ASIP Report

COMP3211 2021 Session One

W12A - Group One

z5216632 - Gabriel Tay Wei Chern
z5209390 - Malavika Pasupati
z5205690 - Quynh Phan
z5232920 - Lindsay Small
z5206677 - Andrew Wong

Project Discussion

Tag Generation

The tag generation algorithm is hardwired into the ASIP design as it is a single, fixed component in the EX stage.

In our project, we have a means to alter the secret key via the PC_LOADEXT command that will take in the new 8 bit secret key from the input and save it into the data memory at the address: `key_addr <= 0x0000`.

The secret key is then loaded into the fixed register that will be then used for the tag generation as follows.

It is also notable that although the secret key is alterable, this ASIP can only be used when data is encrypted in the manner as defined.

By design, there is no instruction to retrieve the secret key, providing an increased security measure to prevent potential attackers from obtaining the secret key used in the tag generation.

Secret Key Loading

Under the current design, the user is assumed to have loaded the secret key from Data Memory before completing any send or receive instructions. That is, the user is assumed to have executed `OP_LOADKEY` prior to executing `OP_SEND` or `OP_RECEIVE`.

If `OP_LOADKEY` is not executed, the subsequent send and receive instructions will assume a secret of `0x0000`.

Externally Provided Secret Key

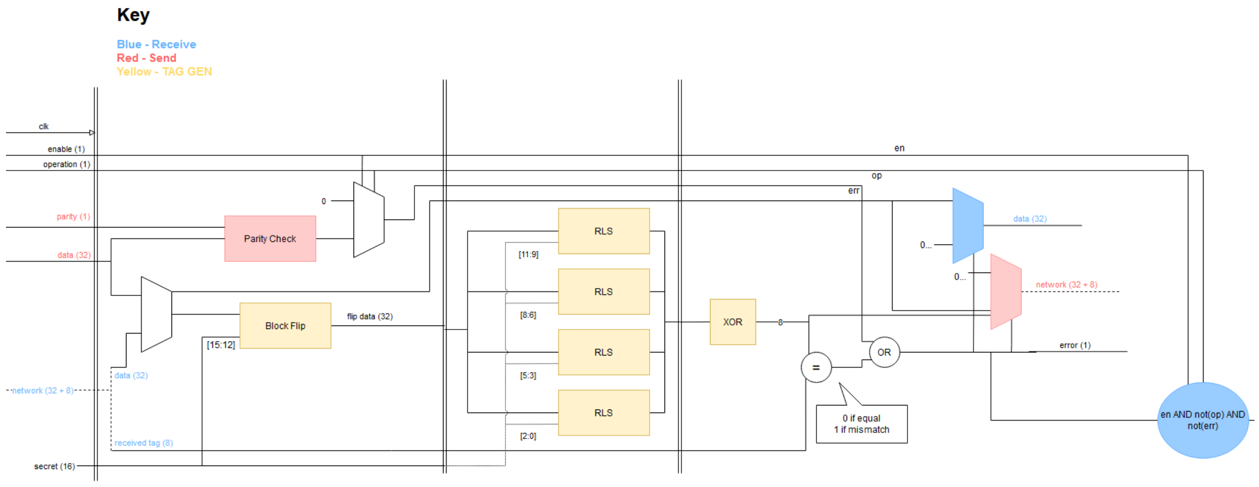
It was suggested that the network ASIP core should allow for a secret key to be provided from an external source (i.e. the main CPU). Originally an instruction (`OP_LOADEXT`) was added into the ISA that would set the secret key register to the value of an external port; however after discussion the instruction was modified to set the secret key register to the value of the data memory. This change simplified the hardware design whilst reducing the overlap of instruction functionality. The modified instruction design also increased modularity, allowing possible future instructions to operate on data that was loaded from the external port.

