Project Proposal

Project Title:	**Implementing RISCV System-on-Chip for Acceleration of Deep Learning Operation Based on ASIC**
University:	**Universiti Teknologi Malaysia**
Department:	**(MJIIT) Department of Electronic Systems Engineering**
Competition Track:	**IC Design**

1. Abstract

The aim of this project is to implement RISC-V to System-on-Chip (SoC) for acceleration of Deep Learning Operation based on ASIC. Instead of using the typical FPGA, we had implemented the system onto ASIC, this is because FPGA is comparatively larger than the ASIC and it will consume more silicon area as well as power than the ASIC. By implementing we are able to reduce the power consumption and increase the speed of the System-on-Chip. This is because when the runtime of the system decreases, the whole system will be more power efficient, hence it can be utilized in many aspects. The Convolutional Neutral Network (CNN) is used to implement the system's framework.

2. Project Introduction

Due to their improved performance over conventional approaches in handling challenging perceptual and control problems, deep neural networks (DNNs) are in very high demand. DNNs neural networks use a process of trial and error and require enormous quantities of data to train. Hence, neural networks gained popularity once the majority of businesses adopted big data analytics and gathered enormous data sets for usage in complex embedded and robotics systems. [1]

Moreover, embedded systems typically impose stringent limitations on the power, size, and weight of their computing platforms, whereas DNNs are computationally intensive and have huge memory requirements. Hardware DNN accelerators are being implemented into embedded system-on-chips to increase the speed and effectiveness of DNN processing, particularly inference operations, in embedded systems (SoCs).

For the core of our design, we will use The Berkeley Out-of-Order Machine (BOOM). BOOM implements the open-source RISC-V ISA and utilizes the Chisel hardware construction language to construct generators for the core[x]. A generator is analogous to an RTL design that has been generalized. One instance of a generator design is the typical RTL design. Because of this, BOOM is a group of out-of-order designs rather than a single core being. Additionally, BOOM uses the Rocket Chip SoC generator as a library to reuse various micro-architecture structures when creating a SoC with a BOOM core (TLBs, PTWs, etc).

The project will make use of the open-source hardware inference engine Nvidia Deep Learning Accelerator (NVDLA), which will be integrated into the Risc-V SoC platform. A free and open architecture called NVDLA supports a common approach to creating deep learning inference accelerators. NVDLA is scalable, highly configurable, and built with a modular architecture to make integration and portability simple[2].

The objective of the project is to execute a faster deep learning operation, like YOLO, Resnet and others algorithms onto a RISC V System-on-Chip. The algorithms use neural networks to provide real-time object detection in various image detection applications due to its speed and accuracy. The RTL will be designed and simulated in FireSim, a fast cycle-exact system simulator which runs on cloud FPGA. The simulated hardware then will be pushed through VLSI flow for the fabrication of the chip. ASIC design flow then will be carried out using Synopsis tools.

From our design specifications, we believe it will have better acceleration performance, compared to GPU and CPU solutions, and under various cache and memory configurations. The ASIC should have good acceleration performance, low power consumption, and its impact of shared memory interference between CPU and NVDLA is significant especially for critical real-time embedded systems. The application-specific integrated circuit is used to design chips with large scale and high complexity, or products with high maturity and large production.

ASIC optimises logic cell resources according to design requirements, and achieves optimal layout and routing to reduce routing delay and cell delay. Under the same conditions, FPGA is larger than ASIC. The FPGA with large look-up tables (LUTs) and static memory (SRAM) will consume a lot of silicon area and consume significantly more power than ASIC. After the design is completed, the design netlist is generated and handed over to the chip manufacturer. After tape-out, the internal logic circuit is fixed, and the function of the chip is fixed.

