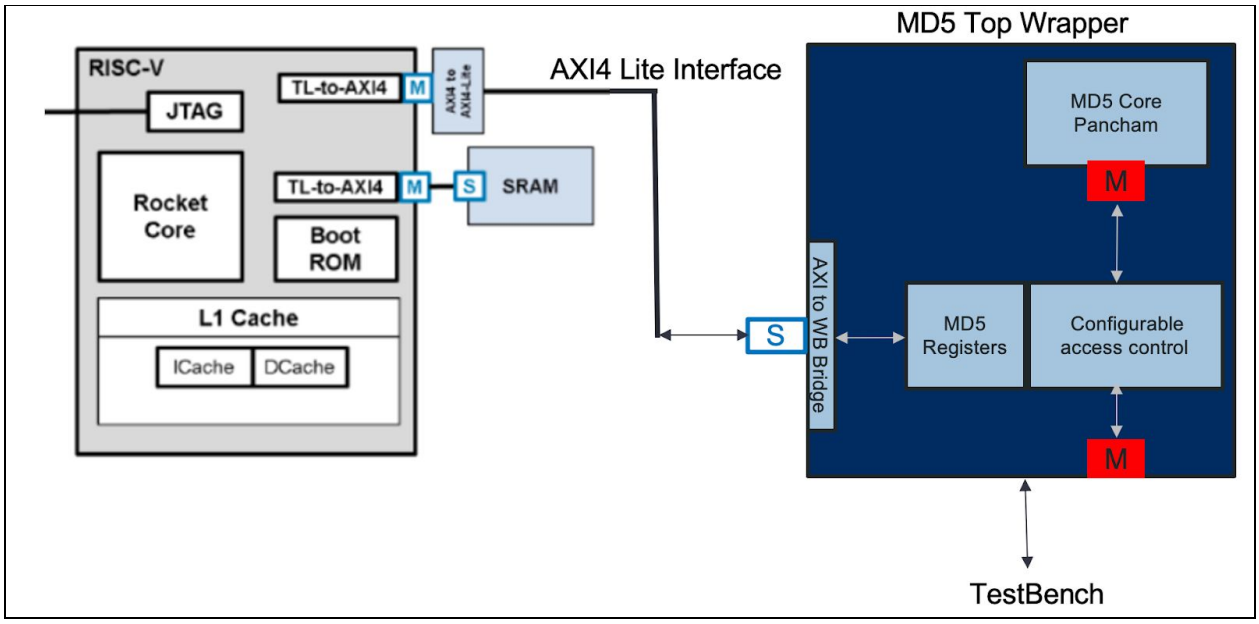
CSE 237D: SoC Security

Final Project Report



Mahima Rathore
Richa Pallavi

Abstract:

Most embedded and mobile computing devices are architected around one or more System-on-Chip (SoC) designs. The SoC architecture involves coordination and communication of a number of pre-designed hardware blocks or cores outsourced by multiple vendors. It could be quite challenging to ensure that these cores are securely initialized, configured, and connected. Also these cores need monitoring to avoid unauthorized access to SoC's confidential data by these cores or their parent vendors. This introduces the concern of security threat in most of the SoCs today.

In this project, we address this critical issue, by extending the SoC architecture to include a secure interface to interact with these cores. Our architecture uses AMBA AXI machines for securing the interface between the SoC processor and the Hardware accelerator cores. We aim at integrating these cores with Master AXI machines to independently access the shared memory of the SoC, at the same time being controlled by the processor through an AXI-lite slave interface. Our design with AXI machines will evaluate and secure SoC against any unauthorized access by these cores. It will monitor and control all transactions between these third party cores and SoC and disable it if needed. Our design serves as a critical stepping stone for an ongoing project for SoC security.

Introduction:

A system on a chip (SoC) is an integrated circuit that integrates all or most components of a computer or other electronic system. These components include a central processing unit (CPU), memory, input/output ports and secondary storage – all on a single substrate or microchip, the size of a coin. It also contains digital, analog, mixed-signal, and often radio frequency signal processing functions. As they are integrated on a single substrate, SoCs consume much less power and take up much less area than multi-chip designs with equivalent functionality. Because of this, SoCs are very common in the mobile computing (such as in smartphones) and edge computing markets.