Previous: `OP_LOADKEY` -> `secretKey = data#0`, `OP_LOADEXT` -> `secretKey = extPort`
Current: `OP_LOADKEY` -> `secretKey = data#0`, `OP_LOADEXT` -> `data#0 = extPort`

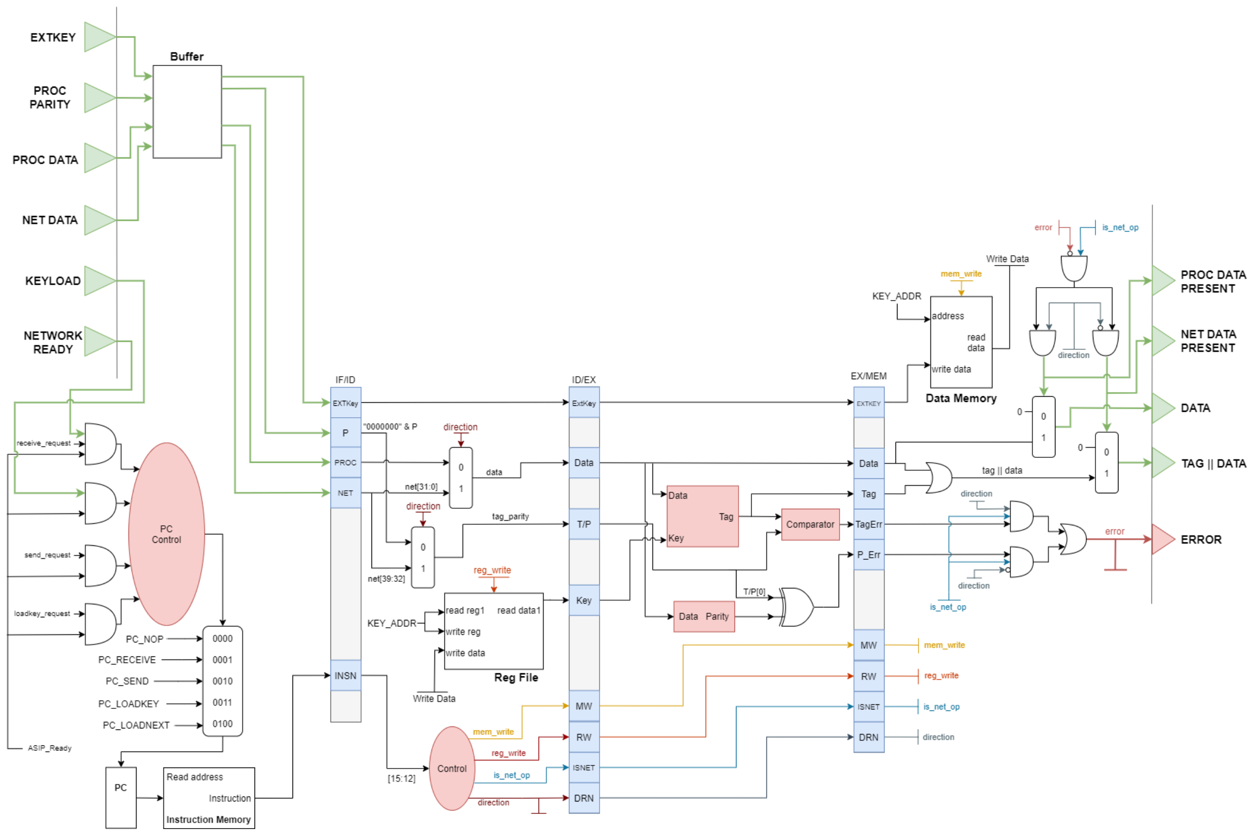
Scope, Revisions, Changes

We (as well as members of other project groups) felt that the assignment specification was unclear and confusing in areas. Consequently, sufficient understanding of the project scope was only achieved late into project development.

As a result of the specification clarifications, the ISA had to be redesigned. This cascaded a series of other changes, including having to redesign the hardware layout and block diagram - delaying and blocking other requisite tasks that needed the ISA and design to be finalised.



Original Design - resembled more of an ASIC than of an ASIP



Final Revised Design

Improvements

Tight Hardware Function Coupling

The hardware implemented in the processor is tightly designed with the ISA in mind, causing tight coupling of its functionality.

For example, the generated tag nor parity bit cannot be stored into data memory or instruction memory. Furthermore, in order to change the tag generation process, such as if tag generation involved byte rotations being completed first, followed by the bit-flipping process, the hardware of the ASIP would need to be altered.