3. Design Methodology

3.1 Design Flow of ASIC digital IC

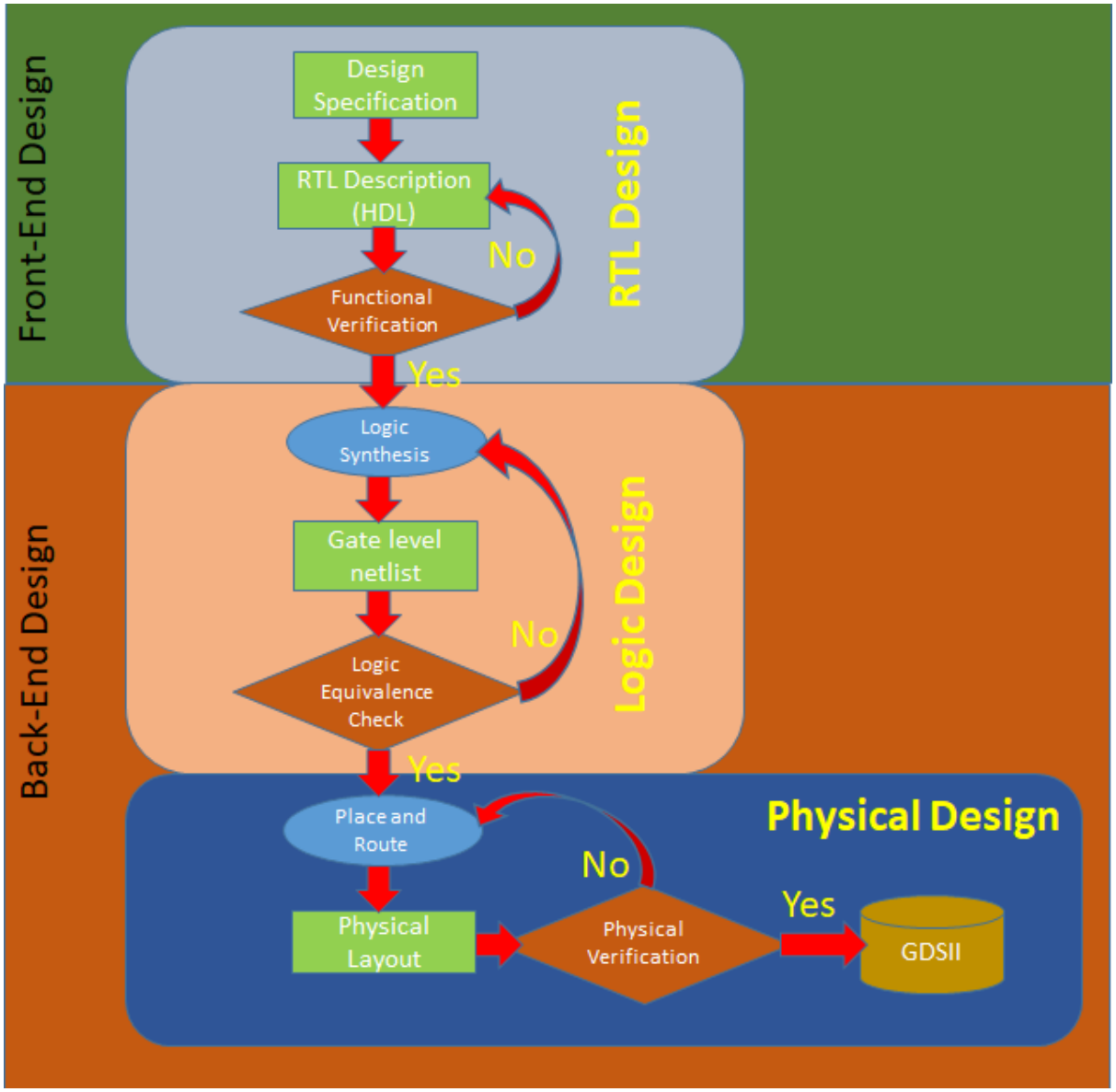

Figure 1 : Design Flow of digital IC

The design flow for an integrated chip design will be applied to design an Application-specific integrated circuit chip. The ASIC is designed according to specific circuit requirements, and a dedicated logic circuit is designed.

ASIC design is divided into front-end design, back-end design, packaging and testing.

3.2 Chisel and Verilog

The Register-transfer level of front-end design will be using Chisel. The Chisel (Constructing Hardware In a Scala Embedded Language) is a hardware construction language embedded in

the high-level programming language Scala. Chisel is an open-source hardware construction language released by the University of Berkeley[1] . It is built on the Scala language and is an application of the Scala domain-specific language. It has highly parameterized generators, which can support advanced hardware design. Chisel is some special class definitions with a collection of predefined objects. When writing a Chisel program, is the same as writing a Scala program. It has many features as follows:

- Hardware construction language
- Embedded in the Scala programming language
- Algebraic construction and wiring
- Abstract data types and interfaces
- Bulk connections
- Hierarchical + object-oriented + functional construction
- Highly parameterizable using metaprogramming in Scala
- Supports layering of domain-specific languages
- Sizeable standard library including floating-point units
- Multiple clock domains
- Generates high-speed C++-based cycle-accurate software simulator
- Generates low-level Verilog designed to pass on to standard ASIC or FPGA tools
- Open source on GitHub with modified BSD license

These features make it a hardware description language that slightly higher-level than Verilog. It is much more powerful than Verilog in some aspects, at least in the field of SoC and processor design.

3.3 Risc-V core and Chipyard

The project will be using Chipyard to configure a Rsic-V core to generate a RTL design. Risc-V is an open-source standard instruction set (ISA, Instruction set architecture) with architecture based on the idea of the reduced instruction set. The main feature of RISC-V is open source compared to other ISA like X86 or ARM. At this stage, many companies have developed hardware and open-source operating systems that support RISC-V. In addition, RISC-V is not only an instruction set architecture. Due to its open-source nature, a huge community has been established. It also has a complete toolchain to finish development, compilation, and simulation.[2]

Chipyard is an open-source framework with a unified framework and Workflow for the agile development of Chisel-based SoCs. Supports production of RISC-V SoCs using Chisel HDL, Rocket Chip SoC generator. This framework has everything from Memory-mapped I/O peripherals to custom accelerators.

