28nm IoT SoC Design Review

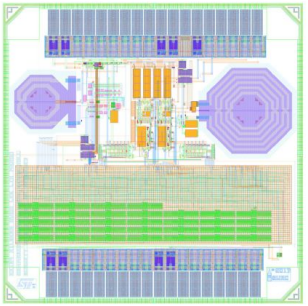
March 16-18, 2021
UC Berkeley EE290C

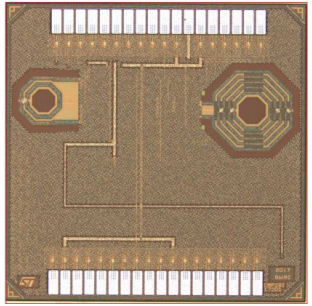
Course History

Tapeout class: taking students from schematic to silicon in one semester

David C. Burnett, Brian Kilberg, Rachel Zoll, Osama Khan, Kristofer S.J. Pister
Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA 94720
Email: {db, bkilberg, rachelzoll, oukhan, pister}@eecs.berkeley.edu

- 2017
  - David Burnett, Osama Khan
  - Taped out analog, RF
- 2018
  - Osama Khan, Edward Wang
- 2019
  - Edward Wang, Aviral Pandey, Hall Chen
- 2021
  - Bora, Ali, Kris
  - Dan Fritchman, Aviral Pandey



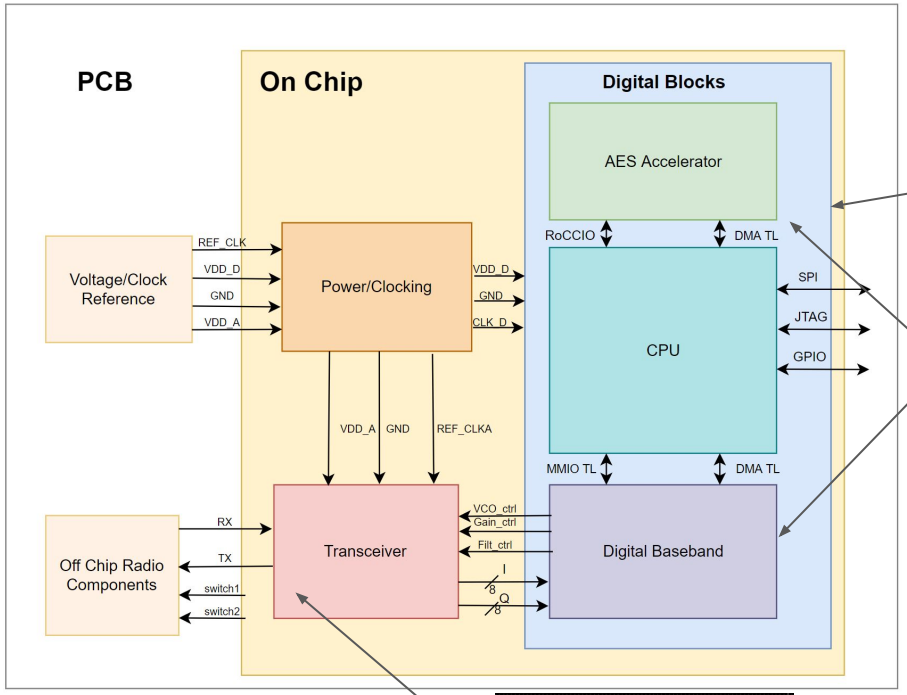


ISCAS 2018

Research Infrastructure

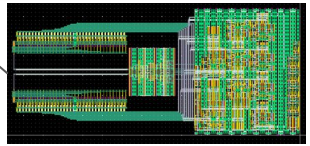
Bora Nikolić
28th-year grad student

Leveraging Research Infrastructure



https://github.com/ucb-bar/chipyard
Processor core, interfaces
Software tools

https://www.chisel-lang.org/
Custom BLE digital baseband, accelerator wrappers



Berkeley Analog Generator (BAG) - generated SAR ADC

Chip Intro & Overview

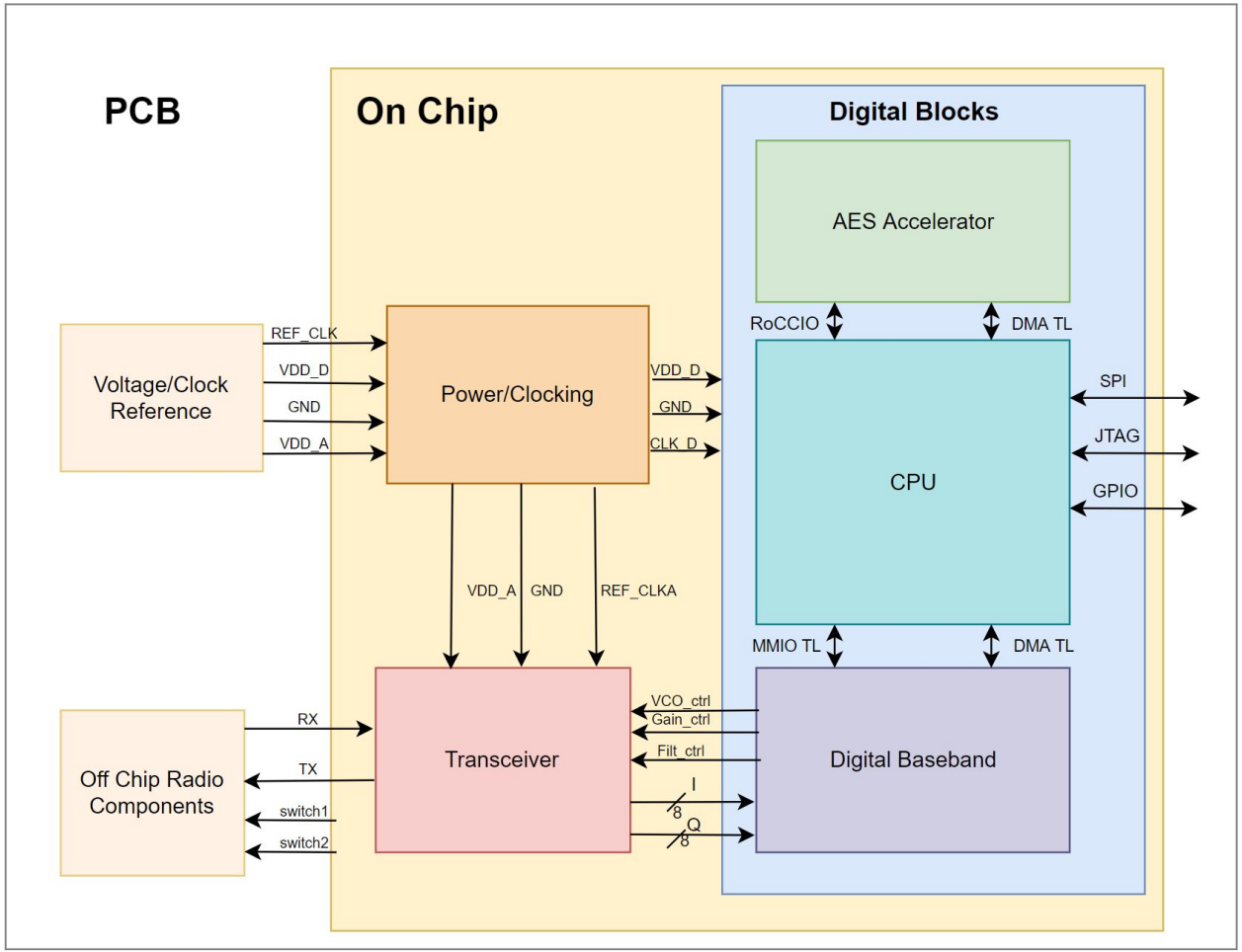
OSCI BEAR

Open-source **S**o**C** for **I**oT with **B**L**E**, **A**ES, and **R**adio

Project Goals

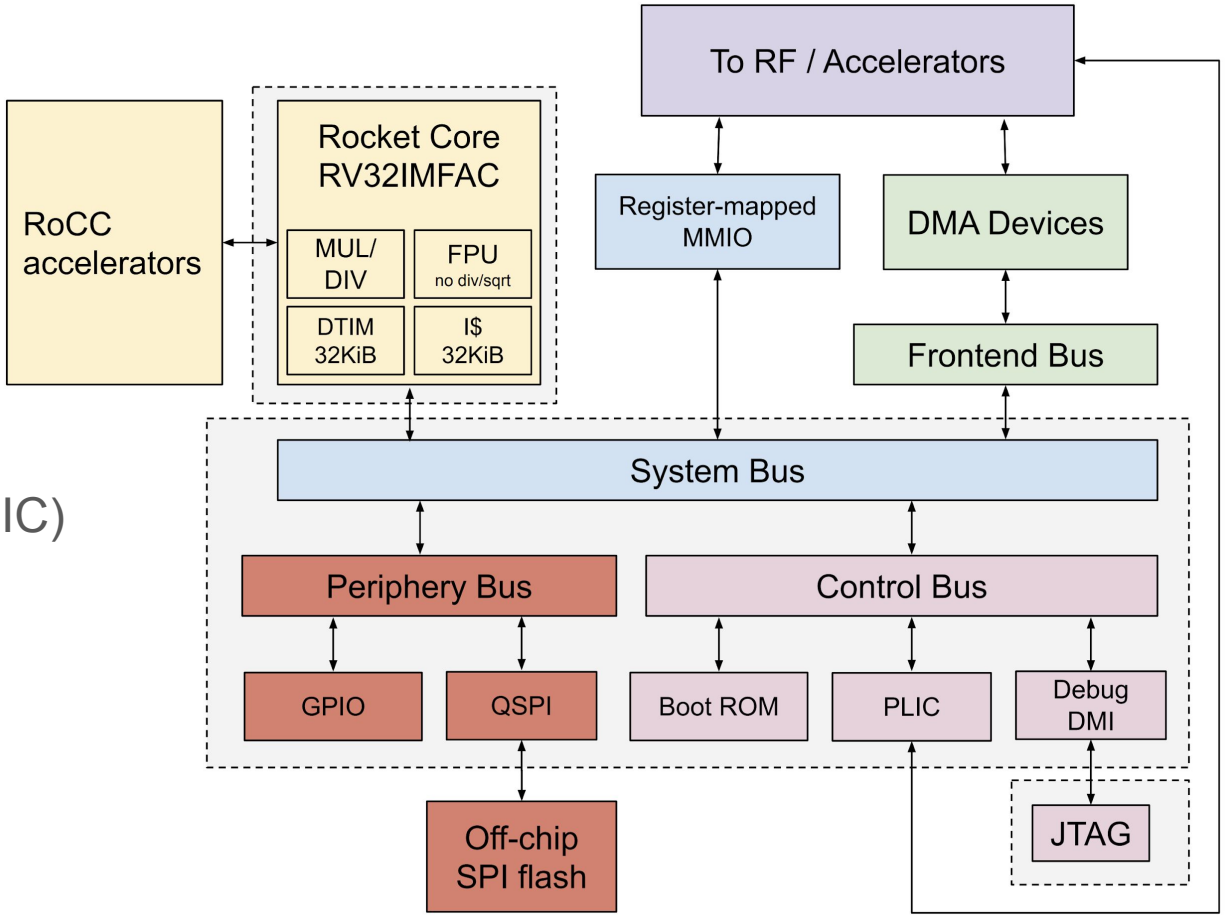
- Lightweight BLE compatible SoC for IoT applications
  - Wearables, localized sensor networks, smart keys, etc
- Demonstrate agile hardware development flow
  - Using Berkeley designed tools
- Tape-out in TSMC 28nm technology
  - 1 mm^2 total die area
  - Off chip radio and clock components will be attached on PCB level

Chip Overview



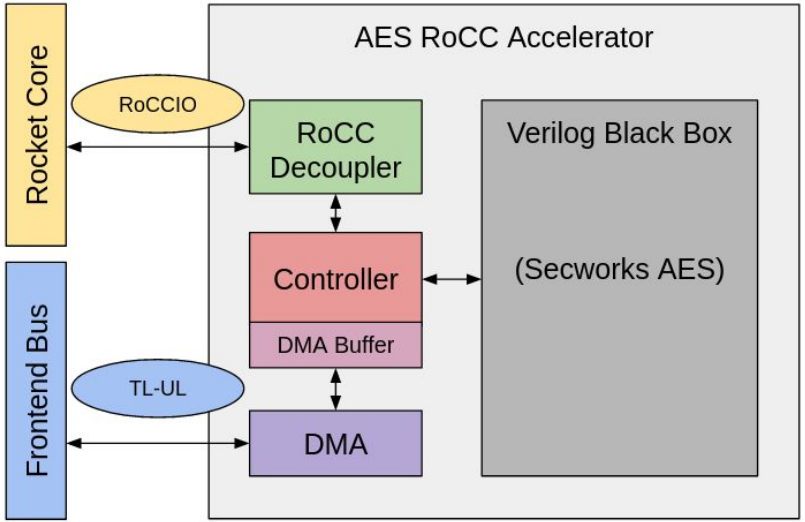
Compute Complex

- RV32IMAFC Core
- 32KB I$
- 32KB Data Memory (DTIM)
- JTAG
- SPI flash
- Interrupt Controller (PLIC)



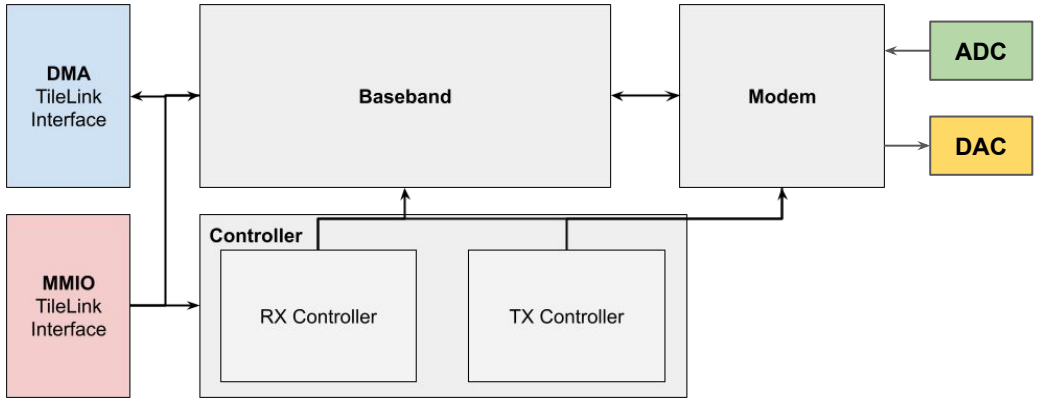
AES Accelerator/Co-processor

- Efficient data security
- Performs AES Encryption/Decryption
  - Different modes orchestrated by software
- Built from open-source Secwork AES
  - Heavily tested
  - Taped-out verilog
- Non-blocking interaction with core
  - Buffer custom RISC-V instructions
  - DMA to handle memory operations
  - Supports both interrupts and polling

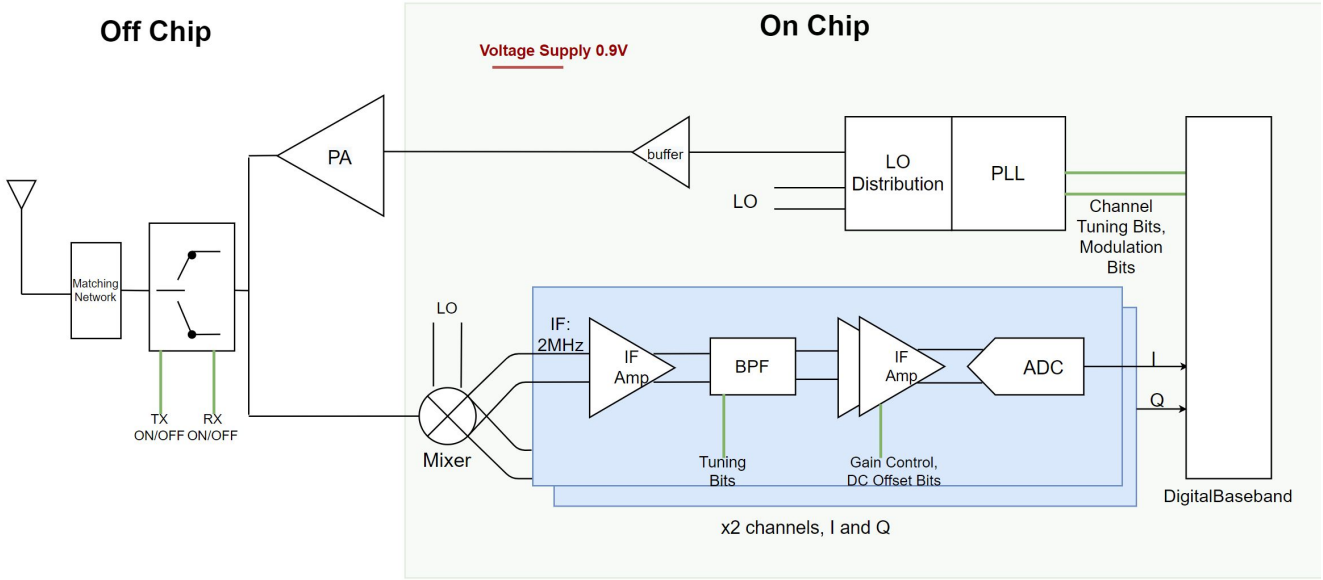


BLE Baseband and Modem

- BLE baseband paired with GFSK modem
- Receives instructions from core via MMIO and data from scratchpad via DMA
  - Notifies core of incoming messages via interrupt
- Supports all packet types for LE 1M specification



BLE Transceiver



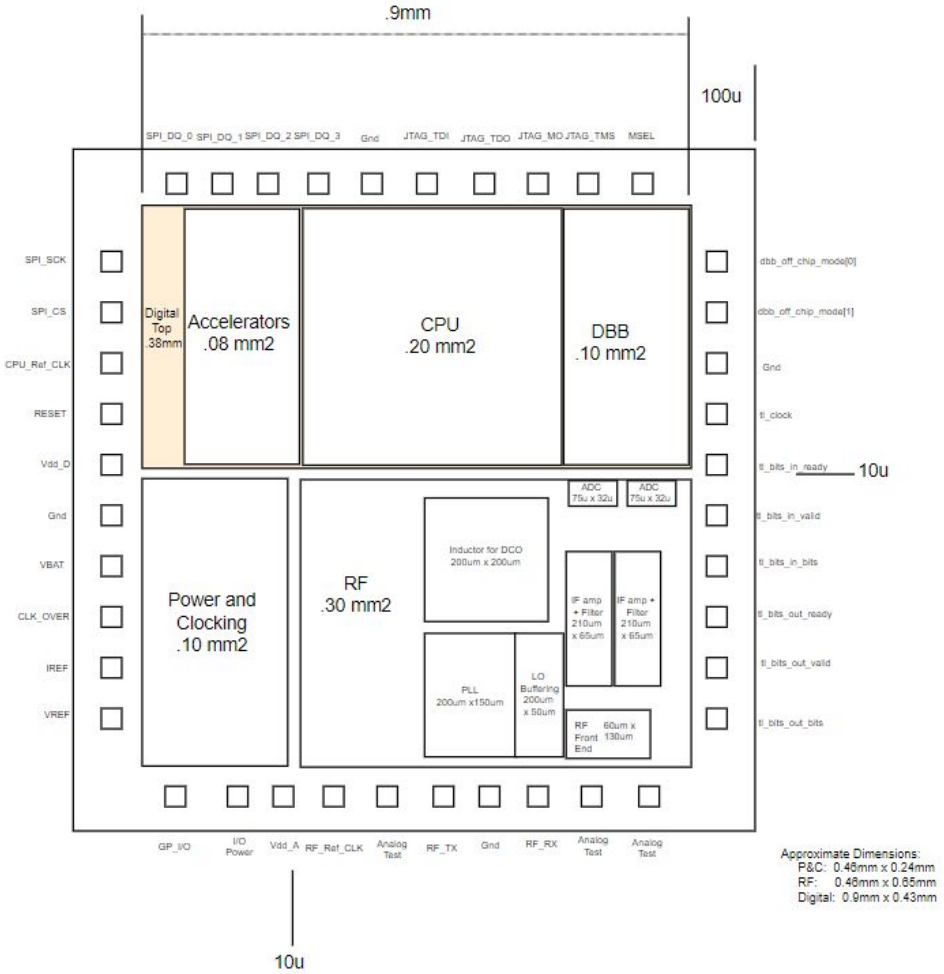
- Low IF receiver architecture with tunable LO
- Analog PLL
- Digital control for GFSK modulation

Target Specs

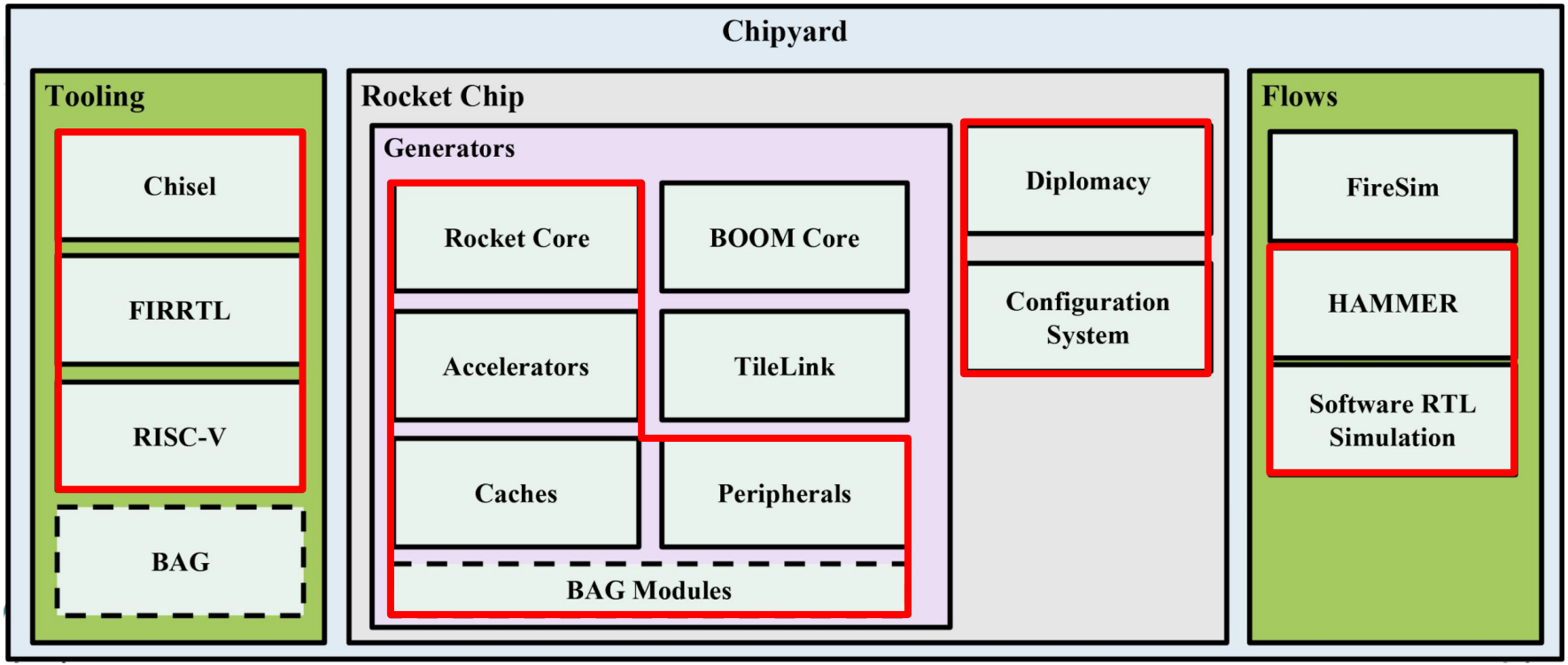
Digital Targets	
Core Clock	200 MHz
Core Power	9 mW
AES Speedup	5x+ speedup from non-accelerated
BLE Transmit Latency	20 μS

