SANTOSH APPACHU DEVANIRA POOVAIAH

Los Angeles, CA | (929) 303-0028 | sdevanir@usc.edu | www.linkedin.com/in/santosh-appachu

EDUCATION

University of Southern California, Viterbi School of Engineering — **Los Angeles, CA**
Master of Science in Computer Engineering — **August 2021-Present**
GPA: **3.5/4.0**
Specialization: Computer Architecture & VLSI design
Coursework: Computer system organization, Computer system Architecture, MOS VLSI circuit design, VLSI System Design (SRAM & Memory design)

Sri Jayachamarajendra College Of Engineering — **Mysuru, INDIA**
Bachelor of Engineering (Electrical & Electronics) — **August 2013-May 2017**
GPA: **9.83 (out of 10) - Gold Medalist (1st rank in EEE department)**
Award: Sukanya Dhananjaya Memorial Endowment Medal for securing the highest GPA and first rank for the academic year 2013-2017

SKILLS

- **Programming Languages:** C, C++, C++11, C++14(advanced), C++17, OOPS, Python, RTL coding, Verilog HDL, Franca IDL (Linux IPC), Verilog Behavioral modeling, Gate/Switch modeling, system tasks simulation and functions, system Verilog class & constraints, functional coverage & Assertions.
- **Software & Tools:** Git, perforce, Visual studio professional, Eclipse IDE, QuestaSim, ModelSim, Cadence Virtuoso, Linux VM, PlantUML, Enterprise Architect, CPPCheck (static code analyzer), Valgrind, Lcov & Gcov (code coverage), Pycharm, Clang-tidy, Docker containers, and Docker images.
- **Operating System:** Microsoft Windows, Linux (Ubuntu 18.04), CentOS7.
- **Framework:** Single cycle CPU, 5 stage pipeline CPU with the late and early branch, Branch prediction models, Out of Order processor (Tomasulo IOI OOE OOC, IOI OOE IOC) with instruction scheduling, Cache Hierarchy & cache mapping techniques, Memory system organization, Synchronous & Asynchronous FIFO, Virtual memory, Chip multicore processor, Cache coherency models (MSI, MESI, MOESI), Invalidation & update protocols, Chip multithreading, superscalar & super pipeline architecture, VLIW & Vector processors, Parallel programming models, MPI(Message passing interface programming), GPU Architecture and CUDA programming, coherence, Cache stashing, synchronization & memory consistency models.
- **Professional Tools & Skills:** Agile & Scrum board for project management, JIRA tool for issue tracking, Polarian for product development and requirement gathering, Agile Software Business Analyst, Certified tester by International software testing qualification board (ISTQB).

WORK EXPERIENCE

NVIDIA Corp (3 months 2 weeks) — **Austin, USA**
CPU+GPU Full System Coherency Verification Intern — **May 2022 - August 2022**

- Worked on generating CPU bare metal elf test files using random code stream stimulus generator BLAZE tool using input Yaml configuration files to test sanity, checkpoint, algorithm, covert, rand_mix, multi-processor read-write gen functionalities on NVIDIA SOCs.
- Worked on running the blaze tests on NVIDIA's post silicon platform using TegraShell and python recipe to load the elf files into the memory location and run multiple tests & capture the test results on the chip using the ping-pong mechanism. Set up the environment to run the CPU bare metal tests on the pre-silicon platform using various simulators like Tsim, Asim, and Dsim simulation platforms that simulate the ARM CPU core complex and generate the core-specific tarmac files to obtain the coverage to check the stimulus goodness.
- Worked on generating BLAZE(random stimulus generator) tests for various instruction sizes using the Tgen tool by writing LVL (level) test scripts. These tests were run on Tsim simulation platform for pre-run and on RTL testbench using co-simulation of Tsim & Simt (simulation tool for CPU).
- Worked on setting up the SCSIM (software simulator) platform on Virtual Machine for one of the NVIDIA SOC to run the CPU+GPU tests using TegraShell connecting to the software platform via a fake USB cable using a dummy hcd driver using Linux named pipe. SCSIM simulator is a Linux image that is booted all the way to the Linux kernel running on Ubuntu18.04VM. Replicated the above testing environment inside Docker container by developing a Linux docker image from scratch and deployed it on NVIDIA's hw artifactory and made it available for the whole CPU team.
- Translated the ARM AMBA Coherency Hub Interface (CHI) protocol and transaction flow into sequence diagrams using PlantUML tool to come up with the test plan for testing CPU+GPU full system coherency network flows for future NVIDIA SOCs.

