RISC-V Microcontroller and Encryption Accelerator with Integrated ADC, DAC and On-Chip References

Du, Larry
Electrical & Computer Engineering
University of Illinois
Champaign, United States
larryrd2@illinois.edu

Gohil, Shiv
Electrical & Computer Engineering
University of Illinois
Champaign, United States
sgohil3@illinois.edu

Ma, Max
Electrical & Computer Engineering
University of Illinois
Champaign, United States
maxma2@illinois.edu

Nguyen, Binh-Minh
Electrical & Computer Engineering
University of Illinois
Champaign, United States
bmn4@illinois.edu

Sasic, Milosh
Electrical & Computer Engineering
University of Illinois
Champaign, United States
msasic2@illinois.edu

Yun, Joshua
Electrical & Computer Engineering
University of Illinois
Champaign, United States
jjyun4@illinois.edu

***Abstract*—This report presents the design and development progress of a RISC-V microcontroller and encryption accelerator featuring integrated Analog-to-Digital Converters (ADC), Digital-to-Analog Converters (DAC), and on-chip voltage and current references. The project includes a comprehensive overview of the design process, documenting both the analog and digital portions of the chip. Key components of the report include a progress summary detailing the milestones achieved, a project timeline tracking major stages of development, individual contributions from team members, and the primary challenges encountered during the design process.**

***Index Terms*—RISC-V microcontroller, RSA, DAC, ADC, Voltage Reference, Current Reference**

I. Project Summary

A. Main Features and Functionality

The chip is a microcontroller with basic cryptography acceleration. It supports basic RISC-V instruction sets and various peripherals that allow it to communicate with common sensor interfaces, and supports interrupts from those sensors. Basic analog features are also supported such as digital to analog (DA) and analog to digital (AD) conversion, with those devices being memory mapped peripherals inside the chip. There is also a embedded RSA accelerator within the chip to improve the performance of encrypting and decrypting data to be sent elsewhere.

B. High-level Architecture

Due to the mixed signal nature of the chip, it is roughly split into two parts: the digital and analog sides.

1) Digital: The digital section is sectioned into three main parts, the core itself, with its caches, the RSA accelerator, and the memory mapped bus which attaches itself to the various peripherals. The core executes instructions given to it from the QSPI interface. Instructions that access memory are intercepted by the memory bus, which decides either to route the request to normal memory to perform an action on a peripheral based on the address in question (RSA accelerator, UART controllers, ADC/DAC control registers, etc).

2) Analog: The analog side of the chip has three subsections laid out with significant noise isolation between the various sections. Those sections are: DAC, ADC, References. The reference part of the chip provides reference voltages and currents to the other analog sections previously mentioned, which then complete their sampling by interfacing with outwards facing pins.

C. Physical Design

See the physical layout. On the left you see the digital design, complete with dense power grid and dummy metals. You can see our two SRAMs for our instruction memory and data memory. On the right you see the analog design, with a much sparser power grid to prevent excessive capacitance on sensitive middle voltage nodes. Taking up the top half you see the large ADC capacitive DAC arrays. Near the bottom houses the DAC. The power supplies are quite small and are hard to see in the image.

Digital Core Statistics:

- Area: (764 x 414) = 316296 um²
- Freq: 125 MHz
- Coremark IPC: 0.68
- Coremark Power: 7.041661 mW
- RSA Power: 8.5238 mW
- GPIO: 10.518 mW
- I2C: 11.6037 mW
- SPI: 9.5864 mW
- UART: 9.0851 mW

D. Design Choices and Justification

We wanted to challenge ourselves by making a mixed signal microcontroller. On the digital side, we decided on I2C, QSPI, UART peripherals to support common microcontroller functionality. RSA was added for research and exploration

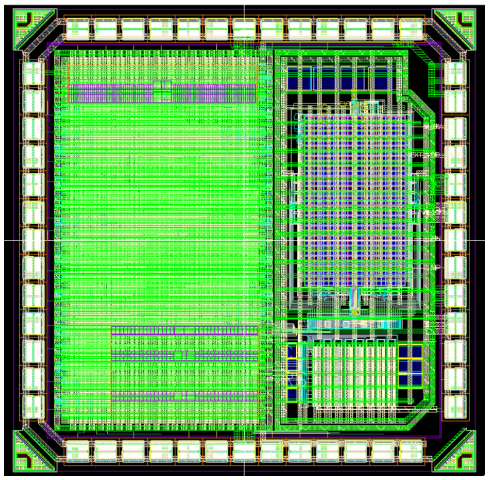

Fig. 1. The chip's top level physical layout and design.

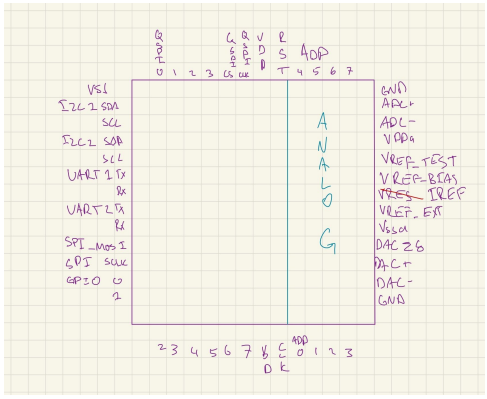

Fig. 2. Pins in the chip.

into possible security interfaces for low power mixed signal microcontrollers. One critical design decision we made was to keep the digital core small, about half of the total allotted area. This allowed ample space for the analog team to develop and design freely.

Another key design decision was for the digital core to implement its own simpler interrupt scheme than the one defined by the RISC-V specification. This is because the use case for a microcontroller will have it always operating at the highest privilege, meaning that the specifications for RISC-V only made the development process more difficult without any real benefit. Since we are the ones controlling how to write the C code, we believed that having an extremely simply interrupt protocol would ensure the highest chance of success while proving that we were capable of enabling interrupts on a processor.

