_done	RW	Set high after all input probabilistic data are fed to the ham FIFO through ham_fifo_din and ham_fifo_wr_en.
0x20	spam_data_valid	RO	0: spam inference computation not done. 1: spam inference computation done and spam_prob is valid.
0x24	ham_data_valid	RO	0: ham inference computation not done. 1: ham inference computation done and ham_prob is valid.
0x28	result	RO	Spam detection result. Result is valid when both spam_data_valid and ham_data_valid are 1. 0: spam email 1: non-spam email

4.4.4 Bare Metal Code

To run the IP block in software, first writes the probabilistic data to the FIFOs in the Spam Detection Inference IP through the AXI interface. Then wait (polling) until the inference computation is finished and read the result (spam or non-spam email). The pseudo-code is shown in Figure 9.

```
spam_list = list of probabilistic data from the spam bag of words from the pretrained model (already taken log)
ham_list = list of probabilistic data from the non-spam bag of words from the pretrained model (already taken log)

// Send in spam probabilistic data
for data in spam_list {
        spam_fifo_din = data
        spam_fifo_wr_en = 1
}
spam_write_done = 1

// Send in non-spam probabilistic data
for data in ham_list {
        ham_fifo_din = data
        ham_fifo_wr_en = 1
}
ham_write_done = 1

// Wait until inference computation finished
while (spam_data_valid == 0 && ham_data_valid == 0) {
        ;
}

// return spam detection inference result: 0 - non-spam; 1 - spam
return result
```

Figure 9. Pseudo code for the Spam Detection Inference IP

5.0 Design Tree

Figure 10. GitHub directory tree

Only the key design files are listed below:

- gui/perf_gui.py: Python performance GUI
- hash_client/client_proj
    - client_proj.sdk/client_w_spam/src/main.c: MicroBlaze code for hash client
    - client_proj.srcs/sources_1/bd/design_1/design_1.bd: block design for hash client
- hash_server
    - server_3/src/main.c: MicroBlaze code for hash server
    - hash_server.srcs/sources_1/bd/design_1/design_1.bd: block design for hash server
- ip_repo
    - hash_func_ip/src/hash_func_S_AXI.v: main module for hashing algorithm IP
    - hash_table_mgr_1.0/hdl/hash_table_mgr_v1_0_S00_AXI.v: main module for hash table IP interface with MicroBlaze

    - hash_table_mgr_mig_1.0/hdl/hash_table_mgr_mig_v1_0_M00_AXI.v: main module for hash table IP interface with MIG
    - sd_card_controller_1.0/hdl/sd_card_controller_v1_0_M00_AXI.v: main module for SD card controller IP
    - spam_detection_inference_1.0/hdl/spam_detection_inference_v1_0_S00_AXI.v: main module for spam detection inference IP

Tips and Tricks

- Make sure to plan well, and have a proper research step at the beginning
- Coordinate the IP block interfaces to ensure all blocks can integrate properly
- Debug systematically and sequentially

Reference

[1] “Antispam hardware: Hosted anti-spam hardware vs software antispam,” Comodo, 07-Nov-2018. [Online]. Available: https://www.comodo.com/business-security/email-security/anti-spam-hardware.php. [Accessed: 01-Apr-2022]

[2] “Hash Tables,” Data Structures Handbook. [Online]. Available: https://www.thedshandbook.com/hash-tables/. [Accessed: 01-Apr-2022].

[3] J. Matthews (19-Nov-2014) sd_controller source code [Source code]. https://github.com/jono-m/mariokart/blob/master/v1/v1.srcs/sources_1/new/sd_controller.v

[4] U. Rudolf (16-Dec-2018) Floating Point Unit source code [Source code]. https://opencores.org/projects/fpu

[5] R.Rivest, The MD5 Message-Digest Algorithm, RFC 1321, MIT LCS & RSA Data Security, Inc., April 1992

[6] MurmurHash, wikipedia.org, https://en.wikipedia.org/wiki/MurmurHash (accessed Jan 29, 2022)

[7] smhasher, (2014), Aappleby, Accessed: Jan 29,2022. [Online] Available: https://github.com/aappleby/smhasher