Recent years have seen rapid proliferation of embedded and mobile computing devices. Such devices come in a variety of form factors, including smartphones, tablets, automotive controls, wearables, medical and fashionable implants, and smart sensors. Given their diversity and personalization, security has emerged as a critical concern for them. Most of these devices contain confidential assets, which must be protected against unauthorized access. Examples of secure or sensitive assets present in virtually all modern computing systems include cryptographic and DRM keys, premium content, firmware, programmable fuses, and personal end-user information. Unauthorized or malicious access to these assets can result in leakage of company trade secrets for device manufacturers or content providers, identity theft for end users, and even destruction of human life. Consequently, it is vital to ensure that secure assets in computing devices are adequately protected.

Most SoCs, from embedded and mobile computing devices, have some fixed functional hardware accelerators coming from multiple different vendors. These are specialised hardware units, meant to carry out a particular task faster than is possible in software running on a general-purpose CPU. These units or cores are known to improve the performance and energy efficiency of a system. However, integrating, configuring and connecting hardware accelerators in an SoC environment can be quite challenging, as it requires additional design, verification and implementation effort. Also these cores shouldn't have access to any confidential or restricted data within the SoC, else it could be manipulated by their parent vendors and this could result in the above mentioned deadly consequences.

In this project, we propose a general architecture which addresses this critical issue. Our project focuses on extending the SoC architecture to include a secure interface to interact and monitor these cores. This architecture uses AMBA AXI machines for securing the interface between the SoC processor and the Hardware accelerator cores. The Advanced eXtensible Interface (AXI), is a part of the ARM Advanced Microcontroller Bus Architecture (AMBA) specifications. It is a handshake mechanism where data can be transferred between Masters and slaves through an interconnect. We integrated these cores with Master AXI machines for independent access to the shared memory of the SoC. At the same time these cores are being controlled by the processor through an AXI-lite slave interface. Our design with AXI machines evaluates and secures SoC against any unauthorized access by these cores. It will monitor and control all transactions between these third party cores and SoC and disable it if needed.

Technical Material:

SoC: The CEP architecture

For this project, CEP is our SoC. The Common Evaluation Platform (or CEP) is intended as a surrogate System on a Chip (SoC) allowing users to test a variety of tools and techniques. This platform was developed by Lincoln Laboratory, Massachusetts Institute of Technology.

The Common Evaluation Platform (CEP) was developed to enable and facilitate the evaluation of various integrated circuit (IC) security-enhancing design and fabrication techniques across a variety of DoD sponsored research and development programs. The CEP is a mission-relevant and license-unencumbered System on Chip (SoC) design with representative scale and features such that it can serve as a surrogate for trusted US Government designs. The CEP is an entirely open-source benchmark design that features:

- Scale: Sufficient SoC complexity to stress and challenge defensive design techniques
- Diversity: DoD Mission-relevant surrogate modules that offer diversity of digital computation functions.
- Releasability: Open-source license compatibility permits free distribution to any performer seeking to evaluate a defensive technique.
- Extensibility: Modular approach to design that offers easy adaptation to meet emerging and future evaluation objectives

This platform uses RISC-V processors and consists of various hardware accelerators for common DoD functions including DSP and secure communications.