On the analog side, we decided on three major components: ADC, DAC, and source references. We chose these components because these are common components required for a basic mixed signal application, such as in medical devices. For each component specifically, the scope and overall design of the component was determined through extensive research on existing technologies and through discoveries while performing simulation. For example, we decided on a differential ADC due to its natural resistance to offset and mismatch variation. When deciding SAR vs flash, we settled on SAR due to its scalability. When deciding the resolution of our ADC and DAC, 8 bits was selected to match many common low power mixed signal microcontroller devices developed in research. Offset calibration was added after discovering a significant offset mismatch following Monte Carlo simulation.

E. Verification and Test Strategy

1) Analog: Analog components were verified through SPICE and spectre simulation techniques. Extensive testbenches were developed with Cadence Xcelium. Each component required different verification suites:

a) ADC: Comparator offset error, gain error, mismatch error, input-referred noise, kickback noise, offset calibration granularity, offset calibration range. Capacitive DAC array mismatch error, charge sharing loss. ADC DNL, INL, ENOB, SINAD, SNR, THD, reset sequencing, calibration sequencing, external voltage sequencing.

b) DAC: DAC DNL, INL, ENOB, SINAD, SNR, THD, settling time, transient noise analysis.

c) Source References: PSRR, Temperature Drift, Process Variation offset, settling time, drive strength.

2) Digital: Digital team used a mixture of directed tests combined with randomized system tests to best cover the design area of the processor in the given amount of time.

Here is a list of the benchmarks and tests used to verify the processor and peripherals: RISC-V benchmarks, example C microcontroller C code, RSA encryption tests, full interrupt system tests, UART/I2C/SPI Peripheral C tests, mixed signal with analog peripherals.

F. Comparison with Proposal

Most of our initial ideas made the final tapeout. What was removed was the asynchronous FIFO and the dedicated TRNG. Everything else, including I2C, QSPI, UART, SPI, interrupts, RSA, and all analog components were included. However, we had changed our SPI peripheral to write only.

II. Post-Silicon Validation Plan

A. FPGA Interface

The boards will interface with an FPGA that is able to have debug visibility into the system, printing out USB values to either a attached computer or screen. The FPGA will also be responsible for the initial programming of the QSPI chip with the desired microcontroller program, as well as monitoring the QSPI during microcontroller reads to verify that the chip is obeying the specifications set by the chip vendor, and give additional debug visibility.

Additionally, we can utilize the Acadia Debug Port to extract additional information from our chip such as Core register reads and Data Sram reads and writes.

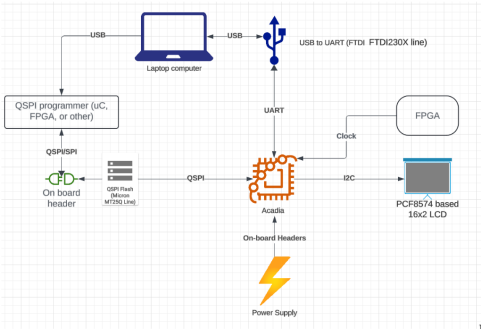

Fig. 3. Bringup Plan Simple Board

B. Board Design

The post silicon validation plan has two parts to demonstrate functionality, a simple bringup board to test basic working functionality, and a board designed to characterize the maximum performance of all the chip components.

The simple bringup board will have a test program such as an LED blinky program that will be designed to exercise all the peripherals in a simple way, minimizing the chance of board complications. An example of the bringup board can be seen in figure 6.

This is in contrast to the profiling board, which will have much more extensive components design to find the maximum ratings of various perhipherals with a emphasis on finding the performance metrics of the analog peripherals. The board design will be done by Joshua, Larry, and Binh-Minh, who have previous board experience, while the RTL can be done by Max and Shiv, while Milosh is able to run actual tests in the lab to benchmark the analog peripherals.

III. Individual Contributions

Larry Du: SAR ADC, Mixed Signal Analysis, Layout
Binh-Minh Nguyen: RISC-V Processor, Peripherals
Milosh Sasic: Current and Voltage References
Joshua Yun: Current Steering DAC, PNR Toolchain
Shiv Gohil: RISC-V Processor, RSA Accelerator, Interrupts
Max Ma: RISC-V Processor, Debug Interface

IV. Major Challenges

A. DAC

The major challenge with the DAC was noise. There are two main types of noise that a DAC has to deal with that degrade the performance of the DAC. The first issue is thermal noise, which is white noise with equal strength for all frequencies. This is directly related to the transconductance, or gm of the transistor. Secondly, there is the issue of flicker noise.

Flicker noise is caused by charge carriers becoming stuck in traps before being suddenly released all at once, causing a gradual decrease followed by a sudden increase in current. This type of noise is strong at lower frequencies, in part because at higher frequencies there is not enough time for a charge buildup to occur and create noise.

Both types of noise are extremely detrimental to the operation of the current mirror, initially causing 10% peak to peak noise errors on the DAC. In addition, since the DAC is a fully thermometer, the noise is able to compound on itself, meaning at the maximum current value of 10mA, there is a maximum noise of 70uA, or the equivalent of 1.5LSBs of noise.

There are multiple strategies used to deal with the problem of noise on the DAC. The main technique used to deal especially with flicker noise is to simply make the transistor larger. Having a larger gate area means that the effects of the traps average over a larger area, meaning that the individual traps' effects on noise are greatly lessened. The other technique used to great effect is to create a low pass filter at specific outputs.

This was achieved in the mirroring circuit with a MOSFET with a precise biasing circuit to emulate an extremely large resistor, along with a small capacitor to create the desired effect. These types of filters were mostly used to deal with thermal noise generation, as they were present with much higher frequencies compared to flickering noise. The last method to deal with noise is to have a lower gm, since gm is defined by 2Ib / Vov, where Ib is the mirrored current, the only option available is to increase the overdrive voltage, which is done by decreasing the W/L ratio. However, doing this has its own tradeoffs in that it further decreases the output swing of the current mirror, limiting our maximum voltage compliance.

B. ADC

The major challenge for the ADC was ensuring accuracy of the charge sharing effect. Because the top plates of the capacitive DAC were floating during evaluation phases, any charge injected onto these plates would accumulate and could eventually cause an incorrect digital decision. The charge sharing accuracy was compromised by several factors, in order of severity: Charge injection from the common mode switch. Kickback noise from the comparator. Transient positive or negative voltage spikes on capacitive DAC bottom plates. VDD/2 accuracy.

