COMP3231 Project Manual

W12A Group One

z5216632 - Gabriel Tay Wei Chern

z5209390 - Malavika Pasupati

z5205690 - Quynh Phan

z5232920 - Lindsay Small

z5206677 - Andrew Wong

COMP3231 Project Manual

W12A Group One

1 Schematics

1.1 ASIP Design Overview

In networked computing systems, the data processed in a processor system can be marred by *soft errors*, such as a temporary condition in the DRAM that unintentionally alters the stored data, or can be under *integrity attack* by adversaries when it is transferred over the network.

To ensure data integrity, an ASIP is placed in-between the processor and the network to ensure that data being sent to the network is soft-error free, and data received from the network is not tampered before being used by the destination processor. This is done by designing the ASIP to offer a double layer protection: one to detect soft errors and another to detect integrity attacks, as seen in Figure 1.1.

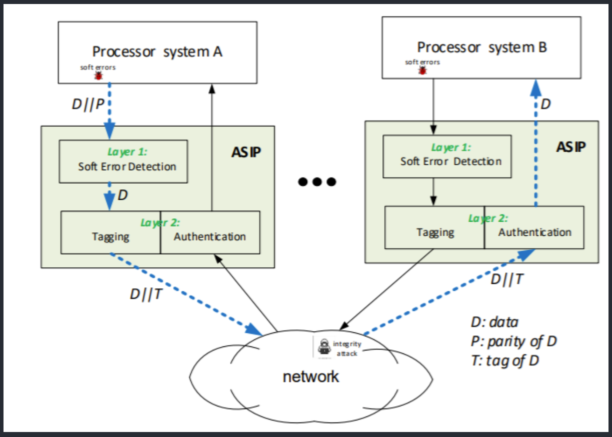


Figure 1.1 System Overview

To detect software errors, 1-bit parity testing is used, as the probability of multi-bit error is minimal and is not induced by malicious action. Before data can be sent to the network, the ASIP calculates the expected parity, given the processor's data, and compares this to the parity bit sent alongside the data, as seen in the figure below. If the received parity bit is the same as the calculated parity bit, the ASIP will attach a tag to the data and send both the data and tag to the network, otherwise an alert will be sent to the processor system as the data is considered to have a software error.

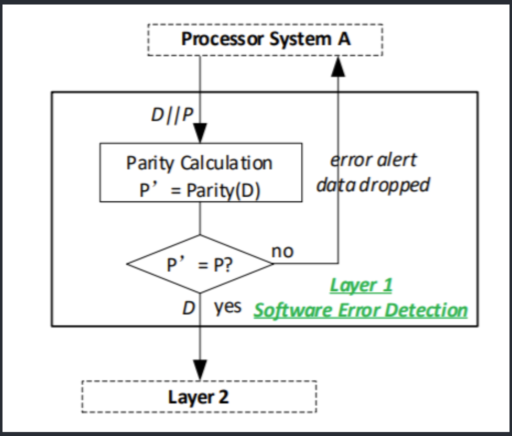


Figure 1.2 Parity Check Logic

To detect any integrity attacks, the ASIP generates a tag from the data received from the network and compares this with the tag sent alongside the data. If the tag generated by the ASIP is the same as the tag received from the network, the ASIP will send the data to the local processor, otherwise the data is dropped and an alert is recorded to be used later for security analysis.

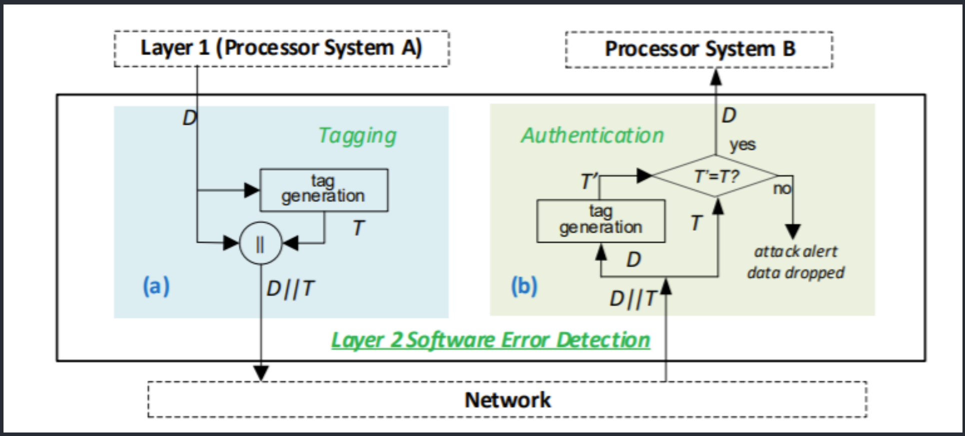


Figure 1.3 Tag Generation and Authentication

The ASIP uses a pipelined processor approach so instructions sent can be processed simultaneously, reducing the processor's cycle time and increasing the throughput of instructions. The block diagram below is an overview of the ASIP designed to process the data being sent to and received from the network.