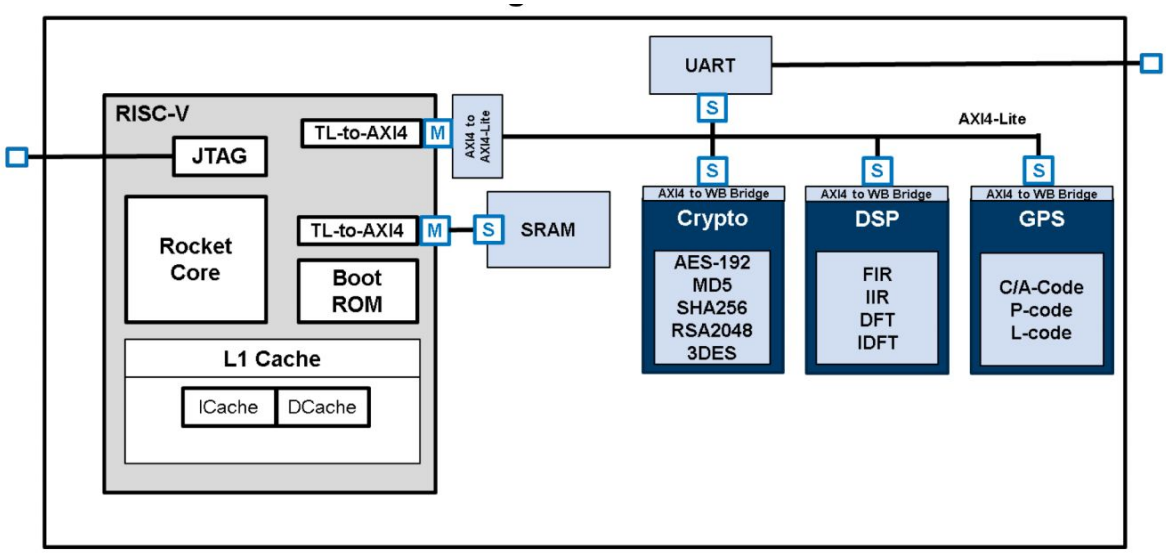


Figure 1: CEP 2.0 High-level Architecture
Image courtesy: CEP, Lincoln Laboratory, Massachusetts Institute of Technology

In summary, the CEP is a DoD-relevant surrogate SoC for IC security technology assessments, and the SETs within the CEP serve as the basis for evaluating the performance and efficacy of those security enhancing technologies.

Interface for Extension: AMBA AXI

The Advanced eXtensible Interface (AXI) is a part of the ARM Advanced Microcontroller Bus Architecture (AMBA) specifications. It is a parallel high-performance, synchronous, high-frequency, multi-master, multi-slave communication interface, mainly designed for on-chip communication. The ARM Advanced Microcontroller Bus Architecture (AMBA) is an open-standard, on-chip interconnect specification for the connection and management of functional blocks in system-on-a-chip (SoC) designs. It facilitates development of multi-processor designs with large numbers of controllers and components with a bus architecture.

In simpler terms, AXI is a handshake mechanism where data can be transferred between Masters and slaves. A typical system consists of a number of master and slave devices connected together through some form of interconnect, as Figure 2 shows. This protocol is burst-based and defines the five independent transaction channels namely read address, read data, write address, write data and write response. An address channel carries control information that describes the nature of the data to be transferred. The data is transferred between master and slave using either:

- A write data channel to transfer data from the master to the slave. In a write transaction, the slave uses the write response channel to signal the completion of the transfer to the master.
- A read data channel to transfer data from the slave to the master.

Each of the independent channels consists of a set of information signals and VALID and READY signals that provide a two-way handshake mechanism. The information source uses the VALID signal to show when valid address, data or control information is available on the channel. The destination uses the READY signal to show when it can accept the information. Both the read data channel and the write data channel also include a LAST signal to indicate the transfer of the final data item in a transaction.

The AXI protocol:

- permits address information to be issued ahead of the actual data transfer
- supports multiple outstanding transactions
- supports out-of-order completion of transactions.

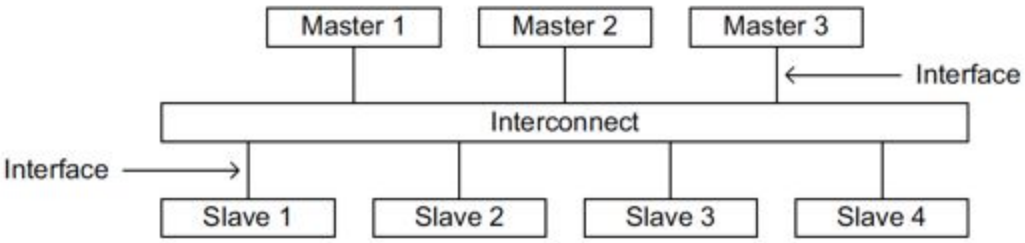


Figure 2: AXI Interface and Interconnect

Project Approach and Solution

CEP architecture itself defines just a simple bus architecture for the hardware accelerators, in which the hardware accelerators do not have direct access to memory, instead, the processors are in charge of moving the data for them (hardware accelerators are slaves on the bus). In order to control and monitor these hardware accelerator cores, AXI machines are added to the architecture.

The steps for extending the architecture are:

- Extend one hardware accelerator (MD5) with a high-performance full AXI master Interface. This mainly means developing an AXI machine with burst capabilities and integrating the machine with the code of each hardware accelerator in the system.
    - This involved developing an AXI Master design.
- The processor on the SoC controls and configures the MD5 by writing it's internal registers through the AXI Lite Slave interface.