Due to the large switch required to precharge the capacitive DAC to VDD/2, the parasitics were not insignificant. When the switch turned on or off, parasitic capacitances observed a charge pumping effect that dumped charge onto the DAC plates. The result was an average voltage of 1.007 V for VIP + VIN, highly unacceptable as the 8 bit LSB resolution is 4 mV. With implementation of a dummy switch at half size, and matching of parasitic capacitances for the P and N mosfets of the switch, charge injection was mitigated to observe an average voltage of 1.001 V.

Kickback noise from the comparator resulted in a momentary positive or negative charge effect on the DAC plates. Although not as severe as the switch charge injection, it still resulted in voltage inaccuracy of around 3 mV. Kickback noise was mitigated by adding a preamplifier.

The large DAC capacitors would overpower the bottom plate switches, momentarily inducing transient voltage spikes on bottom plates that would exceed VDD and VSS. Worse, these spikes would depend on capacitor sizing, current evaluation

state, and the comparator switching decision. Strategies including delayed switching schemes with a coarse comparator, wildly beta values for different switches, and customized sizings for all switches proved relatively effective, but were ultimately excluded from the design due to the lack of time in establishing robustness of these ideas. Instead, the rudimentary strategy of increasing the size of the switches mitigated the transient spikes to an acceptable level.

Degradation in accuracy of the VDD/2 reference acted to change the common mode of the ADC. While the differential nature of the ADC already mitigated this effect, differing common modes would exacerbate the other issues described above by changing parasitic capacitance injection and influencing either the positive or the negative transient spikes in the wrong direction. A switch was implemented that can select between the on-chip VDD/2 reference and an off-chip reference, in case the on-chip VDD/2 variation proves to be too detrimental to ADC performance.

C. Source References

Many challenges were encountered. First and foremost, finding a reasonable architecture for the references. The biggest reason for this issue is the small supply voltage, which causes issues with MOS device saturation. For this reason, many transistors operate in the subthreshold region for the voltage reference, which in turn creates the possibility for larger deviations from ideal values due to process variation. For this same reason, high values of PSRR at lower frequencies is hard to implement, due to the architecture chosen, which heavily relies on the overdrive voltage of the PMOS device separating the supply from the reference. For this reason, a small threshold voltage was chosen here to ensure a high enough voltage reference, but that also maximizes the PSRR and minimizes variation due to process. The main challenges encountered with the current reference were the biasing and noise reduction of the error amplifier, and ensuring proper stability for all possible values of packaging capacitance. These two issues also contributed to the detriment of the specifications for the other, thus careful sizing and placement for this circuit was done to ensure correct functioning. Noise remains a non-negligible issue, which affects the current. Techniques for minimizing noise were employed, such as additional capacitors.

D. Interrupts

There was a significant challenge in trying to merge Interrupts into our processor's fetch stage as the fetch logic wasn't built with interrupts in mind. As a result, a lot of changes and edge cases was needed to be implemented. If we started collaborating on the fetch stage earlier, then we could have thought about these implementation issues ahead of time and circumvented them.

E. Repository Management

We are using a Git repository to store our work. Everyone is working on their own feature branches. This causes some issues when interdependent features are being developed. We have had some issues syncing between different features. We had one big issue with a feature merge being reverted "temporarily" but was not undone correctly. It was pushed to the main branch and caused some issues. This will be mitigated in the future by stricter merge request approval.

F. Design Philosophy

During our digital design process, we had many opportunities to add lots of extra features and capabilities. This would make our design more powerful, but at the cost of complexity. We decided on keeping everything as simple as possible and compartmentalizing parts. For our main execution path (RISC-V core -> Instruction Cache -> QSPI controller), we keep everything as simple as possible since if anything on this path breaks, the entire chip is unusable. Everything else is peripheral, so if they fail, we only lose that capability. This lets us develop more complex modules without risking the entire functionality of the processor.

V. Documentation

A. 8-bit Digital to Analog Converter

The DAC integrated within this controller is an 8 bit current steering fully differential DAC with a current range of 0 to 10 mA. The DAC has a sampling rate of 9 MS/s. The DAC is composed of 255 current mirror cells, due to the high linearity associated with having a full thermometer DAC, though it is possible that we may drop a few thermometer bits to greatly reduce the amount of current sources required.

The DAC is composed of the current cell, which is a high swing cascode current mirror. At the output of the current mirror is a differential pair of MOSFET switches. These pairs of switches direct the current to one of the differential pairs, making the current output fully differential.

To generate the biasing voltages necessary for the high swing mirror, 5 MOSFETS are put in series with tied gates to create an equivalent transistor with W/L values of 1/5 to put the cascode MOSFET well into the saturation region by generating a voltage of Vth + 2Vov.

Since the DAC is a current steering device, it is necessary to have an accurate current source to be mirrored into the DAC current cells. This is achieved using a beta multiplier architecture. The beta multiplier is a pair of cross coupled current mirrors to generate a more accurate current. An external resistor will be provided to generate the required current to properly bias the beta multiplier. In this case, to generate a 10uA reference current, a resistor of 5.5 KOhms will be used.

The DAC also consists of a digital binary to thermometer decoder, which converts a binary number to a vector with the number of ones representing the binary number. This means that the bit width is equal to 2N-1 bits. However, having a single binary to 255 bit thermometer is quite infeasible for both area and fanout. To alleviate these issues, a row column architecture is made, with the rows and columns being a thermometer for 4 bit binary (15 bit thermometer). We then

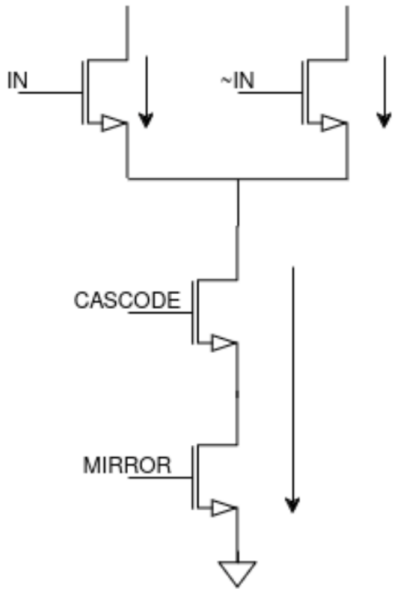

Fig. 4. A schematic of a high swing current mirror with differential pair switches.