Analog Targets	
Power	5 mW
BER	0.1%

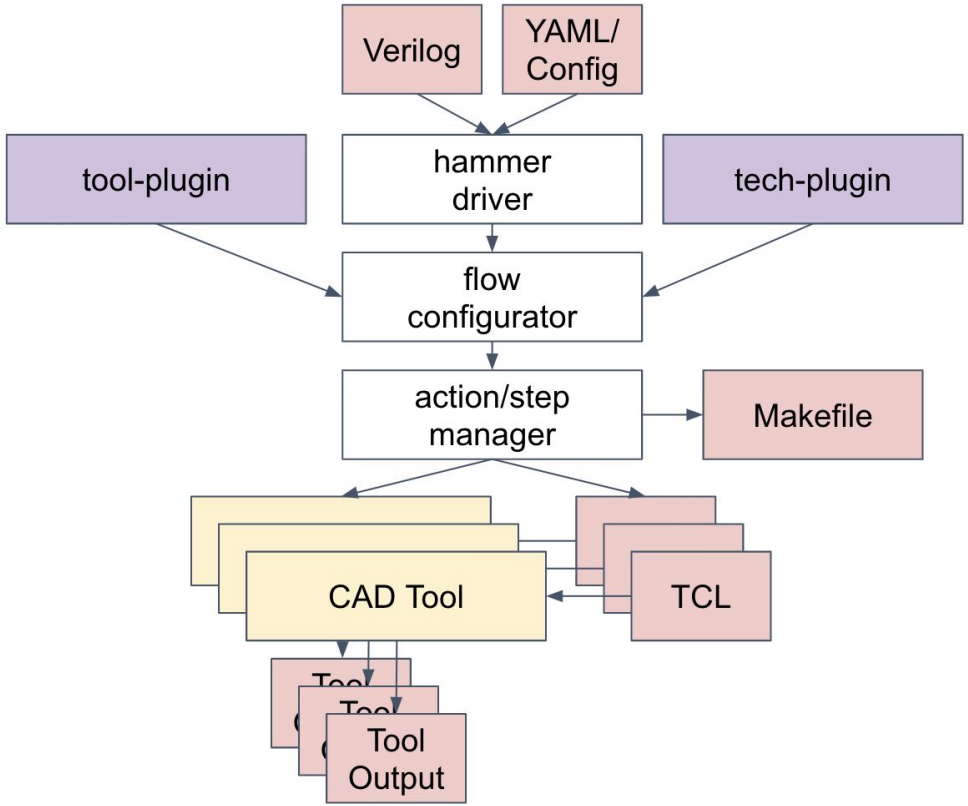
Target Area

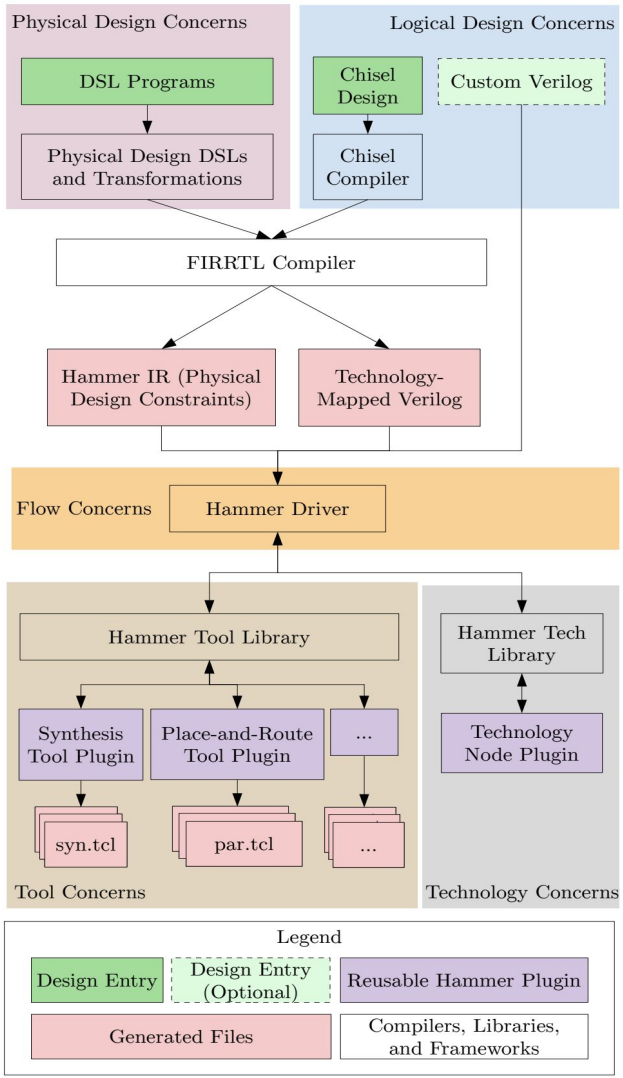

.9mm
100u
SPI_DQ_0
SPI_DQ_1
SPI_DQ_2
SPI_DQ_3
Gnd
JTAG_TDI
JTAG_TDO
JTAG_MO
JTAG_TMS
MSEL
SPI_SCK
SPI_CS
CPU_Ref_CLK
RESET
Vdd_D
Gnd
VBAT
CLK_OVER
IREF
VREF
Digital Top .38mm
Accelerators .08 mm2
CPU .20 mm2
DBB .10 mm2
Power and Clocking .10 mm2
RF .30 mm2
ADC 75u x 32u
ADC 75u x 32u
Inductor for DCO 200um x 200um
IF amp + Filter 210um x 65um
IF amp + Filter 210um x 65um
PLL 200um x150um
LO Buffering 200um x 50um
RF Front End 60um x 130um
dbb_off_chip_mode[0]
dbb_off_chip_mode[1]
Gnd
tl_clock
tl_bits_in_ready
10u
tl_bits_in_valid
tl_bits_in_bits
tl_bits_out_ready
tl_bits_out_valid
tl_bits_out_bits
GP_I/O
I/O Power
Vdd_A
RF_Ref_CLK
Analog Test
RF_TX
Gnd
RF_RX
Analog Test
Analog Test
10u
Approximate Dimensions:
P&C: 0.46mm x 0.24mm
RF: 0.46mm x 0.65mm
Digital: 0.9mm x 0.43mm

Infrastructure: Chipyard



Infrastructure: Hammer VLSI Flow



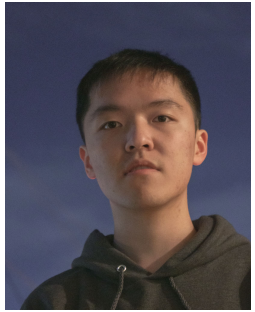


CPU

Team Introduction

Nayiri Krzysztofowicz
1st Yr PhD Student, advised by Bora

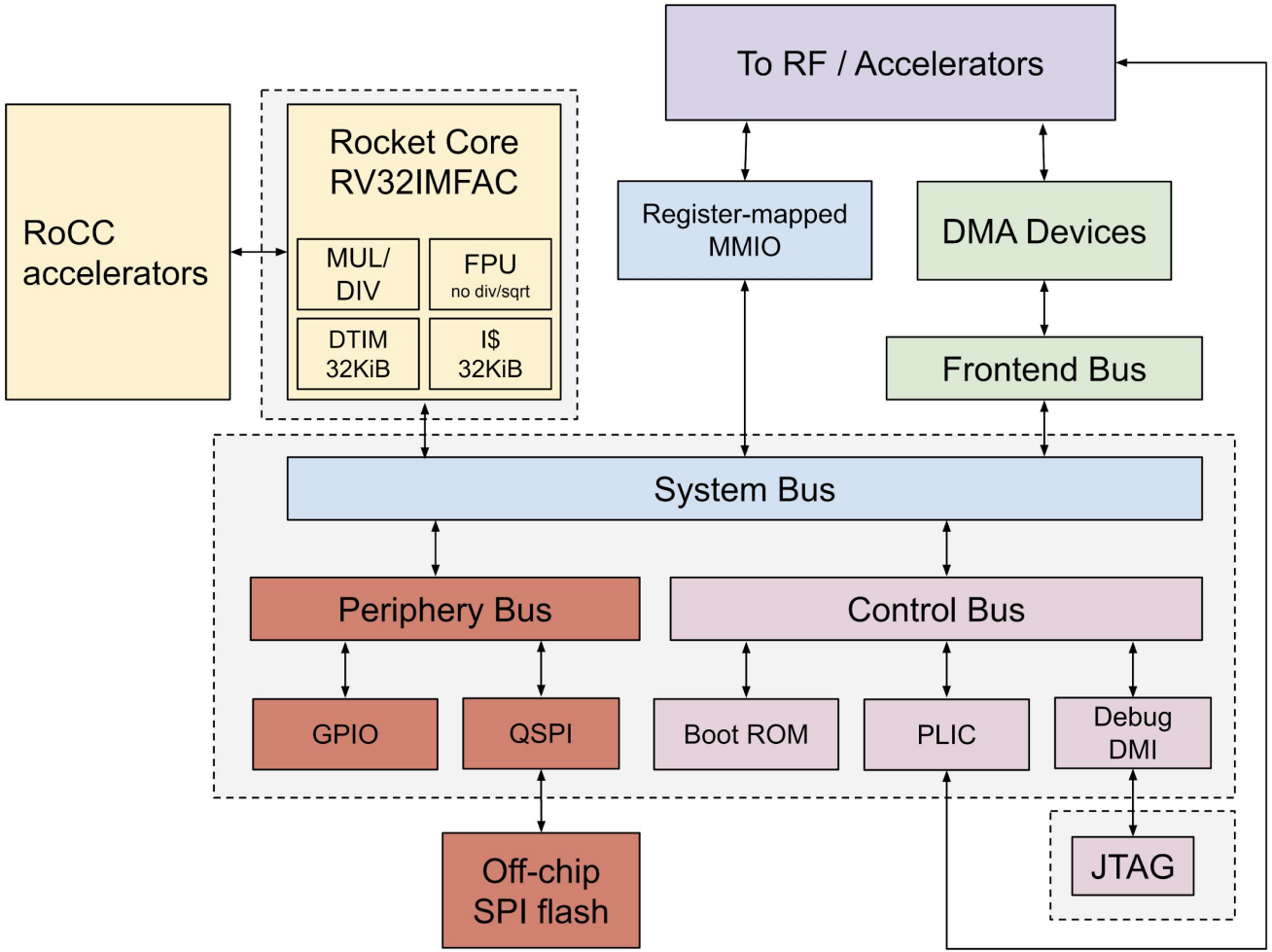
Josh Alexander
4th Yr EECS Undergrad



Cheng Cao
3rd Yr EECS Undergrad

Overview

- RV32IMAFC Core
- 32KB I$
- 32KB DTIM
- JTAG
- SPI flash
  - With XIP
  - Quad read/write



Memory subsystem

- Single core, uses TileLink protocol to connect peripheral devices
- The Data Tightly Integrated Memory in place of the L1D$, and can back the L1I$
- The TSI (Tethered Serial Interface module) can tunnel TileLink messages and provide memory backing through a debugging interface
- The QSPI flash has Execute In Place mode to back the instruction cache
- The debug module has a small SRAM to provide backing in case of other memory blocks are inaccessible

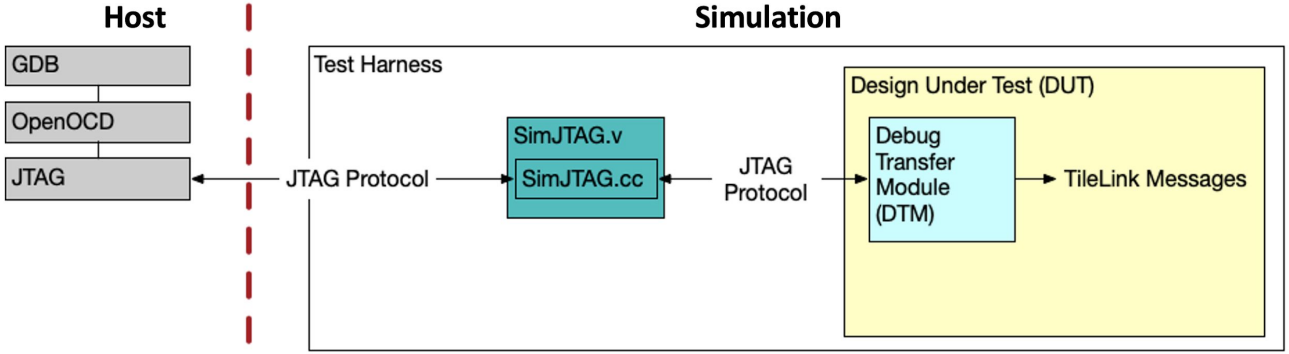
Boot Process (Boot ROM)

Two options:

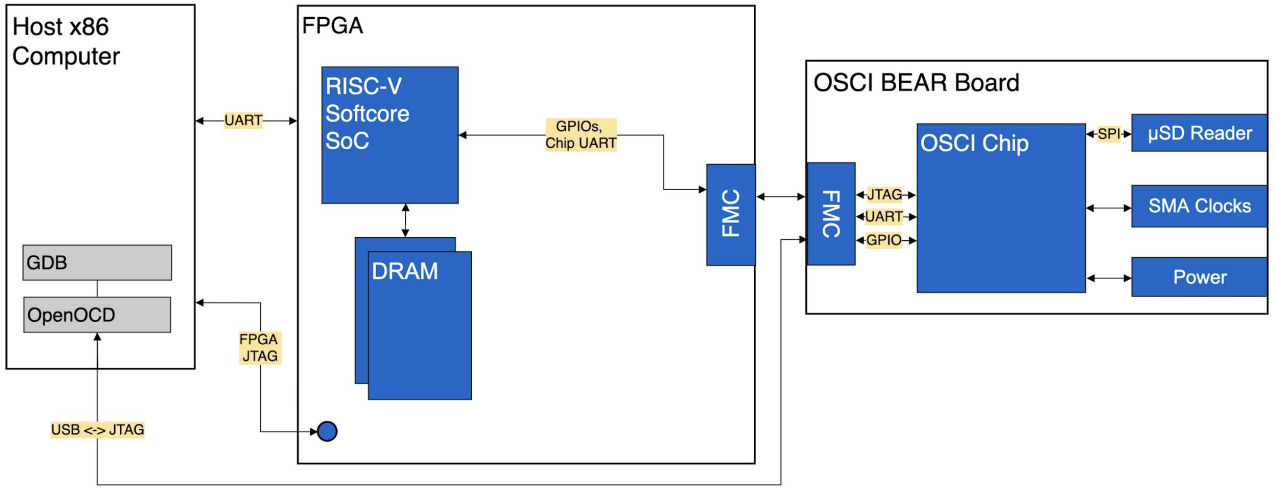
1. Boot select pin is high (Self-boot)
    a. Boot directly into the SPI flash with execute in place mode
    b. The program can copy itself to the DTIM if needed
2. Boot select pin is low (Tethered debug)
    a. Setup a trap handler, and enter a wait for interrupt loop
    b. Use JTAG, TSI, or other external debugging tools to program the on-chip memory
    c. Use JTAG, TSI, or other external tools to trigger an exception
    d. Boot ROM receives exception, jumps to the programmed memory

JTAG Debug

simulation:

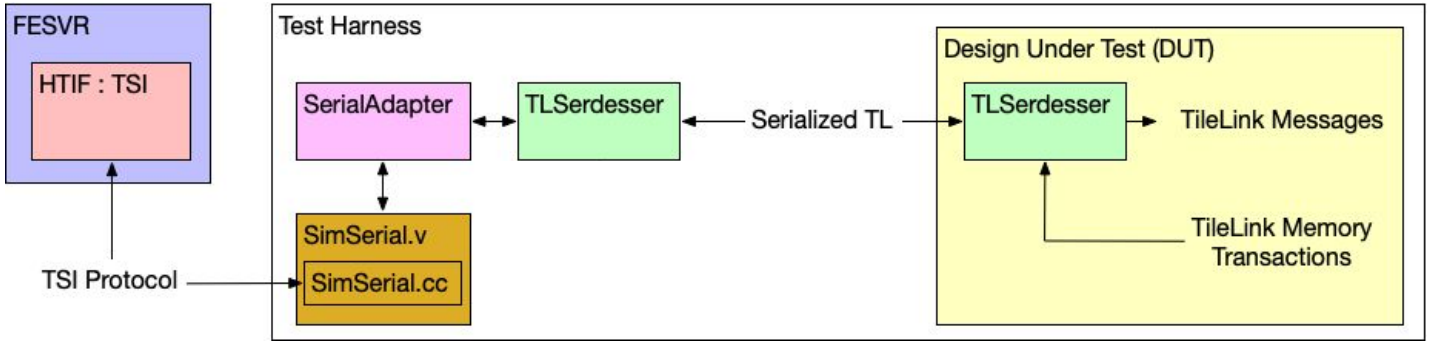


chip bringup:
(**preliminary diagram)

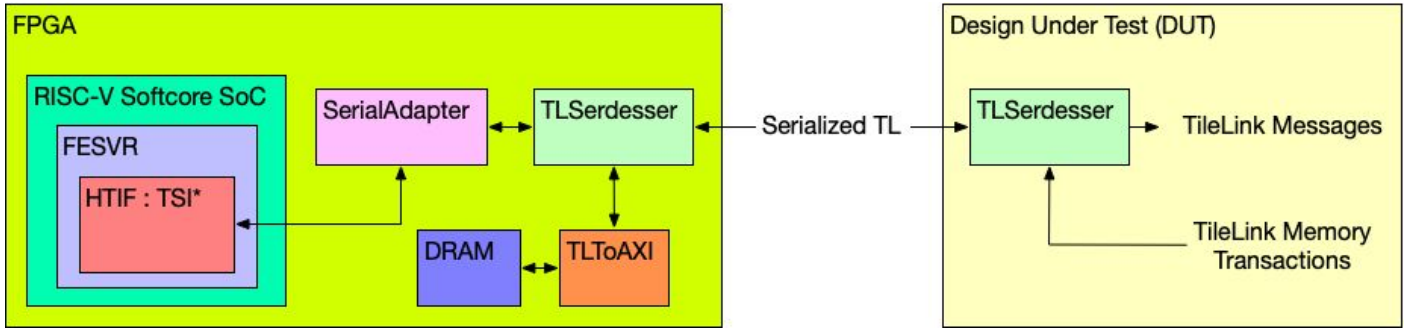


TSI Debug

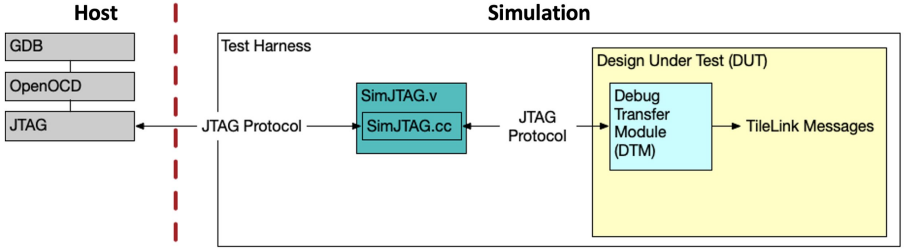
simulation:



chip bringup:



JTAG Simulation



ee290c-software-stack > core > C hellodebug.c > ...

```c
// example taken from: https://github.com/chipsalli

char text[] = "Vafgehpgvba frgf jnag gb or serr!";

// Don't use the stack, because sp isn't set up.
volatile int wait = 1;

int main()
{
    while (wait)
        ;
    int i = 0;
    while (text[i]) {
        char lower = text[i] | 32;
        if (lower >= 'a' && lower <= 'm')
            text[i] += 13;
        else if (lower > 'm' && lower <= 'z')
            text[i] -= 13;
        i++;
    }

    while (!wait)
        ;
}
```

current output:

```
(chipyard) nayiri@bwrcr740-8 /scratch/nayiri/tstech28/chipyard3/sims/vcs (v1p4-digital)
$ riscv64-unknown-elf-gdb hellodebug.riscv
GNU gdb (GDB) 8.3.0.20190516-git
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-pc-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from hellodebug.riscv...
(gdb) set remotetimeout 20000
(gdb) target remote localhost:3333
Remote debugging using localhost:3333
0x00010060 in ?? ()
(gdb) load
Loading section .text.init, size 0x1c2 lma 0x80000000
Loading section .tohost, size 0x48 lma 0x80001000
Loading section .text, size 0x1222 lma 0x80002000
Loading section .text.startup, size 0x18 lma 0x80003222
Loading section .rodata, size 0x258 lma 0x8000323c
Loading section .rodata.str1.4, size 0x40 lma 0x80003494
Loading section .eh_frame, size 0x50 lma 0x800034d4
Start address 0x80000000, load size 5932
Transfer rate: 21 bytes/sec, 847 bytes/write.
(gdb) print text
No symbol "text" in current context.
(gdb)
```

```
(chipyard) nayiri@bwrcr740-8 /scratch/nayiri/tstech28/chipyard3/sims/vcs (v1p4-digital)
$ openocd -f ./emulator.cfg
Open On-Chip Debugger 0.10.0+dev-00849-gcbb15587d (2021-03-08-12:43)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Info : Initializing remote_bitbang driver
Info : Connecting to localhost:37057
Info : remote_bitbang driver initialized
Info : This adapter doesn't support configurable speed
Info : JTAG tap: riscv.cpu tap/device found: 0x00000001 (mfg: 0x000 (<invalid>), part: 0x0000, ver: 0x0)
Info : datacount=1 progbufsize=16
Info : Disabling abstract command reads from CSRs.
Info : Examined RISC-V core; found 1 harts
Info :  hart 0: XLEN=32, misa=0x40801125
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (20639). Workaround: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2183). Workaround: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2191). Workaround: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2203). Workaround: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2209). Workaround: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2205). Workaround: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2175). Workaround: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2205). Workaround: increase "set remotetimeout" in GDB
```

```
(chipyard) nayiri@bwrcr740-8 /scratch/nayiri/tstech28/chipyard3/sims/vcs (v1p4-digital)
$ make run-binary CONFIG=EE290CBLEConfig BINARY=hellodebug.riscv  SIM_FLAGS="+spiflash0=spi.bin +ee290c_bsel=1 +jtag_rbb_enable=1 -V"
Running with RISCV=/scratch/nayiri/tstech28/chipyard3/riscv-tools-install
(set -o pipefail &&  /scratch/nayiri/tstech28/chipyard3/sims/vcs/simv-chipyard-EE290CBLEConfig +permissive +spiflash0=spi.bin +ee290c_bsel=1 +jtag_rbb_enable=1 -V +dramsim +dramsim_ini_dir=/scratch/nayiri/tstech28/chipyard3/generators/testchipip/src/main/resources/dramsim2_ini +max-cycles=10000000  +ntb_random_seed_automatic +verbose +permissive-off hellodebug.riscv </dev/null 2> >(spike-dasm > /scratch/nayiri/tstech28/chipyard3/sims/vcs/output/chipyard.TestHarness.EE290CBLEConfig/hellodebug.out) | tee /scratch/nayiri/tstech28/chipyard3/sims/vcs/output/chipyard.TestHarness.EE290CBLEConfig/hellodebug.log)
Chronologic VCS simulator copyright 1991-2019
Contains Synopsys proprietary information.
Compiler version P-2019.06-SP2-5_Full64; Runtime version P-2019.06-SP2-5_Full64;  Mar 16 11:35 2021
VCS Build Date = May 26 2020 20:23:14
Start run at Mar 16 11:35 2021

NOTE: automatic random seed used: 1607541211
[UART] UART0 is here (stdin/stdout).
```

Simulation & Software

- compile C benchmarks into rv32ui-imafc format
- able to load programs through JTAG
  - next steps: debug helloworld, software stack for cpu benchmarks, aes, baseband
- Chipyard default TSI infrastructure working in simulation
- OS / Software stack
  - RTOS is nice to have & possible to run
  - Hardware specd to be able to run a bluetooth stack on a RTOS
    - 32K data memory, instruction cache, etc.
  - Not our focus and not required, the test software will be baremetal

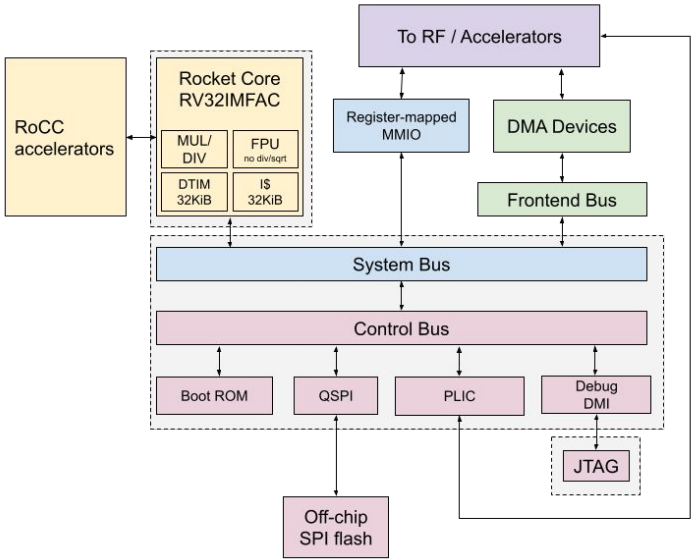
Interface with RF/Accelerators

RF

- Send
  - Send interrupt to send 2-158 bytes
  - Software handles encrypt/decrypt
  - Store accel response in register
- Receive
  - Set base address at onset of transmission
  - Command when to stop
  - Interrupts resolved in order at end
    - How many bytes
    - Valid/Invalid

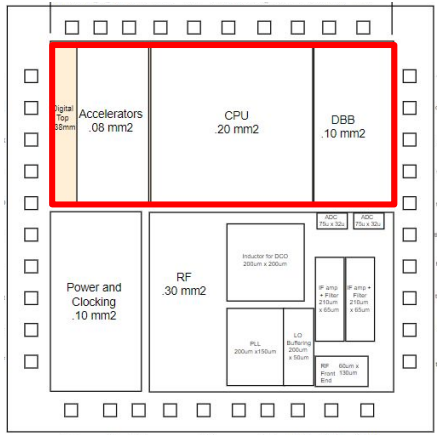
AES

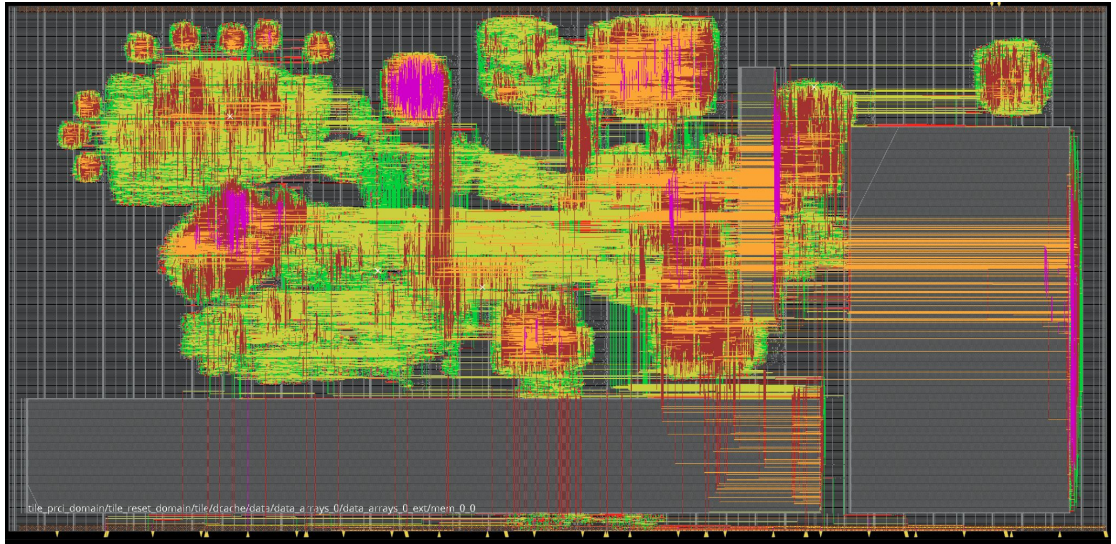
- Interacts directly with Rocket Core
- Uses DMA along with RF
- Space for text to decrypt/encrypt

