Hardware Implementation of Secure Communication in a Bus-Based Multi-Core System Using Tiny Encryption Algorithm

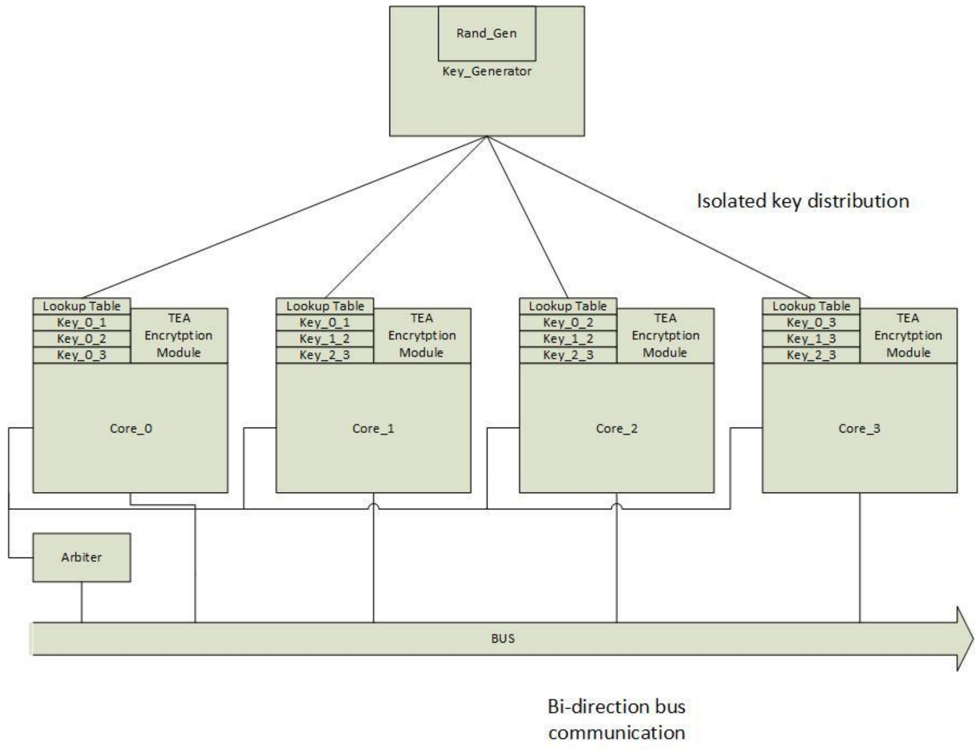
Team DJA DJA

Proposed Solution to Unsecure inter-core communication:

We propose to solve the problem of un-secure inter-core communication by using a hardware encryption method to encrypt data which is transmitted via the bus. Each time a core requests to read contents of the memory from other any other core, the data will be encrypted and then transmitted on the bus.
TEA (Tiny Encryption Algorithm) is a simple and fast symmetric encryption method. In the process of encryption and decryption, only addition, subtraction and XOR are involved, which make it a good hardware encryption method. In all, 16 rounds of encryptions will be executed (we can run multiple rounds in one clock cycle if core clock is slow enough compared to the computation latency).

Since TEA is quite simple to implement, it may not take very long to use brutal force to hack into the bus and snoop on the data. So the target here is to renew the keys periodically to avoid hacking using brute force. We will use a centralized key generator & distributor to implement this. The generator will use random numbers to generate random new keys. We can estimate the number of cycles required using the brute force method to hack into the system, using this data, set the refresh period of the key-generator smaller than this time period.

The overall architecture has been shown below:



Progress:

Key generator:
We implemented a key generation and distribution module. In this module, after a fixed number of clock cycles, a new set of random keys will be generated and stored locally. Once these keys are ready for dispatch, this set of new keys will be sent to all cores/memory controllers ("Masters") to update their respective lookup tables. This module was designed in Verilog and its functionality was tested using VCS.

Encryption/decryption module:
A wrapper has been implemented to handle the communication between:

1. Bus and encryption/decryption module
2. Processor and encryption/decryption module

During the implementation of this wrapper, some small modifications have been carried out on the encryption/decryption module. Some extra tags/descriptors have been added on the bus to inform the listener where the request originates from; also, a FIFO has been added in order to store the sources of requests in order, thereby facilitating queuing. This wrapper is currently under debugging stage (VCS, assertion) and waiting to be integrated on the bus.

Core and Bus system:
The 32-bit open-source core OpenRISC 1200 used in this project was tested stand-alone using the provided testbench. No bugs were obtained, as expected. The Wishbone bus generator obtained from opencores.org generated a VHDL code for the required bus. This VHDL was manually converted to Verilog, to make it compatible with the VCS version supported by CAEN machines. The task of integrating the bus with the core(s) ie, 3 cores and 1 memory module with the bus, is in debug phase (VCS)

Parallely, we tried system-on-chip(SOC) systems provided by OpenCores (e.g. minsoc) in order to integrate our discrete logic onto a single chip. We were successful in running basic test cases on the baseline SoC, but are yet to customize it as per requirements.

Bottlenecks and Challenges and Next Steps:

Owing to incompatibility of the VHDL with the version of VCS used in CAEN computers, we had to carry-out the arduous task of manually converting hundreds of lines of VHDL to Verilog, thereby delaying the final integration process.

One challenge is that, since minsoc requires unique packages which are not inbuilt in CAEN machines, and as new new packages cannot be installed, we can only run it on our machines. However, we don't have licenses to EDA tools such as VCS. Therefore, we are forced to use open source EDA tools but it is time-consuming owing to unfamiliarity with the infrastructure.

Once the current modules under debug are deemed bug-free, we intend to carry out the final integration. Presently, we aim to carry out integration parallely using minsoc as well as manual connections. The better of these two is to be selected based on custom tests.