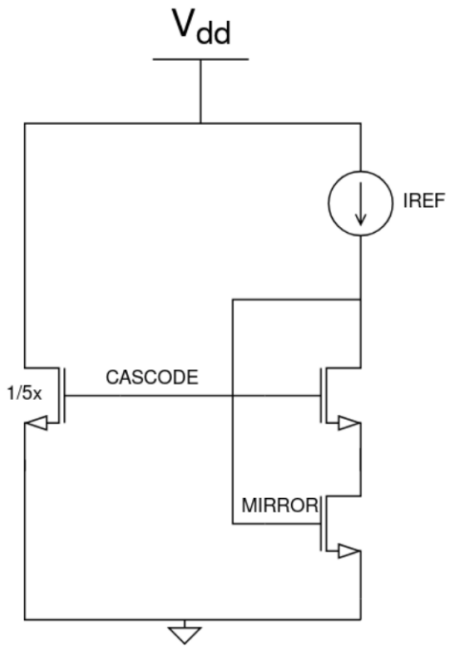

Fig. 5. High swing current mirror biasing circuit. This generates the CASCODE and MIRROR voltages which are then used in the previously shown high swing mirror circuit to mirror the IREF shown here to the DAC current cell.

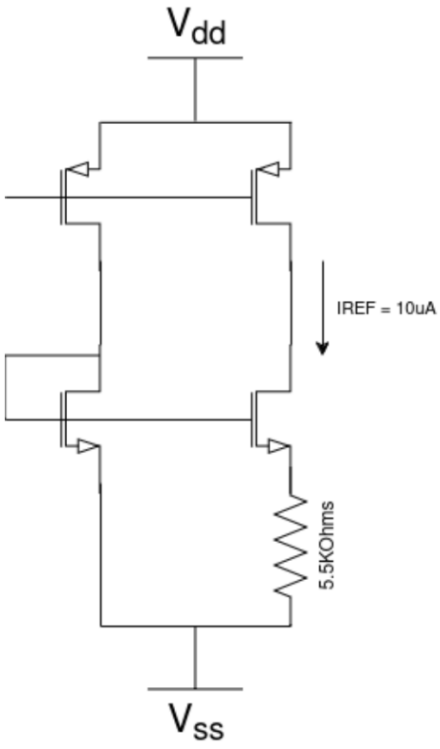

Fig. 6. Beta Multiplier to generate reference current of 10uA. Not shown is the startup circuit required to set the initial current, as well as the voltage amplifier that sets the PMOS voltage bias correctly using feedback.

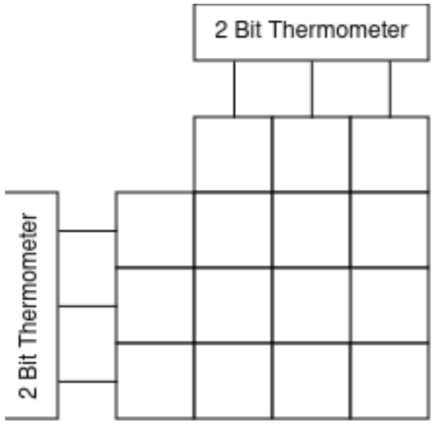

Fig. 7. DAC Matrix architecture as described above. Since we only need 2N-1 units, we remove the 0th DAC cell from the matrix. Each DAC cell has its own small decoder that looks at the aforementioned rows and columns.

have each cell look at its own row, the next row, and its column to determine whether it should turn on.

B. ADC

The ADC is an 8-bit 1 MS/s differential binary-weighted charge redistribution SAR ADC with offset calibration. It uses the capacitive DAC as the sample and hold circuit, eliminating the need to design a separate structure. Control of the ADC is accomplished digitally, and communicates with the microcontroller through a memory-mapped interface.

Under nominal operating conditions, the ADC demonstrates a maximum DNL of 873 mLSB and an ENOB of 7.793. With 10G transient noise, the maximum DNL is 961 mLSB and the ENOB is 7.75. Under 100 Monte Carlo samples, an acceptable 95 samples have DNL under 1 LSB. The ADC performs

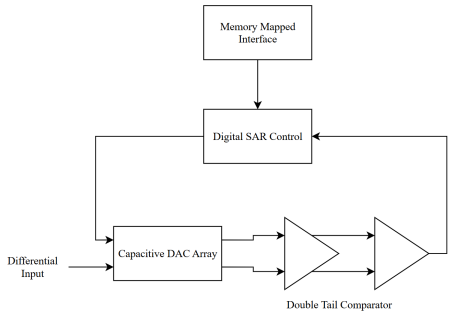

Fig. 8. High-level diagram of the SAR ADC

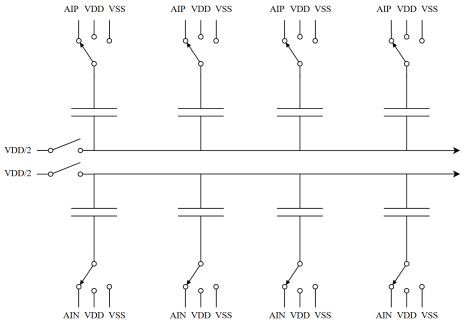

Fig. 9. 3-bit Capacitive DAC

worst in the SS process corner, with a DNL of 979 mLSB. Monte Carlo and process corner ENOB show strong results. VDD ramp and reset logic were also simulated, to ensure proper state on power-up. The digital logic was synthesized and routed with Synopsys tools, and simulated with the ADC to ensure no degradation or glitching.

1) Capacitive DAC Array: The capacitive DAC follows a standard binary-weighted architecture. Define the top plate of the capacitors to be the middle node, attached to the preamplifier-comparator structure. Define the bottom plate to be the nodes with the three-way switches. During the sampling phase, the top plates are pre charged to common mode, VDD/2. The bottom plates follow analog in, both positive and negative inputs, for sampling. During evaluation phases, the top plates are disconnected from VDD/2, leaving them floating and able to perform the charge sharing effect. The bottom plates are connected either to VDD or VSS, depending on the current stage of evaluation and the feedback given from the control block.