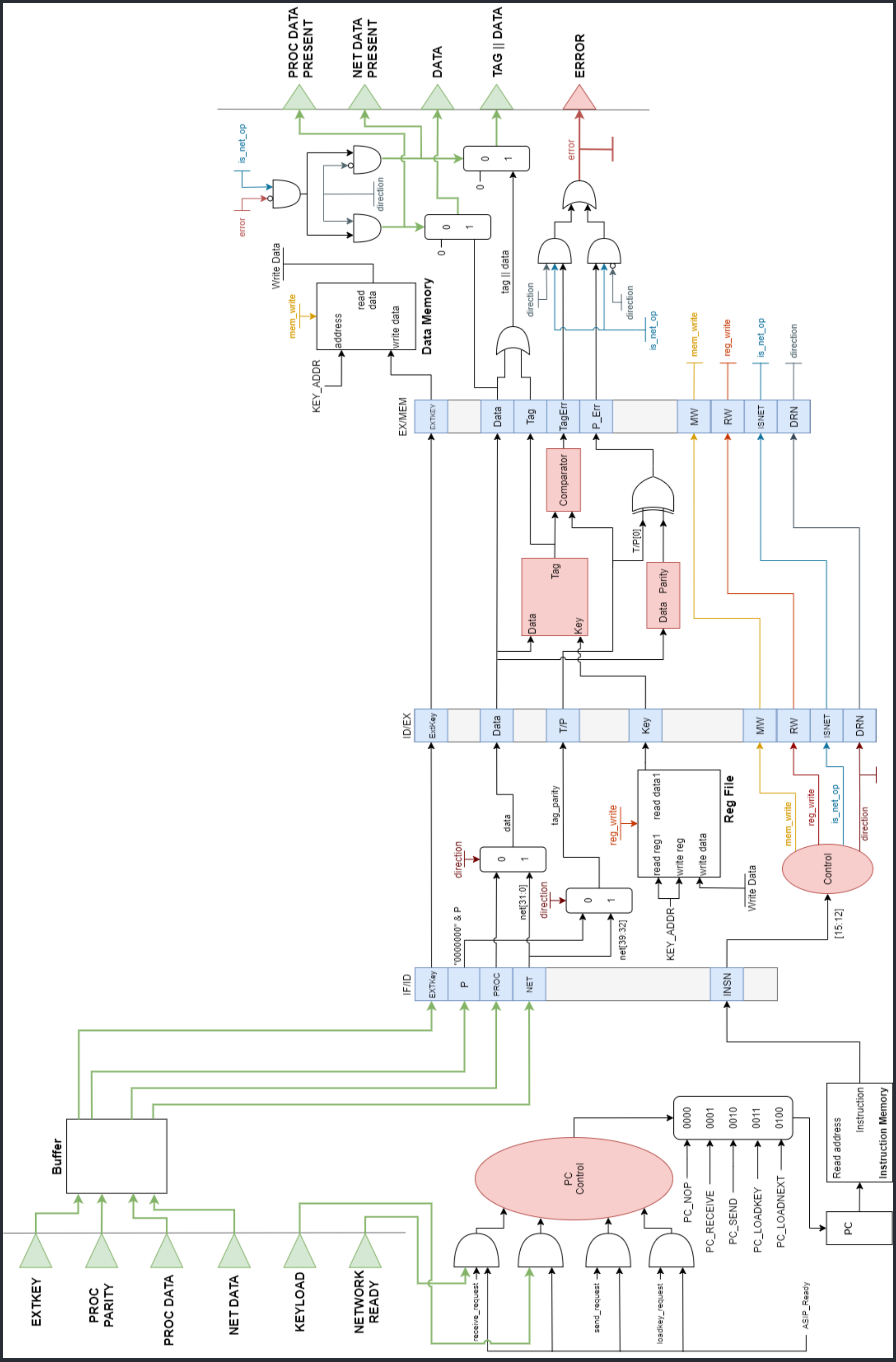


Figure 1.3 Tag Generation and Authentication

1.2 Pipeline Stages

1.2.1 Instruction Fetch (IF) Stage

The purpose of this stage is to fetch the instruction that is to be executed in the ASIP.
The instruction is first processed by the *PC Control* unit, from which the *PC* (Program Counter) will be set to the address of the instruction within Instruction Memory to be executed. !

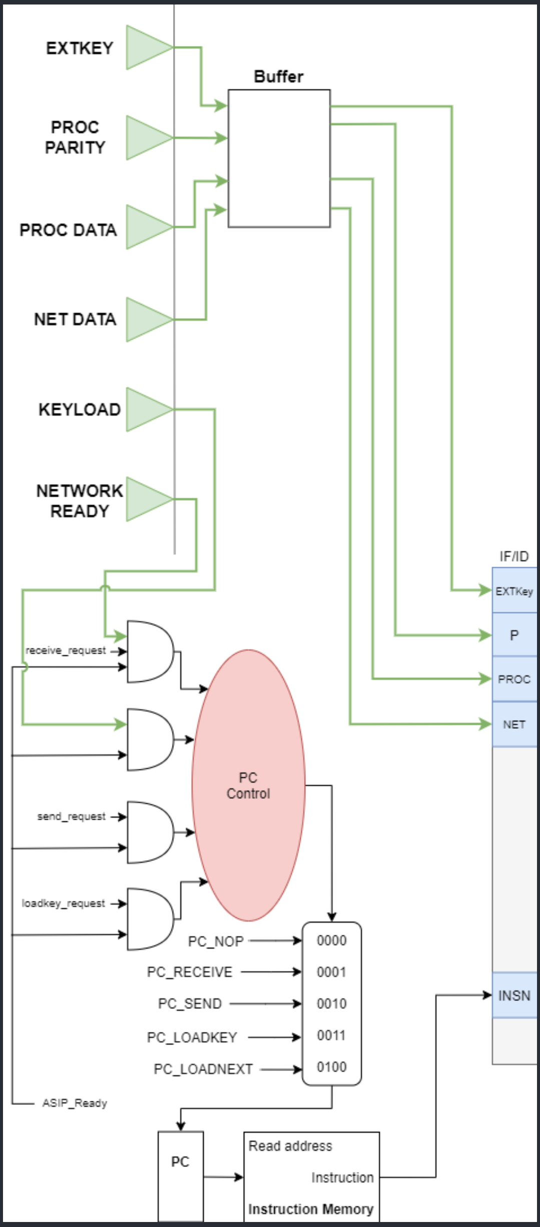


Figure 1.4 Instruction Fetch Stage

The *PC Control* only sends the address to the PC if the ASIP is ready to begin processing (indicated by the ***ASIP Ready*** flag), and in the case where data is to be received from the network, the network must also be ready to send data (indicated by the ***Network Ready*** flag).