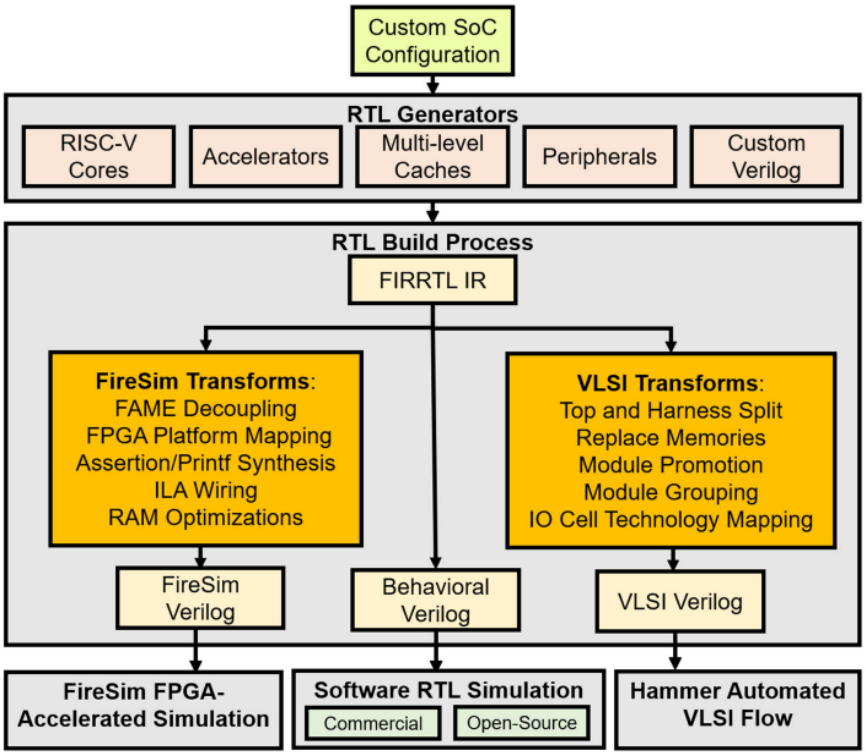

Figure 2 : Chipyard Front-end For RTL

There are multiple separately developed and highly parameterized IP blocks like rocket core, BOOM (Berkeley Out-of-Order Machine) and Hwacha accelerator etc. An SoC RTL design can be formed using these configured and interconnected blocks. This design then can be verified and validated through both FPGA and standard software simulations. The output will push to the portable VLSI design flows to obtain tape-out-ready GDSII data. It also has a workload management system to generate software workloads to test the design.

In this design, our choice will be using Boom from the Chipyard framework as the core CPU for the NVDLA Architecture. The full name of BOOM is Berkeley Out-of-Order Machine. As the name suggests, it is an out-of-order execution core. It is a 7-stage pipeline with a synthesizable and parameterizable RV64GC RISC-V core, which is also implemented by Chisel. The following is the detailed pipeline structure. That are seven stages in the pipeline: Fetch, Decode/Rename, Rename/Dispatch, Issue/Register Read, Execute, Memory and Writeback.[3]

The out-of-order execution of the Mini Boom core can slightly improve performance compared to Rocket's in-order execution. If the configuration of Boom is large, the improvement is more than twice the performance. The following is the struture of Boom core in details.

