Design and Tapeout of Modulation Recognition Accelerator Integrated into RISC-V SoC Using 28nm CMOS

Olivia Yang, Alec Bender, Majd Almudhry, Julia Anitescu, Rebecca Dettmar

Abstract - This paper presents the design and integration of a RISC-V processor and hardware accelerator ASIC for real-time modulation recognition, targeting low-latency and power-efficient classification of radio signals. Two chips were developed from RTL to layout using TSMC 28nm technology, leveraging Catapult HLS and Chisel/CIRCT for high-level synthesis. Version 1 integrates a VexRISC-V CPU and a Residual Neural Network (ResNet) accelerator within a 1mm × 1mm die area. Version 2 extends the architecture by incorporating a Strip Spectral Correlation Analyzer (SSCA) accelerator, increasing the die size to 2 mm × 1.25 mm. Both designs were functionally verified in simulation, with Version 1 passing DRC and LVS checks to qualify for tapeout.

I. INTRODUCTION

The Modulation Recognition Acceleration project aims to integrate two specialized accelerator intellectual properties (IPs) with a RISC-V processor to improve performance for signal processing workloads as well as optimize the overall system for size, weight, and power. The two specialized accelerators are a hardware Residual Network (ResNet) for modulation classification, and a hardware implementation of the Strip Spectral Correlation Algorithm (SSCA) for cyclostationary feature extraction.

With the increasing complexity of wireless communication and constrained spectrum resources, accurate and efficient modulation recognition is essential. Traditional software approaches lack the speed and energy efficiency required for real-time applications, particularly in edge or defense environments. This project addresses these limitations by designing an ASIC that integrates digital signal processing and machine learning in hardware.

Version 1 of this project integrated a VexRISC-V open source CPU with the ResNet, and was taped out with complete pre-silicon verification. Version 2 extended Version 1 by integrating the SSCA IP with Version 1, and was verified in RTL simulation, but did not meet post-synthesis functionality. The specifics of the accelerator IPs, design, and integration will be discussed in section (III), with performance metrics in section (V).

II. PRIOR WORK

There are two previous implementations of this design, one written in Python and running on an NVIDIA Jetson AGX Orin and the other implemented as an FPGA-based SoC of each accelerator separately. The FPGA-based SoC with the ResNet accelerator was 2-5x faster than the reference implementation, with different configurations of the ResNet data flow accounting for the range of speedup. The FPGA-based SoC of the SSCA accelerator was 20x faster than the reference implementation.[1]

III. DESIGN

The proposed system integrates three major functional blocks:

- **ResNet Accelerator**: Implements a convolutional neural network trained to recognize modulation schemes from IQ data inputs. Input data is 512 bits in fixed-point representation, and the accelerator outputs a probability distribution over signal types. RTL is generated using Catapult HLS from a PyTorch model.
- **SSCA Accelerator**: Implements a radix-8 Strip Spectral Correlation Analyzer to extract cyclostationary features. It operates on 133,120 32-bit floating point inputs and is built using Chisel and CIRCT.

- **RISC-V Processor:** A VexRISC-V open source core that is generated using SpinalHDL for configurable features. Implements RV-32I ISA.

These blocks are connected to memory-mapped SRAMs and MMIO control units via a shared Wishbone bus. I/O to and from the chip is handled using the SPI protocol.

IV. VERSION 1 SYSTEM ARCHITECTURE

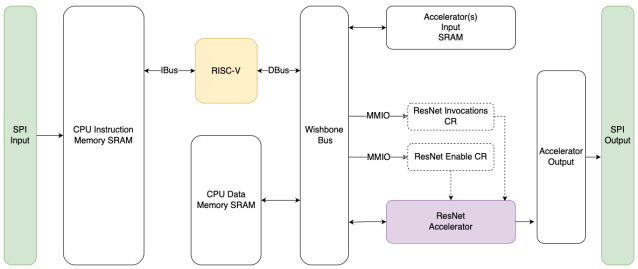


FIGURE I
VERSION 1 SYSTEM BLOCKS DIAGRAM

A. Memory System

The RISC-V core accesses a separate instruction memory and data memory using the VexRISC-V's respective instruction and data bus. The CPU data memory is connected to the shared Wishbone bus and can be read and modified by memory instructions that address into the range dedicated for the CPU data SRAM.

Similarly, Version 1 has input data for the ResNet accelerator that is populated into the input SRAM by using store instructions addressing into the dedicated address range for the ResNet input SRAM. By using this memory-mapped architecture, input data for the hardware accelerator will be controlled by writing software that addresses into specified memory locations.