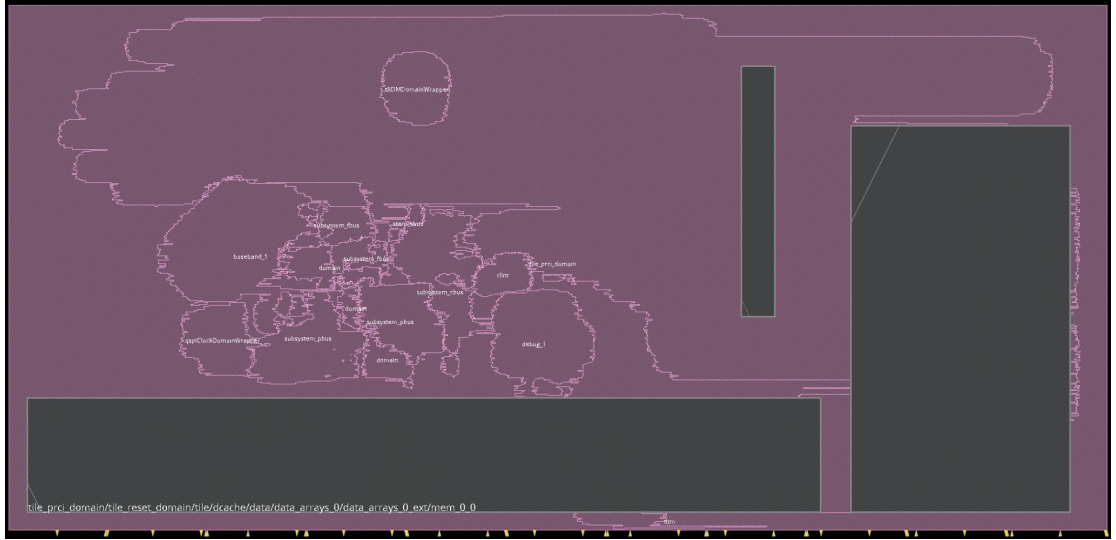


Physical Design

- Digital Top
  - VDD = 0.9 V
  - clock freq = 100MHz
  - LVT devices
- area = 0.193 mm^2
  (allocated 0.38 mm^2)

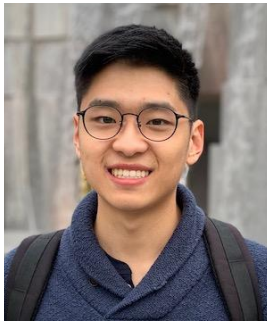






AES Accelerator

Team Introduction



Anson Tsai

5th Yr M.S., advised by Bora

Eric Wu

EECS M.Eng., advised by Kris

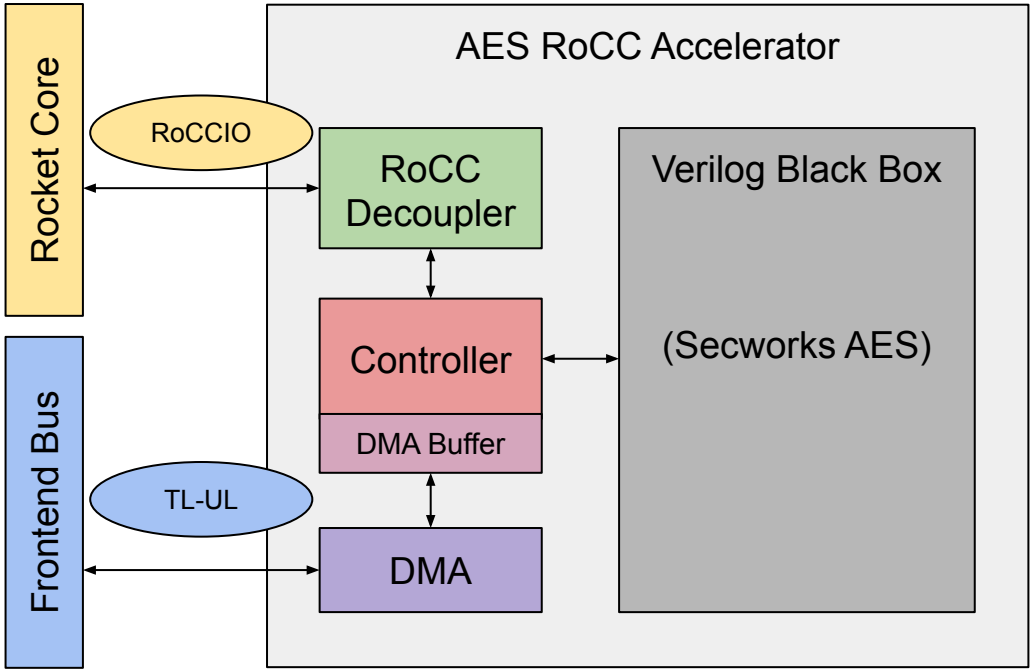
Daniel Fan

3rd Yr EECS undergrad

Overview Design

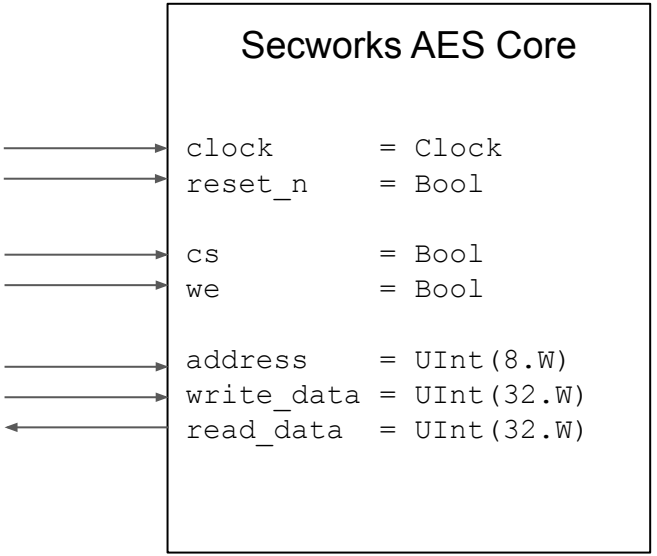
- RoCC (Rocket Custom Co-processor) AES Accelerator
- Building from open-source Secworks AES core
- Designing HW wrapper logic for rocket integration
- Custom RISC-V instructions
- Software stack to compile C code

AES Accelerator Top-Level Diagram



Starting at the Core: Secworks AES

- Performs AES encryption and decryption
  - 128b and 256b keys
- Internal registers to hold key and text
- 51 cycles/block for 128b key
- 71 cycles/block for 256b key
- Well tested and mature
- Taped-out in other designs



AES Core Usage

1. Load the key to be used by writing to the key register words.
2. Set the key length by writing to the config register.
3. Initialize key expansion by writing a one to the init bit in the control register.
4. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the key expansion has been completed.
5. Write the cleartext block to the block registers.
6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.
9. Read out the ciphertext block from the result registers.

AES Core Usage

KEY SETUP

1. Load the key to be used by writing to the key register words.
2. Set the key length by writing to the config register.
3. Initialize key expansion by writing a one to the init bit in the control register.
4. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the key expansion has been completed.

DATA LOAD

5. Write the cleartext block to the block registers.

ENCRYPT/DECRYPT

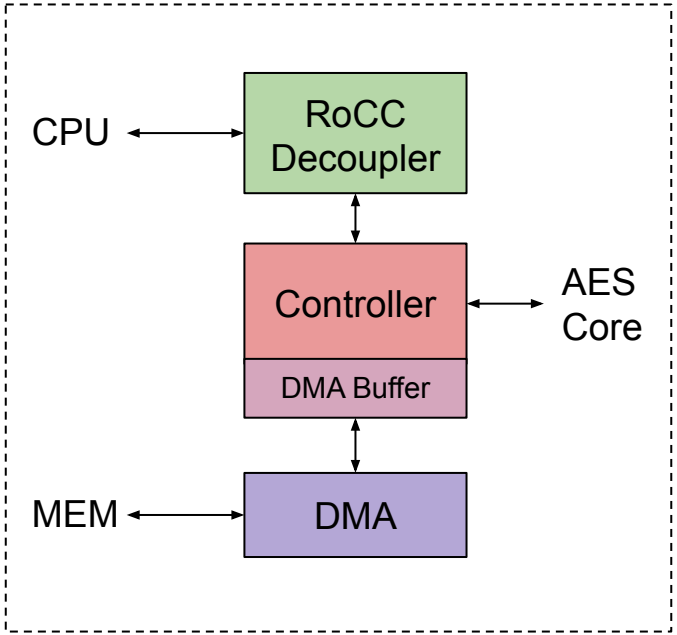
6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.

DATA WRITE

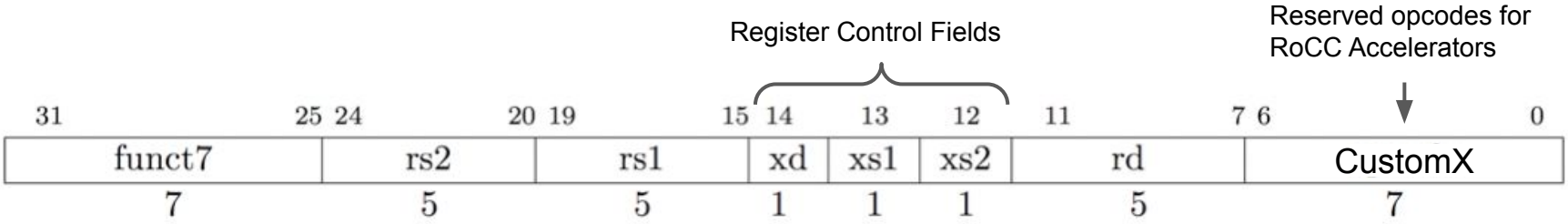
9. Read out the ciphertext block from the result registers.

What we need to implement:

- Instructions
  - Key Load + Setup
  - Data Load
  - Start Encryption/Decryption
  - Query Status
- Decoupler
  - Receive/process instructions for controller
- Controller
  - Handles AES Core operation
  - Queries DMA for data reading/writing
- DMA + Buffers
  - Handles memory operations
  - Buffers DMA packets and outputs 32b blocks



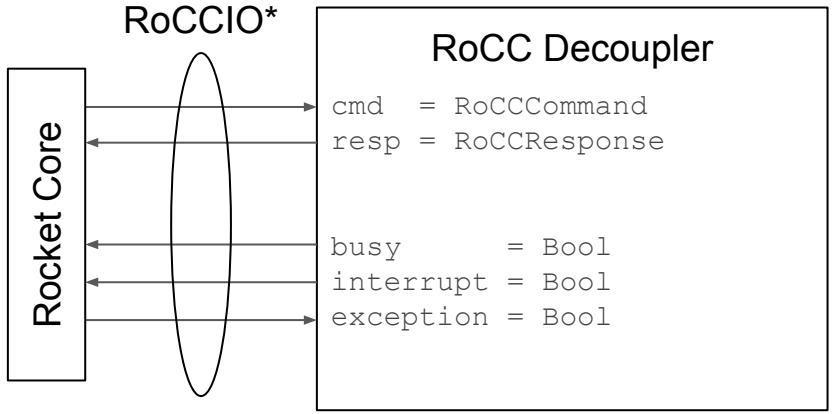
Custom RISC-V Instructions



1. Load key and set key size
   - funct7 - 0 for 128b, 1 for 256b key size
   - rs1 - key address
   - rs2 - N/A
   - rd - N/A
   - (xd,xs1,xs2) - (0,1,0)
2. Load text src and dest address
   - funct7 - 2
   - rs1 - text src address
   - rs2 - text dest address
   - rd - N/A
   - (xd,xs1,xs2) - (0,1,1)
3. Encrypt/Decrypt Blocks
   - funct7 - 3 for encrypt, 4 for decrypt
   - rs1 - # of 128b blocks to process
   - rs2 - N/A
   - rd - N/A
   - (xd,xs1,xs2) - (0,1,0)
4. Query Status (blocking)
   - funct7 - 5
   - rs1 - N/A
   - rs2 - N/A
   - rd - destination register
   - (xd,xs1,xs2) - (1,0,0)

Instruction Interface - RoCCIO

- Rocket Custom Co-processor IO
- Wrapper for RISC-V Instructions
  - Commands (from CPU to accelerator)
  - Responses (from accelerator to CPU)
- Status signals
  - Busy, Exception, Interrupt
- Other interfaces (not used)
  - Page Table Walker (virtual mem)
  - FPU Requests and Response



RoCCCommand

```
funct = Bits(7.W)
rd    = Bits(5.W)
rs1   = Bits(5.W)
rs2   = Bits(5.W)
rd    = Bits(5.W)
…
opcode = Bits(7.W)

rs1_data = Bits(32.W)
rs2_data = Bits(32.W)
```

RoCCResponse

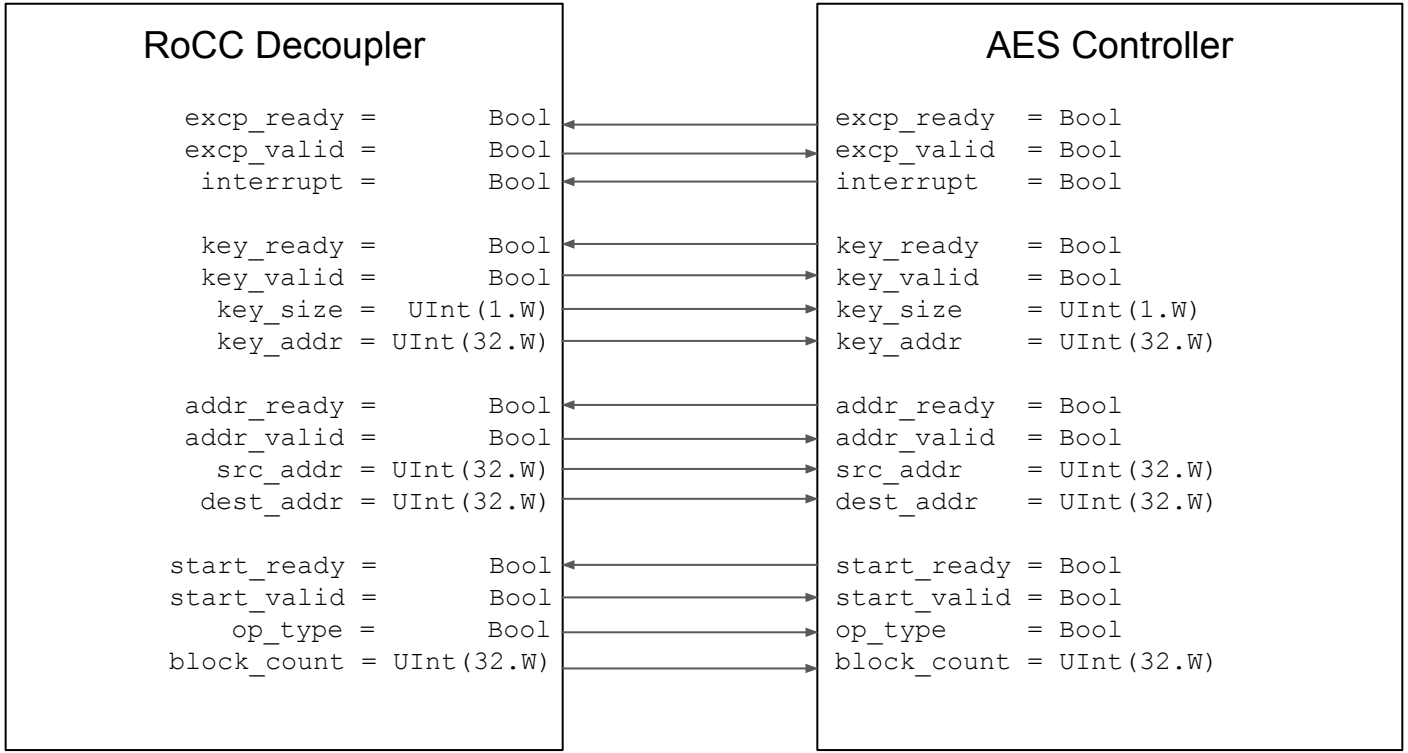
```
rd = Bits(5.W)

rd_data = Bits(32.W)
```

RoCC Decoupler

- Receives/processes instructions
- Buffers data for controller
  - Outputs operate by Ready-Valid
  - Data grouped by operation/step
    - Key, Addr, Start
- Non-blocking interaction with CPU

RV GROUP	OUTPUT (to ctrl)	DESCRIPTION
key	key_addr	Address of Key
key	key_size	Size of Key
addr	src_addr	Address of source data
addr	dest_addr	Address to write result
start	block_count	# of 128b blocks to process
start	op_type	Enc. or Dec.

RoCC Decoupler <> Controller Interface



Controller - AES Core Usage Recap

1. Load the key to be used by writing to the key register words.
2. Set the key length by writing to the config register.
3. Initialize key expansion by writing a one to the init bit in the control register.
4. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the key expansion has been completed.

KEY SETUP

5. Write the cleartext block to the block registers.

DATA LOAD

6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.

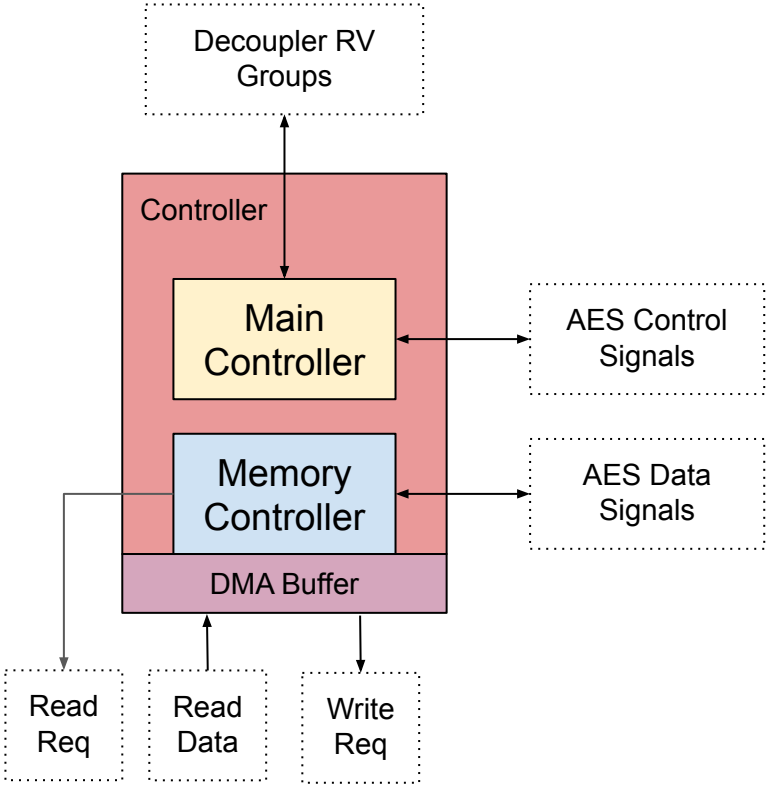
ENCRYPT/DECRYPT

9. Read out the ciphertext block from the result registers.

DATA WRITE

Controller

- Controller consists of a main controller and a memory controller
- Main Controller
  a. Receive information from RoCC Decoupler
  b. Handle the workflow of the AES operations
- Memory Controller
  a. Query key and text data from DMA with DMA buffer
  b. Read/Write data from AES core



AES Core Usage

1. Load the key to be used by writing to the key register words.
2. Set the key length by writing to the config register.
3. Initialize key expansion by writing a one to the init bit in the control register.
4. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the key expansion has been completed.

KEY SETUP

5. Write the cleartext block to the block registers.

DATA LOAD

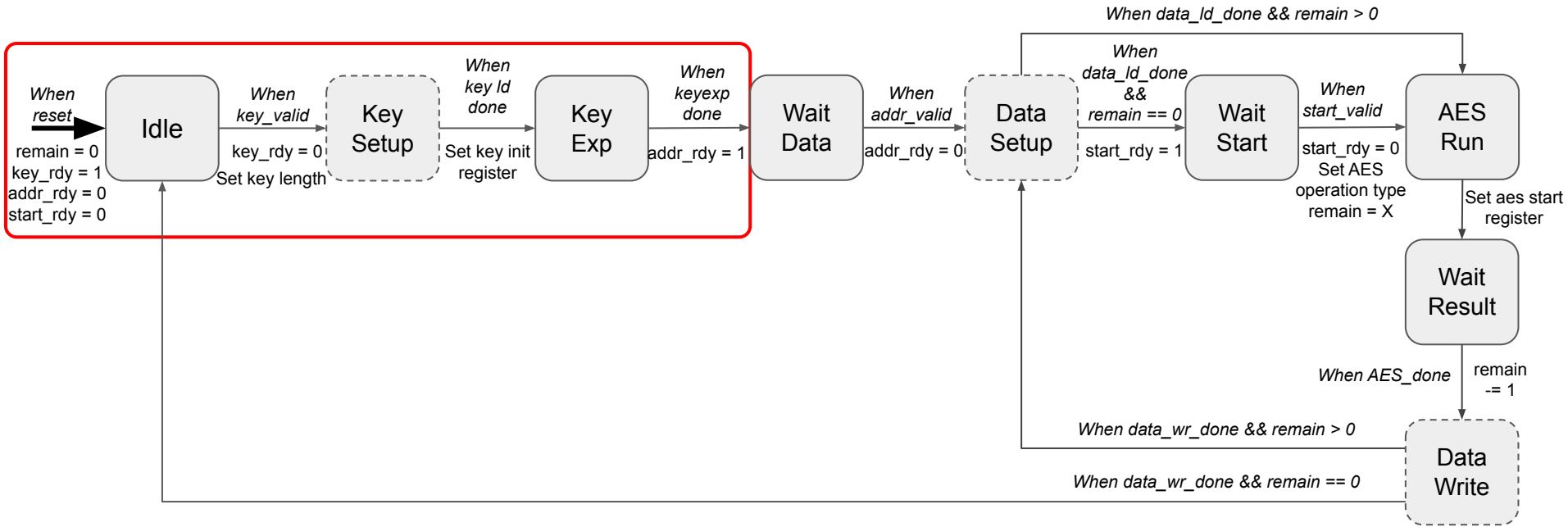
6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.

ENCRYPT/DECRYPT

9. Read out the ciphertext block from the result registers.

DATA WRITE

Main Controller - FSM



AES Core Usage

1. Load the key to be used by writing to the key register words.
2. Set the key length by writing to the config register.
3. Initialize key expansion by writing a one to the init bit in the control register.
4. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the key expansion has been completed.

KEY SETUP

5. Write the cleartext block to the block registers.

DATA LOAD

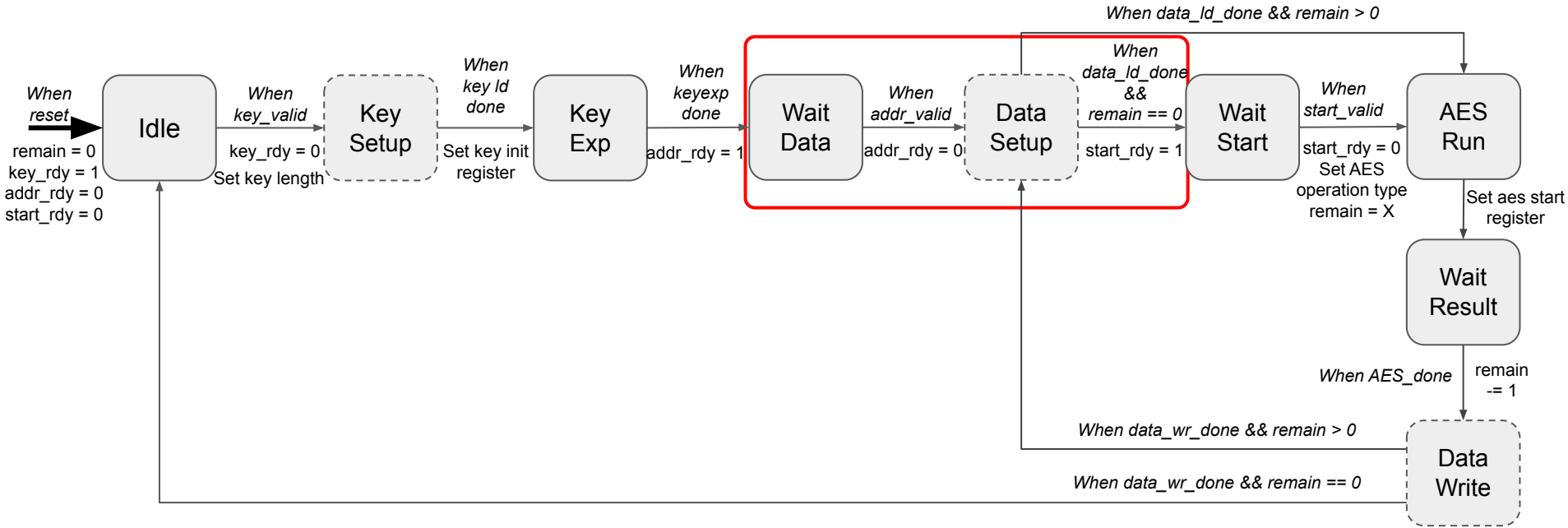
6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.

ENCRYPT/DECRYPT

9. Read out the ciphertext block from the result registers.

DATA WRITE

Main Controller - FSM



AES Core Usage

1. Load the key to be used by writing to the key register words.
2. Set the key length by writing to the config register.
3. Initialize key expansion by writing a one to the init bit in the control register.
4. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the key expansion has been completed.