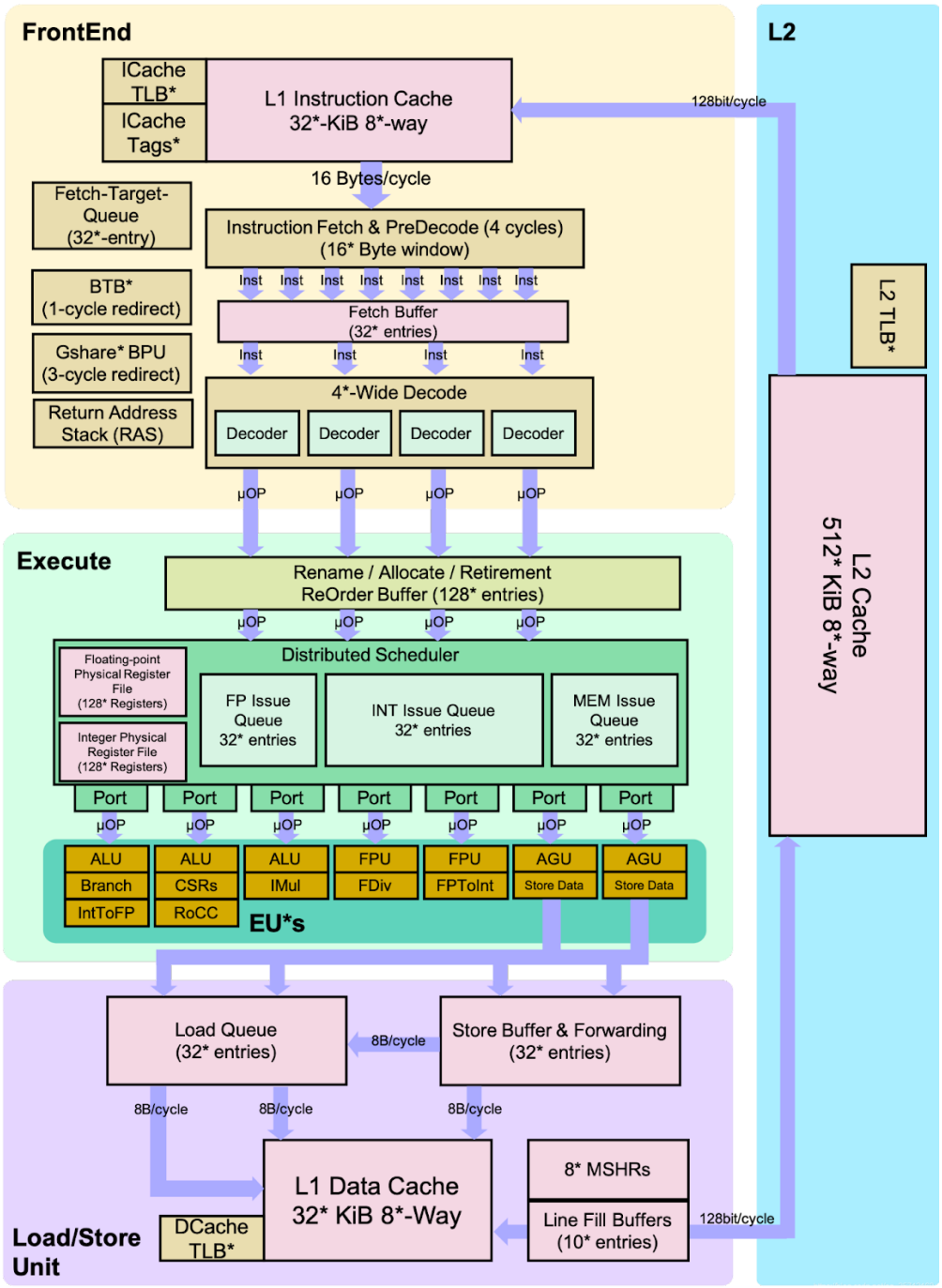

Figure 3 : Architecture of Boom

3.3 NVDLA

The choice of AI accelerator for this project will be using NVDLA from the company Nvidia. This accelerator is also one of component that can port to the Chipyard Framework.

NVDLA (NVIDIA Deep Learning Accelerator) is an open-source architecture released by NVIDIA to address the computing needs of deep learning. Standardized deep learning hardware acceleration modules are designed to facilitate the development of machine learning. The NVDLA is scalable and highly configurable with a modular architecture, which can simplify the process of integration and increase portability.[4]

NVDLA implementations generally fall into two categories:

headless (Small) - Unit-by-unit management of NVDLA hardware occurs on the main system processor.

headed (Large)- Delegates high interrupt frequency tasks to a microcontroller that is tightly coupled to the NVDLA subsystem.