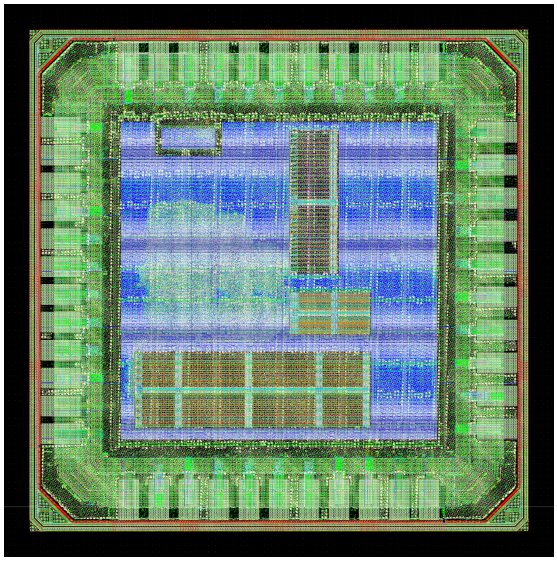
B. Accelerator Control

Control for the ResNet accelerator is handled using memory-mapped IO, where store instructions that are addressed beyond the range of the memory system of the SoC (including CPU data memory and ResNet input SRAM) are interpreted as control signals for the accelerator. A store instruction to the address 0xFFFF_FFFF with a data payload of 0x1 is decoded as setting the enable signal for the ResNet accelerator. A separate control register holding a nine-bit value corresponding to the number of desired ResNet invocations is configured by writing the number of batches as the data payload of a store instruction to the address 0xAAAA_AAAA. This number of invocations must match the number of batches of input data stored into the ResNet input SRAM, which can hold up to 254 batches of input data. When the ResNet accelerator is enabled, all batches will be processed in order. Effectively, these control registers are configured using software and specifically addressed store instructions.

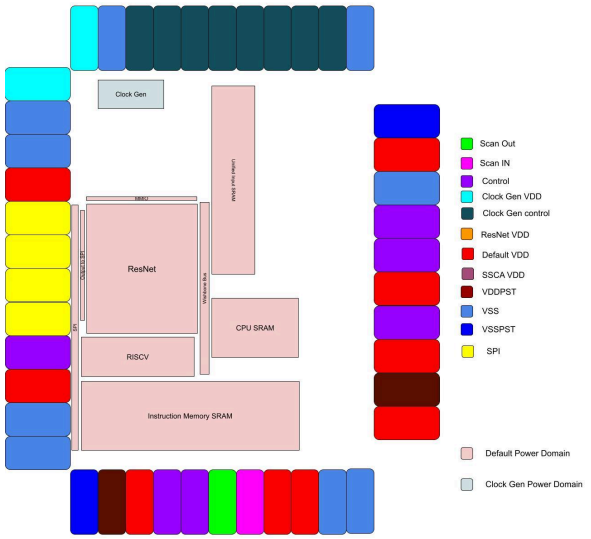
C. Accelerator Output Data

When the ResNet accelerator is enabled, each batch of input data is read from the input SRAM by an input SRAM controller that keeps track of the current read address and handles SRAM control signals, and the input data is processed through an invocation of the ResNet accelerator. When the ResNet accelerator asserts a done signal, the output is stored in an output data register. This register is part of the on-chip SPI slave module, and the data is output live to the off-chip master whenever the chip-select line (NSS) is asserted and the master is requesting data. Following the SPI protocol, the data is shifted serially and sent off-chip via the MISO pin. The output data is sent as four 32-bit words in little-endian order, with the least significant word transmitted first. Since the data is output live, the master must sample the data from the slave periodically.

D. Chip Integration and Layout of Version 1



(a)



(b)

FIGURE II
(a) VERSION 1 FULL CHIP LAYOUT (b) FLOORPLAN

The floorplan of the 1mm × 1mm chip was carefully organized to minimize latency and routing complexity by placing core logic blocks adjacent to the I/O interfaces. Specifically, the VexRISC-V CPU and ResNet accelerator, were positioned near the SPI interface pins to reduce critical path lengths for data ingress and egress. Surrounding each logic block, corresponding SRAMs were placed to ensure local data availability with minimal wire routing congestion. This proximity-driven layout strategy helped reduce parasitic delays and enabled efficient power distribution through domain-specific rails.