2) Double Tail Comparator: The double tail comparator is a traditional design optimized for decent preamplification while maintaining noise immunity and gain. The double tail comparator serves several main purposes: Amplify the differential, thereby minimizing comparator error. Reduce kickback noise from comparator switching. Act as a low capacitance load, minimizing effects of parasitics on capacitive DAC accuracy. Act as a high capacitance source, minimizing adverse comparator load effects. Not shown is the SR latch attached to the output of the double tail comparator, which latches its

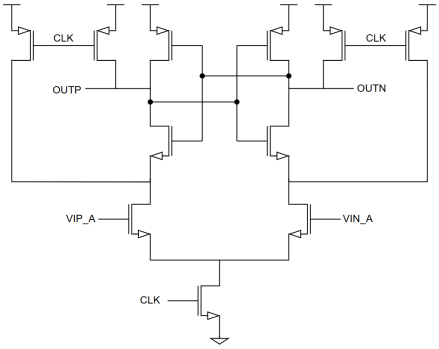

Fig. 10. Double Tail Comparator

decision and prevents setup or hold timing violation to the digital block.

Offset characterization was performed by applying a slow upwards and downwards ramp to the comparator, and averaging the voltage at which the comparator switches its decision. On average, simulated with 200 Monte Carlo samples, the offset is 5 mV, well within the capabilities of the offset calibration. Input referred noise was calculated both with transient noise and PSS tests. Both tests show that the input referred noise is around 500 uV for the comparator.

3) Digital Control Logic: The digital control performs the negative feedback required for the approximation to converge and streams data to the memory mapped interface. A state machine controls the sampling and evaluation phases. During evaluation, it controls the capacitive DAC array switches depending on the current state and comparator decision. The digital block is also responsible for hard and soft ADC shutdown, preventing voltage from exceeding operating values.

C. Source References

In this section, we will discuss the design and progress of the implementation of a voltage reference source for the ADC and a current reference source for the DAC. We opted for these references to be implemented on-chip in order to ease integration of these crucial specifications with the rest of the on-chip devices.

There are several design considerations that we must discuss before we are able to choose and develop a coherent architecture for both the current and voltage references. First and foremost, the supply voltage the references will be operating at is 1V. This voltage was chosen for the entirety of the analog circuitry. References operating at a higher voltage than the rest of the devices on chip could introduce additional sources of noise, and unintentional coupling could degrade the MOS devices of the other on-chip components. Secondly, we wish to optimize our circuits for low variation between the different process corners. Therefore, we will verify that our circuits meet the required variation specs that guarantee correct functioning of the ADC and/or DAC, and do not negatively affect their behavior. Thirdly, if variation is beyond the minimal spec as is often, we should incorporate a means for tuning via off-

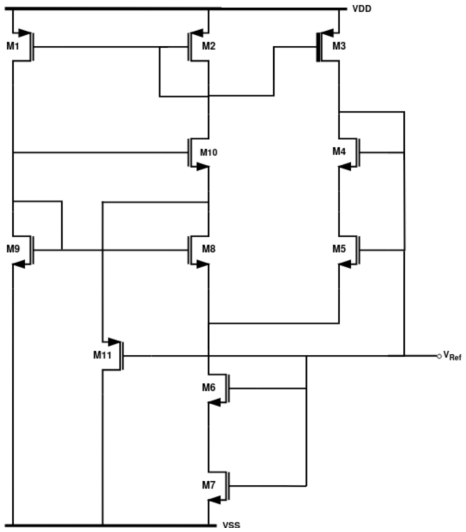

Fig. 11. Voltage Reference Architecture

chip methods. These tuning parameters would be incorporated as a bias voltage that can be adjusted manually, for instance using a potentiometer, and connects to a specific node in the circuit where a manual adjustment in said parameter could change the output to a more favorable value. This in turn would effectively act as a measure that can increase the yield rate of our design. Another measure for decreasing variation is minimizing the amount of different MOS devices, with similar threshold voltages and oxide thicknesses. Lastly, we wish to optimize the circuit operating specifications (power supply rejection ratio, output impedance) such that negative externalities (e.g. noise) are minimized. This requires careful placement and sizing of all devices.

1) Voltage Source: The architecture of the voltage reference follows (Borejko and Pleskacz, 2008). This architecture was chosen due to its ability to operate under low supply voltages, which is crucial for our circumstances. Several modifications have been made to this architecture, duly to the differences in technologies used, and required output specifications. The required voltage reference specifications required for proper functioning of the ADC is the following: Vref = VDD/2 [0.5V] pm 20mV. PSRR is to be maximized to reject as much noise as possible from the power supply. Since the load of the voltage reference will be (for the most part) pure capacitive loads and MOSFET gates, the output impedance will not have to adhere to strict low impedances, however minimizing output impedance is always a plus. The architecture also produces a low current, thus power consumption will be relatively low. One transistor was chosen to have a different threshold voltage, in order to ensure a proper voltage reference (the device is denoted with a thicker line indicating a higher threshold voltage device).

2) Current Source: The current source architecture used resembles that of a classic LDO. However, this is a much simplified version due to a single fact - there is no biasing current to bias the amplifier. In short, this architecture works in

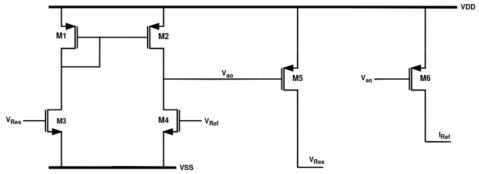

Fig. 12. Current Source Architecture

the following manner: an error amplifier compares the voltage reference, generated by the previously described circuit, with the output of the amplifier. The output of the amplifier works to create a voltage as close as possible to the voltage reference supplied to the amp. The output of the amp in turn is connected back to the input, forming a feedback loop, as well as an off-chip resistor (V_res). The output of the error amp is mirrored to another MOS device, whose current is to be used as a reference (Iref). The reason for choosing an off-chip resistor is simple. First and foremost, this simple practice is done in industry; an off-chip resistor allows for larger resistor values which would constrain the disposable area for other circuitry.

