CPRE 581 Progress Report

Jake Hafele, Gregory Ling, Thomas Gaul
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50014

I. Implementation Progress

We have successfully cloned, built, and created our own Git repository based off of the ChipYard source code. We were able to install the tools and dependencies on our CPRE 581 Linux VMs, which led to running Verilator simulations to validate the toolflow installed correctly. We began using the default RocketChip configuration, and started to learn about the project directory structure.

Once comfortable running Verilator simulations for RISCV assembly programs and generating RTL for the RocketChip, we explored how to generate Verilator simulation results and RTL code for the Boomcore. We learned that there were multiple sizes of the Boomcore, which held multiple different numbers of issue widths and physical registers. After this stage, we were ready to investigate how to implement the generated BOOM Core onto a real FPGA.

This led us to talk more with Jordan McGhee, who has spent a considerable amount of time using Chipyard and the BOOM core with a ZCU106 FPGA development board. Jordan was able to share a zip of his work and a git patch which would allow us to generate RTL for the BOOM Core, which would be able to reconfigure a ZCU106 directly, which we had access to in Durham 310.

Over the weekend, we worked on running Linux on a programmed ZCU106 with a Small Boom core configuration. We were able to successfully echo "Hello World", and attempted running a 2006 SPEC benchmark, which crashed due to latency delays on the loaded SD card. Another option we are considering is using a smaller test program to ensure the test program can run.

In the week before break, we began to generate RTL and Vivado projects for each of our six branch predictors, including TAGE, Gshare, Tournament, Global, Local, and Null predictor configurations. We utilized the patch files provided by Jordan McGhee to generate the Vivado projects for the ZCU106, and were able to attain power, timing, and utilization results for each of the six prediction schemes.

II. Challenges

Throughout the first few weeks of working on the project, we have encountered and addressed a handful of challenges. The first challenge we ran into was Chipyard's size for use. We commonly run out of space on our machine and have to delete items to keep running things with it.

The next issue we ran into was the size of the chips once they were done and the challenge of configuring a project to the FPGAs available to us. Our initial plan was to run on a ZED Board, but that was too small to run a BOOM Core due to the gate count and did not have a setup pre-made by Chipyard. An alternative would be to run a Rocket Core on it which it would have barely had the gate count for, but that would have required configuring the ZED board and making more branch predictor units than the BOOM Core. After talking to Jordan, he said getting a board configured for Chipyard was no small undertaking, and he recommended not doing so. Another alternative plan was to run simulations with Verilator or GEM5 and just get the physical characteristics from Vivado. The simulation options would take way too much time to run any Benchmark to the point 5 hours into a simple Quicksort run on Verilator yielded no completion. To solve this problem, we got access to an FPGA that was already configured with Chipyard and was big enough to run the BOOM Core.

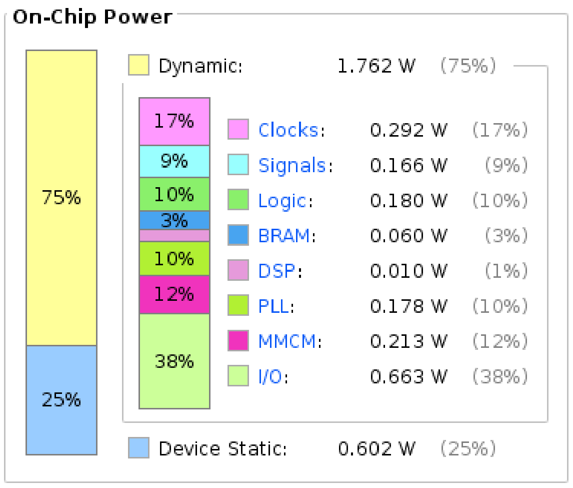
We ran into another issue where the provided SD Card which had an instance of FireMarshal and Linux had a broken kernel driver, which was to be put on the FPGA after reconfigured. To fix this, we wiped the SD card and copied over a fresh FireMarshal configuration which was provided again by Jordan McGhee. The only supported file system for FireMarshal is HFS+, which is only accessible on Mac OS, meaning we had to use Gregory's laptop to reload the SD card.