V. VERSION 2 SYSTEM ARCHITECTURE

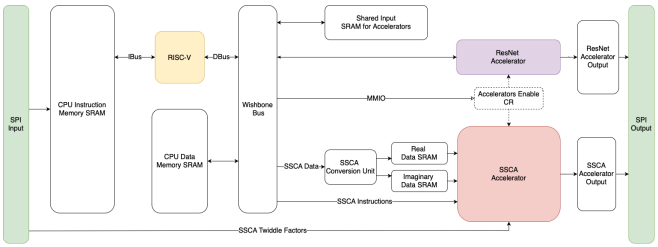


FIGURE III
VERSION 2 SYSTEM BLOCKS DIAGRAM

For the integration of the expanded system that incorporates the SSCA accelerator along with the CPU and ResNet accelerator, several design changes were made to global integration as well as at the SSCA block level.

A. *SSCA Instruction Memory*

At the global integration level, one change that was made is the connection of the SSCA instruction memory to the Wishbone bus. The SSCA requires its own set of custom instructions which can be reconfigured, and these are streamed using the same SPI interface as the CPU instructions. The SSCA instructions follow a similar data flow as the input data for the accelerators, where store instructions are processed by the CPU which then stores data to a memory-mapped SRAM that is dedicated for the SSCA instructions.

B. *SSCA Local Memory Pre-population*

At the SSCA block level, a local memory SRAM must be populated before an invocation of the SSCA accelerator is called. The full local memory must be initialized with the correct twiddle coefficients required for the fast Fourier transform upon which the SSCA is based before any calculation is initiated. To integrate this into the overall system, a separate SPI interface is dedicated for this purpose, with a complete set of SPI pins set aside for the SPI slave module that handles writing to this local memory SRAM, and an initialization mode is used to indicate when to write those coefficients directly into the local memory.

C. *Floating Point Converter & Shared Input SRAM*

The SSCA requires 32-bit floating-point data, while the ResNet requires 16-bit fixed-point data. To accommodate this while still guaranteeing the same data is used for both accelerators, the chip accepts input data as fixed-point and a fixed-to-float converter is connected to the SSCA's input buffer.

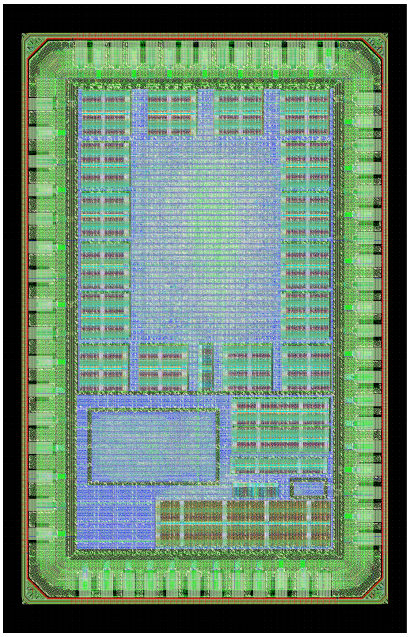
Additionally, since the ResNet accelerator and the SSCA accelerator use the same input data, both use a single input data SRAM. Control units are implemented to facilitate the data transfer and reads from the shared SRAM to the ResNet or SSCA accelerator.

D. *Accelerator Control and MMIO*

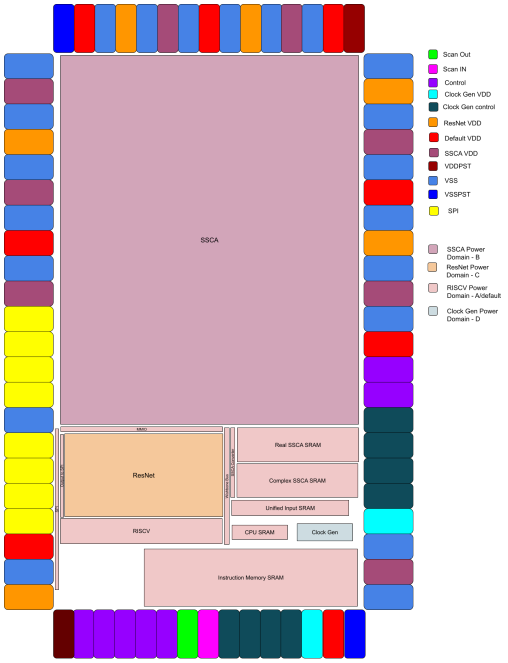
In the expanded system, the accelerator control signals are shared between both the SSCA and ResNet accelerators. When the accelerator enable control register is set, both accelerators are enabled.