After receiving the address from the *PC Control* unit, the *PC* sends the address to the *Instruction Memory* component to access the instruction code that is used to execute the command.

A buffer register is also used to first store the incoming data before it is passed into the IF/ID Pipeline Register. This is to account for the one clock cycle delay that arises when the *Instruction Memory* fetches the set instruction.

1.2.2 Instruction Decode (ID) Stage

The purpose of this stage is to decode the inputted instruction and to prepare the data needed for the execute stage of the ASIP for the given instruction.

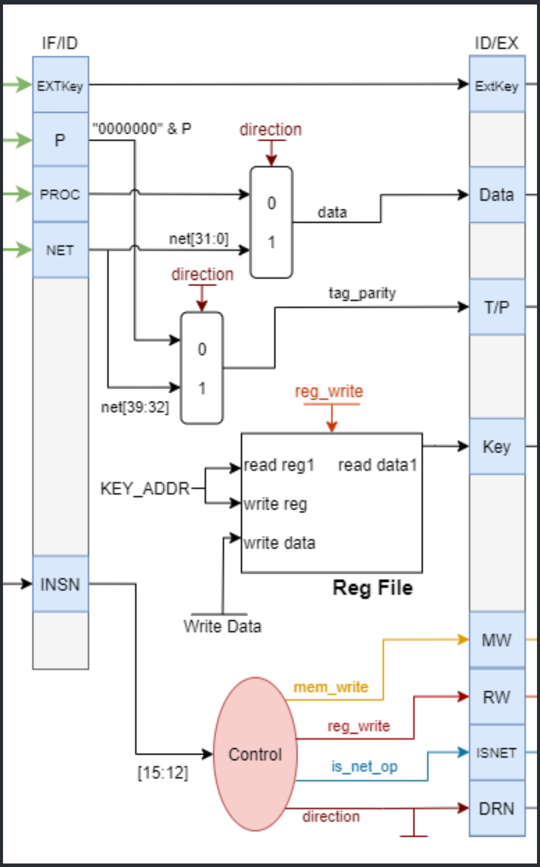


Figure 1.5 Instruction Decode Stage

The instruction fetched from the *Instruction Memory* component is sent to the *Control Unit*, which will send the necessary control signals to successfully execute the instruction.

Data is also forwarded to the EX stage. The data source (Network or Processor) is selected according to the instruction, and the relevant data is forwarded to the ID/EX Pipeline Register. This includes either the tag or parity bit, depending on the direction of data transfer, as outlined in. Note that the parity bit is extended to an 8-bit number before forwarding.

Additionally, in this stage, the secret key is loaded for the tag generation process that is completed in the *Execute* (EX) stage of the pipeline.

1.2.3 Execute (EX) Stage

The purpose of this stage is to execute the instruction given to the ASIP.

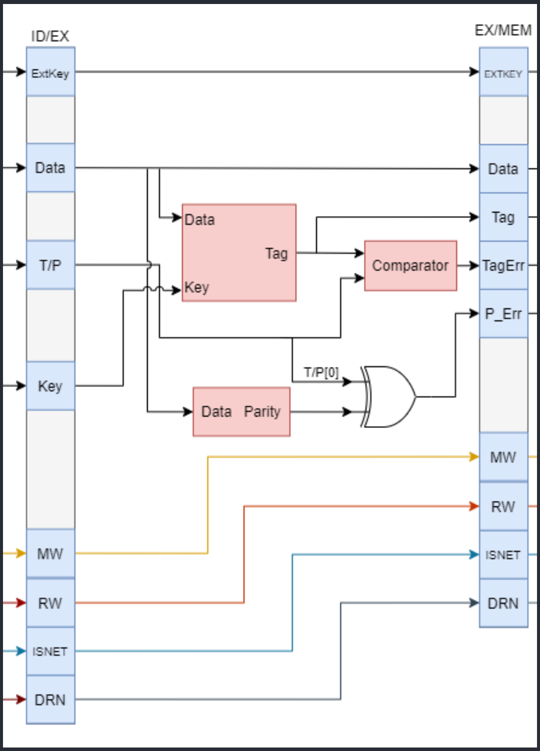


Figure 1.6 Execute Stage

The data given in the input is passed into the *Parity Calculation* component to obtain the *parity bit* and the *Tag Generation* component to generate a *tag* according to the secret key.

The parity bit calculated and the tag generated are compared with the parity or tag data given in the input to see if they are equal. If not, an error flag, TagErr or P_Err is set in the EX/MEM pipeline register

1.2.4 Memory Access (MEM) Stage

The purpose of this stage is to access the memory of the ASIP to store the external key, if instructed to. This is also the final stage of the ASIP, in which the data will be outputted to the network or the processor system depending on the instruction given.7