The off-chip impedance must be modeled as the resistor used for reference generation, as well as the package and trace capacitance in parallel. This capacitance can reach values anywhere between 1pF and 100pF, thus in order to guarantee a stable, non-oscillating reference current, we must include a smaller, on-chip capacitor, connected in parallel to the resistor. With this capacitor, as well as the off-chip resistor, we model the phase margin of the circuit to be at least 50 degrees. In the case that the capacitance of the packaging is higher than the capacitor value located on-chip, the phase margin will complete to above 60 degrees, which is generally considered the golden standard for stable and fast feedback systems. Larger margins will reduce the settling time of the circuit, however, with a proper power sequence, this will not be an issue.

D. RSA Encryption/Decryption Accelerator

The RSA scheme is based on the idea of modular exponentiation where Encryption and Decryption is handled by the mathematical equation:

c=m^emod n; m = c^dmod n; (e*d) = 1 mod ϕ(n) (1)

Our RSA accelerator applies this equation for both Dec. and Enc. using a modular exponentiation technique known as right to left binary. The method is an iterative algorithm which ensures that on a given clock cycle we only need a maximum of either two multipliers or dividers.

Our implementation of the right to left method is denoted above where we iterate through x cycles where x represents the bit length of our key. In each cycle, we always calculate:

base = base * base mod n (2)

However if the 0th bit of the key, e, is 1, then we must also calculate:

result = base * result mod n (3)

For each cycle of these two calculations, we right shift the key by 1 and repeat. Once the key is 0, then we have our final

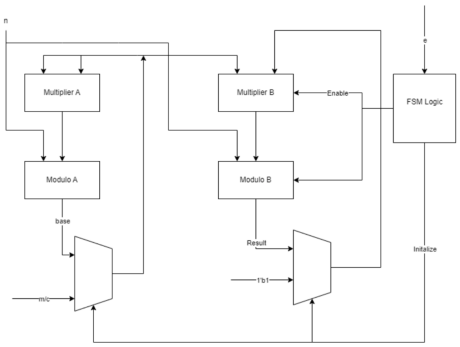

Fig. 13. High level diagram detailing RSA right to left binary method hardware implementation

result stored in the variable called result. This is the basis of our accelerator where we currently support 16-bit keys due to our area limitations. As a result, we have completion speed of (12*16)=192 clock cycles for encryption or decryption. Note that the initial value for base is the message value and for result is 1.

Since RSA is integrated as a memory mapped peripheral, to start an encryption or decryption four separate loads are required to start the process: message or ciphertext, key, modulo_n, and finally the start flag. Read from the same memory address and the accelerator will return an idle flag to denote ongoing calculations; the next 4 byte address will be the output message once the calculator is complete.

E. RISC-V IMC Core

The RISC-V architecture has become increasingly popular due to its open-source nature and extensibility. In Acadia, we support a 32-bit RISC-V IMC microprocessor. The motivation behind this was to create a power efficient, but functional core that targets embedded systems and IoT applications.

The core architecture is designed around the IMC extension, incorporating key components like a hardware multiplier, a hardware divider, and a compressed instruction decoder. The system is designed with a pipeline structure, allowing for instruction fetch, decode, execute, memory access, and writeback stages.

One interesting part about this core is the implementation of compressed instructions. Compressed instructions can cause misalignment of instruction in memory. Specifically, these instructions cause misalignment within cache lines of the instruction cache. To solve this, a modified instruction cache state machine was designed in order to fetch from flash twice upon a cache line misalignment. This would only happen when the most significant 16-bits of the cache line were accessed. In this case, we fetched the least significant 16-bits of the next cache line to form a complete word. After dealing with fetching misaligned instruction, we supported compressed instruction decoding. The basic model is to do an initial stage of 16-bit compressed instruction decoding before regular 32-bit instruction decoding. To implement this, we transform 16-bit compressed instructions to their 32-bit

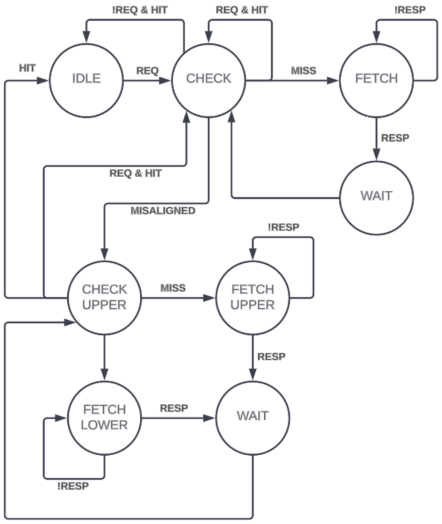

Fig. 14. Instruction Cache Controller, C Extension

counterparts within the decode stage of the processor. This design decision was done in order to simplify integration and reduce complexity. After this transformation, we can execute compressed instructions as per usual in the rest of the pipeline. From our testing, the C extension allows for a reduction in code size by 30%. The memory system that the core supports is a typical Harvard architecture. There is a separate instruction memory, implemented using a QSPI Flash. There is also a separate data memory, implemented using a 8KB SRAM. Moreover, load and stores are not able to read and write to instruction memory. This design decision was done in order to simplify the QSPI controller interface talking to the Flash. Of course, the processor supports a basic memory map for peripheral accesses.

F. Clocks and Resets

The main processor clock will be driven from an externally provided base clock at roughly 100 MHz. The peripheral clocks are all divided down from this base clock using configurable prescalers. There is one main external, asynchronous reset pin provided to the chip. Resets for the processor and peripherals are all asserted asynchronously and deasserted synchronously using special standard cells. For reset sequencing, the main processor reset is deasserted after all peripherals on slower clocks to support requests immediately once the processor begins to execute.