When the accelerators are enabled, an input handler coordinates reads from the shared input data SRAM to move batches of input data to the ResNet accelerator and all input data into the SSCA accelerator's input buffer since the SSCA requires its input buffer to be full before it starts computing. Once calculations are completed, the SSCA flags that it is done and the output data can be found in the local memory. A stream-out mode, when activated, allows addresses in local memory to be read directly to the top-level and streamed out through the SPI interface, so no dedicated output buffer is required.

E. *Chip Integration and Layout of Version 2*



(a)



(b)

FIGURE IV
(a) VERSION 2 FULL CHIP LAYOUT (b) FLOORPLAN

In addition to the inclusion of the SSCA and an expanded SPI interface, Version 2 introduces dedicated power domains for each of the major logic blocks, enabling improved power efficiency through selective power gating. The design from Version 1 is fully retained but spatially condensed into the bottom third of the chip, freeing up the upper two-thirds to accommodate the new SSCA module. One of the primary layout challenges in integrating the SSCA was its requirement of 20 separate SRAM blocks (16 banks of local working memory, an additional small scratch memory, and instruction memory as part of the preexisting IP; further, two SRAMs were included to store type-converted input data), totaling over 250 KB of on-chip storage. To address this, the SRAMs were strategically arranged in a ring around the SSCA logic, minimizing routing complexity and reducing access latency to support high-throughput spectral analysis.

VI. Simulation Results

A. Functionality Simulation (Block-Level)

We were able to verify the functionality of both the ResNet and the CPU individually, as well as the entire chip-level design. For the ResNet, the testbench converts the provided test vector inputs from floating point to 16-bit fixed point, feeds it to the core ResNet RTL, and converts the output from 34-bit fixed point to floating point. The output is then compared to the expected output generated from a software golden model. The following results is what we got from testing, which is accurate enough for the purposes of this class:

TABLE I
RESNET SIMULATION RESULTS FOR PROVIDED TEST VECTORS

Expected Output	Simulated Output
0.3183179795742034912	0.317378
0.05524525791406631470	0.0558033
-0.3927243053913116455	-0.39354

For the CPU, we ran several assembly programs borrowed from 18-447 that focus on testing basic CPU functionality and verified the results by monitoring the resulting register file state. In RTL simulations, the design passes basic arithmetic tests, branch tests, and memory tests.

B. Functionality Simulation (Chip-Level)

When conducting chip level verification in simulation, we wrote RV-32I assembly programs that test major aspects of our intended workloads: CPU workloads, ResNet workloads, and SSCA workloads. These programs were then input to the CPU Instruction SRAM by serially transmitting the 32-bit instructions over SPI from the testbench's off-chip SPI master to the on-chip SPI slave using the SPI protocol.

For CPU workloads, we used assembly programs as described above that test basic functionality of the RISC-V processor. We then compared the register file of the VexRiscV to the expected register dump for the program being tested.

For ResNet workloads, we wrote assembly programs that store batches of input data to the address range dedicated for the ResNet input SRAM, then use MMIO instructions for setting the ResNet enable control register and configuring the register holding the number of invocations. The output via SPI is then captured by the testbench using a SPI master that selects the chip and waits for the 4 words of output data per invocation to be transmitted. The output data captured off-chip is then compared to the golden model's test vector output.

For testing the SSCA workload from the chip's top level, two different programs of 32-bit SSCA-specialized instructions were loaded into the SSCA's instruction memory using the input SPI. These programs were set to do different things: one to read exclusively from the input buffer, and one to read exclusively from local memory. This was to confirm if both types of the functionality of the SSCA were performing as expected and did not interfere with the functionality of the other. The resulting values stored in local memory once the SSCA was done computing were then corroborated by Larry Tang's testbench, to ensure that the simulated SSCA was leading to the correct output numbers. Finally, scanout of local memory was tested from the top level, to ensure that it was not interfering with the functionality of the ResNet and CPU, and could be sent to the outside of the chip.

These simulation steps were completed at the RTL, post-synthesis, and post-PnR level. All ResNet and CPU workload tests passed for RTL, post-synthesis and post-PnR simulation, and all SSCA tests passed for RTL simulation.

C. Power Simulation

We ran Voltus on our design, which reported the following breakdown of the total power:

TABLE II
VERSION 1 POWER RESULTS

Internal Power	6.1473mW	72.0845%
Switching Power	1.9762mW	23.1736%
Leakage Power	0.4043mW	4.7419%
Total Power	8.5279mW	100%

The VDDA power domain which consists of all the compute logic and the memories consumes 53%, the VDDCLKGEN power domain, hosting the clock generator

consumes 0.1651%, and the IO consumes the remaining power.