KEY SETUP

5. Write the cleartext block to the block registers.

DATA LOAD

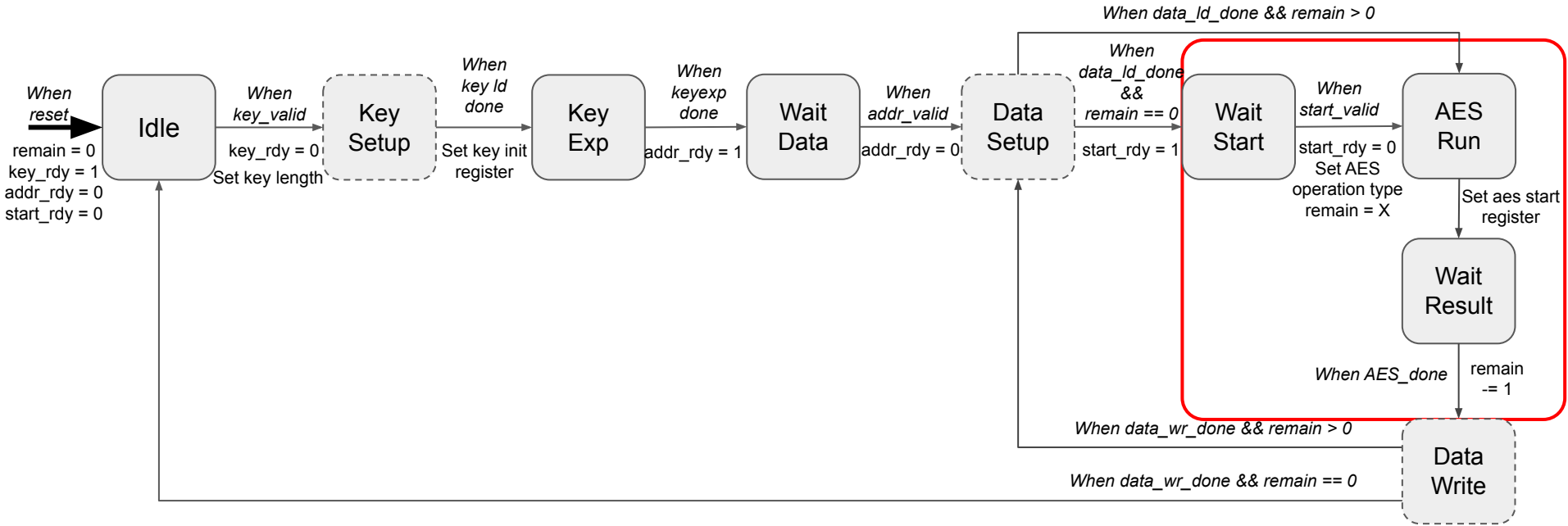
6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.

ENCRYPT/DECRYPT

9. Read out the ciphertext block from the result registers.

DATA WRITE

Main Controller - FSM



AES Core Usage

1. Load the key to be used by writing to the key register words.
2. Set the key length by writing to the config register.
3. Initialize key expansion by writing a one to the init bit in the control register.
4. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the key expansion has been completed.

KEY SETUP

5. Write the cleartext block to the block registers.

DATA LOAD

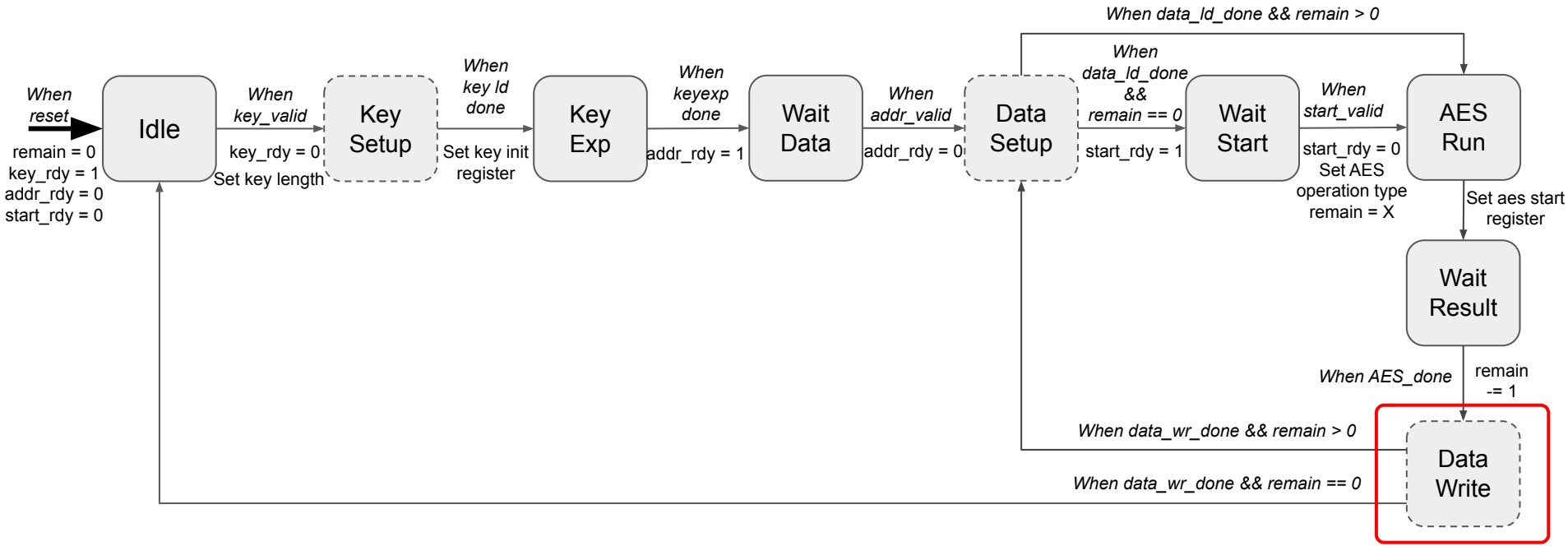
6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.

ENCRYPT/DECRYPT

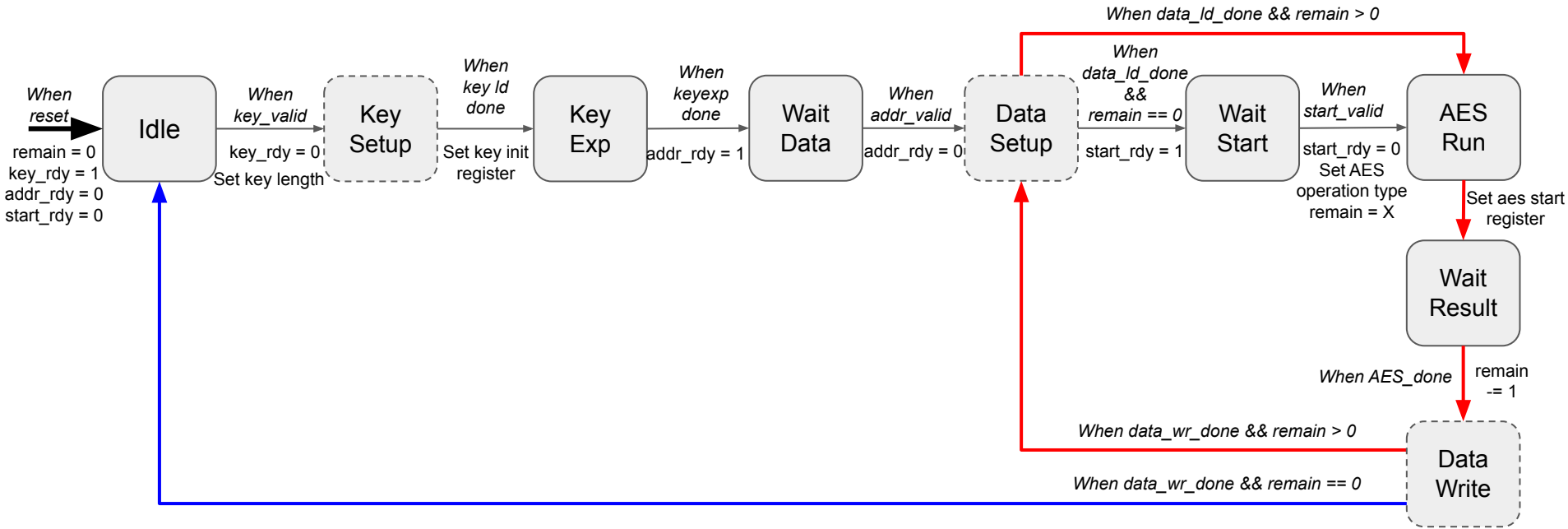
9. Read out the ciphertext block from the result registers.

DATA WRITE

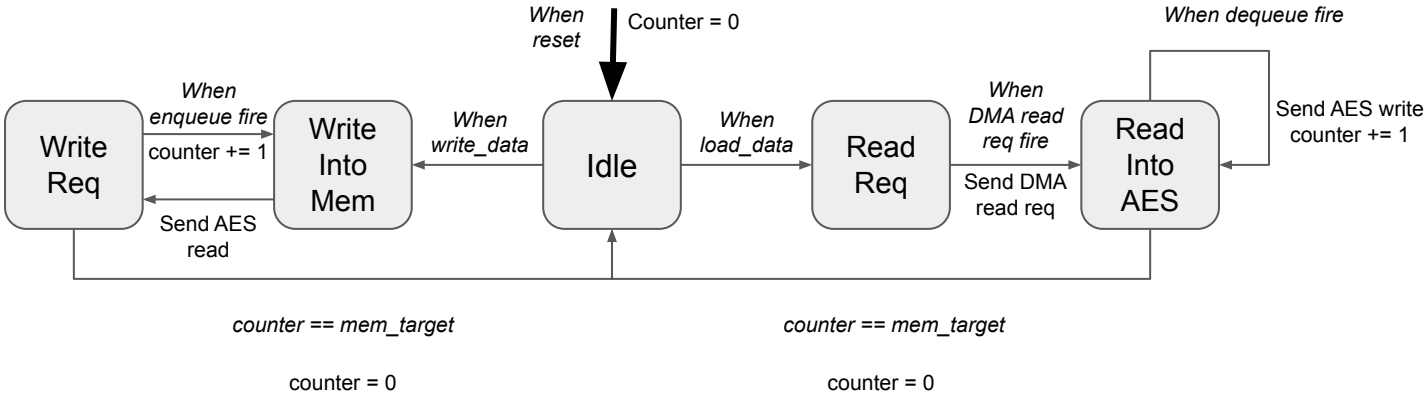
Main Controller - FSM



Main Controller - FSM

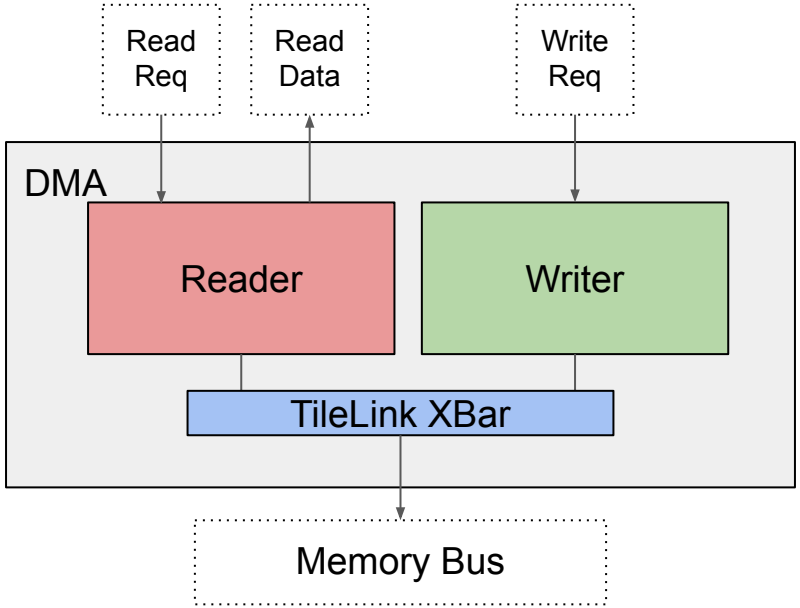


Memory Controller - FSM



DMA

- Services memory requests from controller
- Contains reader and writer sub-blocks
  - Concurrent read + write operations
- Interfaces with memory bus via TileLink
  - TileLink is a chip-scale interconnect in Rocket
- Shared DMA design in baseband module
  - More details presented in that presentation



DMA Read Request

```
addr = UInt(32.W)
totalBytes = UInt(9.W)
```

DMA Read Data

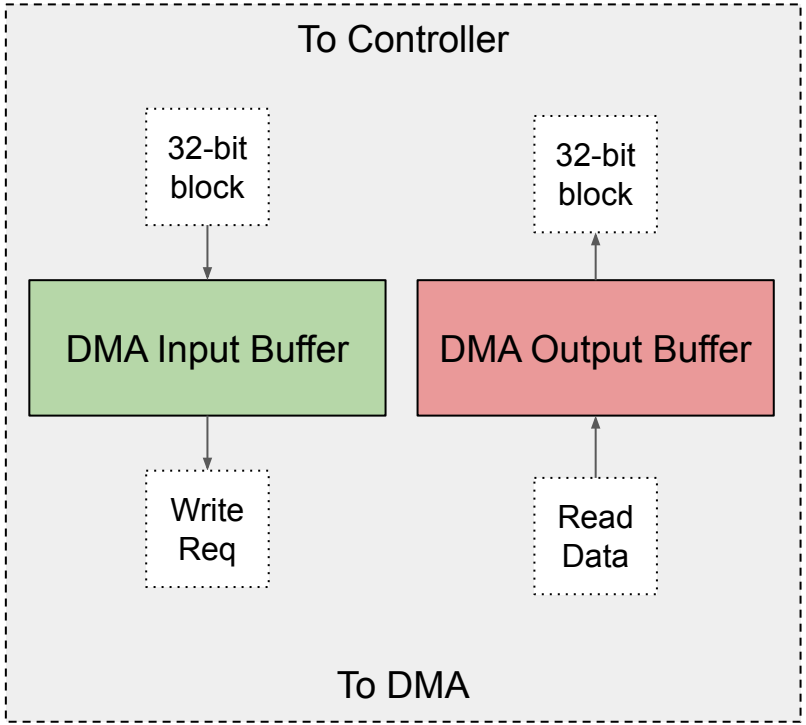
```
data = UInt(X.W)
```

DMA Write Request

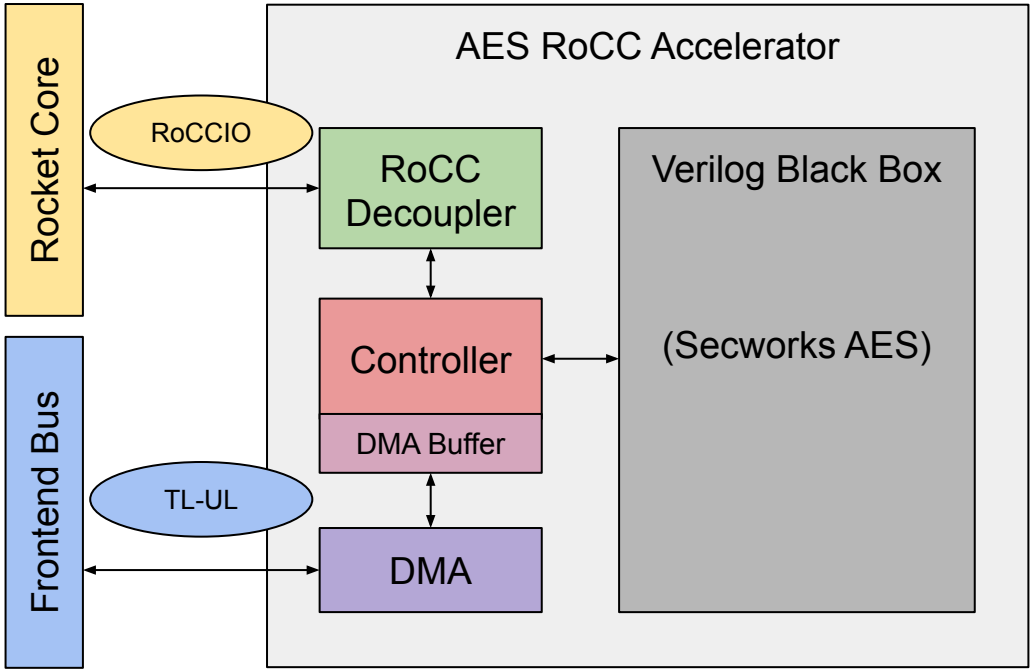
```
addr = UInt(32.W)
data = UInt(X.W)
totalBytes = UInt(Y.W)
```

DMA Buffers

- AES Core operates with 32-bit blocks
- DMA “packet size” may be different
- DMA Input Buffer
  - Takes 32-bit blocks and breaks into packet-sized blocks if required
- DMA Output Buffer
  - Takes packet-sized blocks and breaks/combines into 32-bit blocks



AES Accelerator Top-Level Diagram

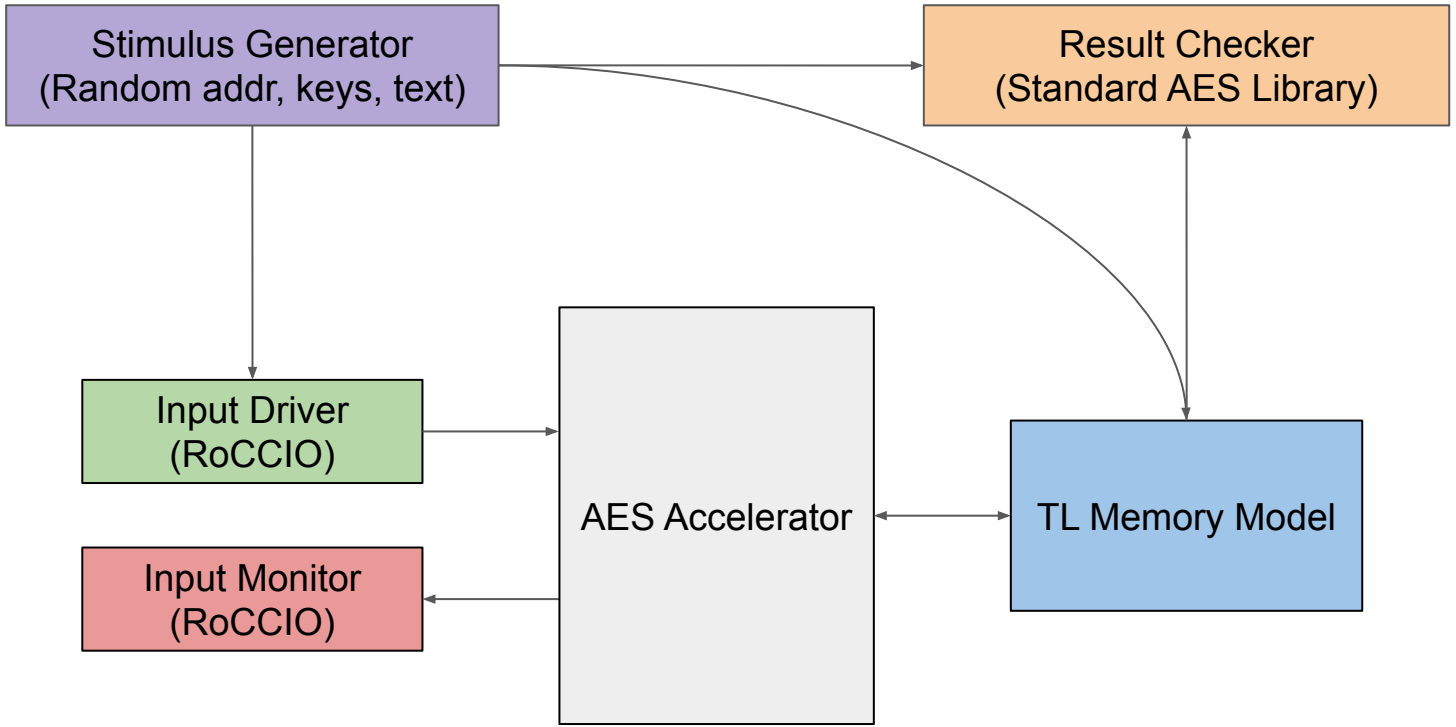


Verification: Secworks AES Core

- Given TB alongside RTL
- Flexible design
  - Option to add keys/input/expected
  - Generate own randomized keys/input
  - AES Library to calculate expected

```
*** TC 22 ECB mode test started.
cycle: 0x0000000000000d69
State of DUT
------------
ctrl_reg:   init   = 0x0, next   = 0x0
config_reg: encdec = 0x0, length = 0x1

block: 0xb6ed21b9, 0x9ca6f4f9, 0xf153e7b1, 0xbeafed1d

*** TC 22 successful.

*** TC 23 ECB mode test started.
cycle: 0x0000000000000e55
State of DUT
------------
ctrl_reg:   init   = 0x0, next   = 0x0
config_reg: encdec = 0x0, length = 0x1

block: 0x23304b7a, 0x39f9f3ff, 0x067d8d8f, 0x9e24ecc7

*** TC 23 successful.

*** All 16 test cases completed successfully

*** AES simulation done. ***
```

Verification: Accelerator Testbench Design



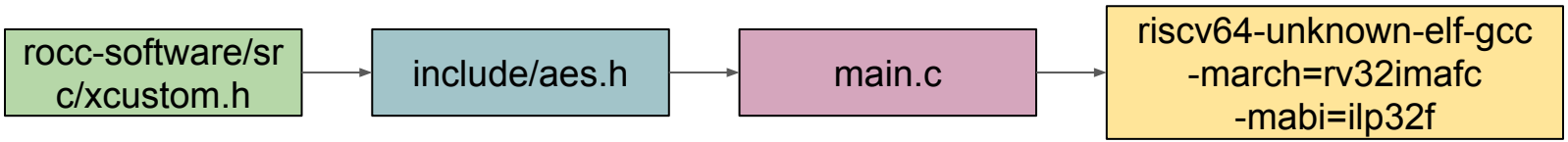
Software

- How does a user code for the accelerator?
  - Include the C header to access accelerated crypto routines
  - Different routines implement different cipher modes in software
  - Routines also handle key setup and polling
  - Provides a level of abstraction above embedded assembly

RoCC Assembly

- Handy pre-existing repo (IBM/rocc-software) defines some useful RoCC generating macros
- Header file implements our extensions with said macros

```
#define aes_extended_keysetup128(key_address) \
    ROCC_INSTRUCTION_0_R_R(AES_OPC, key_address, 0, 0)

#define aes_extended_keysetup256(key_address) \
    ROCC_INSTRUCTION_0_R_R(AES_OPC, key_address, 0, 1)

#define aes_extended_addressload(src_address, dest_address) \
    ROCC_INSTRUCTION_0_R_R(AES_OPC, src_address, dest_address, 2)

#define aes_extended_encryptblocks(num_blocks) \
    ROCC_INSTRUCTION_0_R_R(AES_OPC, num_blocks, 0, 3)

#define aes_extended_decryptblocks(num_blocks) \
    ROCC_INSTRUCTION_0_R_R(AES_OPC, num_blocks, 0, 4)

#define aes_extended_querystatus(status) \
    ROCC_INSTRUCTION_R_R_R(AES_OPC, status, 0, 0, 5)

#define aes_extended_queryinterrupt(int_type) \
    ROCC_INSTRUCTION_R_R_R(AES_OPC, int_type, 0, 0, 6)

#define aes_fence() \
    asm volatile("fence")
```



Encryption modes

- The accelerator only implements the block cipher, so we have to implement individual cipher modes in software
- For ECB, we can tell the accelerator the number of blocks we want to encrypt
- For other modes like CBC, CTR, we have to encrypt blocks one at a time

```
static int AES_ECB_encrypt(uint8_t* key, int keylen, uint8_t* src, uint8_t* dest, int length) {
    if (keylen == 128) {
        aes_extended_keysetup128(*key);
    } else if (keylen == 256) {
        aes_extended_keysetup256(*key);
    } else {
        return 1; // other key lengths not supported
    }
    aes_extended_addressload(src, dest);
    aes_extended_encryptblocks((int) (length / 16));  // AES block length == 128 bits == 16 bytes
    int status = 1;
    while (status)
        aes_extended_querystatus(status);
    return 0;
}
```

Status

- Implementation - Completed
  - All sub-blocks are implemented and sanity checked with unit-level tests
- Software Stack
  - Completed intrinsics of custom RISC-V instructions
  - In process of implementing various AES Modes
- Verification
  - Subcomponent verification completed
  - Currently setting up accelerator-level testbench
  - Integrated into SoC with software tests pending

RF Baseband

Team Introduction

Ryan Lund

5th Year M.S. Student

Griffin Prechter

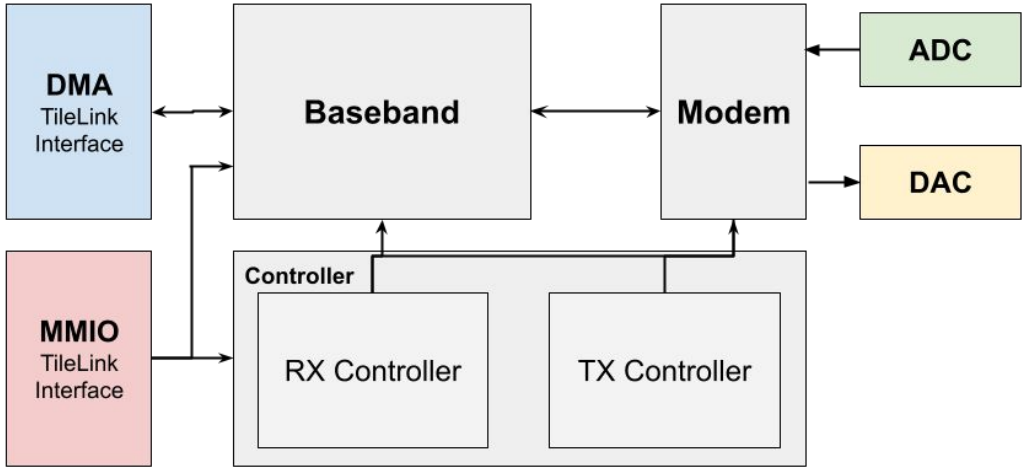
5th Year M.S. Student

Functional Goals

- Send and receive 1M Uncoded Link Layer Packets
  - With a design that can be expanded for more packet types (2M, LE Coded) over time
- Manage RF tuning constants
  - Some are CPU set while automatically tuned (AGC)
- Resilience to real world circumstances
  - Demodulation resilient to noise
  - Packet disassembler gracefully handled malformed packets
- Target 20μS transmit latency (command start to analog RF start)
- Meet BLE timing requirements
  - Allow CPU to have enough control over TX and RX; appropriate interrupts