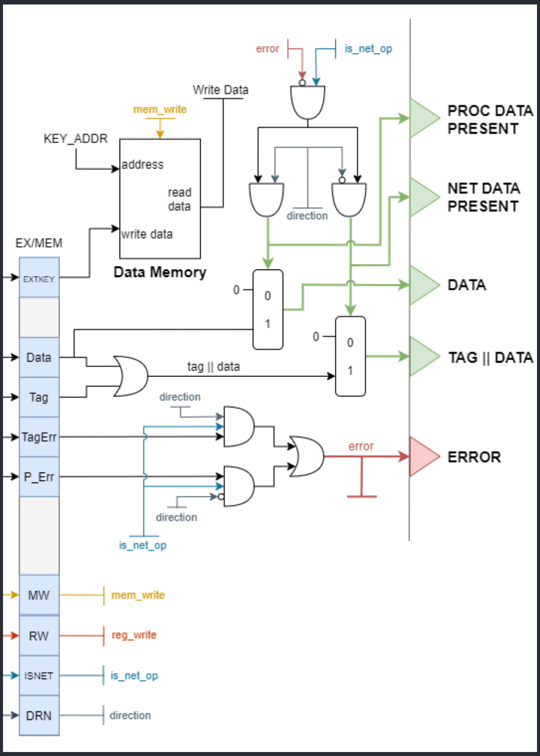


Figure 1.7 Memory Stage

If the ASIP is instructed to send data to the network, it will send the data and the tag, and will set the *Net Data Present* flag to signal that there is data to be sent. If the parity bits were not the same when compared in the EX stage, (i.e. the *P_Err* flag is set), the ASIP will raise an error signal, notifying the processor of a software error, and not sending the data or tag to the network.

If, however, the ASIP is instructed to receive data from the network, it will forward the data to the processor and will set the *Proc Data Present* flag to signal that there is data on the bus. If, however, *TagErr* is set, the ASIP will instead raise an error signal, notifying that there has been an integrity attack, and not sending through the corrupted data.

1.3 Pipeline Registers

1.3.1 IF/ID Pipeline Register

Input Port	Formula
EXTKey	ExtKey
P	ProcParity
PROC	ProcData
NET	NetData
INSN	Instruction_Memory

1.3.2 ID/EX Pipeline Register

Input Port	Formula
ExtKey	IF/ID.EXTKey
Data	(~direction)•IF/ID.PROC + direction•IF/ID.PROC
T/P	direction•IF/ID.NET + (~direction)•(“0000000” || P)
Key	reg_write•RegFile[key_addr]
MW	mem_write
RW	reg_write
ISNET	is_net_op
DRN	direction

1.3.3 EX/MEM Pipeline Register

Input Port	Formula
EXTKEY	ID/EX.EXTKey
Data	ID/EX.Data
Tag	Tag (Generated by tag generator)
TagErr	Tag•ID/EX.T/P
P_Err	ID/EX.Tag•ID/EX.T/P
MW	mem_write
RW	reg_write
DRN	direction

1.4 ASIP Components

1.4.1 Program Counter (PC) Control

The PC Control unit processes input signals to determine the value sent to the PC, and ultimately the instruction to be processed[1].

Input(s)	Output(s)
Request Signals, ASIP_ready, network_ready	Next PC Value

1.4.2 Program Counter (PC)

The PC inputs addresses referring to Instruction Memory based on the PC Controller output, therefore determining the next instruction

Input(s)	Output(s)
PC next instruction MUX	PC Next Instruction

1.4.3 Instruction Memory

The Instruction Memory unit process instructions based on the PC value and sends the instruction code required to execute the instruction [2] .

Input(s)	Output(s)
PC value	Instruction OP_Code

1.4.4 Control Unit

The Instruction Memory unit process instructions based on the PC value and sends the instruction code required to execute the instruction [3] .

Input(s)	Output(s)
Instruction code [bit 15:12]	Control signals (mem_write, reg_write, is_net_op, direction)

1.4.5 Register File (RegFile)

The RegFile unit is used to solely access the secret key register (Key_Addr, 0x0000) for reading or writing. The key can be passed on to later stages of the processor or can be accessed when the original key is to be overwritten by an external key

Input(s)	Output(s)	Enabler
*Register to read from (KEY_ADDR)**Register to write to (KEY_ADDR)**New key data*	*Key data*	*reg_write* - HIGH when new key data is to be used.

1.4.6 Parity Unit

The Parity Unit retrieves the parity bit from the data received by the processor. This is done by performing an XOR operation between all data bits. See the code below for reference.

Input(s)	Output(s)
Data from Processor	Expected Parity Bit

```
n : integer := 8

signal temp: std_logic_vector(n-1 downto 0);
begin
    temp(0) <= data(0);

    -- XOR each bit together
    gen: for i in 1 to n-1 generate
        temp(i) <= temp(i-1) xor data(i);
    end generate;

    parity <= temp(n-1);
```

1.4.7 Tag Generator

The Tag Generator unit takes in the data from the user and produces a tag using the secret key that is stored in the secret key register. The overview of the tag generator operation is shown below:

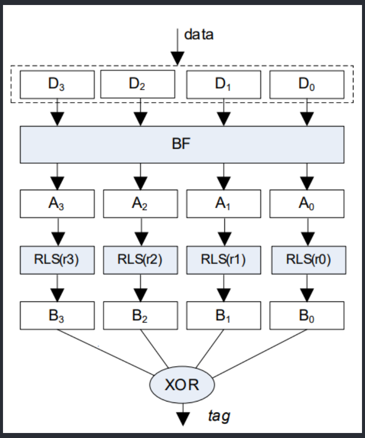


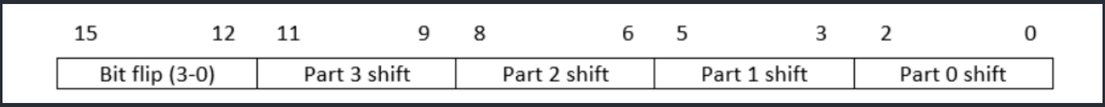
Figure 1.3 Tag Generation Flowchart

Input(s)	Output(s)
Data [31:0] Key data [15:0]	Tag [7:0]

1.4.7.1 Tag Generation Using the Secret Key

The Tag Generator is a key component in the authentication of data. An essential aspect of Tag Generation is the Secret Key, which encapsulates the tagging algorithm.