- This required developing registers internal to MD5 and a register access mechanism by the AXI4 lite interface.
    - AXI4 lite Slave design also had to be developed for this step.
- The control registers written by slave interfaces then used to configure the AXI Master.
    - One register is used to activate or initiate the Master
    - The other two are used to set up the source & destination addresses, which is used by the master for reading and writing the shared memory.
- AXI Master then reads data from the source address (value given by register 1) of the shared memory (testbench) & writes it into an internal buffer of the hardware accelerator (MD5) top wrapper, which is input to the MD5 core called pancham.
    - Pancham uses a wishbone interface to communicate with the AXI Master.
    - It processes the data sent by the master and outputs it into an output buffer of the hardware accelerator (MD5) top wrapper.
- AXI Master then writes this data from the output buffer into the shared memory (testbench) at the destination address (value given by register 2).

This step by step approach as shown by Figure 3, will result in an extended hardware accelerator which is configured (by the processor) using the AXI interface and performs reads and writes also using an AXI interface. This allows us to set boundaries in the memory access on each hardware accelerator. We want the boundaries to be customizable, so we would extend the AXI Master with a configuration port to be customizable by the processor. This is being implemented as a part of this ongoing project, not included in this project report.

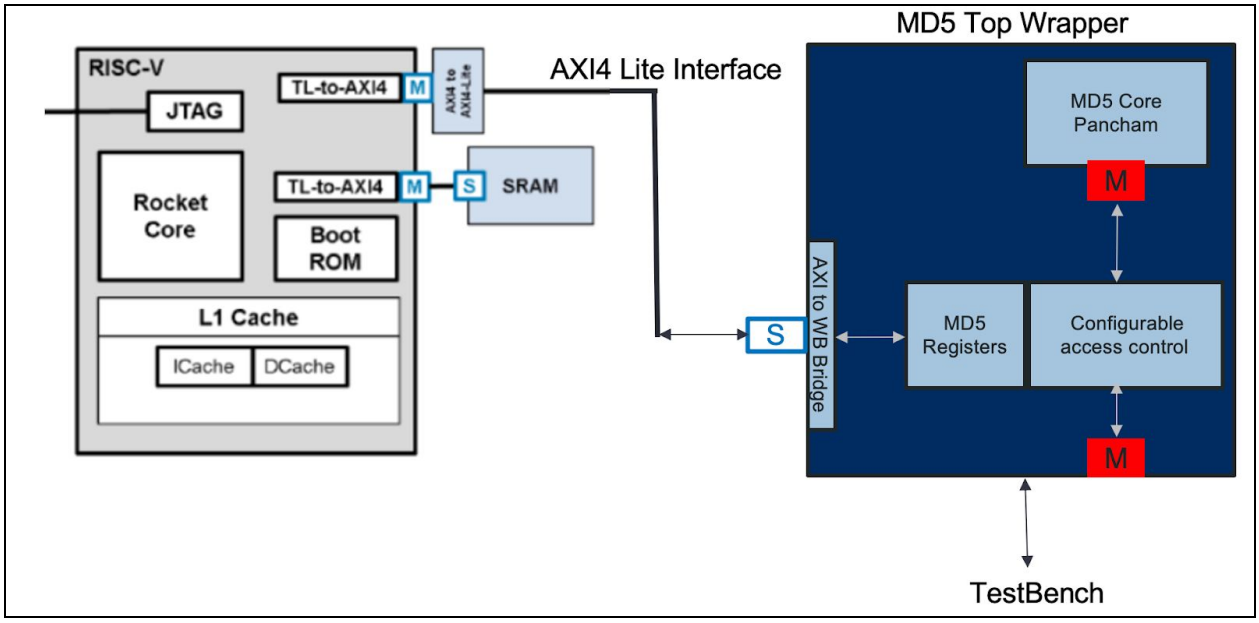
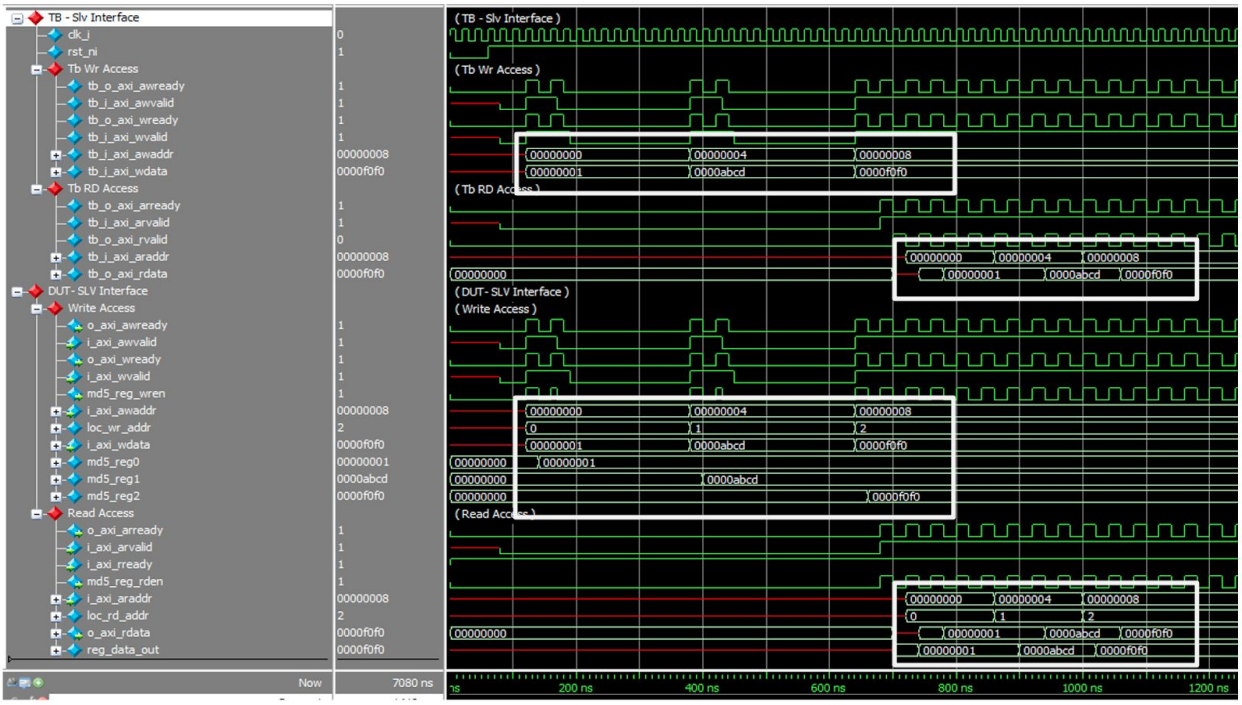