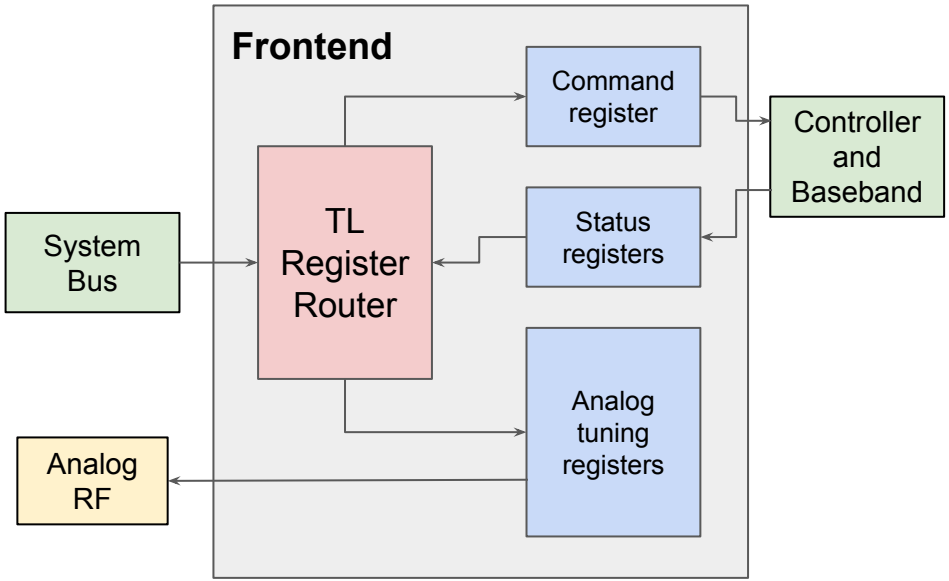
Design: Overview

- Five main functional blocks
  - DMA
  - MMIO Frontend
  - Baseband
  - Controller
  - Modem
- Interface between digital and analog
  - Modem interacts with ADC/DAC
  - Frontend contains tuning registers for RF components



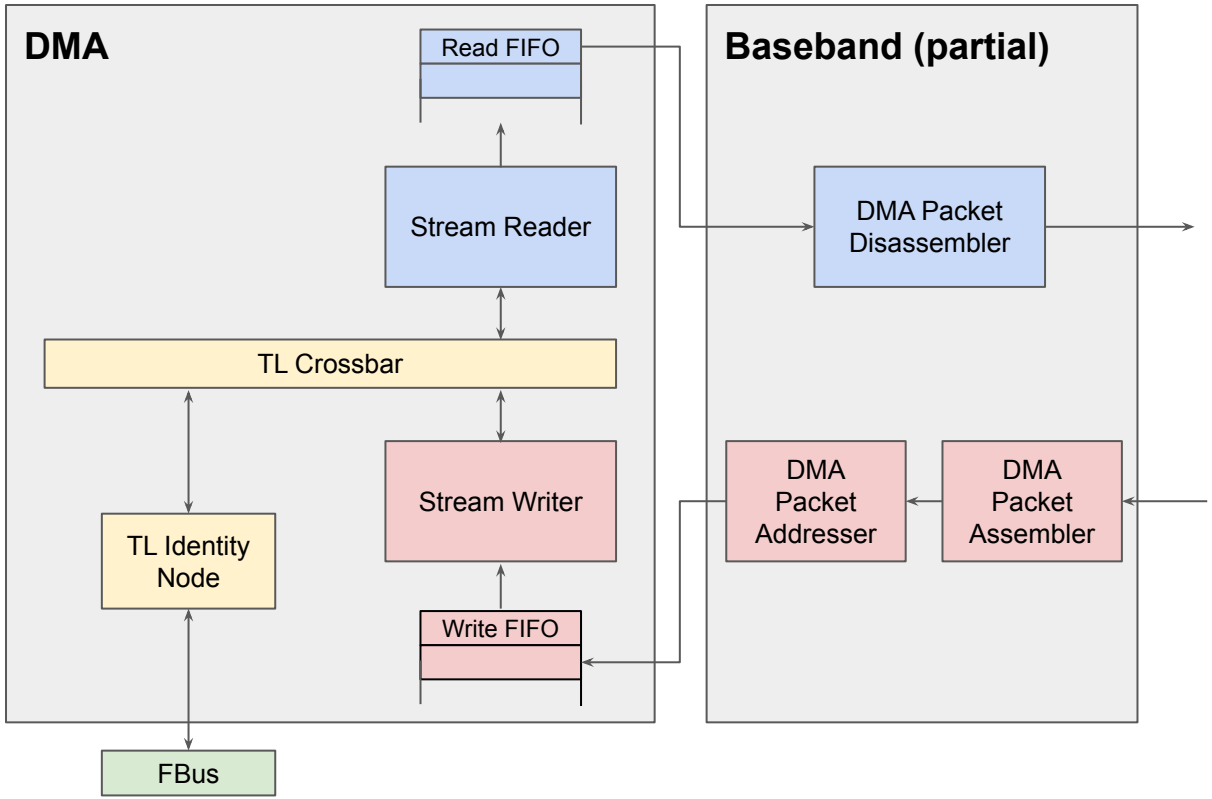
Design: MMIO Frontend

- TL register router component exposes registers as memory mapped devices
  - Command register from CPU
  - Status registers for CPU
  - Analog tuning parameters (e.g. I/Q filter resistors)
- Houses connection to interrupt bus and PLIC
- Parameterizable to allow for attachment at different base addresses



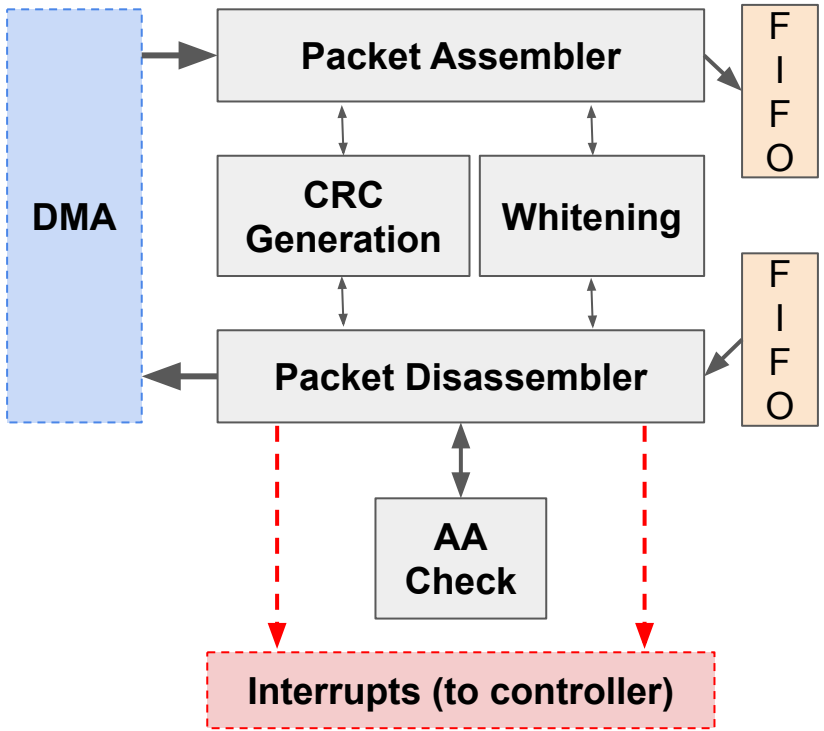
Design: DMA Interface

- Stream reader and writer generate TL transactions
  - Get, put, put partial
- Single command for reads
- Multiple commands for writes
  - TL bus width of data per command
- Allows for simultaneous reading and writing



Design: Baseband

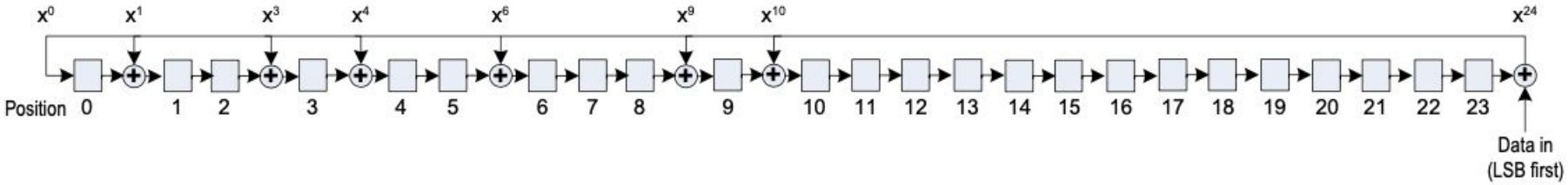
- Send Command:
  - Packet Assembler begins the bit-streaming process, pulling data from specified address via DMA.
- Receive Command:
  - Packet Disassembler begins to wait for preamble; will write to specified address via DMA.
- Debug Command:
  - Both TX and RX chains active, loopback enabled to allow for loopback testing.
- PD sends interrupts to alert the CPU of packet reception and success/failure.



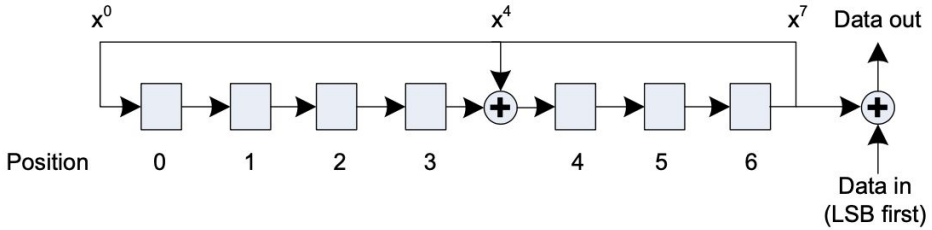
Design: Baseband

Bit Stream Processing

- Cyclic Redundancy Check (CRC): implemented as an LFSR

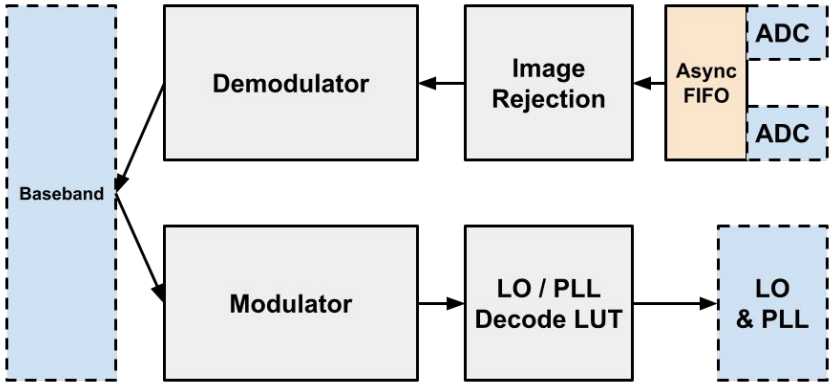


- Data Whitening: intended to avoid continuous streams of all 1s or all 0s, implemented as an LFSR



Design: Modem

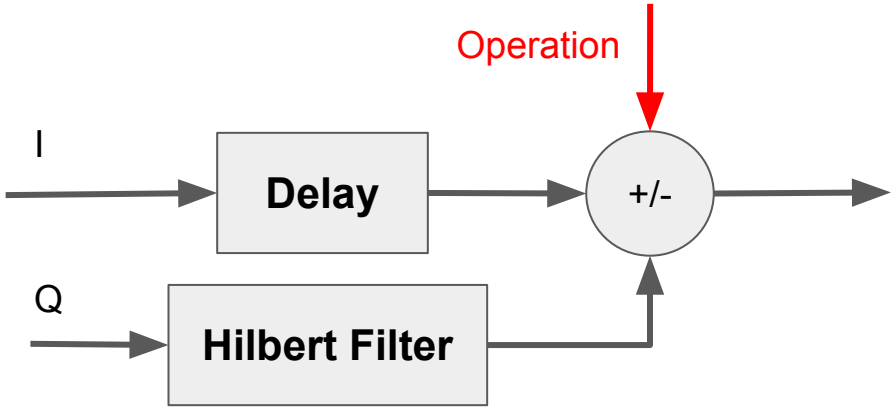
- Designed for GFSK
- TX side
  - Gaussian filter on bitstream input from baseband (with per bit over-sampling)
  - Resulting value mapped to LO / PLL control signals via programmable LUT
- RX side
  - Async FIFO accounts for non-sync ADC logic
  - AGC performed on FIFO output
  - Image rejection uses Hilbert Filter
  - Non-coherent demodulation with bandpass and envelope detection



Design: Modem

Digital Image Rejection

- Hilbert Filter implemented as a 29 tap FIR filter with fixed point arithmetic
- CPU-configurable control signal to choose between adding and subtracting between I and Q signals.



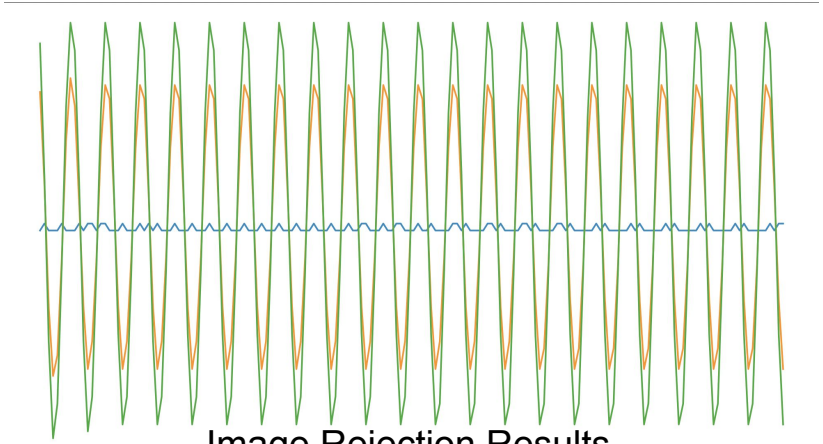
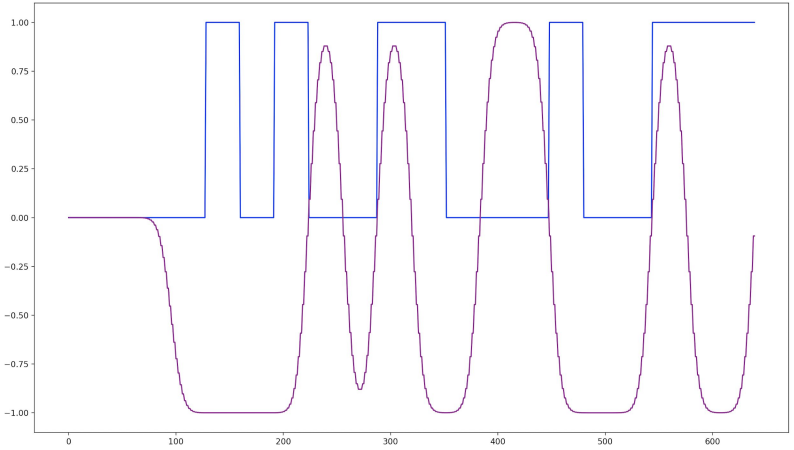


Image Rejection Results

Design: Modem

Modulation and Demodulation

- Modulation performed using integral tap Gaussian FIR filter
- Demodulation performed using dual bandpass filters at W_0 and W_1 followed by envelope detection
  - Decision made by symbol time majority



Data Input vs Gaussian FIR Output

Design: Controller

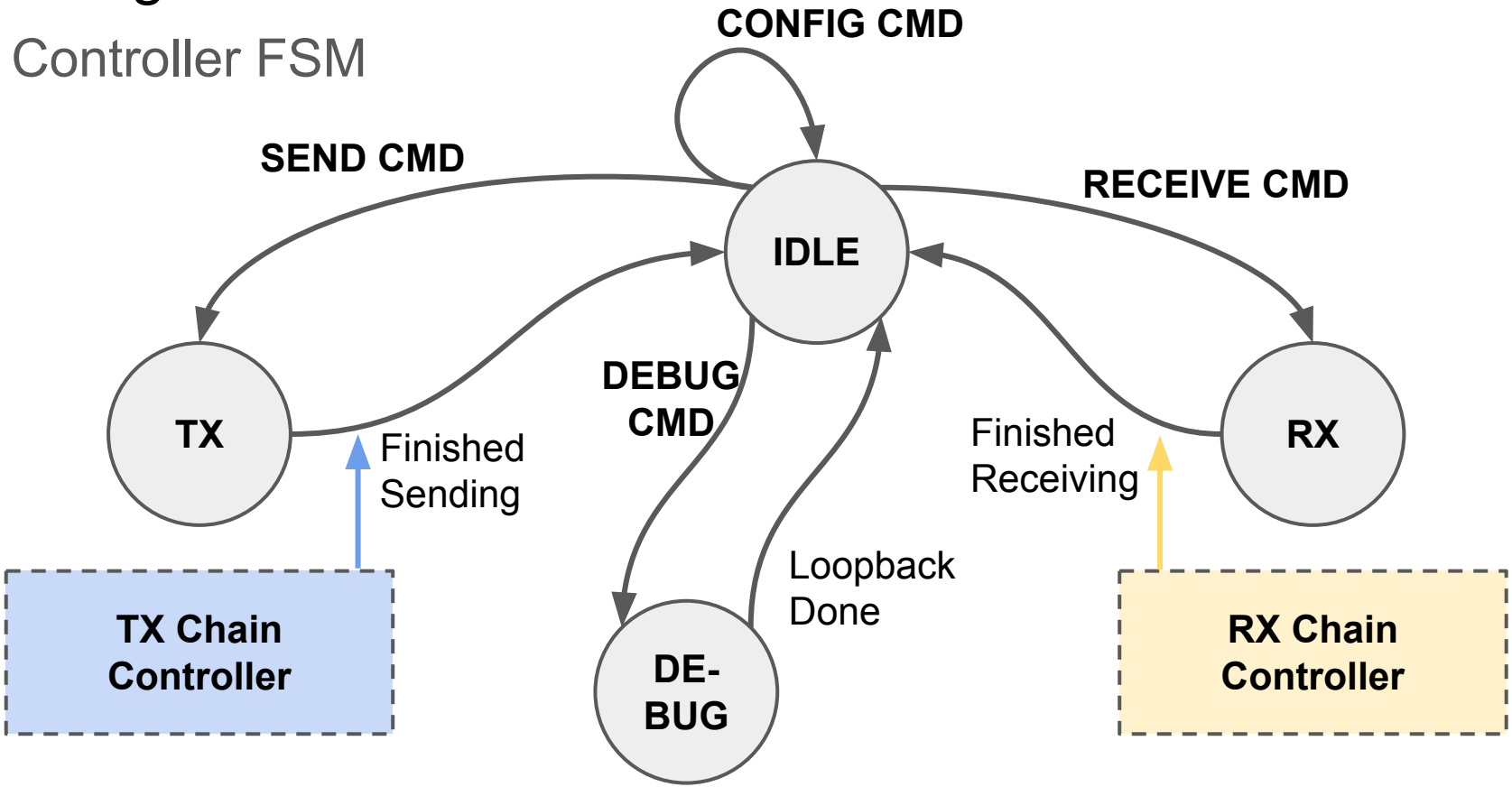
- Commands contain 4 fields
  - Inst. 1: Primary function
  - Inst. 2: Sub-function
  - Data: Small values
  - Additional data: Wide values
- Synchronizes DMA interface, baseband, and modem for command execution
  - Configures loopbacks for debug execution
- Manages interrupts and status registers

Bits	31-8	7-4	3 - 0
Field	Data	Instruction 2	Instruction 1

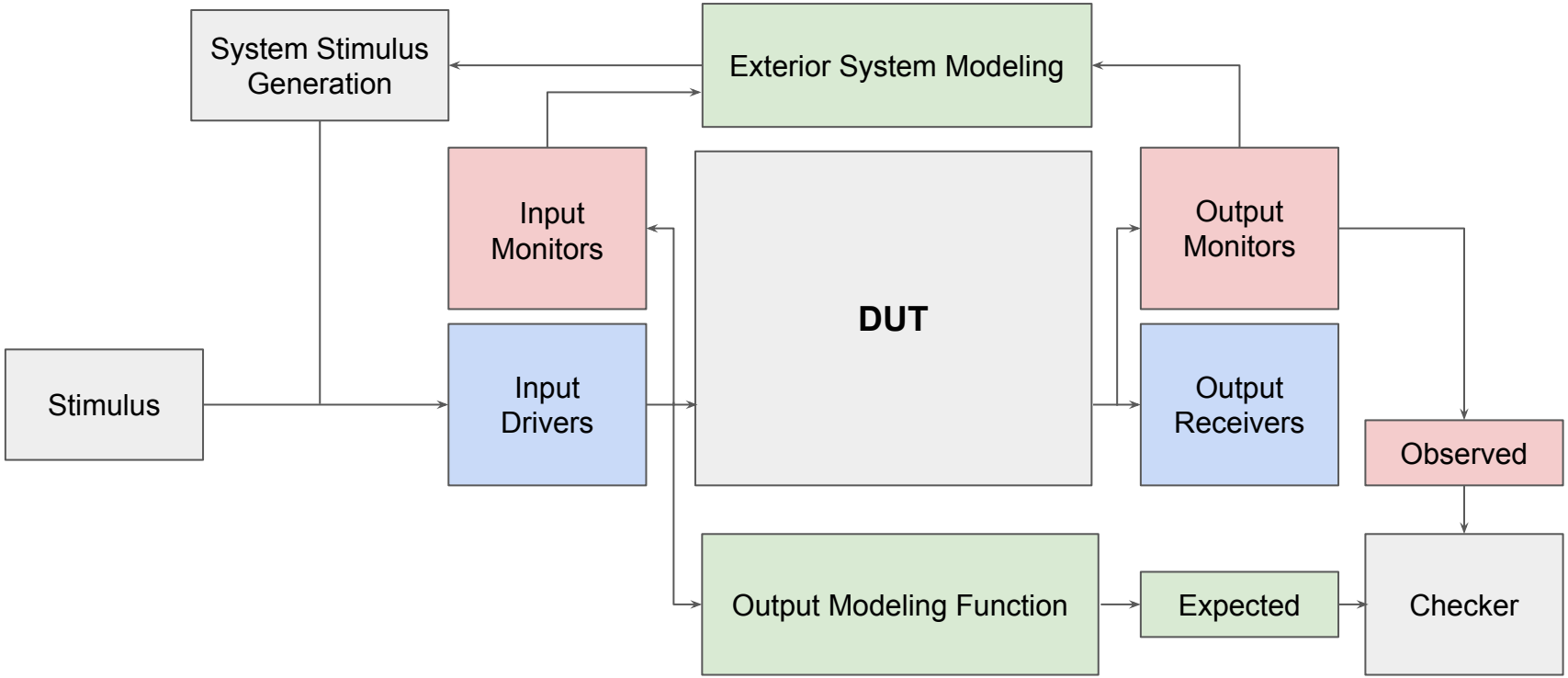
Bits	31-0
Field	Additional Data

Design: Controller

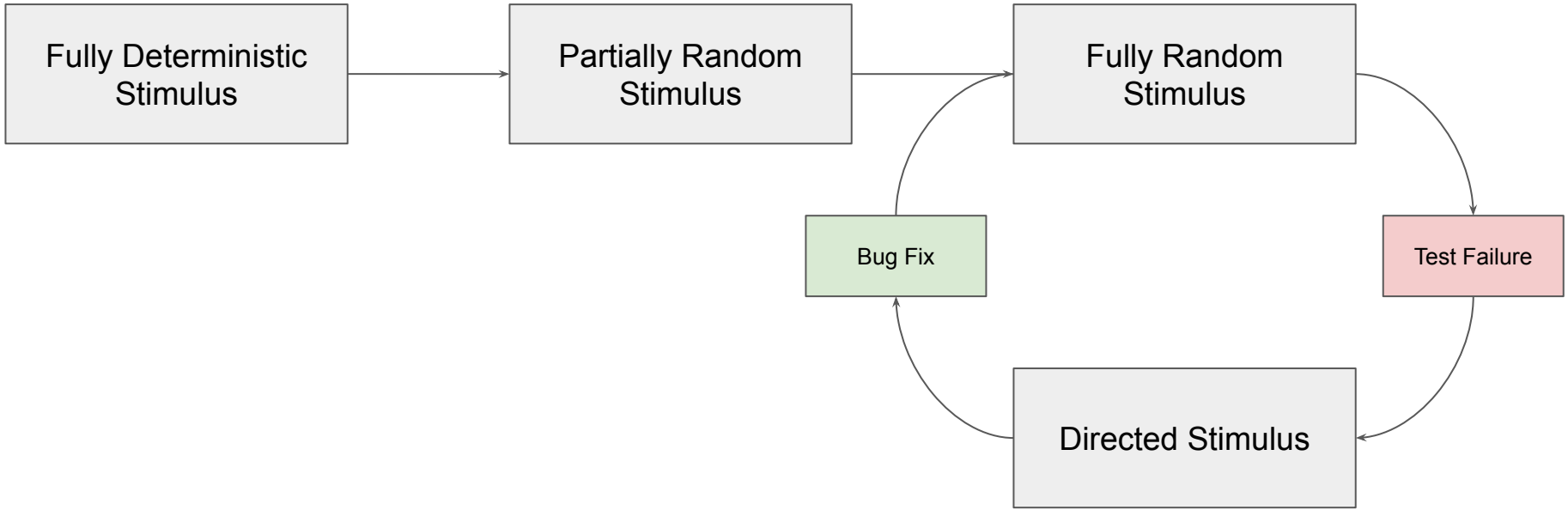
Controller FSM



Verification: Testbench Design



Verification: Stimulus Generation



Software

- MMIO controls are easy to interface with in C
  - No special ISA extensions required
  - Generic mmio.h file allows for register reads and writes from 8 to 64-bits
- Series of macros defined to streamline interface
- Work to be done to integrate with chosen RTOS

```
// Address map
#define BASEBAND_INST 0x8000
#define BASEBAND_ADDITIONAL_DATA 0x8004
#define BASEBAND_STATUS0 0x8008
#define BASEBAND_STATUS1 0x800C
#define BASEBAND_STATUS2 0x8010
#define BASEBAND_STATUS3 0x8014
#define BASEBAND_STATUS4 0x8018

…

// Instruction macro
#define BASEBAND_INSTRUCTION(primaryInst, secondaryInst, data)
((primaryInst & 0xF) + ((secondaryInst & 0xF0) << 4) + ((data &
0xFFFFFF00) << 24))

// Primary instructions
#define BASEBAND_CONFIG 0
#define BASEBAND_SEND 1
#define BASEBAND_RECEIVE 2
#define BASEBAND_RECEIVE_EXIT 3
#define BASEBAND_DEBUG 15

// Secondary instructions
#define BASEBAND_CONFIG_CRC_SEED 0
#define BASEBAND_CONFIG_ACCESS_ADDRESS 1
#define BASEBAND_CONFIG_CHANNEL_INDEX 2
#define BASEBAND_CONFIG_ADDITIONAL_FRAME_SPACE 3
#define BASEBAND_CONFIG_LO_LUT 4
```

```
reg_write32(BASEBAND_ADDITIONAL_DATA, 5);
reg_write32(BASEBAND_INST, BASEBAND_INSTRUCTION(BASEBAND_CONFIG, BASEBAND_CONFIG_CHANNEL_INDEX, 0));
```