Appendix A: Project Results

```
*********************************************
      SPAM EMAIL DETECTION - menu
*********************************************
'start' - Start spam email detection - emails in mem
'type' - Start spam email detection - let user type email
'admin1' - Insert a new key/values to hash table
'admin2' - Update existing entry in the server
'admin3' - Delete an entry in the server
'quit' - Exit
*********************************************
>>
SENDING EMAIL CONTENT BELOW:
let s face it age should be nothing more than a number it s okay to want to hold on to your young body as long as you
Setting up GUI client connection
>> Waiting for GUI server to accept connection
*********************************************
Connection to GUI server established
*********************************************
Packet sent to GUI successfully, 2 bytes
---------------------------------------------
```

Figure A-1: screenshot of client terminal displaying the menu and GUI network connection

```
PRINT SENDING DATA:  00 84 CD 00 74 65 6C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0A
Packet sent successfully, 44 bytes
---------------------------------------------
************************************************************
```

Figure A-2: Content of data packet from a client

```
************************************************************
Packet received, 43 bytes
************************************************************
__VALID DATA SET (hex)__:  C0 B6 66 63 C0 E1 76 95
>> C0B66663 C0E17695 IN HEX:  00 42 54 65 72 65 68 00 00 00 00 00 00 00 00 00 00 00 00 00 00
************************************************************
END LOOP
```

Figure A-3: Content of data received from server (hash table)

```
Sending response to client 0:
   Type: 0
   Address: 0xF0 D
   Key: irra□dev□□□□□□□□□□□□□□□□□□□□□□□□□□□□
   Data: C1275F38C1309D50

Sending opcode 2 to GUI

Received request from client 0:
   Operation: 0
   Address: 0xEFF9
   Key: □tey□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
   Data: 0000000000000000

Sending opcode 0 to GUI

Sending opcode 1 to GUI
```

Figure A-4: Screenshot of server terminal displaying the response packet to a client and a request packet from a client

```
************************************************************
Packet received, 43 bytes
************************************************************
__VALID DATA SET (hex)__:  C1 04 B7 CC C1 03 48 78
spam:ham:<<-601.745510 -608.179663>><-8.294872 -8.205193>
IN HEX:  00 EF F9 00 74 65 79 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C1 04 B7 CC C1 03 48 78
************************************************************
END LOOP
Packet sent to GUI successfully, 2 bytes
----------------------------------------
inside:0 106 106
SPAM EMAIL DETECTED
Packet sent to GUI successfully, 594 bytes
----------------------------------------
Packet sent to GUI successfully, 2 bytes
----------------------------------------
DONE PROCESSING
FREE DYNAMIC MEMORY SPACES
```

Figure A-5: Screenshot of client terminal displaying the response packet from server, and the result of spam email detection algorithm

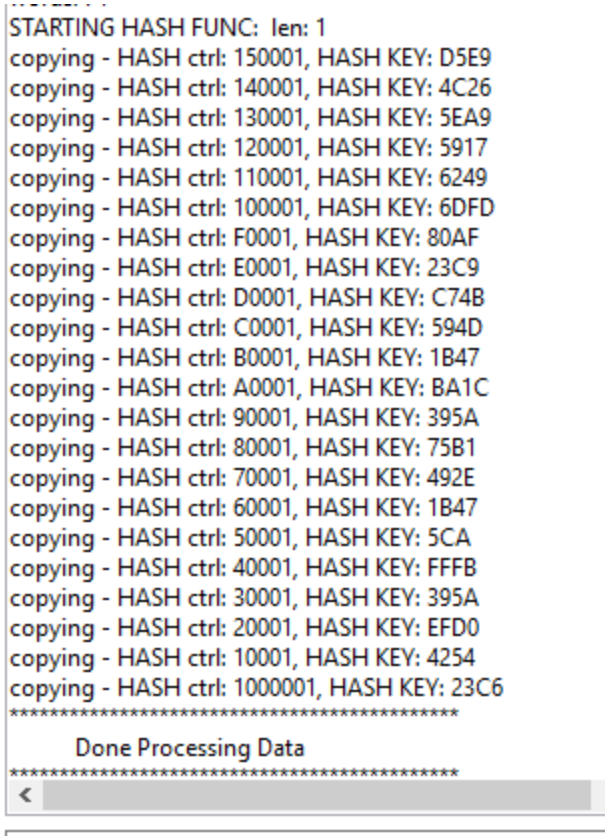


Figure A-6: screenshot of client terminal displaying the list of hash keys generated from an email