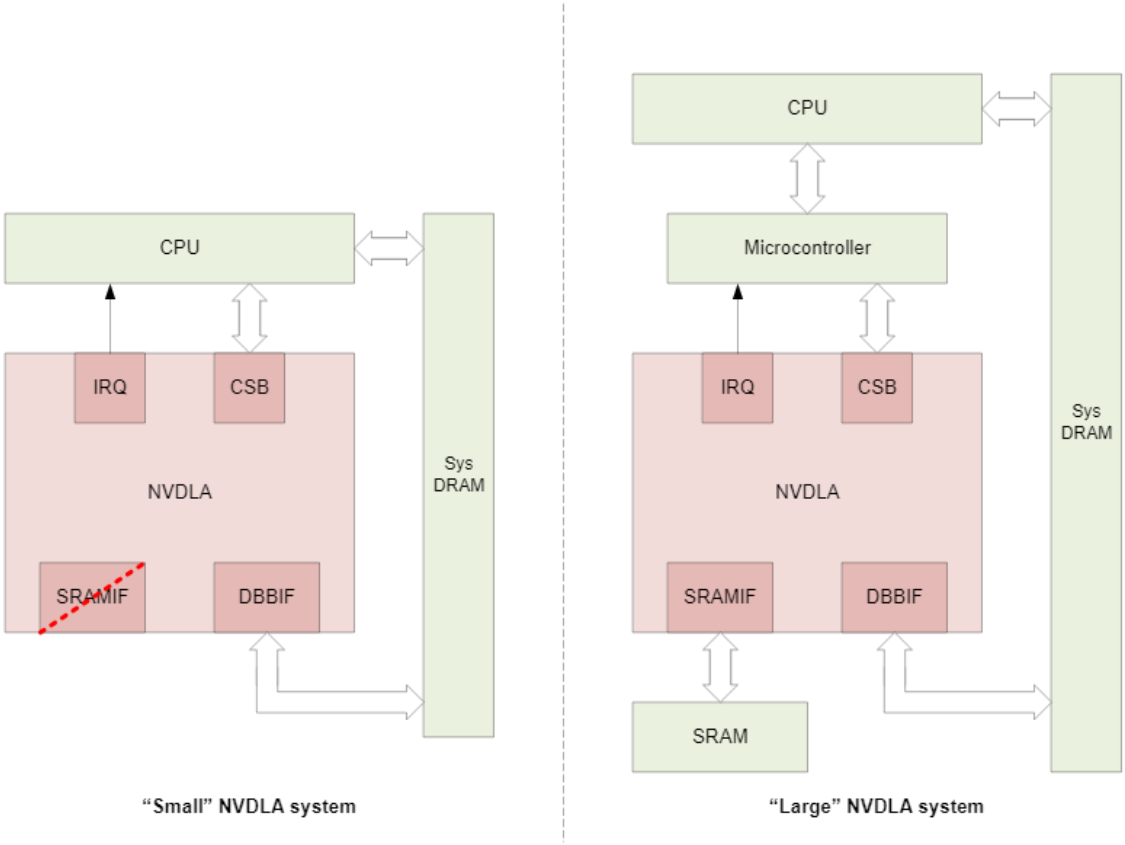

Figure 4 : Implementations of Small and Large NVDLA

These two implementation of NVDLA is open source on Github with both hard ware and software. Our design will use the small NVDLA to integrate with a Risc-V for acceleration of Deep Learning Operation with a Risc-V core. It is more energy efficient compare to the large implementation.[5]

Most of the computational work of deep learning inference is based on mathematical operations, which can be divided into four main parts: convolution, activation, pooling, and normalization. NVDLA hardware consists of the following components:

- Convolution Core - An optimized high-performance convolution engine.
- Single Data Processor - Single point of lookup engine for activation functions.

- Planar Data Processor - Planar averaging engine for pooling.
- Channel Data Processor - Multi-channel averaging engine for advanced normalization functions.
- Dedicated Memory and Data Reshape Engines - Memory transformation acceleration for tensor reshape and copy operations.

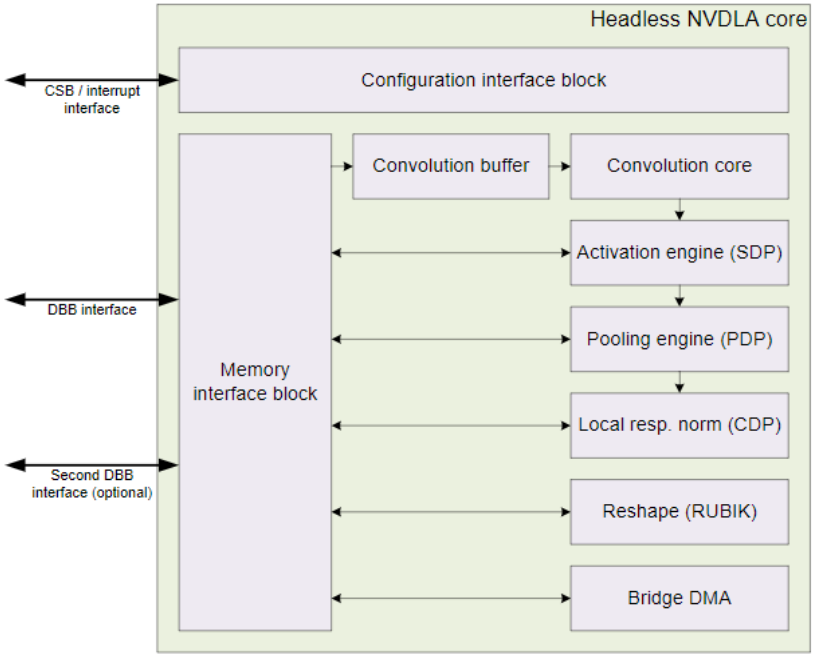

Figure 5 : Internal architecture of NVDLA core

3.4 TileLink

NVDLA has an integrated AXI4-based direct memory access (DMA) engine (Data Backbone Interface and Configuration Space Bus). It is used to read neural network layer weights and input feature maps from system memory, and to write output feature maps to system memory. Since Risc-V Chip uses the TileLink bus, this AXI4 interface need to convert to TileLink.

TileLink is a new chip-level bus interconnection standard proposed by SiFive, a chip start-up company started by Berkeley university. It allows multiple masters to access memory and other slaves in a consistent memory-mapped. The design goal of TileLink is to provide a high-speed, scalable on-chip interconnect with low latency and high throughput for SoCs to connect general-purpose multiprocessors, coprocessors, accelerators, DMA, and various simple or complex equipment. [6]