Into the Weeds: How the Baseband is Integrated

- Baseband attachment required custom Chisel
  - Four sections that need connecting: DMA to SBus, MMIO to FBus, Interrupt to IBus, Analog IO to ChipTop
- Accomplished via mixin, IO punch throughs, and harness binders (for simulation)
  - The magic: This is all automated, the connections can be made invariant to Baseband changes
  - Allows us to accommodate a continual stream of analog interface changes

```
trait CanHavePeripheryBLEBasebandModem { this: BaseSubsystem =>
  val baseband = p(BLEBasebandModemKey).map { params =>
    val baseband = LazyModule(new BLEBasebandModem(params,
fbus.beatBytes))

    pbus.toVariableWidthSlave(Some("baseband")) { baseband.mmio }
    fbus.fromPort(Some("baseband"))() := baseband.mem
    ibus.fromSync := baseband.intnode

    val io = InModuleBody {
      val io = IO(new
BLEBasebandModemAnalogIO(params)).suggestName("baseband")
      io <> baseband.module.io
      io
    }
    io
  }
}
```

```
class WithBLEBasebandModemPunchthrough(params: BLEBasebandModemParams =
BLEBasebandModemParams()) extends OverrideIOBinder({
  (system: CanHavePeripheryBLEBasebandModem) => {
    val ports: Seq[BLEBasebandModemAnalogIO] = system.baseband.map({ a =>
      val analog = IO(new
BLEBasebandModemAnalogIO(params)).suggestName("baseband")
      analog <> a
      analog
    }).toSeq
    (ports, Nil)
  }
})
```

Status

- Baseband
  - Fully implemented
  - Highly tested on a unit level
  - Integrated into SoC with software tests pending
- Modem
  - Work in progress, constantly evolving
  - Targeting completion by end of Spring Break
- Software stack
  - In preliminary phase
  - To date, used mainly for SoC integration testing of components

Integration

Introduction

Troy Sheldon

3rd Year EECS Undergrad

Kareem Ahmad

5th Yr M.S., advised by Prof. Sophia Shao

Focus on HW for ML

Jackson Paddock

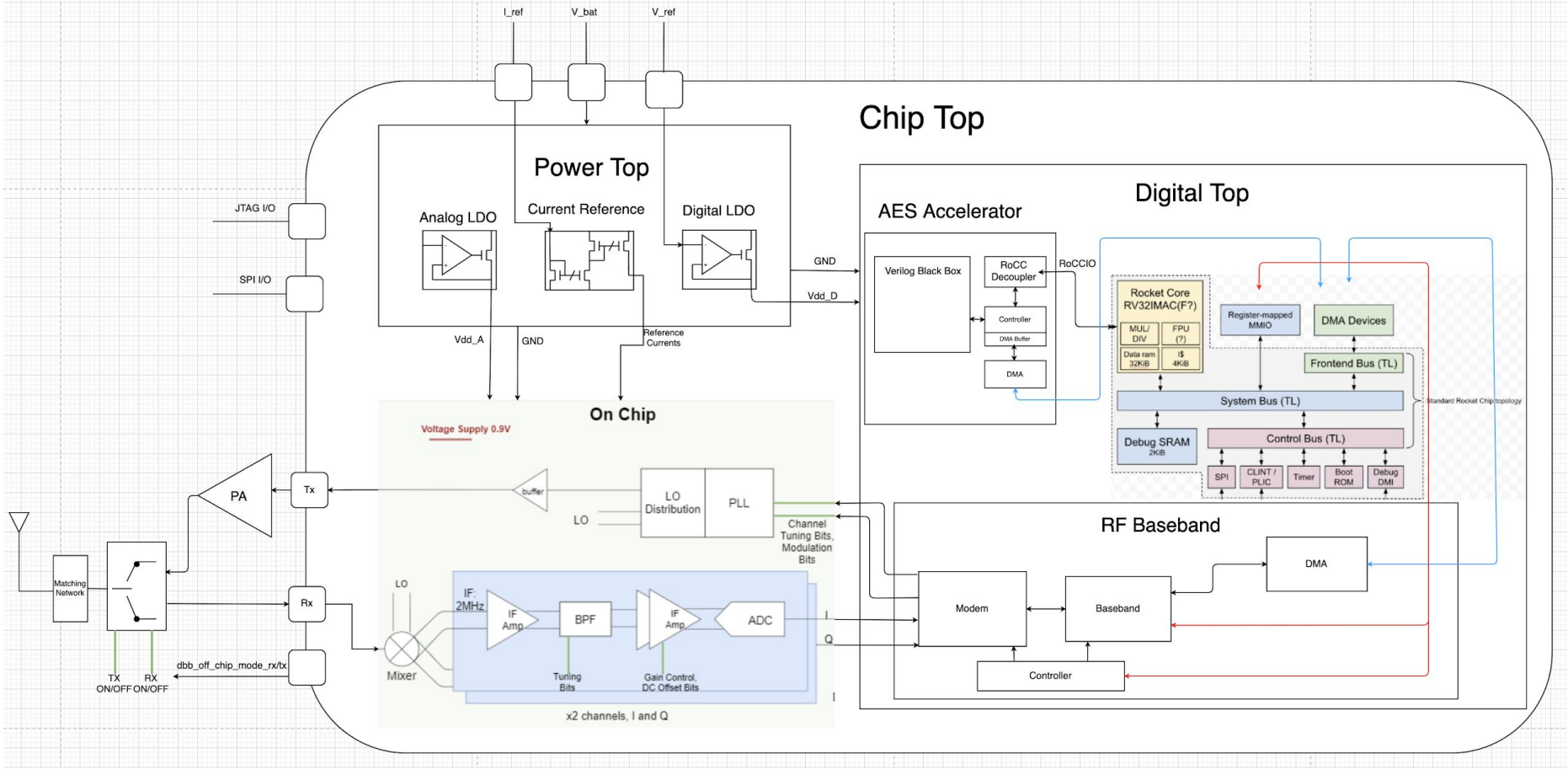
5th Yr M.S., advised by Prof. Pister

Focus on analog circuit design

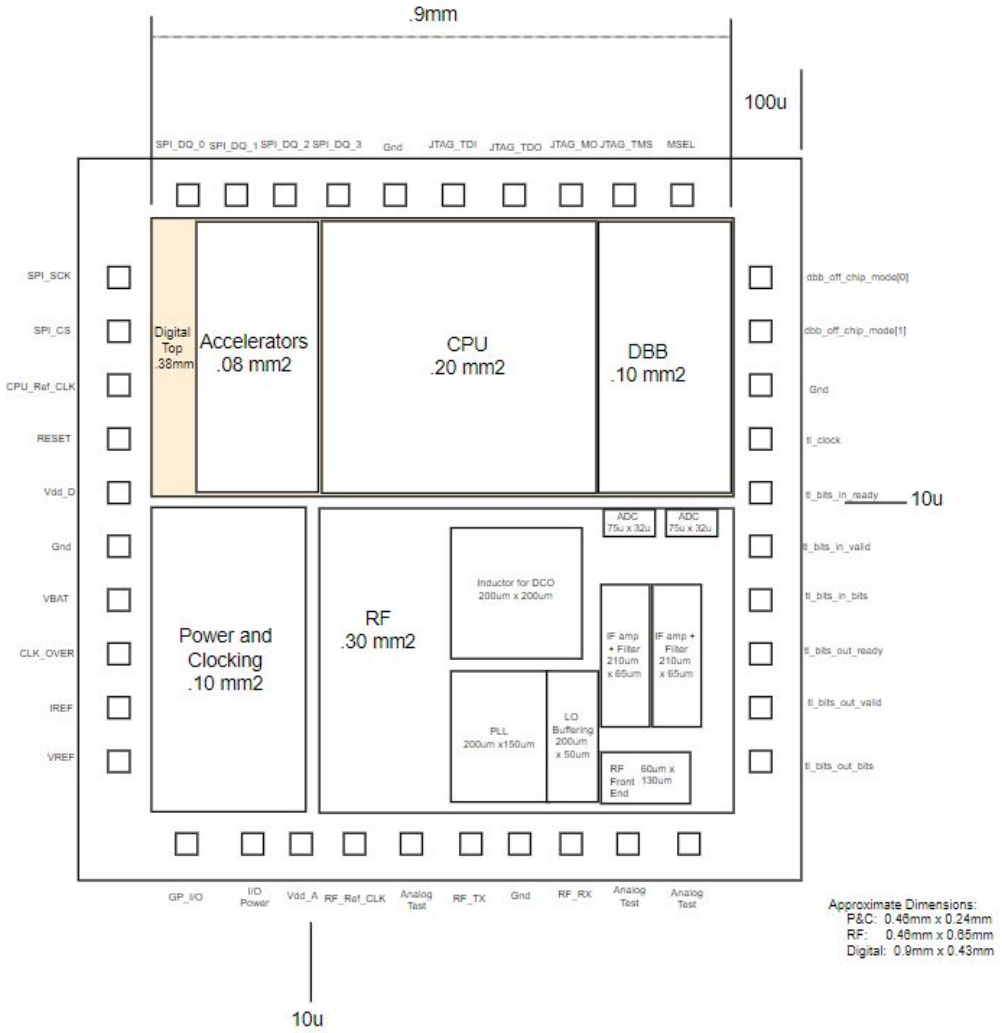
Dylan Brater

3rd Year EECS Undergrad

Top Level Diagram



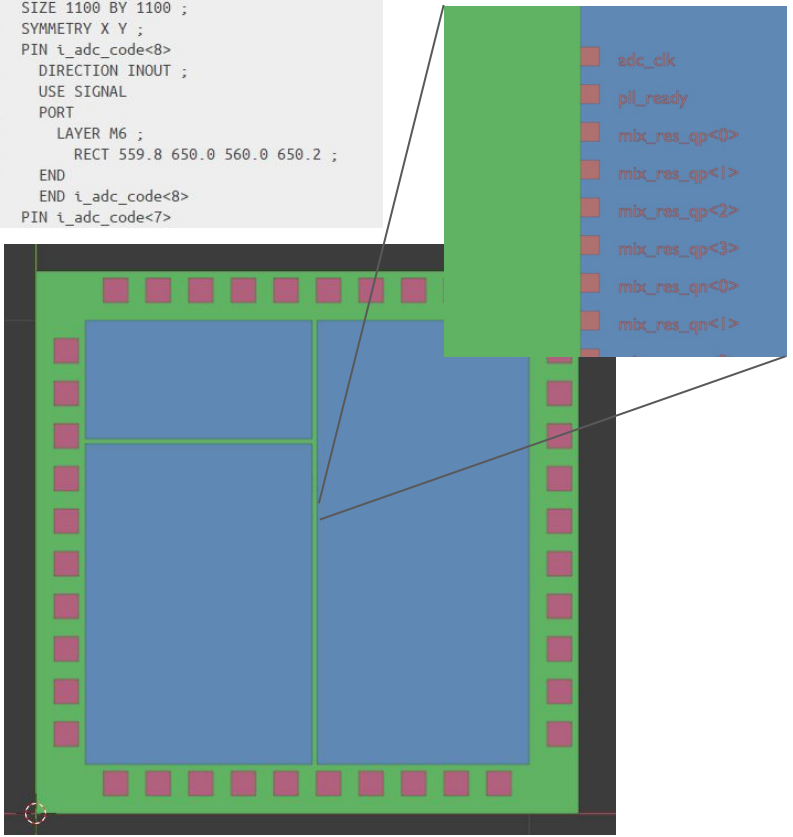
Floorplan


.9mm
100u
SPI_DQ_0
SPI_DQ_1
SPI_DQ_2
SPI_DQ_3
Gnd
JTAG_TDI
JTAG_TDO
JTAG_MO
JTAG_TMS
MSEL
SPI_SCK
SPI_CS
CPU_Ref_CLK
RESET
Vdd_D
Gnd
VBAT
CLK_OVER
IREF
VREF
Digital Top .38mm
Accelerators .08 mm2
CPU .20 mm2
DBB .10 mm2
Power and Clocking .10 mm2
RF .30 mm2
Inductor for DCO 200um x 200um
ADC 75u x 32u
ADC 75u x 32u
IF amp + Filter 210um x 65um
IF amp + Filter 210um x 65um
PLL 200um x150um
LO Buffering 200um x 50um
RF Front End 60um x 130um
dbb_off_chip_mode[0]
dbb_off_chip_mode[1]
Gnd
fl_clock
fl_bits_in_ready
10u
fl_bits_in_valid
fl_bits_in_bits
fl_bits_out_ready
fl_bits_out_valid
fl_bits_out_bits
GP_I/O
I/O Power
Vdd_A
RF_Ref_CLK
Analog Test
RF_TX
Gnd
RF_RX
Analog Test
Analog Test
10u
Approximate Dimensions:
P&C: 0.46mm x 0.24mm
RF: 0.46mm x 0.65mm
Digital: 0.9mm x 0.43mm

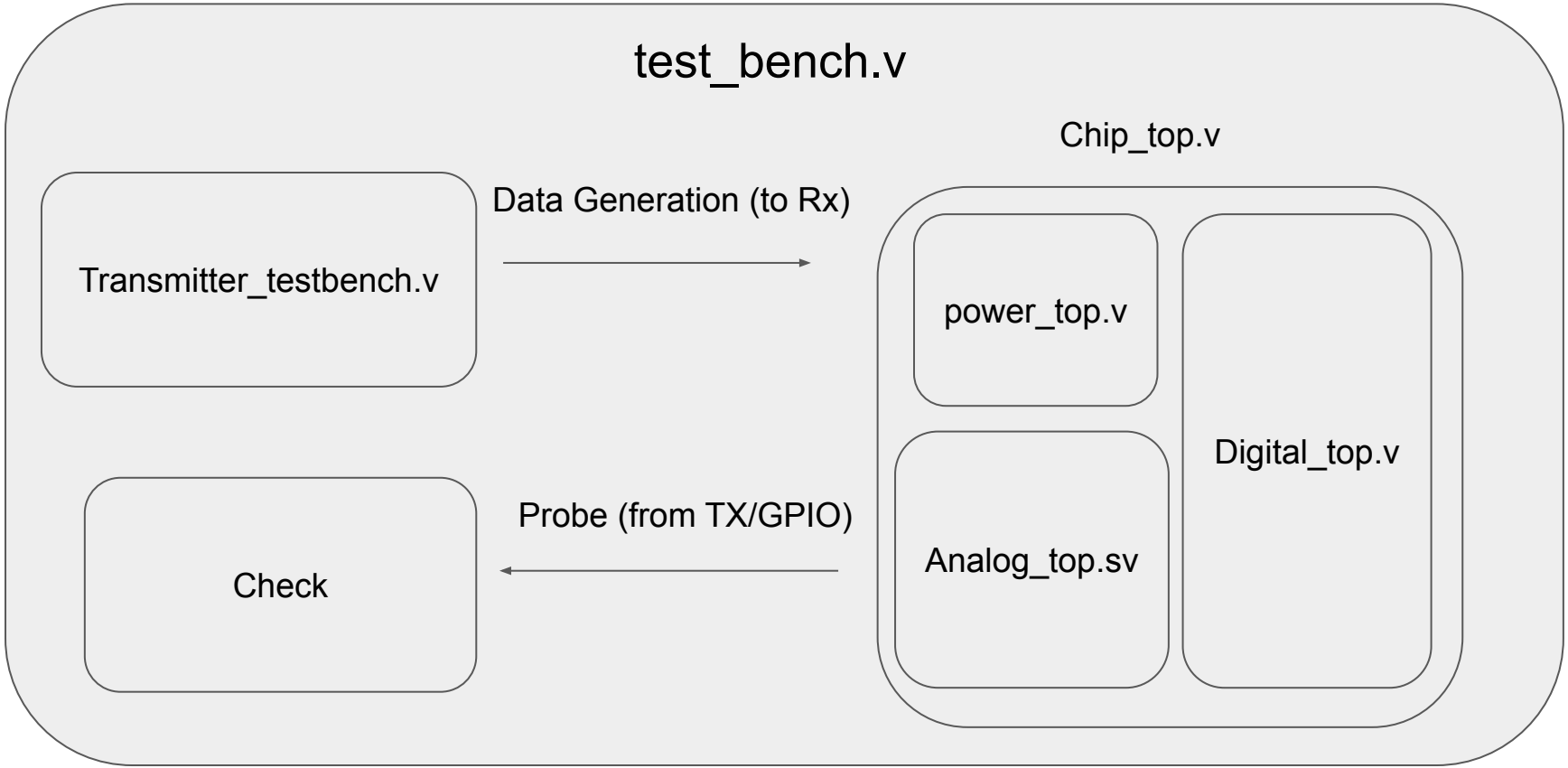
Layout and Planning

- Manual Layout
- 9 Metal layers
- M8 and M9 = Power Grid
- M6 and M7 = Inter-block connections

```
VERSION 5.6 ;
BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;

MACRO PinLocations
  CLASS BLOCK ;
  ORIGIN 0 0 ;
  FOREIGN PinLocations 0 0 ;
  SIZE 1100 BY 1100 ;
  SYMMETRY X Y ;
  PIN i_adc_code<8>
    DIRECTION INOUT ;
    USE SIGNAL
    PORT
      LAYER M6 ;
        RECT 559.8 650.0 560.0 650.2 ;
    END
    END i_adc_code<8>
  PIN i_adc_code<7>
```

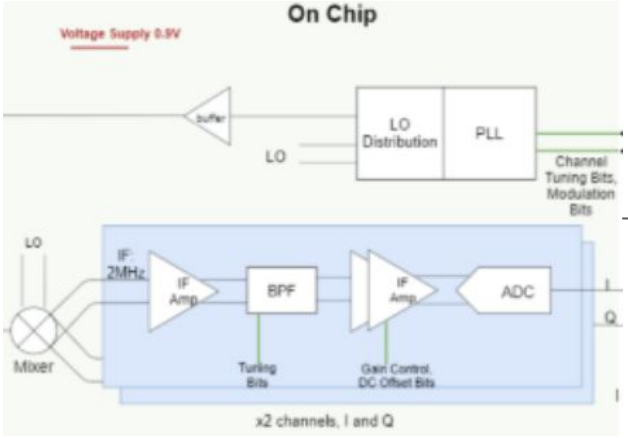


Verification Methodology



Verification Methodology

On Chip RF IC



Behavioral Verilog Model

Analog_top.sv

testbench.v

Digital Baseband

VCO_ctrl
Gain_ctrl
Filt_ctrl
I
8
Q
8

Verification Methodology

Early Stage Goals:

- Minimal design that only replicates Rx
- Ignores tuning bits from Baseband
- Receives a high frequency simple square wave from Receiver and outputs a lower frequency square wave from adc->modem

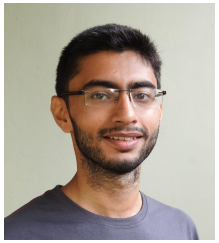
End Goals:

- Replicates both Tx and Rx
- Receives full sinusoid signal to be modulated based on tuning bits from baseband
- Outputs descretised sinusoid from adc->baseband
- Models LO and outputs full sinusoid from Tx->off chip

RF Transceiver

Team Introduction

Alex Moreno
Ph.D. Student

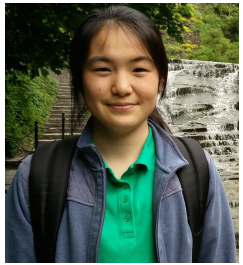


Shreesha S
Ph.D. Student

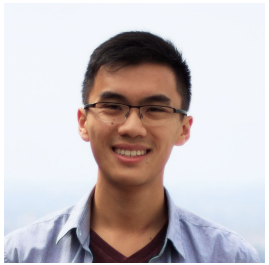
Kerry Yu
4th Year Undergrad

Leon Wu
4th Year Undergrad

Sherwin
4th Year Undergrad

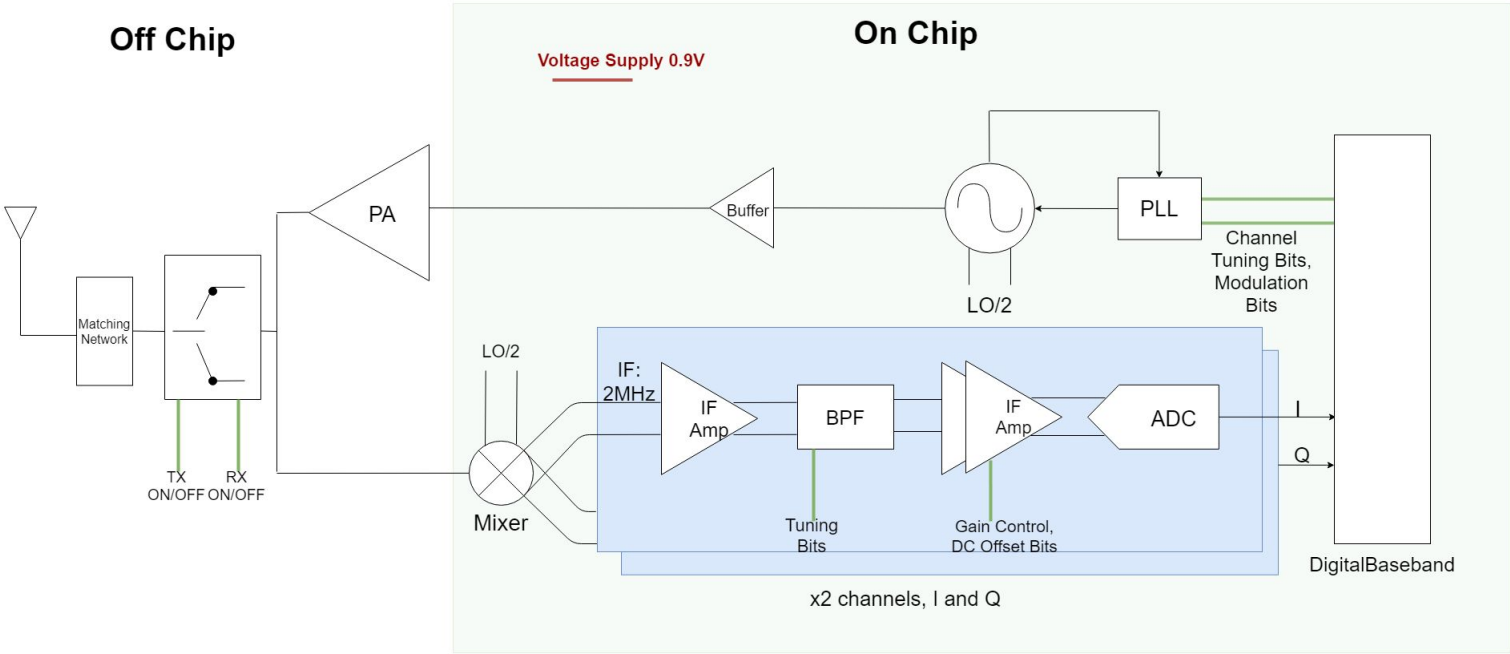


Felicia Guo
Ph.D Student



Jeffrey Ni
4th Year Undergrad

Overview



Target <5mW power consumption

RX Specifications

- Input range of -70dBm ~ -10dBm
  - Translates to a gain range of 2-60dB
- BER of 0.1% under the following conditions (-67dBm input signal power)
  - Translates to SNR of 12.5dB

Interference Band	$\frac{P\,carrier}{P\,interference}$	Pinterference
Co-channel	21dB	-89dBm
Adj. 1Mhz	15dB	-82dBm
Adj. 2MHz	-17dB	-44dBm
Adj. >= 3MHz	-27dB	-34dBm
Image	-9dB	-58dBm
Image +- 1MHz	-15dB	-52dBm

Band	Pinterference	Meas. Res
30MHz - 2000MHz	-30dBm	10MHz
2003MHz - 2399MHz	-35dBm	3MHz
BLE channels here		
2484MHz - 2997MHz	-35dBm	3MHz
3000MHz - 12.75GHz	-30dBm	25MHz

TX Specifications

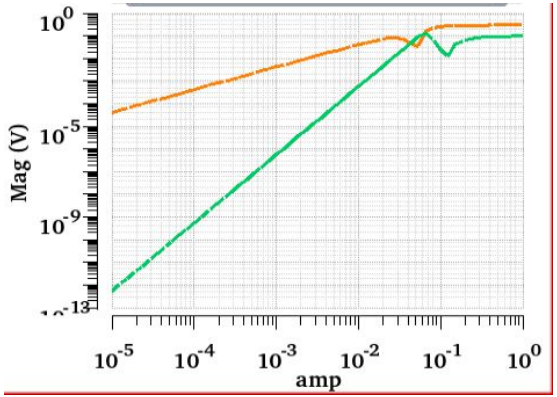
- Symbol period: 1us
- Gaussian Mask - BT=0.5
- Modulation index between 0.45 ~ 0.55
  - 0.495~0.505 for ‘stable modulation’
  - Target 0 = fc-250kHz; 1 = fc+250kHz
- Max center freq. drift: 150kHz
- Maximum freq drift: 50kHz
- Maximum drift rate: 400Hz/us
- Transmission power met using off chip PA