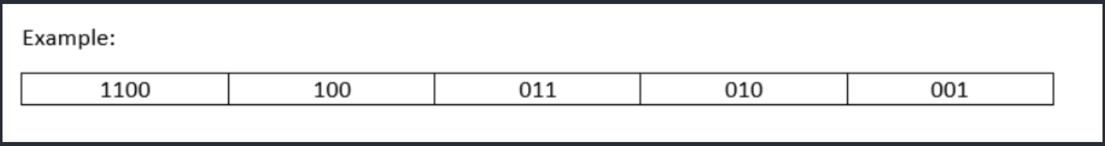
The Secret Key for the designed ASIP is of the fixed structure in Figure 1.4.



15 12	11 9	8 6	5 3	2 0
Bit flip (3-0)	Part 3 shift	Part 2 shift	Part 1 shift	Part 0 shift

Figure 1.4 Formulation of Secret Key

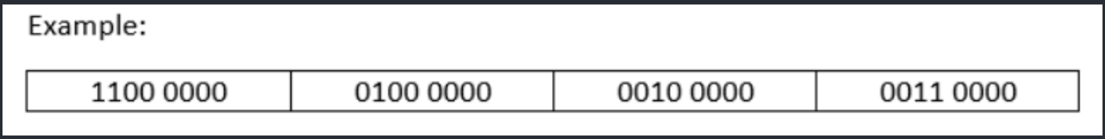
An example of secret key following this formula can be seen below:



Example:

1100	100	011	010	001

Incoming 32-bit data can be split evenly, as seen below into four segments for Tag Generation or Authentication.



Example:

1100 0000	0100 0000	0010 0000	0011 0000

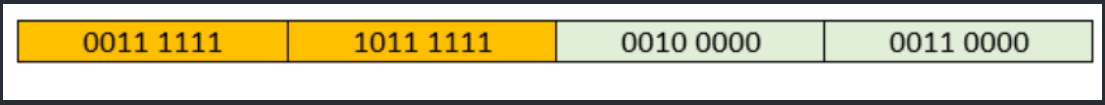
The above example will be referred to throughout the following explanations.

1.4.7.2 Bit Flip

The Bit Flip stage utilises the most significant four bits of the Secret Key, bits 15:12, or "1100" in the given example. Each bit corresponds to a segment.

If the bit is high ('1'), all bits in the segment are flipped, such that '1' becomes '0' and '0' becomes '1'. Conversely, if the corresponding bit is low ('0'), no bits in the byte segment are flipped (i.e. '1' will remain '1' and '0' will remain '0').

Examining the example above, we can see that the first two segments (from MSB to LSB) will be flipped, resulting in the data seen below:

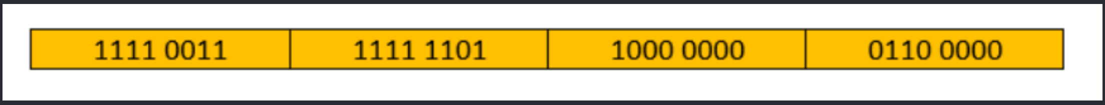


0011 1111	1011 1111	0010 0000	0011 0000

1.4.7.3 Rotate Left Shift

In this stage of the tag generation, a simple Rotate Left Shift component is used, with a variable roatation value. Each byte of the data will be individually rotated to the extent indicated by its corresponding rotation emount in the secret key.

In the example above, the first byte corresponds to **0b100**, or 4_{10}, meaning the first byte should be shifted 4 times. Note that this stage continues on from the Bit Flip stage, and hence the above example would be shifted to result in the following:



1111 0011	1111 1101	1000 0000	0110 0000

1.4.7.4 XOR

Following the left shits, the tag generation component XORs all four segments to form a final 1-byte tag to be used in authentication of the data.

For all bytes collectively, the XOR operation will essentially record whether the number of set bits in the 0th up to the 7th position is even or odd.

Referring to the above example, three bytes have their MSB set, meaning the resulting tag will have a '1' in the MSB position. As such, the tag will be **1110 1110**.

1.4.8 Data Memory

The data memory component is primarily used to store the Secret Key, which is stored at *Key_Addr*, 0x0000.

Hence, this key_addr is hardcoded into the address input of the data memory component. This means that any time a new key is stored, the existing key will be overwritten.

Input(s)	Output(s)	Enabler
Key_addr (via Data Address port) *New key data*	*Key data*	*Mem_write* : HIGH when new key data is to be stored

2 Instruction Set Architecture (ISA)

2.1 I/O Registers

Register	Definition
procData	32-bit bus carrying data from local processor Assumed possible errors Utlised during OP_SEND
netData	40-bit bus containing network data [31:0] tag [39:32] Data sent from network and not necessarily secure Utilised during OP_Receive
procParity	Processor-sourced data parity Utilised during OP_SEND
extKey	16-bit secret key received from an external source Utilised during OP_EXTLOAD
procOut	32-bits data received from the network output to local processor if secure 0 if data has been tampered or data not OP_RECEIVE instruction
netOut	Data and generated tag concatenated (Tag39:32Data31:0) 0 if parity error or not OP_SEND instruction

2.2 Flags

Signal	Definition
Error	Flag set by ASIP if data has been tampered or corrupted If set, data will not be sent to processor or network (direction • TagErr) • (!direction • P_Err)
ASIP_ready	Indicates that ASIP is ready to receive and process instructions
key_load	External input fed into control system by local user, set if loading an external key
network_ready	Input signal from network to indicate it is ready to receive data from the local processor
send_request	Input signal to indicate the local processor would like to send data to the network
receive_request	Signal to indicate a processor over the network would like to send data to the local processor
loadkey_request	Signal from local processor user to indicate the user would like to load the secret key that is currently stored in Data Memory, into the ASIP for tag generation and authentication
proc_data_present	Set if there is data to send to processor, once the data has been authenticated by the ASIP
net_data_present	Set if there is data to send to network, once the data has been authenticated by the ASIP

2.3 PC Control Summary

The PC Control System implements the following truth table, which can be seen summarised as a flow chart in Figure 2.1.