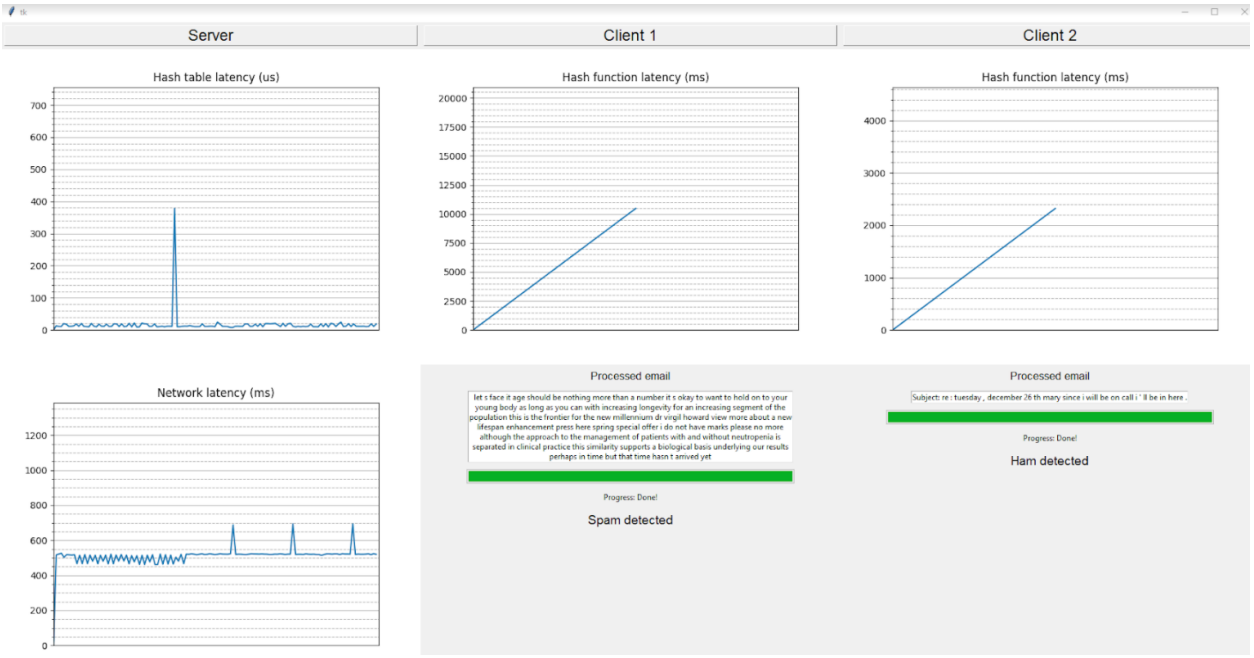
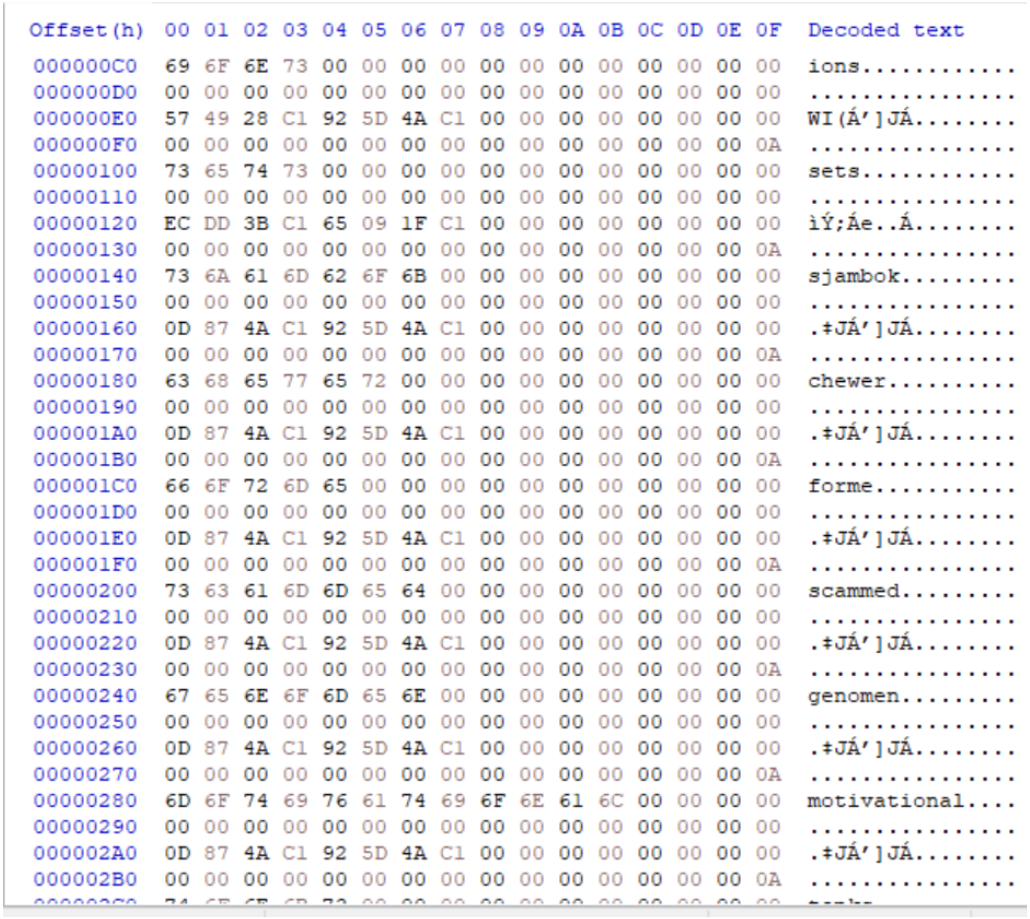