Receiver Link Budget + Modeling

	Rin (ohms)	Rout (ohms)	Gain (dB, 20log)	Max_in (V)	Output Noise (V integrated)	Total Gain (dB)	Total SNR (dB)	input voltage	signal amp
Antenna		50.00	0		-	-6.021			
Matching Network	50.00	225.00	0		-	-12.041			**2.50E-04**
Mixer	225.00	400.00	30	0.007	2.14E-04	17.959	17.919	7.00E-03	7.91E-03
Buffer	1.00E+99	1,000.00	0	0.1	7.00E-05	17.959	17.334	2.18E-03	7.91E-03
VGA	1.00E+99	1,000.00	12	0.1	8.00E-04	29.959	15.845	1.29E-03	3.15E-02
BPF	1.00E+99	1,000.00	2	0.5	1.75E-04	31.959	15.610	9.74E-04	3.96E-02
VGA	1.00E+99	1,000.00	12	0.1	8.00E-04	43.959	**15.348**	**3.83E-04**	1.58E-01

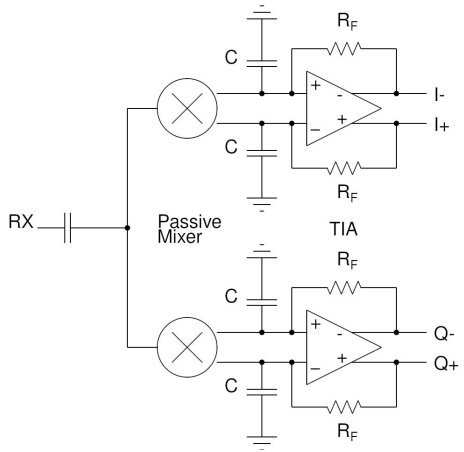
- Modelling also in virtuoso
  - Combination of verilogA and discrete ideal components
  - Similar to link budget in variables that can be adjusted

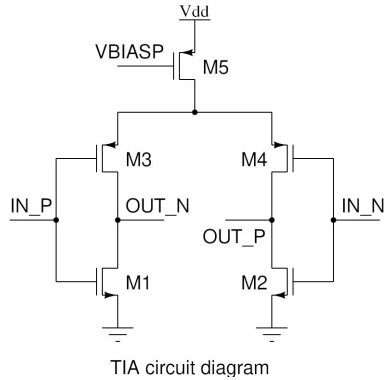
```
module nonlin_noise_va2 (inp, inn, outp, outn);
    input inp, inn;
    output outp, outn;

    parameter real Av=10;
    parameter real second=0;
    parameter real max_vin=0.001;
    parameter real noise=10**(-4);

    real a1 = 10**(Av/20);
    real a2 = a1/second;
    real a3 = 4*(-1)*a1/(3*max_vin*max_vin);

    electrical outp, outn, inp, inn, p1, p2;
    analog begin
        V(outp, outn) <+ max(min(a1*V(inp,inn)+ a2*V(inp, inn)*V(inp, inn)
                          + a3*V(inp, inn)*V(inp, inn)*V(inp, inn), a1*max_vin*1.1), -max_vin*a1*1.1);
        V(outp, outn) <+ white_noise(noise, "noise");
    end

endmodule
```



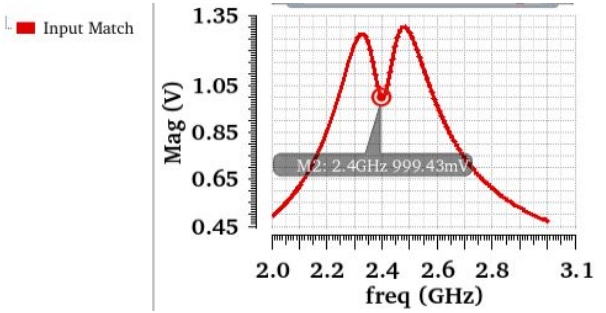
Mixer

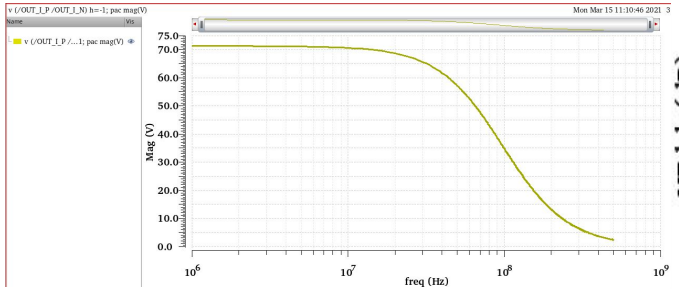
- Passive mixer for low power
- Differential Voltage gain: 37dB
- Maximum input voltage: ~6mV
- NFdsb: 3.56 dB integrated
- Power consumption: ~1mW
  - ~500uW for the LO buffers
  - ~450uW for the TIA

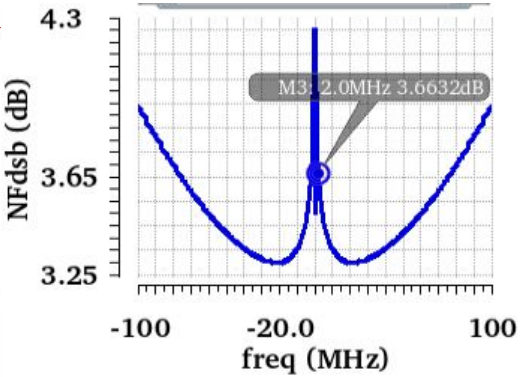




TIA circuit diagram

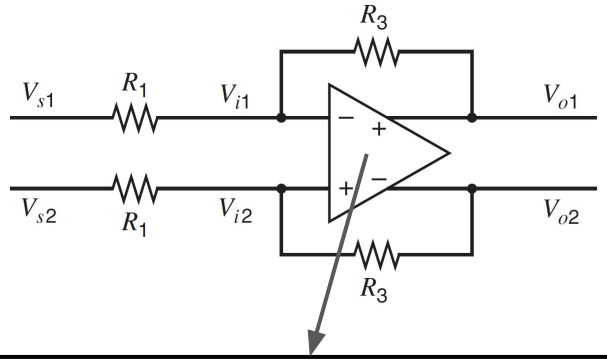


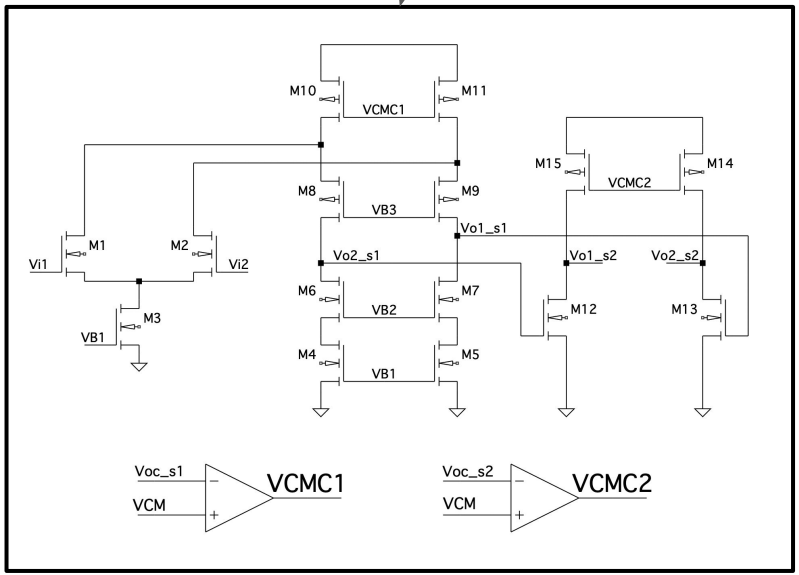




VGA

- VGA design
  - 0-30 dB gain range for each VGA
  - R_1 steps in 6 dB, R_3 steps in 2 dB
  - BW(30 dB) = 6.85 MHz with 300pF load
  - $R_{1,min}$ = 5 Kohm, $R_{3,min}$ = 80 Kohm
- Fully differential amplifier design
  - $V_{i,CM} = V_{o,CM} = 0.6V$
  - A = 98 dB, BW = 3.15 kHz
  - Current consumption = 270uA
  - Miller compensation, PM = 57
  - Input swing 0.3-0.9 V; output swing 0.15-0.8 V
- Noise @ 30dB gain
  - Input spot(2MHz) = 15 nV/√Hz
  - Integrated output noise (1-3 MHz) = 678 uV





VGA

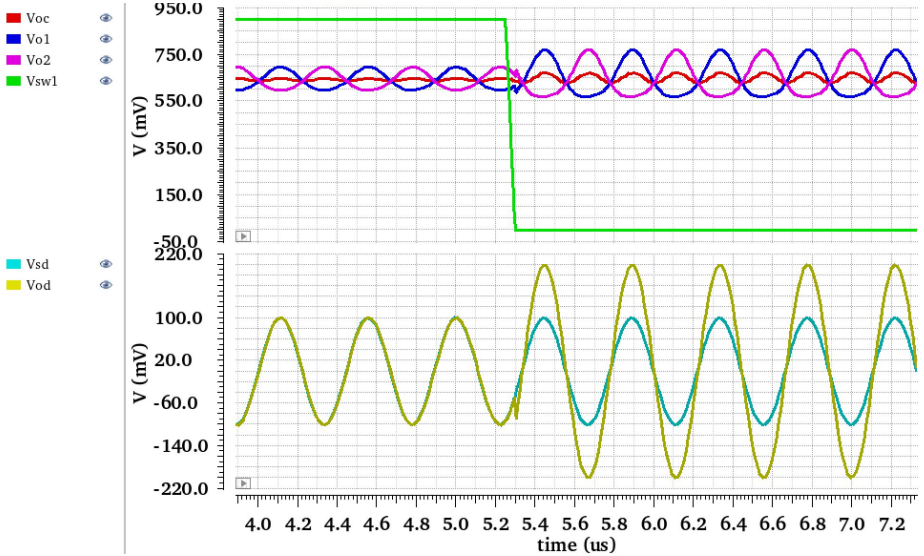
Monte Carlo @ 30dB gain:

	mean	std dev	max	min
Equivalent input offset (mV)	0.006	**1.25**	3.58	-3.47
PSRR (dB)	60.6	9.69	110	**46.2**
CMRR (dB)	82.0	11.1	130	**56.6**

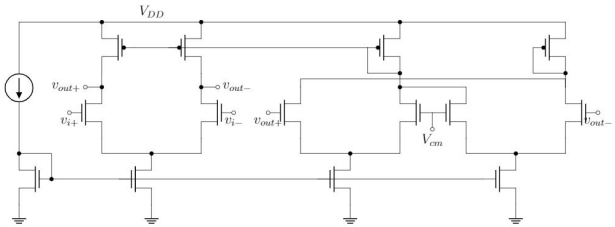
Corner simulation @ 30dB gain:

	TT(27°)	Worst case
DC gain (dB)	30	29.9
Gain @ 2MHz (dB)	29.5	28.6
3dB BW (MHz)	5.44	3.39
output noise (uV)	684	813

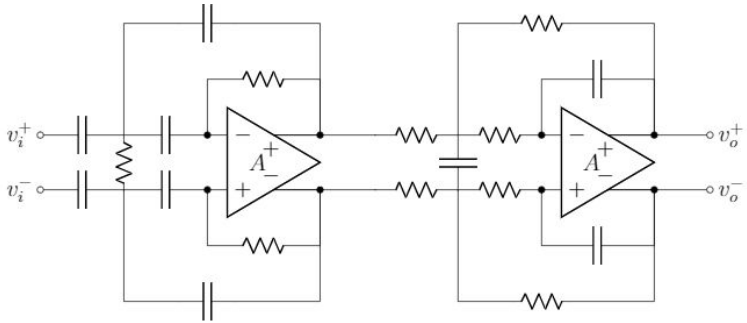
Transient analysis:
Switching from unity to 6dB gain



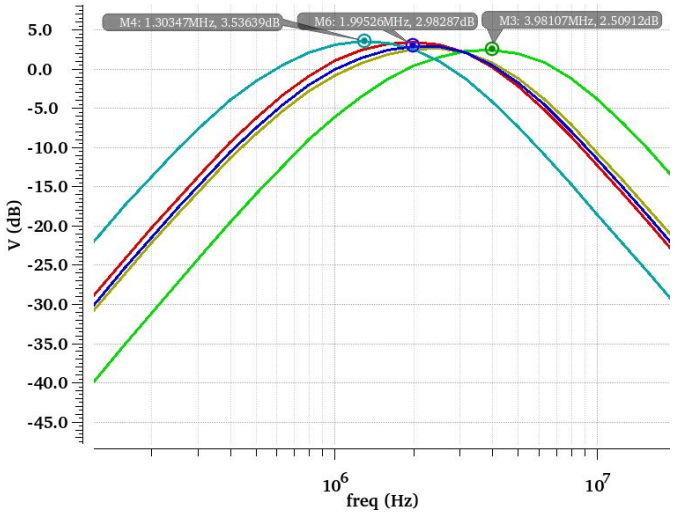
Bandpass Filter



- Multiple feedback high pass followed by low pass
  - Opamp: single stage with CMFB
- Gain of 3dB unloaded, and about unity when loaded by next stage
- Resistors adjustable +-50% to account for variation and maintain passband across temperature



- Noise:
  - Input spot @2MHz: 80nV/√Hz
  - Output integrated (1-3MHz): 170uVrms
- Group delay: 168ns (0) to 139ns (1)



No tuning.

LO:

- F = 4.8 GHz
- P = 360 uW
- Vpp = 0.814 V
- Q = 15
- L = 3.472 nH

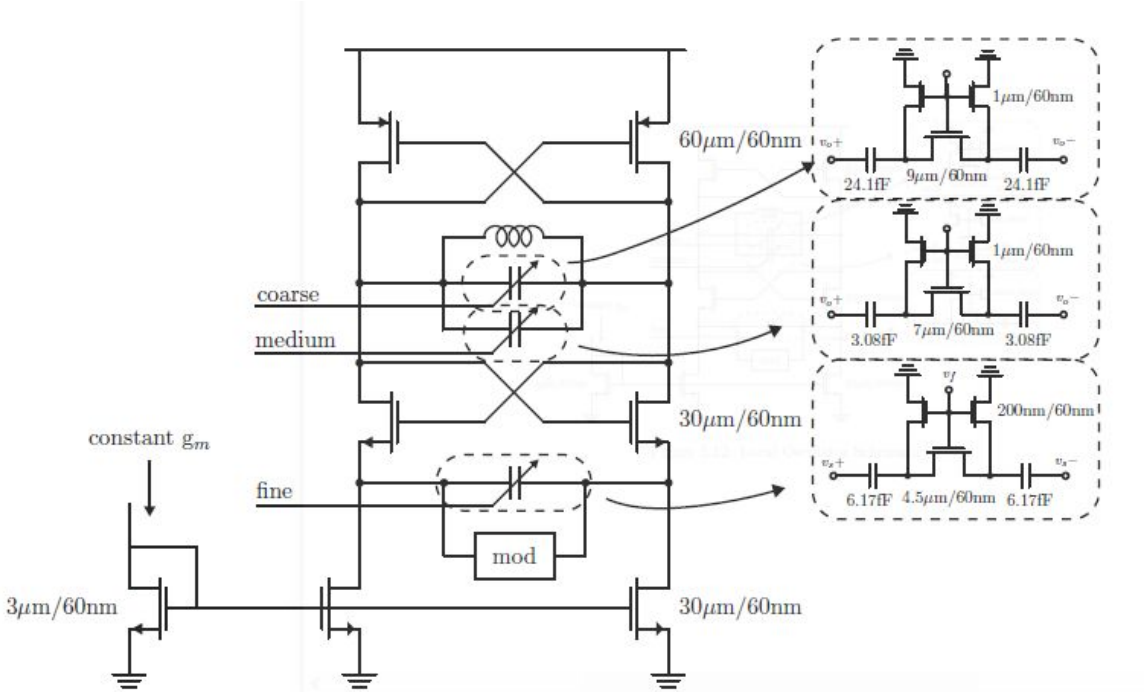
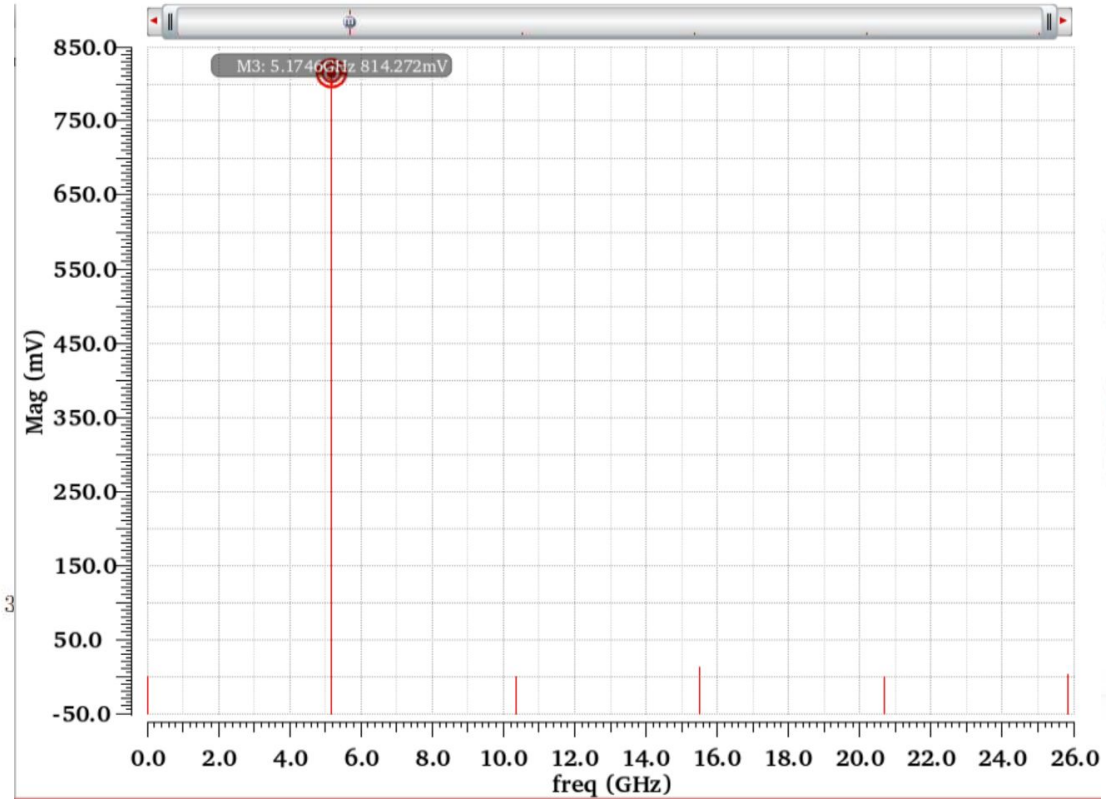


Figure 5.12: Local Oscillator Schematic

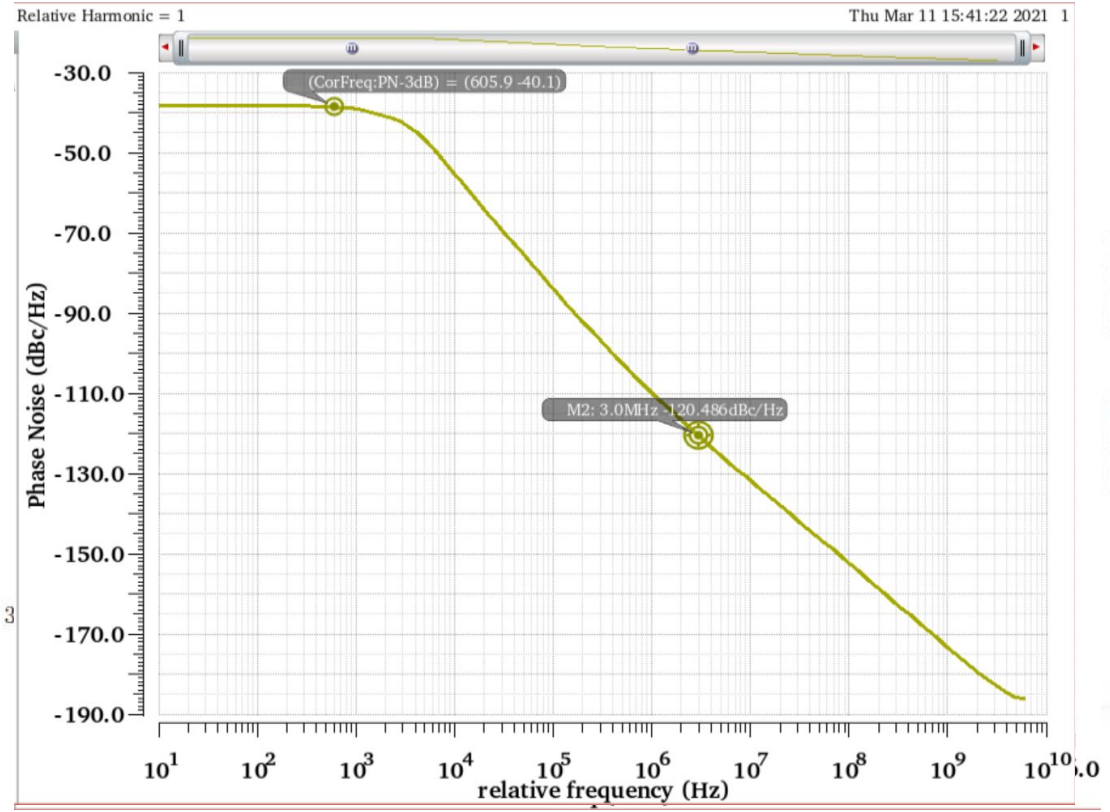
LO:

- F = 4.8 GHz
- P = 360 uW
- Vpp = 0.814 V
- Q = 15
- L = 3.472 nH



LO:

- F = 4.8 GHz
- P = 360 uW
- Vpp = 0.814 V
- Q = 15
- L = 3.472 nH



LO:

- F = 4.8 GHz
- P = 360 uW
- Vpp = 0.814 V
- Q = 15
- L = 3.472 nH

Code		-27	27	70
0000	Freq (GHz)	5.134	5.126	5.120
1111		error	4.575	4.572
0000	PN (dBc/Hz)	-121.7	-120.8	-119.8
1111		error	-119.8	-118.9
0000	Amp	815.417	801.446	777.969
1111		error	585.71	589.93

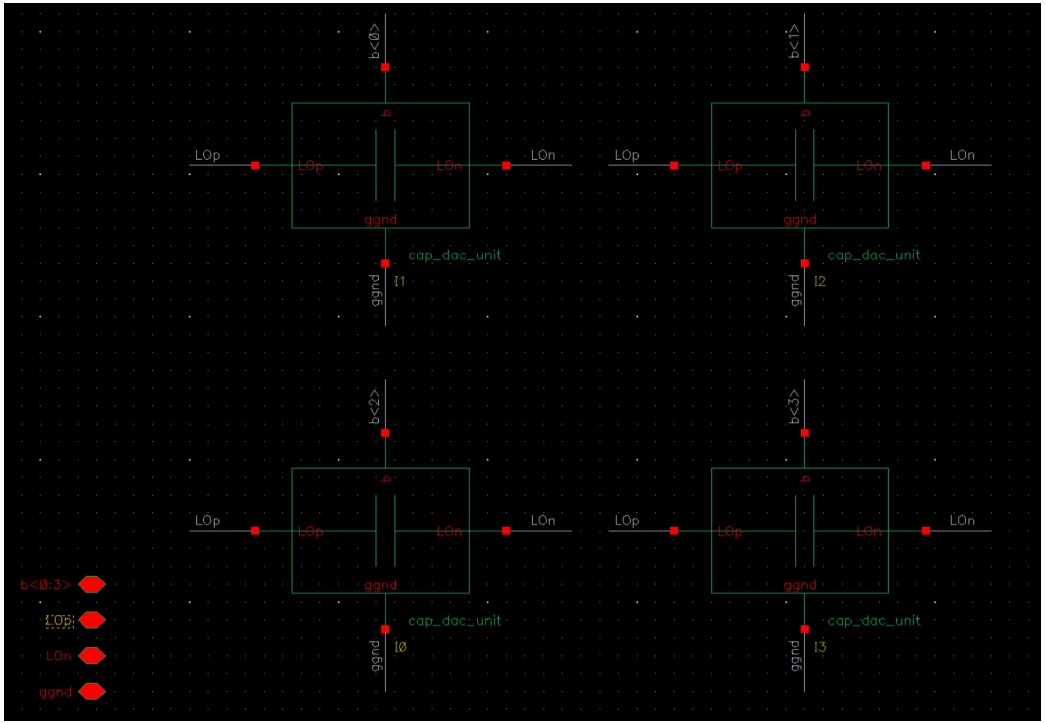
LO:

- F = 4.8 GHz
- P = 360 uW
- Vpp = 0.814 V
- Q = 15
- L = 3.472 nH

Code		-27	27	70	FF+-27	SS+70
0000	Freq (GHz)	5.134	5.126	5.120	5.926	error
1111		error	4.575	4.572	5.307	error
0000	PN (dBc/Hz)	-121.7	-120.8	-119.8	117.2	error
1111		error	-119.8	-118.9	-117.4	error
0000	Amp (mV)	815.417	801.446	777.969	856.788	error
1111		error	585.71	589.93	779.946	error