ROBERT BOSCH BUSINESS AND ENGINEERING SOLUTIONS (4 years 1 month) — **Bengaluru, INDIA**
Senior Software Developer — **July 2017 -August 2021**

- Designed C++ object-oriented interfaces for Media Service applications. Creation of Franca IDL interfaces for Energy Management, Firmware Update, Developed firmware update for BOSCH smart home appliances using C++14 to check for updates, and on availability of a new update download the packages and update appliances with the new firmware. Network libraries used - Curlpp, libcurl, BOOST SML, REST API, BOOST DI, JSON parser.
- Worked on Linux desktop bus IPC communication with GENIVI common API as the wrapper (stub and proxy class). Worked on the Unit testing framework (GoogleTest & GoogleMock) to verify the C++ developed code and generate the functional, line, and branch coverage.

PROJECTS

Design and Implementation of Static & Dynamic Branch Predictors | Intel Pin Tool | C++ — **August 2021**

- Implemented various static and dynamic branch predictors to predict if the branch instructions were taken/not taken using the Intel pin tool.
- Always taken, 2bit-Global predictor, 2bit-Bimodal buffer predictor, 2bit-Correlated predictor, and (M, N) BPB were implemented as part of this.
- The predictor models were developed using the C++11 programming language. Simulation tests were performed on the models using a dynamic binary instrumentation framework by Intel. It was used to instrument and analyze the total branch references, the total number of predictions, and the branch **prediction rate.**

Design of 32 bit 5 stage pipelined CPU using RTL coding | Verilog | ModelSim — **October 2021**

- Implemented in-order 5-stage pipeline CPU design using Verilog. The structural coding technique was adopted. The design has 5 pipeline stages: Instruction Fetch (IF), Instruction Decode (ID), Execution Stage 1 (EX1), Execution Stage 2 (EX2), and Write Back Stage (WB). The EX1 included X-3 **ALU** and EX2 included X+4 ALU**.** The same design was also developed using the behavioral coding technique
- Internally Forwarding Register File, Hazard Detection Unit(Stalling logic), and Forwarding Units were also included in the design.
- The pipeline included a simple one-hot opcode encoding, and supporting instructions of SUB3, ADD1, ADD4, MOV, and NOP operations.

Design of 16bit single clock synchronous FIFO | Verilog | Questasim — **December 2021**

- Designed single clock (n+1) and n bit FIFO with 4bit Write and Read pointer. Accounted for RACE condition in the design.
- Simulated the FIFO design using producer and consumer test benches in Questasim. Mitigated ambiguous situations of almost empty or almost full scenarios in FIFO by using (n+1) bit pointer for read and write pointer to determine Full and Empty conditions.

Microarchitecture Simulation of Out-of-Order Superscalar Processor | Simple-Scalar | Cacti | Real Estate Estimator — **February 2022**

- Performed simulation of a superscalar processor by modifying the configuration knobs such as fetch, decode, commit width, cache size, associativity, reservation stations, ROB, RUU & functional units using a simple-scalar simulator. Used the cacti tool to estimate the access time & power consumption of cache and RUU structures. Used real estate tool to estimate the transistor count and area consumed for various micro-architecture configurations.
- Ran Perl, Compress & Anagram benchmark programs on every custom simulated processor to analyze the number of committed instructions and MIPS value for each configuration. Performed iterative simulations to get the best design for a superscalar processor with the highest MIPS.

Vector Addition using GPU Kernel function & CUDA Memory | CUDA Programming — **April 2022**

- CUDA programming of vector addition using GPU kernel function, kernel grids, warps, and blocks using Global, shared, and constant memory.
- Point-to-Point and collective communication explicitly using Message-passing Interface (MPI) programming in shared-memory multiprocessors.