This design choice was however chosen to aid in the simplicity and performance of the processor. To decouple the hardware, more signals and components would need to be implemented to cater for data addressing and redirection, which would require more work to redesign the pipelined processor. Such a redesign would allow just the firmware (i.e. instruction memory) to be modified, allowing the hardware to be reused as-is.

Key Loading Enforcement

As a result of the design choices for the ASIP, the processor will not automatically initialise itself with the secret key (i.e. during reset / boot). Instead, this instruction (`OP_LOADKEY`) must be manually issued; failure to do so will result in the secret key being zero (`0x0000`).

To fix this issue, the PC subsystem of the processor should be modified to run a series of instructions rather than being statically controlled by the incoming instruction control signals.

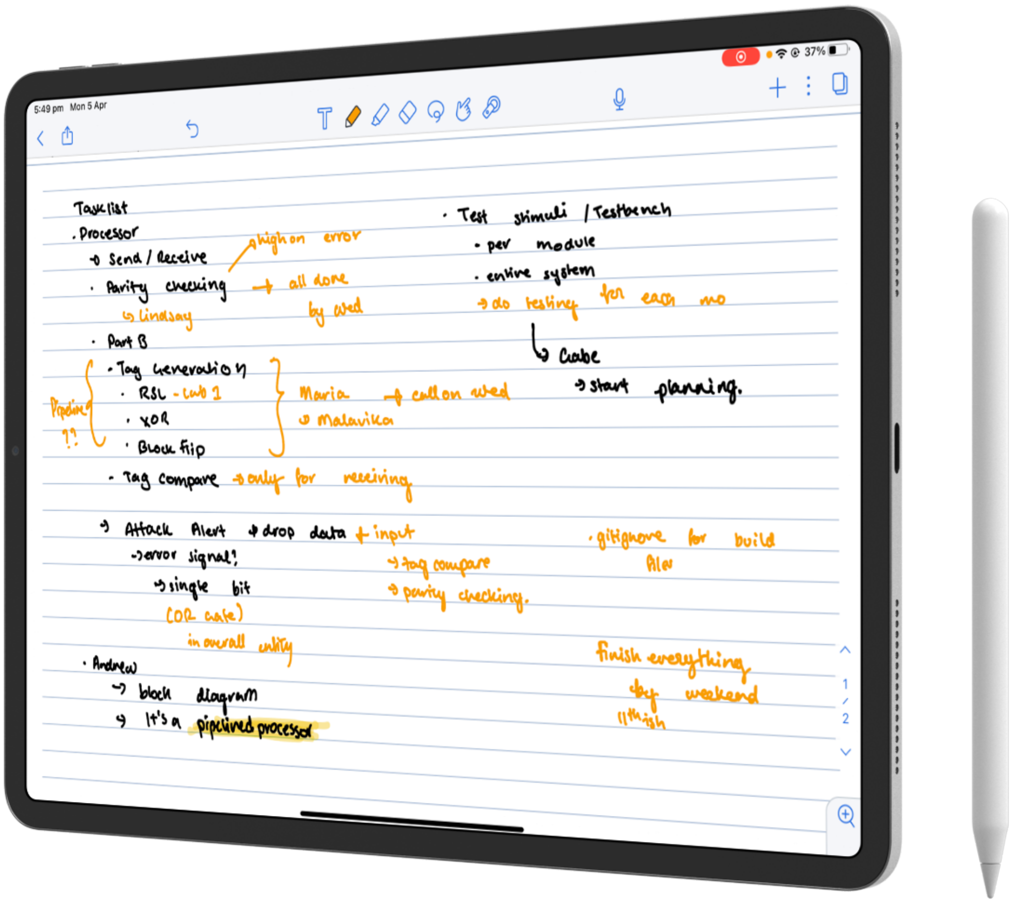
Performance

The network ASIP has a maximum clock speed of 250 MHz. Whilst reportedly fast compared to other observed ASIP designs, improvements can be made to increase the maximum clock speed - bound by the long execution stage delay. The current implementation of tag generation involves all of its processes being completed in the same clock cycle - becoming the critical path of delay.