ASIP_ready	network_ready	loadkey_request	keyload	receive_request	send_request	Next Instruction
0	-	-	-	-	-	PC_NOP
-	0	0	-	0	1	PC_NOP
1	-	1	-	-	-	PC_LOADKEY
1	-	0	1	-	-	PC_LOADEXT
1	1	0	0	1	-	PC_RECEIVE
1	0	0	0	1	1	PC_SEND
1	-	0	0	0	1	PC_SEND
-	-	0	0	0	0	PC_NOP

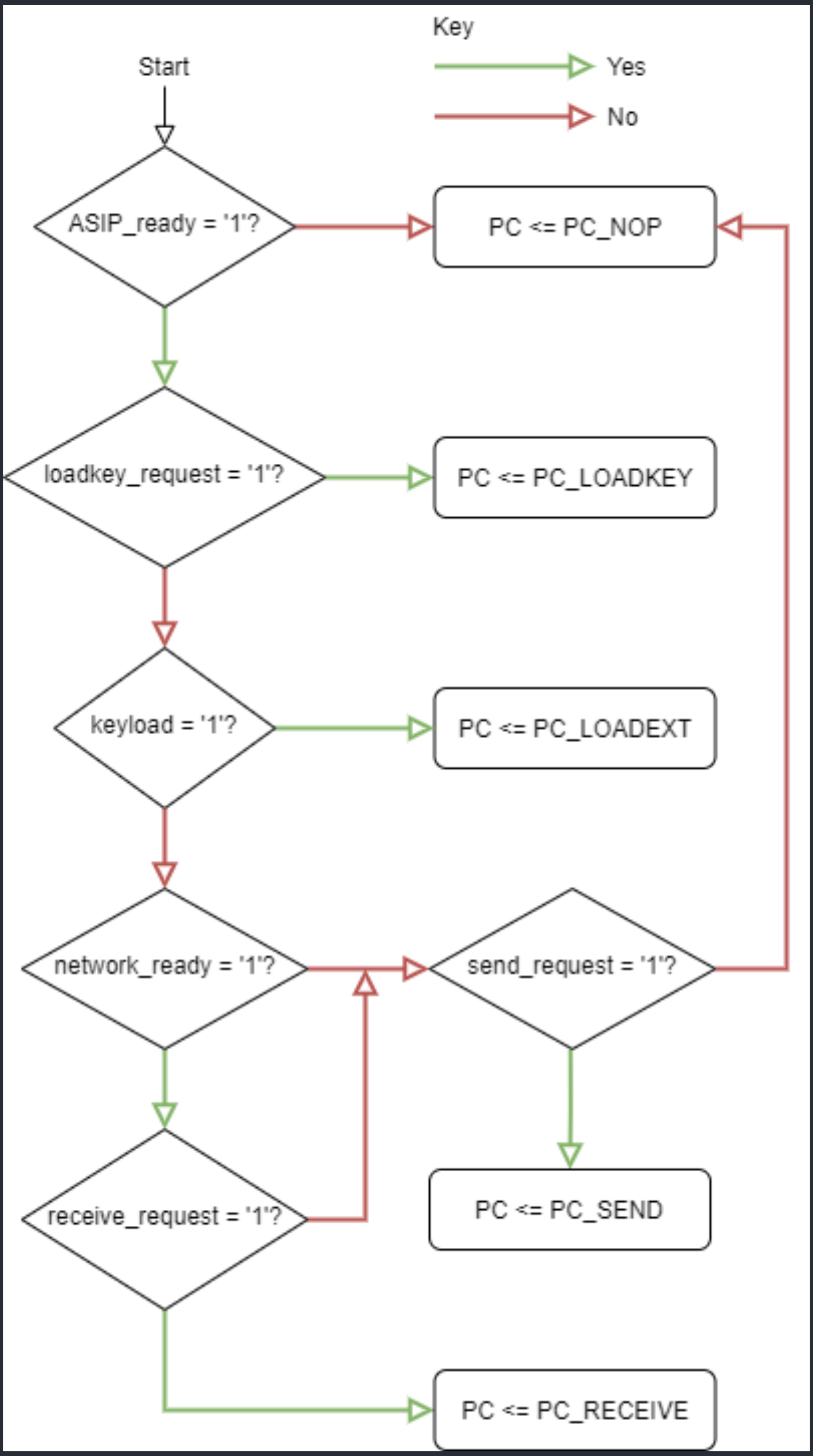


Figure 2.1 PC Control Logic

As seen in the truth table above priority of instructions is given as:

1. OP_LOADKEY
2. OP_LOADEXT
3. OP_RECEIVE
4. OP_SEND

Instructions of higher priority will be chosen to run in place of instructions of lower priority in cases where multiple, valid requests are received simultaneously.

Ultimately, if the ASIP is not ready, as indicated by the ASIP_ready flag, the ASIP will perform a NOP (no-op), and will not process the next request.

If the network is not ready, the ASIP may complete all instructions except for send requests, which require their output to be sent to the network.

On each new clock cycle, the PC Control unit will analyse the incoming signals and flags. Depending on the flags set, as above, the PC mux will be used to set a new PC address, in accordance with the next instruction to be run.