Figure 3: CEP Extended Architecture

Results:

The following waveforms are used to demonstrate the results achieved:

1. ***Successful register access by the processor using AXI4 lite slave interface.***

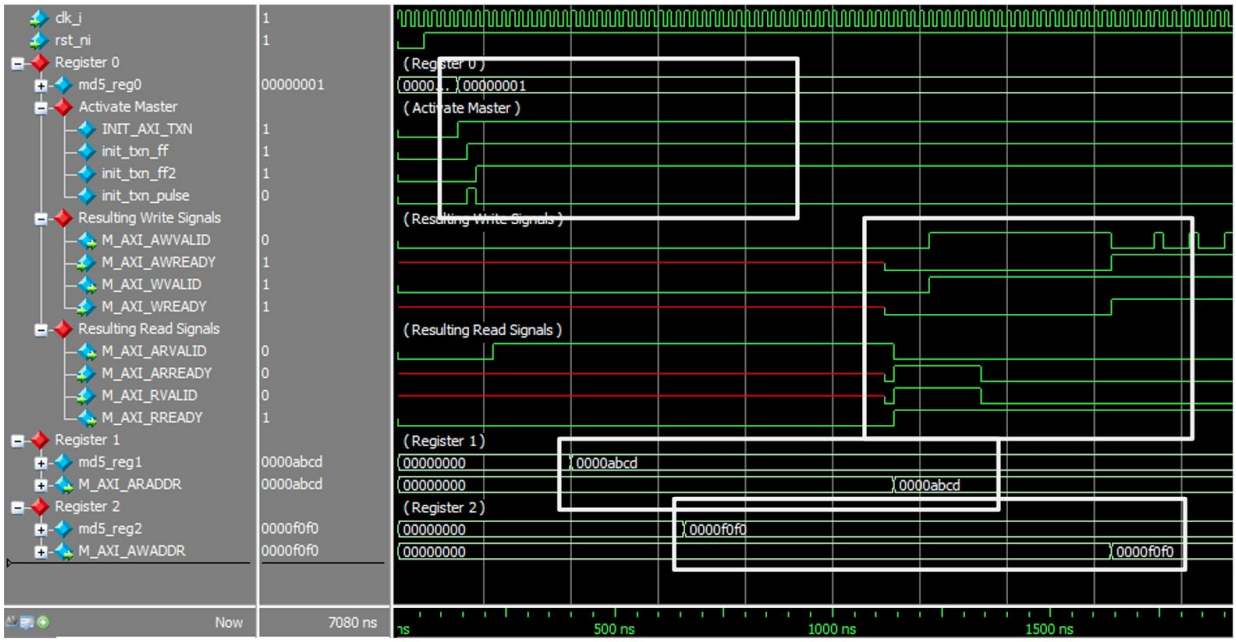


This waveform shows the register write and read from the testbench level as well as the DUT (design under test) level.

- The write address and data are driven by the processor (testbench) on the tb_i_axi_awaddr & tb_i_axi_wdata signals. This is received by the design as seen at the i_axi_awaddr & i_axi_wdata signals. This address is converted to a local address (loc_wr_addr signal) and the register mapped to that address is picked and is written. md5_reg0/1/2 are the internal registers as shown in the waveform.

- The processor/testbench drives the read address on the tb_i_axi_araddr signal, which the design converts to a local address (loc_rd_addr signal) and the register mapped to that address is picked and is read. The read data is driven by the design on the tb_o_axi_rdata bus to the testbench.

2. *Successful configuration of AXI Master using the control registers of MD5.*



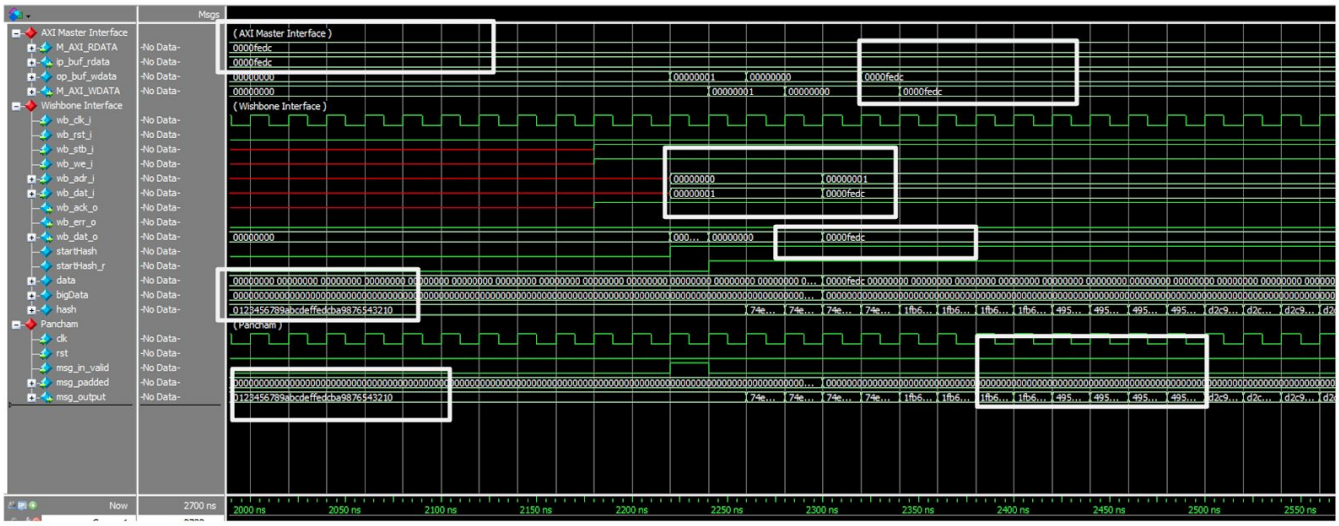
This waveform demonstrates configuration of the AXI master based on the control registers of the hardware accelerator written by the processor (testbench).

- Register 0 is responsible for activating or initiating the AXI Master. When written with '1', it triggers a logic which generates the init_txn_pulse signal which inturn is responsible for generating AXI Master's read & write protocol signals like READY, VALID and RESPONSE.

- Register 1 specifies the source register which is used by AXI Master to read from the shared memory. When this register is written, it drives the M_AXI_ARADDR bus with the read address value based on the AXI read protocol.