G. Memory Map Decoder

Here is the memory map for our data bus. Each 32 bit address is broken down into two parts, an offset (8KB, MMAP_ALIGNED_WIDTH) and a base width. Each peripheral device is assigned one base address, and the offset bits are used to address them. 8KB was chosen as that was our biggest "device", the data SRAM. When a processor makes a request (read or write) to an address, the request is forwarded to the

```
// Memory Map Address Space
localparam MMAP_ALIGNED_WIDTH = 13; // 8KB Aligned
localparam MMAP_BASE_WIDTH    = 32 - MMAP_ALIGNED_WIDTH;

//count for how many bits away from 32 we have for interrupts
localparam INVERSE_INT_COUNT    = (32 - INTERRUPT_LINES);

localparam MMAP_START_BASE_ADDR = 19'h3_8000;
localparam DATA_SRAM_BASE_ADDR  = 19'h3_8000;
localparam GPIO_BASE_ADDR       = 19'h3_8001;
localparam PRESCALER_BASE_ADDR  = 19'h3_8002;
localparam INT_BASE_ADDR        = 19'h3_8003;
localparam RSA_BASE_ADDR        = 19'h3_8004;
localparam MMAP_END_BASE_ADDR   = 19'h7_8000;
```

Fig. 15. Memory Map Decoder Snippet

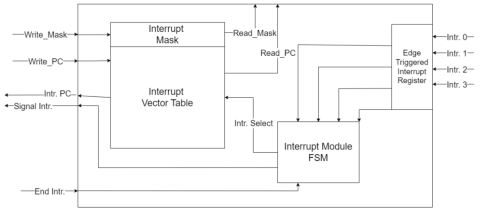

Fig. 16. Interrupt Architecture Diagram

corresponding device, and the device's output ports are then connected to the processor's port until it finishes responding. If the address is not mapped, the memory map decoder will respond to the request. If it's a write request, it responds to the processor with no side effects, if it's a read request, it responds with all zeros as the data.

H. QSPI Flash Controller

We used Stanley Wu's QSPI Flash controller as our flash controller. It is used to connect our instruction cache to an off-chip flash memory that holds our program. It was adapted to match our cache line size. In our test models, we have it connected to a behavioral model of a Winbond W25Q128JV to source our test programs.

I. Data SRAM

Our design does not have the capability of loading/storing from our off-chip flash interface (simpler interface). Therefore, we must have some space within the chip to use as data memory. We implement this as an 8-KB SRAM. Since it is on-chip SRAM, it is already fast so we do not need a data cache.

J. Interrupt Controller

Our Interrupt Controller is meant for both external devices and internal peripherals to interface with our IMC Core and trigger interrupts to the program core. Above is an internal representation of our Interrupt Module. We designed our module to host the Interrupt Vector Table inside the module itself instead of memory. However, since this module has access to the memory bus. The programmer is able to read and write to both the Interrupt Mask and IVT using load and store instructions.

Our processor uses an edge triggered interrupt response, where any rising edge will be registered by the module. This means that if an interrupt is currently being serviced, incoming signal will be latched, and will be serviced once the current Interrupt is finished.

Additionally for interrupt priority, priority is decided upon interrupt line id where devices connected to the first interrupt line are given highest priority while devices in the lowest line are given the least priority.

How our design handles interrupts is that once an interrupt is registered and we are in the idle_state, it will first loop through every single interrupt line register in order starting from index 0. On the first raised interrupt which has its corresponding mask bit enabled, it immediately stops and enters the in_service state. If no such condition is met, then it continues to loop in this state.

In the in_service state, we index the IVT and pass the corresponding PC to the CPU. We also signal an interrupt to the CPU at the same time and force the fetch_stage to start fetching from the interrupt PC.

The old PC, x1, and x2 are saved in its own hardware registers at this point. We also enable an interrupt_flag that is passed onto each stage and tied to every instruction. This ensures that for any branches that have yet to be resolved, branches with an interrupt_flag of 0 will update the original saved PC whereas an interrupt_flag of 1 will change the current PC in if_stage.

Once an mret is signaled by the programmer, the CPU flushes any erroneous instructions in decode and fetch. It also sends an end interrupt signal to the Interrupt module to have the FSM return to an idle_state and remove the interrupt signal from our register. The mret also causes our saved PC, x1, and x2 to be properly populated back into its corresponding locations.

K. UART

We used the UART TX/RX controller from OpenCores project https://opencores.org/projects/uart2bus We discarded the debug and bus controller only using the TX/RX and baud rate generator from the project. We wrote our own UART master that connects those three modules with two FIFOs acting as TX/RX buffers for more realistic use cases. We wrote our own memory-mapped bus interface to the controller, letting us connect it to our memory map decoder.

L. I2C Controller

We ported over an FPGA I2C controller from ECE437 to use our IO drivers (i2c_transceiver.sv). We trimmed it down for our use case and added a wrapper to connect it to our memory mapped interface (i2c_peripheral.sv).

M. Acadia Debug Interface

The ADP is the Acadia Debug Port, which essentially supports the JTAG specification with additional functionality. The main components that make up the ADP include data, address, instruction, bypass, and device ID registers. The ADP supports multiple DFT features including scan chain, boundary scan, processor core register reads and writes, and SRAM

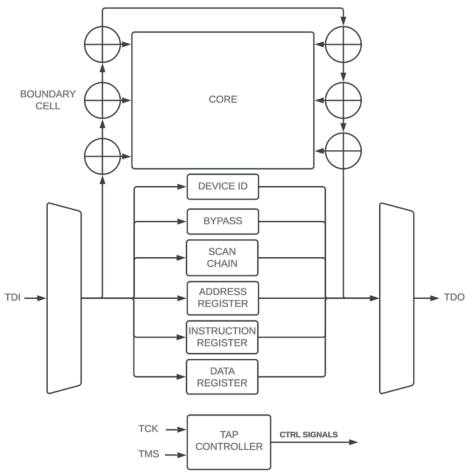

Fig. 17. ADP Architecture Diagram

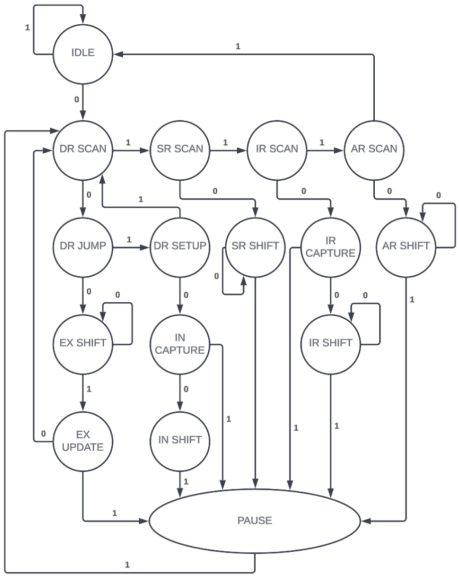

Fig. 18. ADP TAP Controller

reads and writes. Below describes the instruction map for the ADP and simple operation for each instruction.