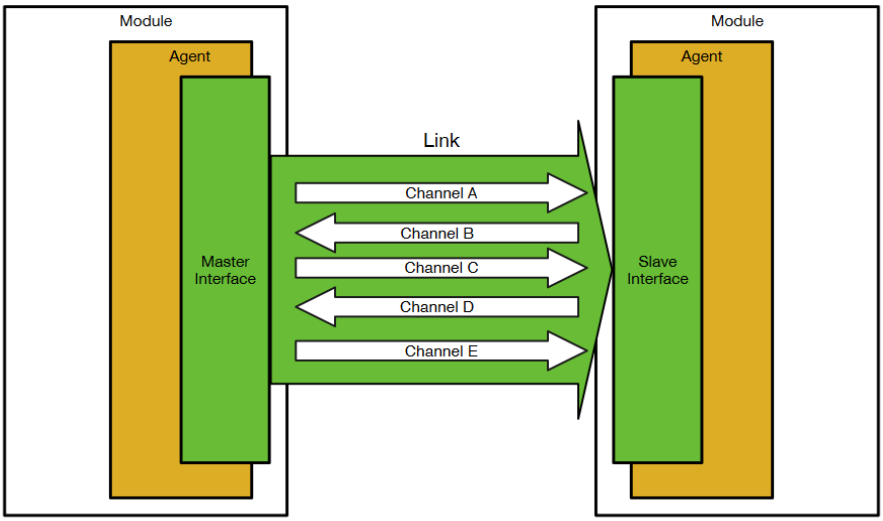

Figure 2.4: The five channels that comprise a TileLink link between any pair of agents.

Figure 6 : TileLink between Agents

3.5 Tool chain for deployment

NVIDIA cooperated with SiFive to introduce a firesim-nvdla project with deployment of Yolov3. This project ran a RiscV+NVDLA architecture on the FPGA cloud server platform. The Darknet was modified, and special tools were used to extract the subgraphs in Yolov3 that can be run with DLA to generate a Loadable file. The Yolov3 was successfully run, and Yolov3 can reach 7 frames in the large configuration. But the toolchain is not open source and is limited to the Darknet framework. Then, another toolchain is required in this project for development of Asic.

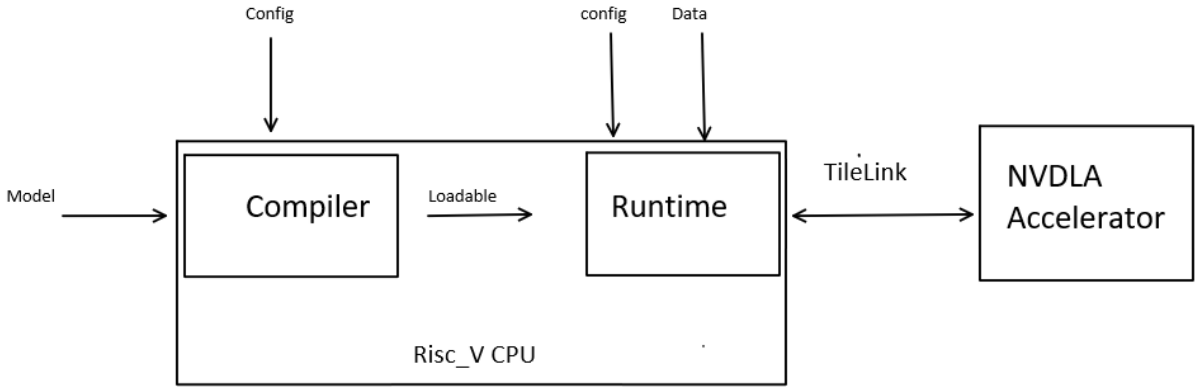

Figure 7 : Deployment algorithm process

The deployment of a trained model required compiler and runtime inference to link with NVDLA. The compiler is provided by Nvidia and the runtime inference can use the open-source Tengine framework. Tengine is an open-source runtime framework for edge device inference by Open AI Lab.[7] This project realizes the fast and efficient deployment of deep learning neural network models on embedded devices. It uses C language for the core module development with customized limited resources process of embedded devices for better compatibility in cross-platform deployment in many AIoT applications. At the same time,

completely separate front-end and back-end designs are adopted. Which benefits fast deployment on CPU, GPU, and NPU, and reduces evaluation and migration costs.