Another issue we ran into involved using the Linux computers in Coover. On one of our machines (Jake), we did not have the gtc dependency installed, and did not have sudo access. The constraint was that we needed the Linux Coover instance to source Vivado, so we were unable to use our 581 VM's to create and synthesize our design. To solve this, we used Gregory and Thomas's computers to generate Vivado reports and bitstreams, and worked in parallel.

III. Results

We learned that the Verilator simulations would be unrealistically long, and improbable to simulate our branch prediction performance metrics. Dr. Duwe also stated that running a Boom Core on an FPGA could be up to 100X faster.

We were able to successfully attain synthesis results in Vivado, which included power, utilization, and timing results for each of our 6 branch configurations. Figure 1 represents the power utilization of the ZCU106 for the TAGE branch predictor configuration. It can be seen that a majority of the power consumption comes from dynamic switching power.


On-Chip Power

🟨	Dynamic:	1.762 W	(75%)
🟪	Clocks:	0.292 W	(17%)
🟦	Signals:	0.166 W	(9%)
🟩	Logic:	0.180 W	(10%)
🟦	BRAM:	0.060 W	(3%)
🟪	DSP:	0.010 W	(1%)
🟩	PLL:	0.178 W	(10%)
🟥	MMCM:	0.213 W	(12%)
🟩	I/O:	0.663 W	(38%)
🟦	Device Static:	0.602 W	(25%)

Figure 1: TAGE Predictor ZCU106 Power Consumption

We also were able to generate a utilization report for each branch predictor, with a sample Utilization mapping of the ZCU106 seen in Figure 2. It should be noted that the size of the Small BOOM core would be too large to load on other FPGA's, and while only around half of the ZCU106 is used, it is still a marginal amount of utilization which would exceed other supported boards such as the Arty100T FPGA development board.

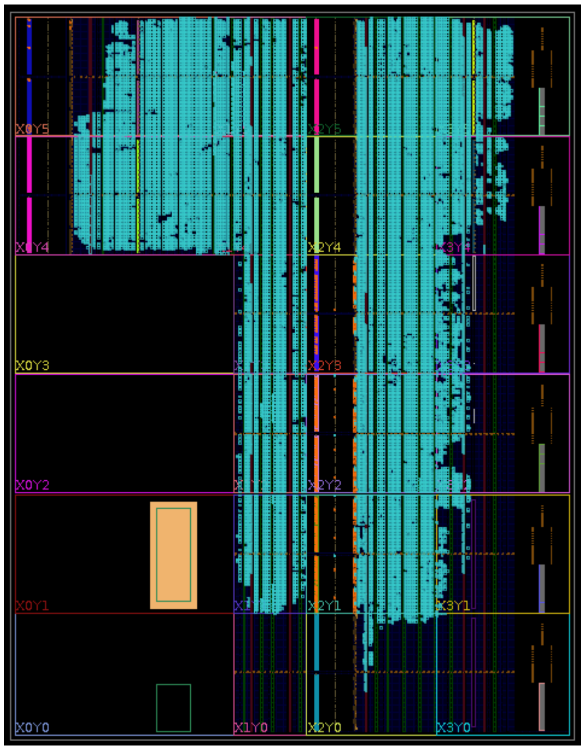


Figure 2: TAGE Predictor ZCU106 Utilization

Figure 3, 4, and 5 represent the synthesized results of all six branch configurations for the Small BOOM Core ZCU106 synthesis. As expected, the null predictor scheme had the least amount of logic cells utilized. Surprisngly, the global predictor had a lower power consumption. The TAGE prediction scheme had a much larger utilization count than most predictors, specifically including many more registers utilized, leading to a higher power consumption.