- Register 2 specifies the destination register which is used by AXI Master to write in the shared memory. When this register is written, it drives the M_AXI_AWADDR bus with the write address value based on the AXI write protocol.

3. Successful end to end flow using AXI Master & Pancham.



This waveform shows an end to end flow where the AXI Master reads data from the memory (testbench). This read data is written into the input buffer, used as an input by the MD5 core, Pancham to process and output to an output buffer. This output buffer data is written by the AXI Master into the memory. Following is the detailed flow:

- The AXI Master interface rdata has value FEDC in hex which is written to the input buffer.

- The input buffer is fed to the wishbone interface (of pancham) and wb_dat_i shows holding the same value when address and other design control signals (like stb & we) are set.

- The input to wishbone interface/pancham is 512 bits and the output bus is 128 bits. There are 16 arrays of 32bit (max data bus) which store values depending on the address and hash provided to give 128 bits output. Similarly these 128 bits are captured in 4 arrays of 32bit (DW bus) and written to the output buffer.

- Big data is 512 bits holding 16 arrays of data (32bit each) which is sent to pancham when mesg_in is high and we can see the same 512 bits of message padded.

- The mesg_ouput from pancham is the same, reflected on the hash value in the wishbone interface. This is further divided in chunks of 32bits and stored in wb_data_o (32 bits) and fed further to the output buffer in the MD5 top and written to memory by AXI Master.

Milestones & Deliverables:

Ownership: All the milestones are joint milestones. Both the team members will be working together on all of them. This was done to encourage better discussion, brainstorming and debugging for various design code/specifications.

Milestones:

Milestone 0 **-** A complete plan for the project SoC Security after understanding the past developments within the project.
Progress: Completed

Milestone 1 **-** Demonstrating working simulation & successful build of the current SoC - CEP Architecture.
Progress: Completed with a workaround for build.
Workaround:

- Migrated some files (immediately required for developing AXI Machines) to our local machines.
- Used ModelSim on our systems for their compilation. But without any testbench for this design architecture correct compilation or simulation couldn't be done.

Milestone 2 **-** Development & verification of AXI4 lite Slave reading & writing registers internal to hardware accelerator.

- *PART A:* Develop internal registers for hardware accelerator core-MD5 and a register access mechanism for AXI4 lite slave.
- *PART B:* Develop testbench to emulate the processor to read and write the MD5 registers through AXI4 lite slave interface.

Progress: Completed

Milestone 3 **-** Development & verification of AXI4 full Master, controlled by control registers of hardware accelerator & its interaction with Pancham.

- *PART A:* Design AXI Master code and configure it using the control registers of MD5. Also perform data manipulation and conversion into a wishbone interface for pancham to receive the input and process it.
- *PART B:* Develop testbench to emulate the memory for the AXI Master to read and write from. Implement AXI Master and Pancham interaction using axi to wishbone conversion mechanism.

Progress: Completed

Grading:

Assigned grades for milestones in mid quarter:

Grade	Milestones	Completion
B+	0, 1, 2(part A)	Completed
A-	0, 1, 2(part A & B), 3(part A)	Completed
A	0, 1, 2(part A & B), 3(part A & B)	Completed

Conclusion:

We were successfully able to extend the CEP SoC architecture with the AXI Master and Slave machines to control and monitor the interface between the SoC and the hardware accelerator cores. We made the hardware accelerator configurable by the processor on the SoC, in turn controlling the AXI Master implemented in the hardware accelerator.

However this is just one aspect for securing the SoC. This design provides a critical stepping stone to completely secure the SoC which is being worked on in this ongoing project.

References:

1. CEP Architecture for Security Assessments: https://www.darpa.mil/attachments/06_Brendon%20Chetwynd%20-%20Common%20Evaluation%20Platform%20(CEP)%20-V2.pdf
2. CEP Code from github: https://github.com/mit-ll/CEP
3. Basak, Bhunia, Ray, “A Flexible Architecture for Systematic Implementation of SoC Security Policies”, *2015 IEEE*
4. Restuccia, Pagani, Biondi, Marinoni, Buttazzo, “Is Your Bus Arbiter Really Fair? Restoring Fairness in AXI Interconnects for FPGA SoCs”, *J. ACM, 2019*
5. AMBA® AXI™ and ACE™ Protocol Specification, *ARM*