This instruction will be obtained from instruction memory, which is shown in the excerpt below:

```
if (addr_in == 1) insn_out = 0x1000 // OP_RECEIVE
   else if (addr_in == 2) insn_out = 0x2000 // OP_SEND
   else if (addr_in == 3) insn_out = 0x4000 // OP_LOADKEY
   else if (addr_in == 4) insn_out = 0x8000 // OP_LOADEXT
   else insn_out = 0x0000 // OP_NOP
```

To account for the 1 cycle delay between updating the PC and retrieving the instruction, a buffer is utilised to store incoming data, which may otherwise be lost.

2.4 Control Signals

Control Signal	Enabled	
mem_write	Data memory write enable If enabled, the original key data will be overwritten with the external key data	
reg_write	Register write enable If enabled, secret key in register will be overwritten	
is_net_op	Indicates whether the operation being performed is a network operation, i.e. send or receive data HIGH if a network operation	
direction	Direction of data passage: from network to local processor or vice versa LOW if data passage is from local processor to network HIGH if data passage is from network to local processor	

2.5 Instruction Format

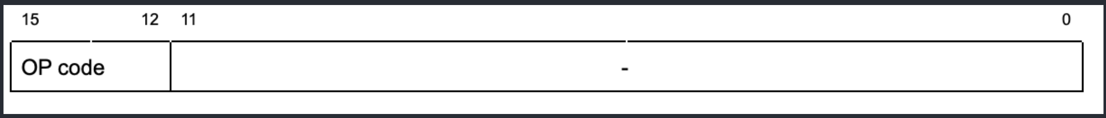


Figure 2.2 Formulation of ASIP Instruction

As seen in Figure 2.2, instructions consist of a three-bit OP Code. The special instructions from the processor can be viewed below.

3 Instruction Set Overview

3.1 OP_LOADKEY

Description

Loads key from data memory, into secret key register, overwriting previous data.

Operation

KeyReg ← Mem[key_address]

Operands

none

Instruction Memory Address

0x3

4-bit OpCode

0100

Input Flags

ASIP_ready	network_ready	loadkey_request	keyload	receive_request	send_request
1	-	1	-	-	-

Control Signals

mem_write	reg_write	is_net_op	direction
0	1	0	-

Output Flags

Error	P_Data_present	Net_data_present
-	-	-

3.2 OP_LOADEXT

Description

Loads an external key into data memory, overwriting any existing key. This does not automatically update the special key register. To do this, the user must run the **OP_LOADKEY** instruction.

Operation

Mem[key_address] ← extKey

Operands

extKey - new key value

Instruction Memory Address

0x4

4-bit OpCode

1000

Input Flags

ASIP_ready	network_ready	loadkey_request	keyload	receive_request	send_request
1	-	0	1	-	-

Control Signals

mem_write	reg_write	is_net_op	direction
1	0	0	-

Output Flags

Error	P_Data_present	Net_data_present
-	-	-

3.3 OP_SEND

Description

Sends data from the local processor to a network-connected processor.

Operation

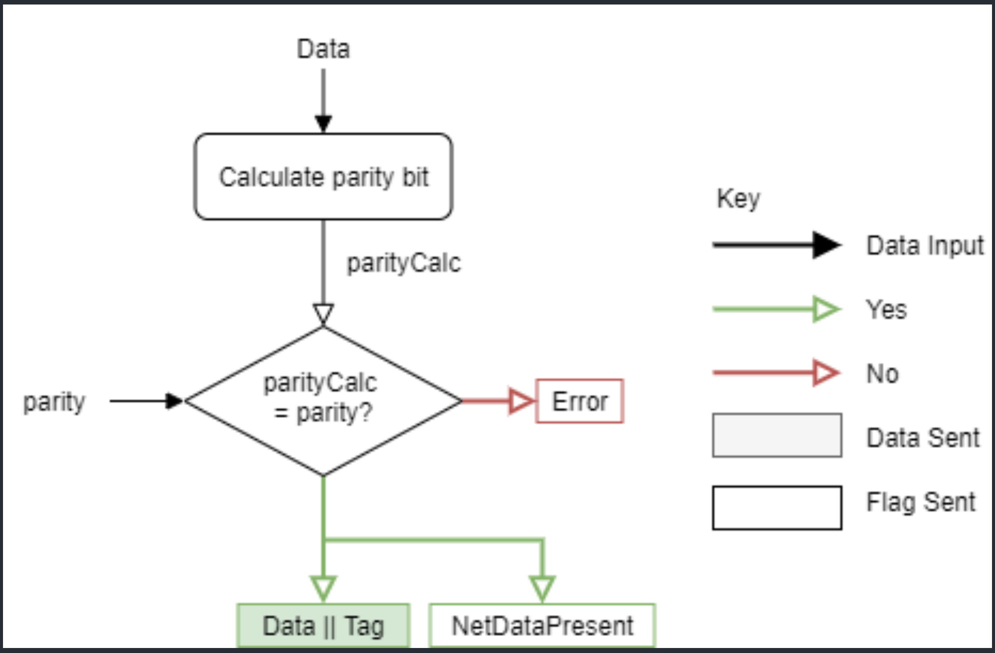


Figure 3.1 OP_SEND Operation Logic

Operands

procData

Instruction Memory Address

0x2

4-bit OpCode

0010

Input Flags

ASIP_ready	network_ready	loadkey_request	keyload	receive_request	send_request
1	-	0	0	0	1

Control Signals

mem_write	reg_write	is_net_op	direction
0	0	1	0

Output Flags

Error	P_Data_present	Net_data_present
<=>	0	1

3.4 OP_RECEIVE

Description

Authenticates data sourced from a network-connected processor to send to the local processor, if secure.