- NULL_MODE: Reserved instruction; Guarantee debug mode is disabled.
  - Set IR to NULL
- DEVICE_ID_READ: Read 32-bit device ID
  - Set IR to DEVICE_ID_READ
  - Shift out SR 32 times
- SCAN_MODE: Access the scan chain
  - Set IR to SCAN_MODE
  - Set ADP TDI pin to disable/enable scan appropriately
  - Shift out SR xxx times depending on scan chain
- BOUNDARY_SCAN: Enables control of boundary scan cells
  - Set IR to BOUNDARY_SCAN
  - Control TAP to execute internal or external tests with
  - Capture, shift, and update state appropriately
- BYPASS_MODE: Simply pass TDI to TDO for sanity or daisy chaining chips
  - Set IR to BYPASS
  - Shift out SR xxx times depending on bypass functionality
- DEBUG_READ: Output internal register / data SRAM values to TDO
  - Set IR to DEBUG_READ
  - Set AR to memory mapped address
  - Shift out DR 32 times or xxx times
- DEBUG_WRITE: Write to the internal register / data SRAM values from TDI
  - Set IR to DEBUG_WRITE
  - Set AR to memory mapped address
  - Shift in DR to set write data
- STEP_MODE: Step through core operation cycle by cycle using TDI
  - Set IR to STEP_MODE
  - Set ADP TDI pin to stall/allow core execution

For some notable instructions, the TDI pin for the ADP interface has a specific functionality. During the SCAN_MODE instruction, the TDI pin acts as the scan enable pin for the scan chain. During BOUNDARY_SCAN and STEP_MODE instructions, the TDI pin acts as the stall signal for the main processor core. The most important piece of the ADP is the TAP, which is the Test Access Port. The TAP essentially implements a very simple state machine that is controlled using the TMS pin. The state machine is designed such that in any state, it takes up to six cycles of TMS high to reset back to idle. Thus, the state machine can never become in deadlock. In order to isolate processor execution using the debug port, the TAP will assert a debug mode signal when used. This switches the main system clock and reset to that of the ADP in order to perform synchronous reads and writes. Returning to the IDLE state in the TAP controller will allow the core to operate normally.

The most basic bring-up for ADP is reading the IDCODE register, which contains Acadia's DEVICE ID, 0x69ECE498.

N. Compile/Linking Programs

By implementing a Harvard memory architecture, we are unable to load initialized program data directly into SRAM. As a result, we cannot have a .data section for initialized variables or a .rodata section for initialized constants. Thus, we implemented a custom linker script to compile programs for our processor. All static variables must be in the .bss section, which is uninitialized and in SRAM. The .data and .rodata sections are mapped here for compatibility reasons but do not contain the initialized values.

Below details the memory sections that are located within our processor

- FLASH, 0x60000000, 16MB
- SRAM, 0x70000000, 8KB
- GPIO, 0x70001000, 8KB
- PRESCALER, 0x70002000, 8KB
- INTERRUPT, 0x70003000, 8KB
- RSA, 0x70004000, 8KB

Note, each peripheral is given 8KB but not all addresses must be utilized. Below details the specific sections of SRAM important to the C compiler in mapping variables. The most interesting thing is that we cannot support initialized variables in the traditional sense. In most processors, initialized variables are loaded from Flash into SRAM; however, our processor doesn't support this. Instead we have to use LUI and ADDI instruction to load constants to uninitialized data. Thus, all programs are compiled to only have uninitialized variables in the .bss section and we manually initialize them using instructions.

SRAM Sections

- .bss (0x70000000)
- .data
- .rodata
- .heap
- .stack (0x70001000)

FLASH Sections

- .text (0x60000000)
- .text_end (0x60800000)

O. Programming Model

With the memory sections defined by the linker script in mind, the programmer is responsible for loading the correct initialized values using explicit instructions. In the supported RV32IMC architecture, loading initialized values can be done by using the LUI and ADDI instructions. The C compiler should generate these instructions from assignments to uninitialized variables. Thus, the programmer must only use uninitialized variables and data structures when programming for the RISC core.

For dynamic memory support, the .heap section starts after the .bss, .data, and .rodata sections. Dynamic memory will be allocated in the .heap section and potentially extends to the end of the 8KB SRAM, which is the location for the start of the .stack section. These sections behave as the normal concepts of heap and stack. Currently, there is no library support for dynamic memory allocation functions like malloc() and free().

VI. Testing Instructions

A. Digital

Demo repo path is in bmn4@ece-498hk-02.ece.illinois.edu:/home/bmn4/working_digital . Navigate to <repo>/digital/sim for simulation instructions. <repo>/digital for other instructions.

B. Analog

Navigate from the cloned repository into acadia/analog. From inside this directory in the terminal, open Virtuoso. From Virtuoso's Library Manager tool, you will be able to see the library acadia. Inside this library are all the analog components that will be used in the final project. Refer to dac_tb for DAC testing. Refer to CMOS_4_Iref_tb and CMOS_1_Refs_tb for source reference testing. Refer to adc_cap_dac_8_bit_diff_top_tb for analog testing. After clicking these testbenches, open the Maestro view with either ADE Explorer or ADE Assembler. In this program you are able to run tests by going to Simulation -> Netlist and Run. Additionally, history may be viewed and loaded in ADE Assembler to view past waveforms

Acknowledgment

The team would like to thank Professor Dong Kai Wang and graduate TA Stanley Wu for their monumental help in facilitating the class. We would also like to thank Professor Mostafa Ahmed for his help for the analog team. Finally, we would like to thank Apple for sponsoring this tapeout.