To reduce this delay and hence increase the maximum clock speed, the processes involved in tag generation could be split into multiple stages which would therefore decrease the critical path delay. Such a change would provide overall speed benefits - especially for instructions that do not require tag generation process, but will also allow for the individual stages of tag generation to be used for other future instructions.

For example, the XOR component required in tag generation could be used for a hypothetical equality comparator instruction.

Project Development



5:49 pm Mon 5 Apr

Tasklist

. Processor

→ Send / Receive

· Parity checking → all done by wed → high on error

↳ Lindsay

· Part B

Pipeline ?? · Tag Generation

· RSL - cwb 1

· XOR

· Block flip

Maria → call on wed

↳ Malavika

- Tag compare → only for receiving

→ Attack Alert + drop data + input

→ error signal?

→ single bit

(OR gate)

in overall entity

→ tag compare

→ parity checking.

· Andrew

→ block diagram

→ It's a pipelined processor

· Test stimuli / Testbench

· per module

· entire system

→ do testing for each mo

↳ Gabe

→ start planning.

· gitignore for build files

finish everything by weekend 11th ish

Required Tasks

Tasks were organised into four categories: design, implementation, testing and documentation.

- Design
  - Create ISA
  - Create block diagram
- Implementation
  - Create Tag Generation component
    - Create Block Bit Flip component
    - Create Rotate Left Shift
    - Write XOR logic
  - Create Parity Generation component
  - Create core to model the block diagram
- Testing
  - Create component test benches
  - Create integration test benches

- Documentation
    - Create slide deck presentation
    - Write ISA Manual
    - Write Report

Project Schedule / History

Week 5
15/03 - Group project specifications released

Week 6
23/03 - Initial group meeting

Week 7
30/03 - Tuesday Standup
03/04 - Saturday Standup (rescheduled because of Good Friday)
Deliverables: ISA,Block Diagram

Week 8
06/04 - Tuesday Standup
09/04 - Friday Standup
Deliverables: Parity Generation, Tag Generation

Week 9
13/04 - Tuesday Standup
16/04 - Friday Standup
Deliverables: Core, Testbenches

Week 10
20/04 - Tuesday Standup
21/04 - Presentation Meeting
23/04 - Project Submission
Deliverables: Presentation, Manual, Report

Roles

Tasks were evenly assigned to team members to ensure a fair contribution of each member. Additionally, each member was involved in all processes / task categories, to ensure that each member had the chance to code as well as to contribute to documentation.

For each task category, certain members were assigned as 'task leaders' - where they would be responsible for the overall direction of a certain task. This did not eliminate differing opinions, but rather mitigated stale decision outcomes.

Design Task Leader - Andrew
Implementation Task Leader - Lindsay
Testing Task Leader - Gabriel
Documentation Task Leader - Quynh, Malavika

Testing

Testing is a critical part of the project in ensuring that the components within the project are working as prescribed.
In order to ensure that all things were working smoothly in a categorically sequential progression, we decided that we will test all the individual components and move upwards towards the greater complexity of combined components.

The testings sequence was tested in the following order as described below:

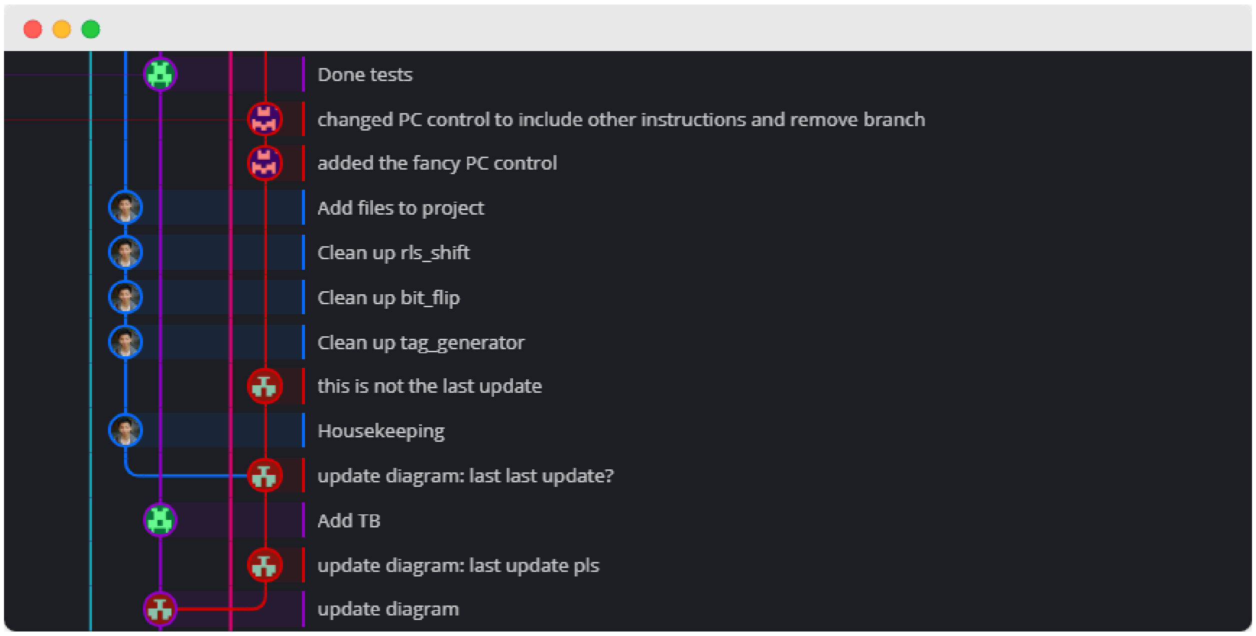
The testing of the project involved testing the individual components of the project:
Parity Testing
Block Flip Testing
Rotate Left Shift Testing
Tag Generation Combined Testing

This is continued with the combined overall components of the ASIP:
Send Testing
Secret Testing
Parity soft error Testing
Receiving tampered data Testing

Project Management Strategies

Vivado (VHDL) Project Version Control

A private Git repository was created to facilitate the sharing and tracking of VHDL code changes.
This allowed multiple members to simultaneously work on different files of the project.



Done tests
changed PC control to include other instructions and remove branch
added the fancy PC control
Add files to project
Clean up rls_shift
Clean up bit_flip
Clean up tag_generator
this is not the last update
Housekeeping
update diagram: last last update?
Add TB
update diagram: last update pls
update diagram

Meetings

Given the current nature of distance and remote learning, bi-weekly standup meetings were held online over Microsoft Teams, occuring on Tuesdays and Fridays. These standup meetings sought to outline completed and upcoming work, as to keep everyone informed of the project's current status.

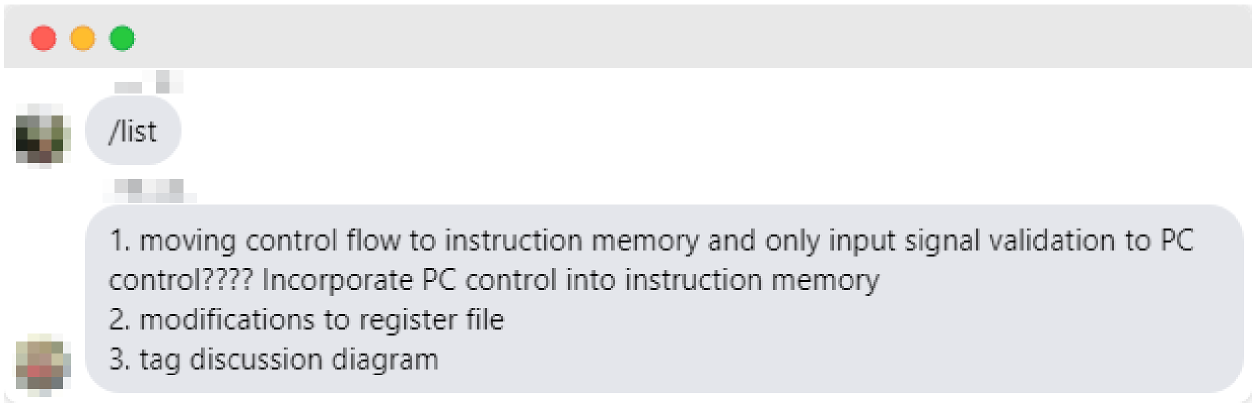
Due to work and external commitments, team members were occasionally unable to join the standup meetings - however through the Git logs as well as team communication they were able to be brought up to date.