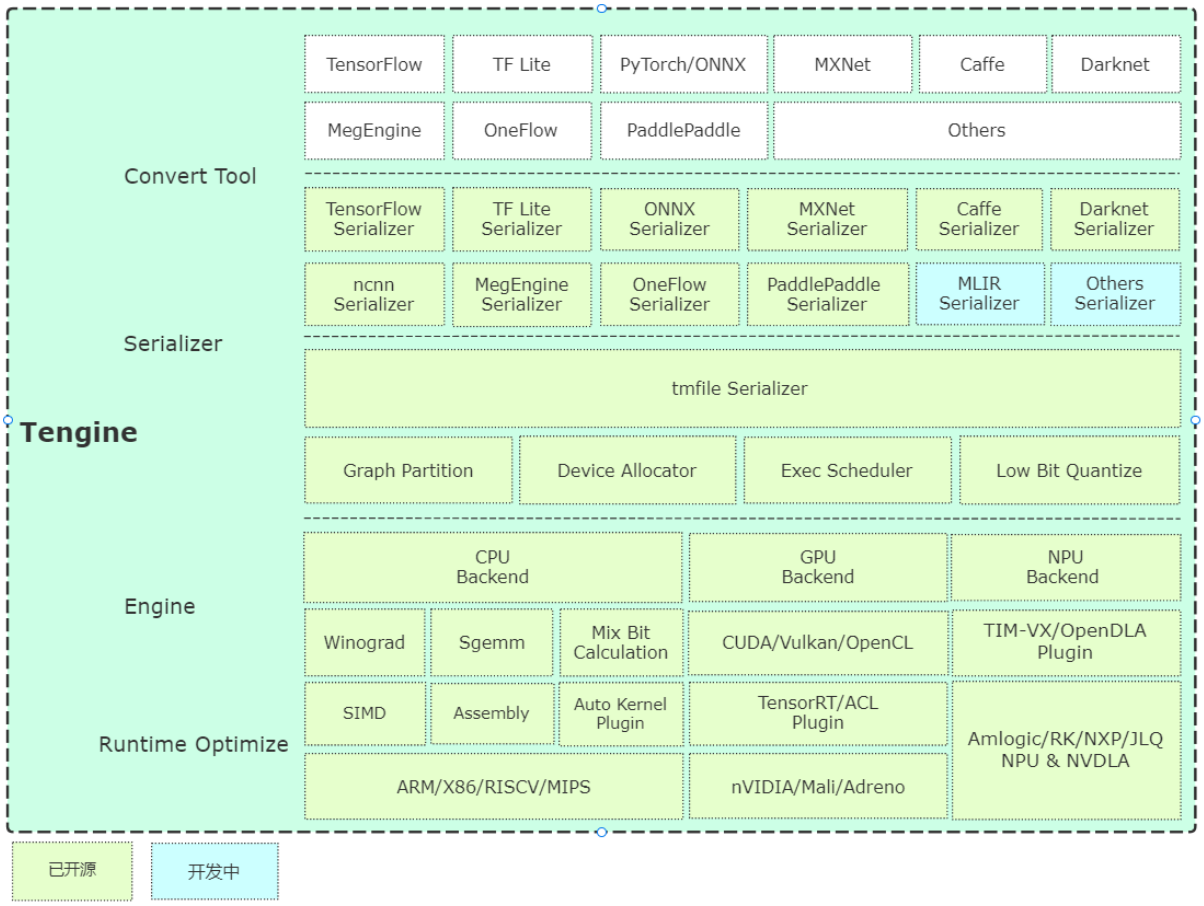

Figure 8 : Tengine Tool Chain

3.6 Back-End Design

After the Front-End Design is completed with a RTL Design synthesized and verified, the design will proceed to Back-End Design, as referred from Figure 1 : Design Flow of digital IC. The back-end design synthesis tool that we will be using is Synopsis. [10]

Synopsys' Design Compiler will be used to constrain a complex design for area and timing, apply synthesis techniques to achieve area and timing closure, analyze the results and generate design data that works with physical design or layout tools. Design compiler timing and area constraints file based on schematic and specification will be created. Syntax of the constraints will be verified then apply the constraints to design and validate its correctness. To check whether the RTL and Netlist are functionally equivalent or any differences in the design without using testbench, formal verification is used.

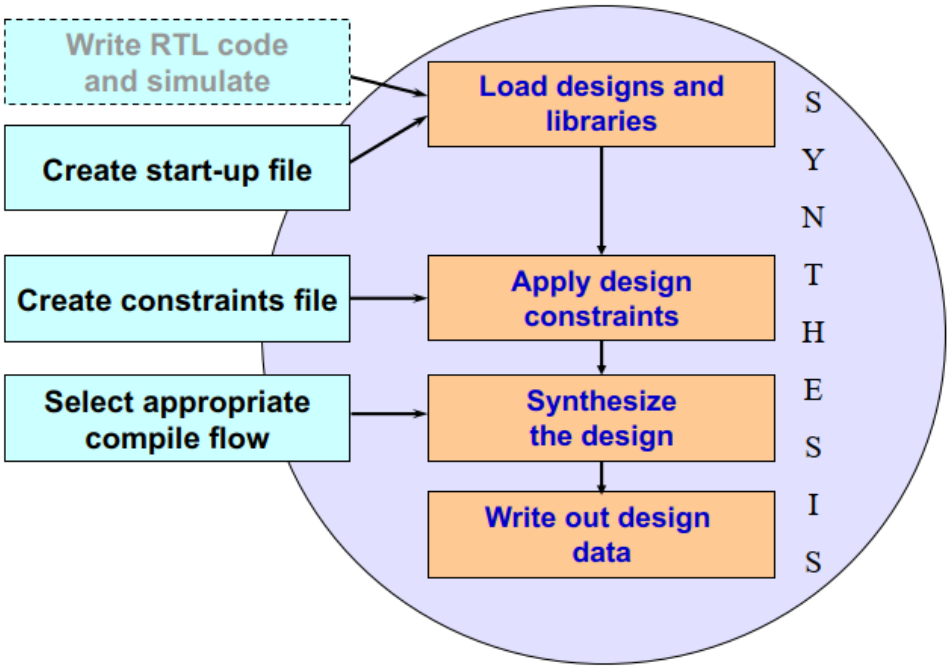

Figure 9 : Design Compiler (DC) Flow

IC Compiler II will be used to do the physical design. NDM design library will be created and load the gate-level netlist. RC parasitic tables will be loaded, placement site and routing layer settings will be checked and modified before applying the UPF. floorplan- and scan- DEF files will be loaded and confirm the placement site and routing layer settings. Timing setup will perform MCMM setup and confirm the implementation readiness including the zero-interconnect timing sanity check.