Figure A-7: Screenshot of performance GUI



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000000C0	69	6F	6E	73	00	00	00	00	00	00	00	00	00	00	00	00	ions.....
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000000E0	57	49	28	C1	92	5D	4A	C1	00	00	00	00	00	00	00	00	WI(Á']JÁ.....
000000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....
00000100	73	65	74	73	00	00	00	00	00	00	00	00	00	00	00	00	sets.....
00000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000120	EC	DD	3B	C1	65	09	1F	C1	00	00	00	00	00	00	00	00	ìÝ;Áe..Á.....
00000130	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....
00000140	73	6A	61	6D	62	6F	6B	00	00	00	00	00	00	00	00	00	sjambok.....
00000150	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000160	0D	87	4A	C1	92	5D	4A	C1	00	00	00	00	00	00	00	00	.‡JÁ']JÁ.....
00000170	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....
00000180	63	68	65	77	65	72	00	00	00	00	00	00	00	00	00	00	chewer.....
00000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001A0	0D	87	4A	C1	92	5D	4A	C1	00	00	00	00	00	00	00	00	.‡JÁ']JÁ.....
000001B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....
000001C0	66	6F	72	6D	65	00	00	00	00	00	00	00	00	00	00	00	forme.....
000001D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000001E0	0D	87	4A	C1	92	5D	4A	C1	00	00	00	00	00	00	00	00	.‡JÁ']JÁ.....
000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....
00000200	73	63	61	6D	6D	65	64	00	00	00	00	00	00	00	00	00	scammed.....
00000210	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000220	0D	87	4A	C1	92	5D	4A	C1	00	00	00	00	00	00	00	00	.‡JÁ']JÁ.....
00000230	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....
00000240	67	65	6E	6F	6D	65	6E	00	00	00	00	00	00	00	00	00	genomen.....
00000250	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000260	0D	87	4A	C1	92	5D	4A	C1	00	00	00	00	00	00	00	00	.‡JÁ']JÁ.....
00000270	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....
00000280	6D	6F	74	69	76	61	74	69	6F	6E	61	6C	00	00	00	00	motivational....
00000290	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
000002A0	0D	87	4A	C1	92	5D	4A	C1	00	00	00	00	00	00	00	00	.‡JÁ']JÁ.....
000002B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	.....

Figure A-8: image of binary file generated from pretrained probabilistic models for spam detection