Chat Group

An online chat group was established to allow members to communicate with each other quickly when required. Impromptu meetings could also be held when required

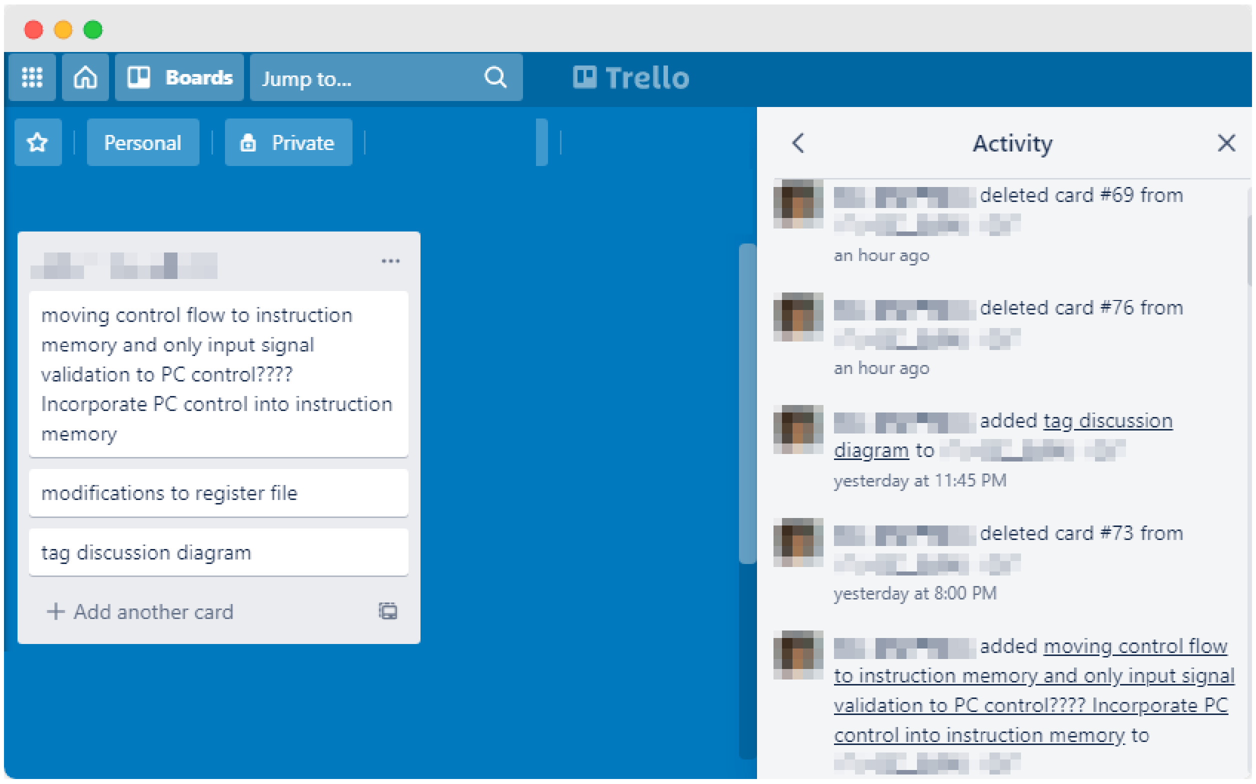
Scratchpad

A scratchpad was created to allow members to share data and notes between each other - such as spontaneous questions, change requests, important notices. This scratch pad was accessible via Trello, as well as through the online chat group.



/list

1. moving control flow to instruction memory and only input signal validation to PC control???? Incorporate PC control into instruction memory
2. modifications to register file
3. tag discussion diagram



moving control flow to instruction memory and only input signal validation to PC control???? Incorporate PC control into instruction memory

modifications to register file

tag discussion diagram

+ Add another card

deleted card #69 from
an hour ago

deleted card #76 from
an hour ago

added tag discussion diagram to
yesterday at 11:45 PM

deleted card #73 from
yesterday at 8:00 PM

added moving control flow to instruction memory and only input signal validation to PC control???? Incorporate PC control into instruction memory to