Operation

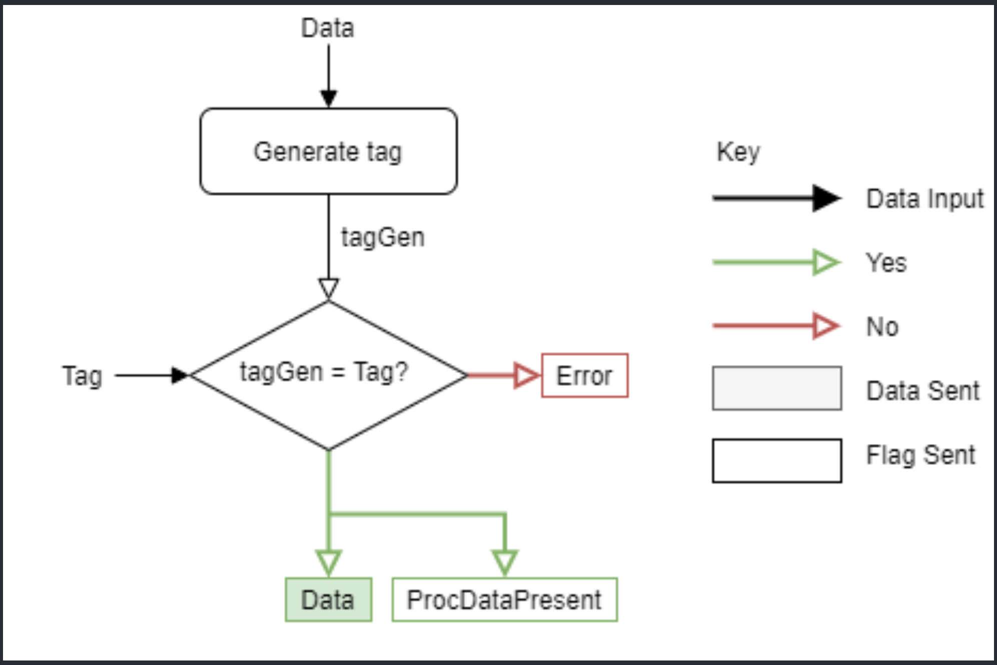


Figure 3.2 OP_Receive Operation Logic

Operands

netData

Instruction Memory Address

0x1

4-bit OpCode

0001

Input Flags

ASIP_ready	network_ready	loadkey_request	keyload	receive_request	send_request
1	-	0	0	0	1

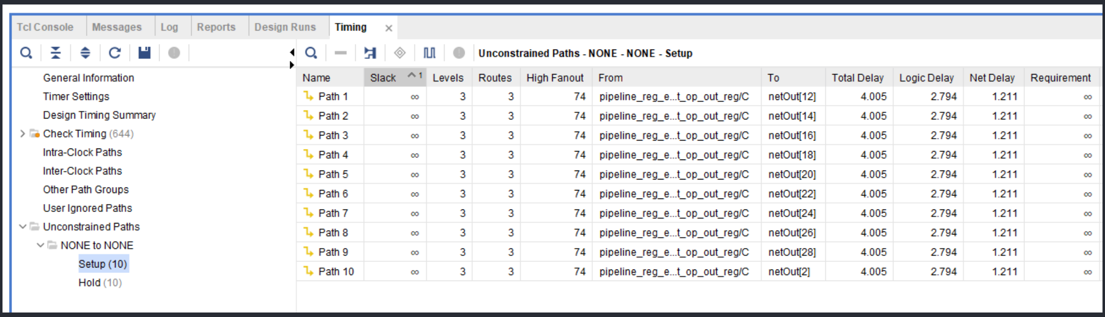
Control Signals

mem_write	reg_write	is_net_op	direction
1	0	1	1

Output Flags

Error	P_Data_present	Net_data_present
<=>	1	0

4 Performance


Tcl Console | Messages | Log | Reports | Design Runs | Timing

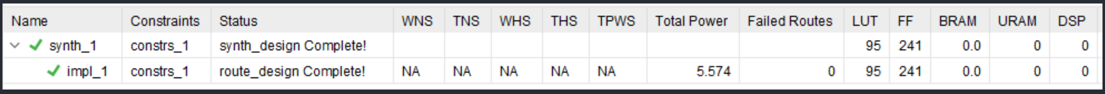
General Information
Timer Settings
Design Timing Summary
Check Timing (644)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths
NONE to NONE
Setup (10)
Hold (10)

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[12]	4.005	2.794	1.211	∞
Path 2	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[14]	4.005	2.794	1.211	∞
Path 3	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[16]	4.005	2.794	1.211	∞
Path 4	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[18]	4.005	2.794	1.211	∞
Path 5	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[20]	4.005	2.794	1.211	∞
Path 6	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[22]	4.005	2.794	1.211	∞
Path 7	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[24]	4.005	2.794	1.211	∞
Path 8	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[26]	4.005	2.794	1.211	∞
Path 9	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[28]	4.005	2.794	1.211	∞
Path 10	∞	3	3	74	pipeline_reg_e...t_op_out_reg/C	netOut[2]	4.005	2.794	1.211	∞

Figure 4.1 Performance Table

The ASIP has a critical path delay of 4.005ns. This translates to an approximate processor speed of 250 MHz.



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!								95	241	0.0	0	0
✓ impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	5.574	0	95	241	0.0	0	0

Figure 4.2 Synthesis Table

The design had a final count of 95 Look Up Tables (LUTs) and a total of 241 Flip Flops (FFs) for its hardware implementation

1. Refer to section 2.3 for further details ↩
2. Refer to section 2.5 for instruction format ↩
3. Refer to section 2.4 for control signals ↩