For floorplanning, the block shape of the design will be defined, with its voltage areas placement inside the block. Then, placement to determine macro locations will be performed. Defining the block pin locations. The design congestion and netlist connectivity will be analyzed and run the PG prototyping. “place_opt” settings will be applied to perform placement and optimization. Timing and design quality of results(QoR) will then be analyzed. During the CTS setting, clock tree balancing will be set up. Non-Default routing rules, DRC constraints and clock-timing, clock tree synthesis and datapath optimization will be applied to the design. It's either running the CTS or CCD flow in our design.

Routeability checks onto placed design clock tree will be carried out and apply the routing options. Routing secondary PG nets and control via optimization. Analysis of the design for timing with SI enabled other power and crosstalk optimization. Execute sign-off DRC checking and fixing, standard cell filler insertion and sign-off metal fill insertion.

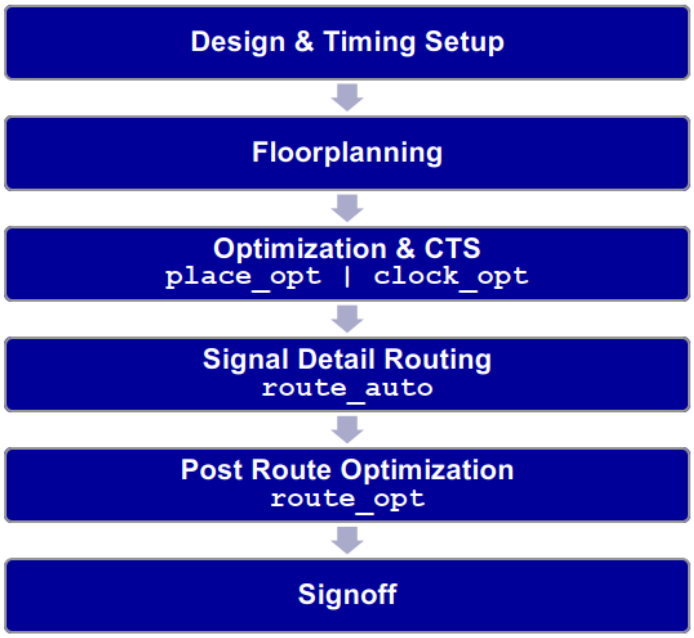

Figure 10 : IC Compiler II Flow

Prime Time will be used to accomplish Static Timing Analysis (STA) on our chip level design netlist. The design, library, parasitic data and constraints need to be read in. Before producing the Interpreting Reports for Summary, Timing and Noise reports, STA constraints need to be debugged. On Chip Variations(OCV) is set and Advanced Waveform Propagation (AWP) for signoff accuracy is enabled. Path Based Analysis(PBA) is applied.

3.7 Improvement

The NVDLA architecture was developed in 2017, so there are still some technical improvements that can be done. Such as limited optimization of AI computing data flow and storage walls, and the usage of AI-optimized memory. The pre-built workloads from the NVDLA software provided in the software stack are not synced to the latest release of NVDLA hardware.

This project targets to test more configurations of both NVDLA with Risc-V cores in the prototyping stage and seek more improved modifications for the suitable accelerator to proceed to the next stage of backend design.

References

[1] Mingyu Gao et al. 2017. Tetris: Scalable and efficient neural network acceleration with 3D memory. OSR 51, 2 (2017), 751–764.

[2] Nvidia. 2018. NVIDIA Deep Learning Accelerator. http://nvdla.org/primer.html

[3] “GitHub - chipsalliance/chisel3: Chisel 3: A Modern Hardware Design Language.” https://github.com/chipsalliance/chisel3 (accessed Nov. 14, 2022).

[4] “Welcome to Chipyard’s documentation (version ‘1.8.1’)! — Chipyard 1.8.1 documentation.” https://chipyard.readthedocs.io/en/stable/index.html (accessed Nov. 14, 2022).

[4] “GitHub - riscv-boom/riscv-boom: SonicBOOM: The Berkeley Out-of-Order Machine.” https://github.com/riscv-boom/riscv-boom (accessed Nov. 14, 2022).

[5] F. Farshchi, Q. Huang, and H. Yun, “Integrating NVIDIA Deep Learning Accelerator (NVDLA) with RISC-V SoC on FireSim,” *Proceedings - 2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications, EMC2 2019*, pp. 21–25, Feb. 2019, doi: 10.1109/EMC249363.2019.00012.

[6] “NVDLA Primer — NVDLA Documentation.” http://nvdla.org/primer.html (accessed Nov. 14, 2022).

[7] “SiFive TileLink Specification,” 2017.

[9] “GitHub - OAID/Tengine: Tengine is a lite, high performance, modular inference engine for embedded device.” https://github.com/OAID/Tengine (accessed Nov. 14, 2022).

[10] “Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions.” https://www.synopsys.com/ (accessed Nov. 14, 2022).