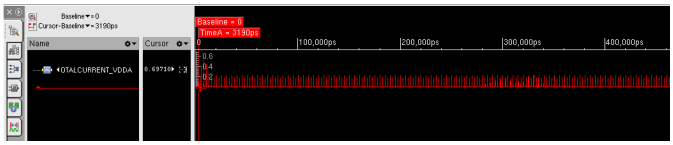


FIGURE V
WAVEFORM OF TRANSIENT CURRENT

D. Comparison with Previous Work

Overall, the performance of each individual block as well as the whole chip all satisfies or exceeds our target specifications while room for improvement remains.

E. Timing Reports

```
                    timeDesign Summary
--------------------------------------------------------------------

+--------------------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  1.108  |  1.326  |  2.395  |  1.108  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |
|          All Paths:|  16880  |  16078  |   485   |   363   |
+--------------------+---------+---------+---------+---------+

+----------------+-------------------------------+------------------+
|                |              Real             |       Total      |
|    DRVs        +------------------+------------+------------------+
|                |  Nr nets(terms)  | Worst Vio  |  Nr nets(terms)  |
+----------------+------------------+------------+------------------+
|   max_cap      |      0 (0)       |   0.000    |      0 (0)       |
|   max_tran     |      0 (0)       |   0.000    |      0 (0)       |
|   max_fanout   |      1 (1)       |     -32    |      1 (1)       |
|   max_length   |      0 (0)       |      0     |      0 (0)       |
+----------------+------------------+------------+------------------+
```

FIGURE VI
SETUP TIME REPORT

```
--------------------------------------------------------------------
                    timeDesign Summary
--------------------------------------------------------------------

+--------------------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.004  |  0.004  |  0.041  |  1.425  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |
|          All Paths:|  16880  |  16078  |   485   |   363   |
+--------------------+---------+---------+---------+---------+

Density: 18.263%
       (99.748% with Fillers)
--------------------------------------------------------------------
```

FIGURE VII
HOLD TIME REPORT

F. Chip Stats

TABLE III
SUMMARY OF CHIP STATISTICS

Process	28nm
Die Area	1mm^2
IO	43
Memory	610 Kb
Clock Frequency	200 MHz
Power	8.5279 mW

VII. ETHICAL ISSUES

There are certain privacy issues associated with any technology used to decode intercepted wireless transmissions. The intent of this project is for use in a defence context, but there are no features inherent to a demodulator itself which prevent it from being used to decode civilian communications, as many modulation schemata are used for both civilian and military transmission. In an extreme case, it could be potentially used for mass monitoring of all communication frequencies. It also could potentially intercept frequencies related to medical devices, such as pacemakers.

We plan to work around this by restricting our scanning frequencies and radii to those utilized by the edge and defence environments in which it is deployed, as these frequencies are not generally used by general-purpose communications, nor by medical devices.

VIII. CLASS FEEDBACK

Feedback that we have for the course include:

- Additional office hours with a single TA over multiple days rather than a single office hour each week with all the TAs
- Earlier notice of the available projects before the start of the semester in order to devise teams more efficiently and start capstone projects sooner

IX. Individual Contributions

The following table summarizes the roles and responsibilities of each team member throughout the chip development process:

TABLE IV
ROLES AND RESPONSIBILITIES IN CHIP DEVELOPMENT

RTL Design and Verification	
RISC-V CPU	Olivia
ResNet	Alec, Majd
SSCA	Julia, Rebecca (+ Larry)
SPI	Alec, Olivia
Wishbone Bus	Olivia
SRAM	Alec
Top-level Verification (Pre, Post Synthesis)	
ASIC Version 1	Olivia
ASIC Version 2	Julia, Rebecca
PnR Flow Development	
ASIC Version 1 Layout	Alec
ASIC Version 2 Layout	Alec
ASIC Version 1 Signoff	Olivia, Majd
ASIC Version 2 Signoff	Alec
ASIC Version 1 DRC, LVS	Everyone

X. Acknowledgement

The authors would like to express their sincere gratitude to Apple, Dr. John G. Wohlbier, and Larry Tang for their generous support and for providing the essential resources that made this project possible.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

XI. References

[1] J. G. Wohlbier et al., "Co-design for Edge AI: Modulation Recognition," presented at the 2025 Government Microcircuit Applications and Critical Technology Conference (GOMACTech), Pasadena, CA, Mar. 17-20, 2025