IV. Changes to Proposal

As suggested by Dr. Duwe, we decided to add a TAGE branch predictor as one of our evaluations since ChipYard includes TAGE as a default predictor for the BOOM Core. We also decided to not implement a Random predictor as this would lead to more time developing the design in Chisel, versus evaluating our results for more commonly used predictors, such as

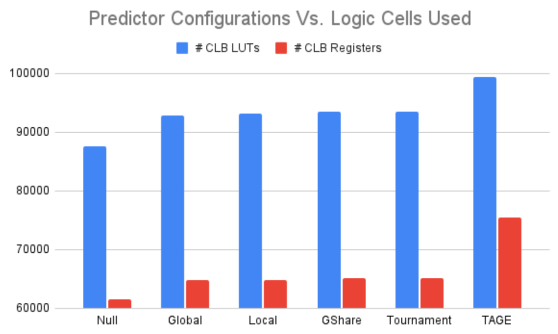


Figure 3: Branch Prediction Utilization Results

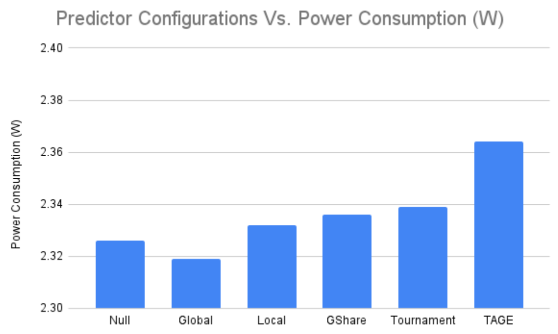


Figure 4: Branch Prediction Utilization Results

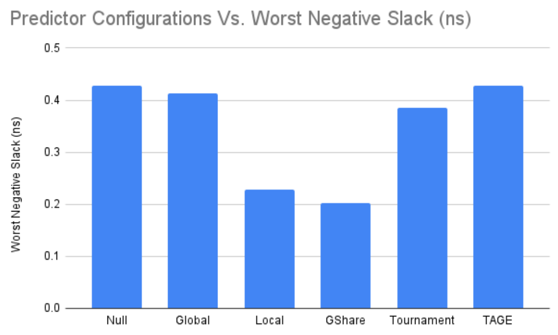


Figure 5: Branch Prediction Utilization Results

TAGE, Gshare, and Tournament configurations. The overall plan has stayed the same, with more clarity on how to implement the Boom Core on an FPGA by using the ZCU106 in Durham 310, utilizing Jordan McGhee's previous work. We still plan to implement the same set of branch predictors in Chisel, and analyze their power, area, and timing delays in synthesis using Vivado. Alongside this, we still plan to evaluate performance metrics using SPEC benchmarks by utilizing FireMarshal and Linux on the BOOM Core to run our results.

V. Individual Contributions

A. *Gregory*

- Installed Chipyard on X Drive, tools build on the linux lab computers. Vivado generated a bitstream and provided implementation details including utilization reports and a pretty picture of the FPGA usage.
- Found installation instructions for FireMarshal, used Jordan's prebuilt firemarshal for the ZCU106 to boot the FPGA with out bitstream
- Cross compiled SPEC2006 bzip2 benchmark and tried running it - it crashed horribly, but started to run for 30 minutes.

B. *Thomas*

- Installed Chipyard repository and run sample Verlator simulation
- Compiled Quicksort Algorithm test to RISC-V and ran it on a Small BOOM with Verilator
- Started to learn Chisel and understand architecture for processors within Chipyard.
- Helped brainstorm alternatives to resolve issues of size and configurations
- Helped synthesize various branch predictor configurations in Vivado

C. *Jake*

- Contacted Jordan McGhee for ZCU106 Chipyard Interface
- Installed Chipyard repository and ran sample Verilator simulations
- Researched more about Chisel and how to model branch predictors in BOOM Core configs
- Gained key card access to Durham 310 for ZCU106 access
- Helped determine viable FPGA solutions for running BOOM Core
- Helped synthesize branch predictor configurations in Vivado for ZCU106