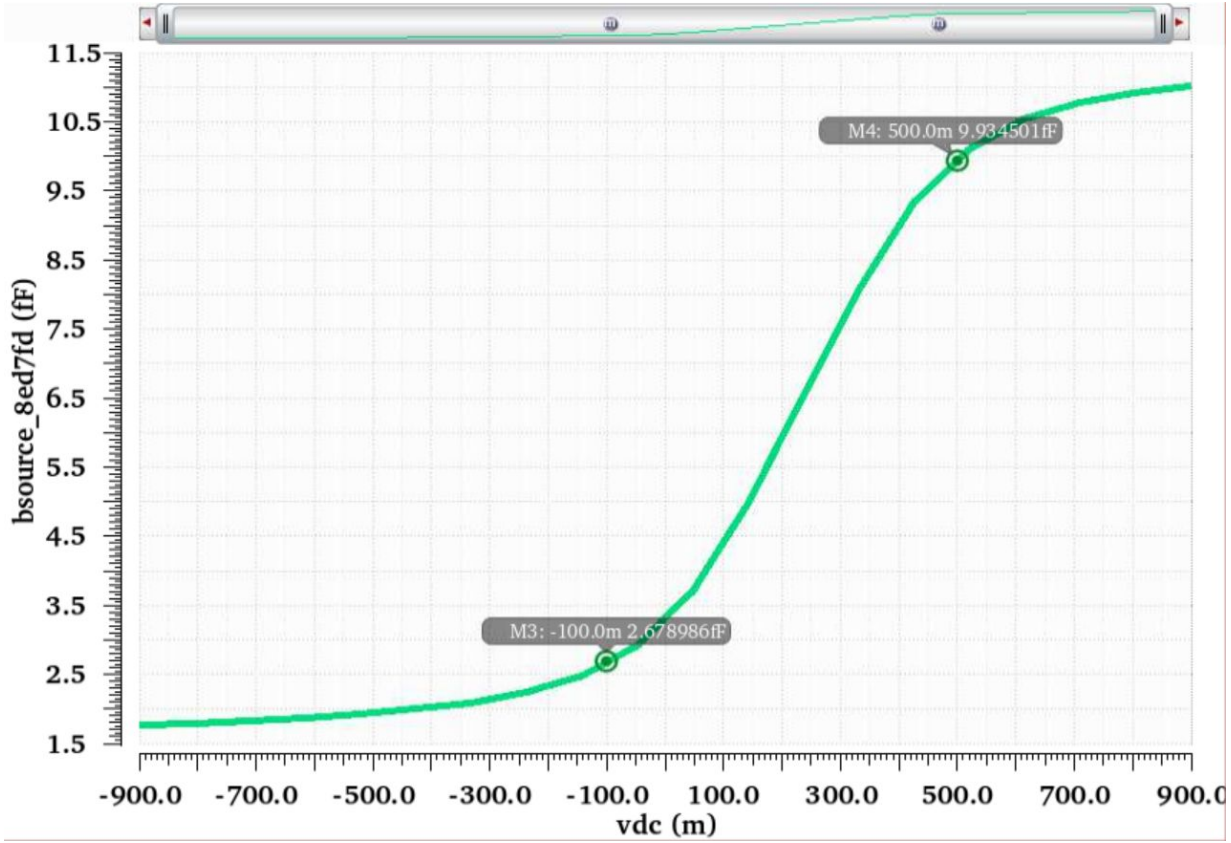
LO: Cap DAC

- 4 bit Cap Bank
- Fmin target = 4.56 GHz
- Fmax target = 5.21 GHz
- ~40 MHz per unit cap

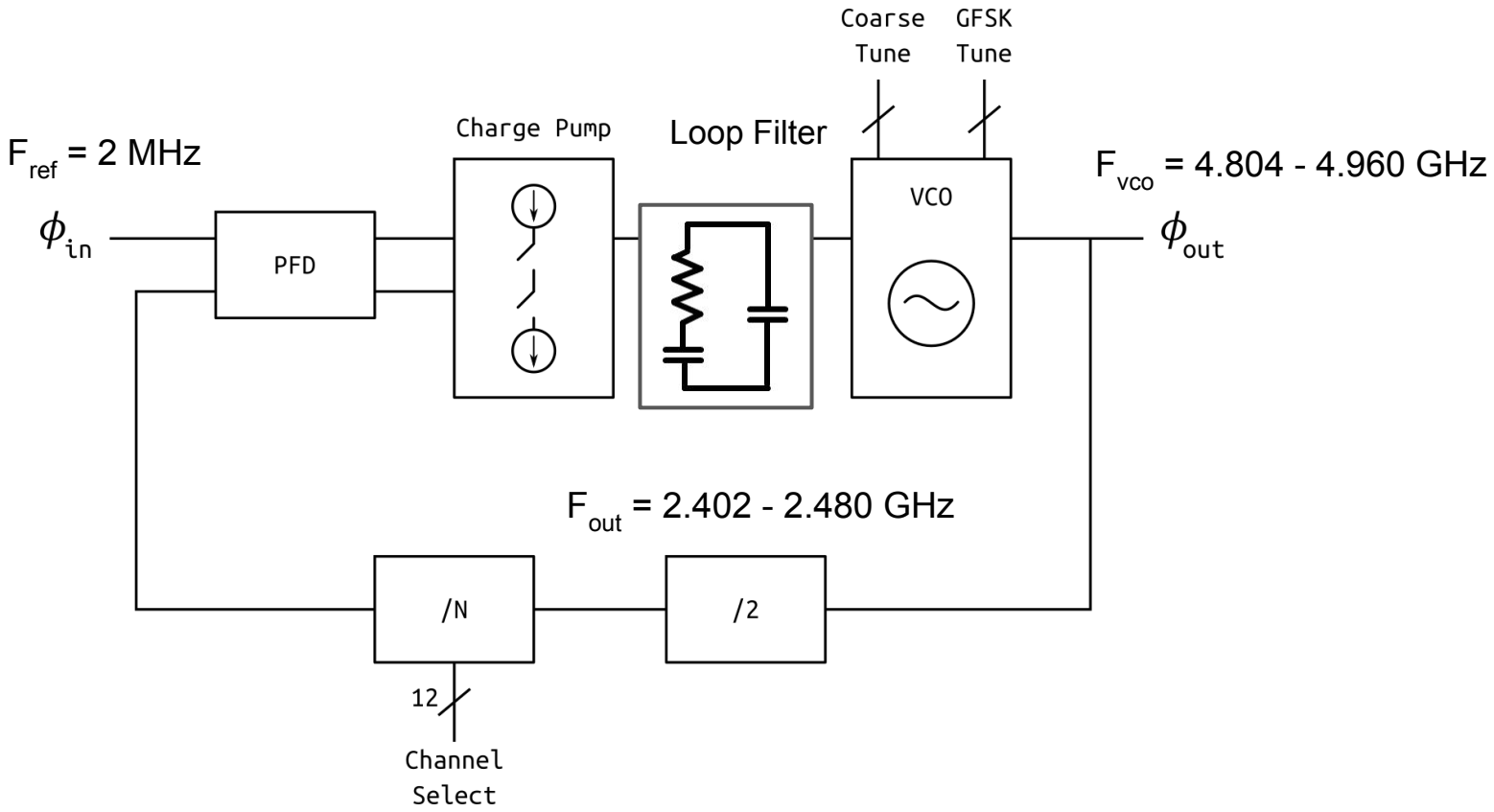


LO: Varactor

- Range 200mV-800mV
- Delta C= 7.2556 fF
- ~70MHz
- KVCO=115MHz/V

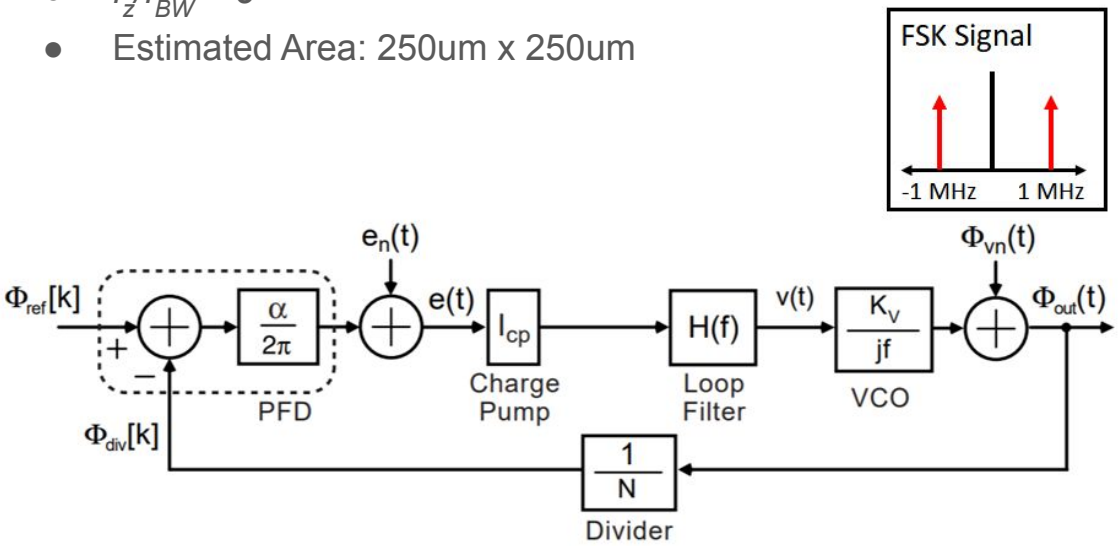


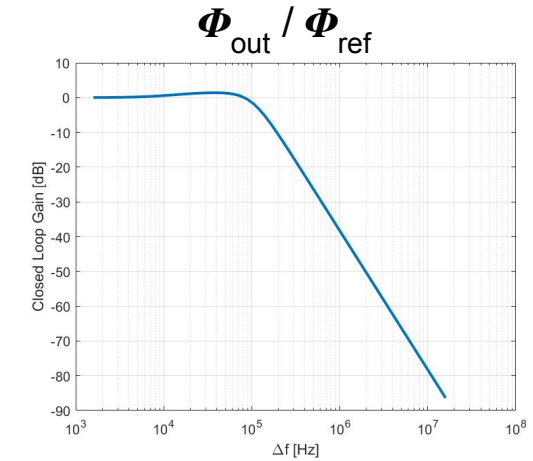
PLL

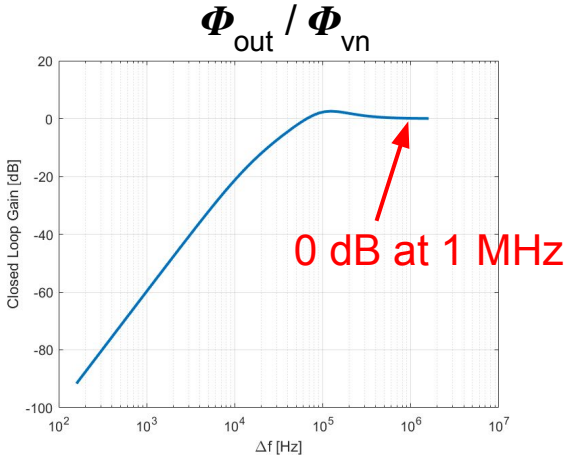

Coarse Tune
GFSK Tune
Charge Pump
Loop Filter
F ref = 2 MHz
VCO
F vco = 4.804 - 4.960 GHz
ϕ in
PFD
ϕ out
F out = 2.402 - 2.480 GHz
/N
/2
12
Channel Select

PLL Loop Filter

- Closed loop bandwidth placed low enough to stay locked while not rejecting 1 MHz GFSK during TX
- f_{BW} = *100 kHz*
- f_z/f_{BW} = *8*
- Estimated Area: 250um x 250um

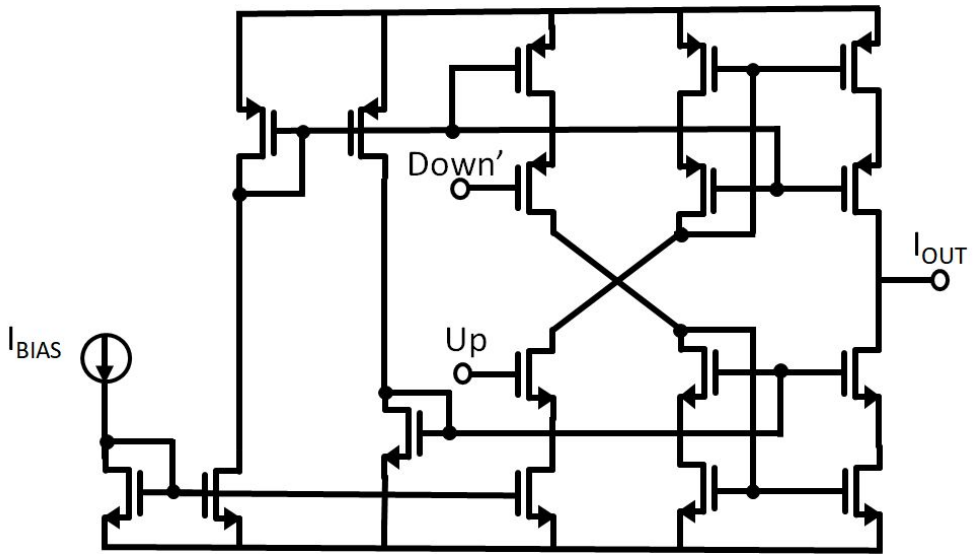




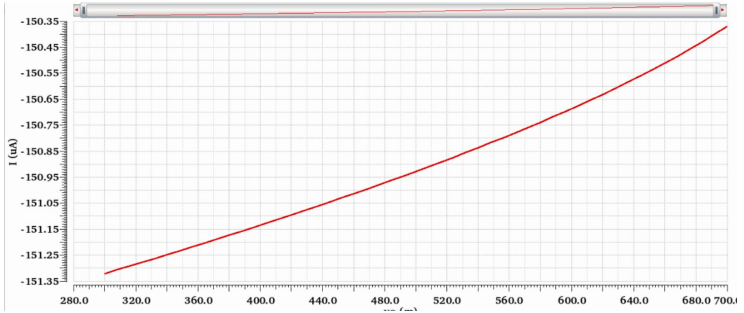


Charge Pump

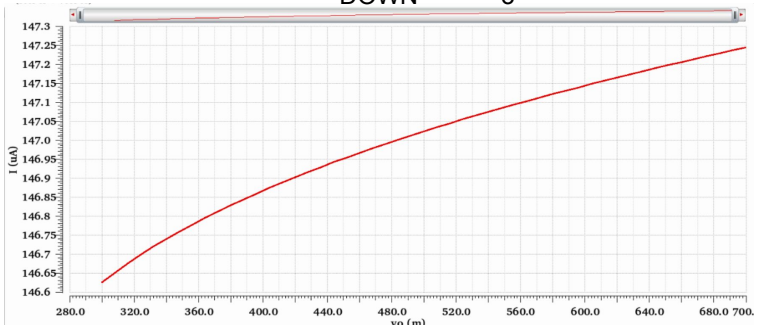
- Cascode current source with wide swing current mirrors
  - Designed for I_{OUT} = 150 uA
  - Estimated Power Consumption: 300 uW



I_{UP} vs V_o



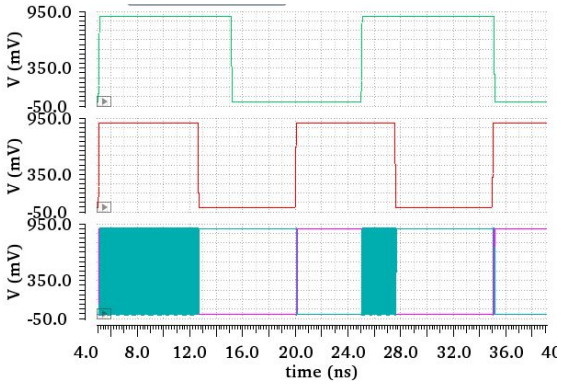
I_{DOWN} vs V_o

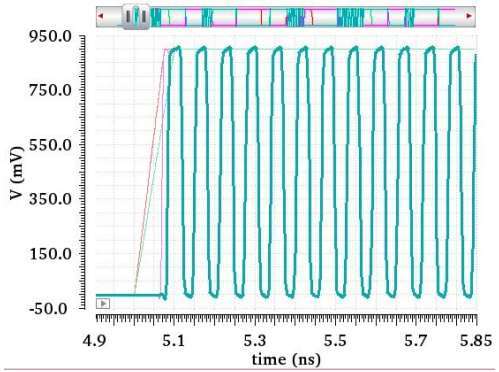


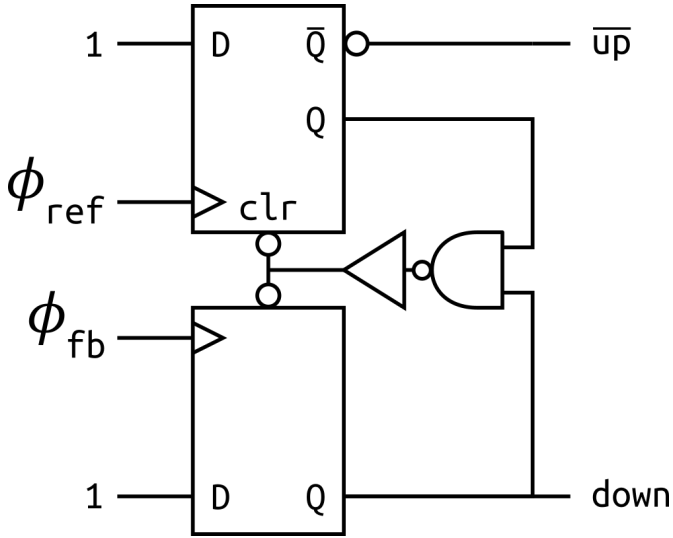
Phase/Frequency Detector

Stability issues when ϕ_{ref} and ϕ_{fb} both high
17GHz oscillations on up and down

FB / REF / UP / DOWN



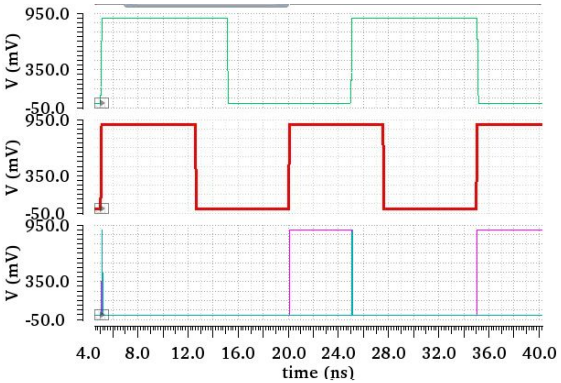


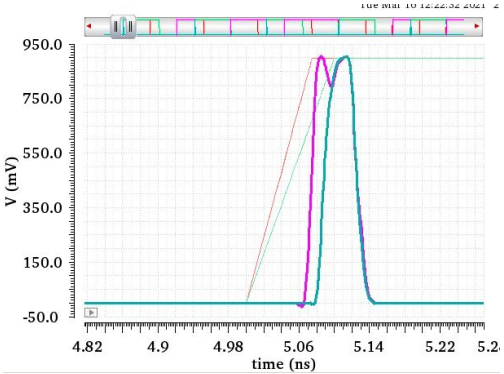


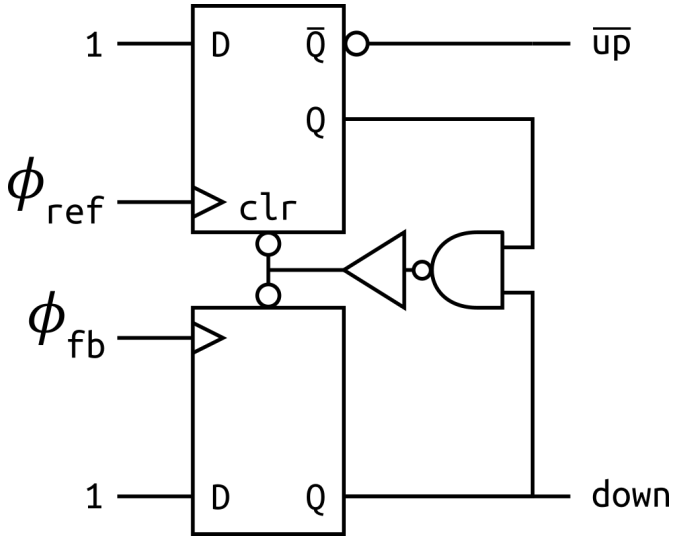
Phase/Frequency Detector

Fixed by adding buffers and using slower NAND

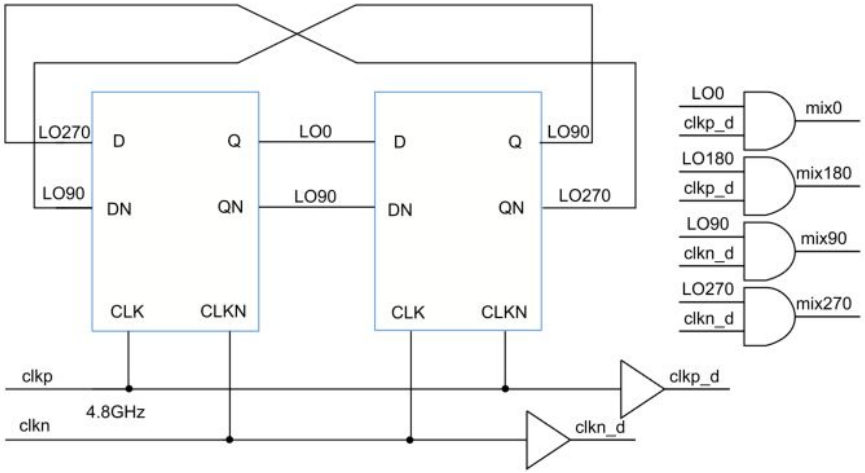
FB / REF / UP / DOWN

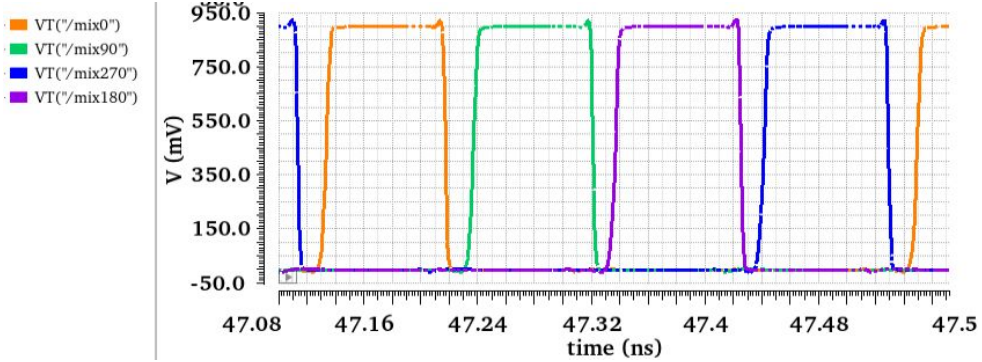


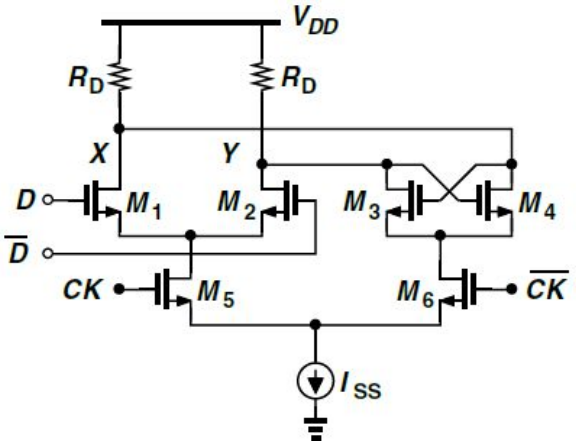




Pre-scalar/Mixer LO Distribution



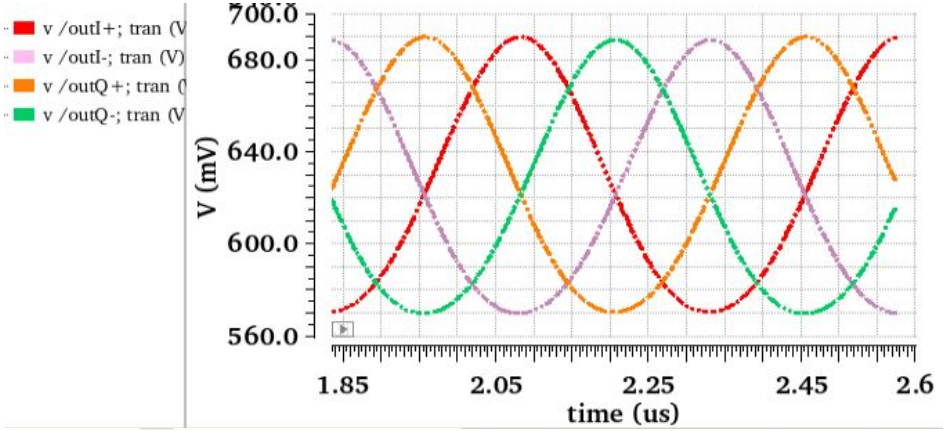


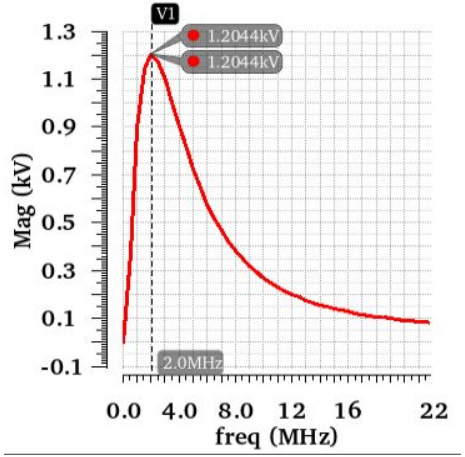


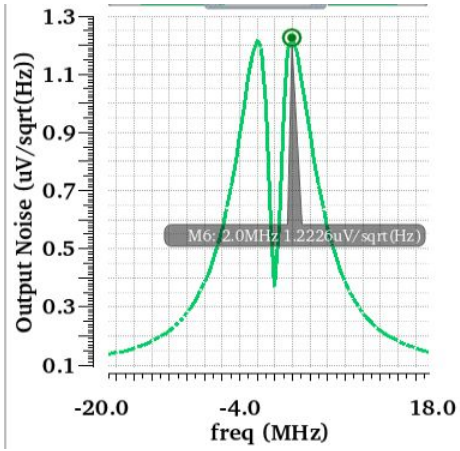
- CML Logic for better differential output balance
  - <4ps output pulse width variation
- Takes 1 clock cycle to settle
- 130uW power consumption

Current Receiver Results

- Functional first pass integration measured at -70dBm input
- Sufficient Gain (35dB voltage gain measured)
- NFdsb of 4.2dB
- Power Consumption: 2.817mW
- Next Steps
  - Operating range expansion
  - Use real components for passives







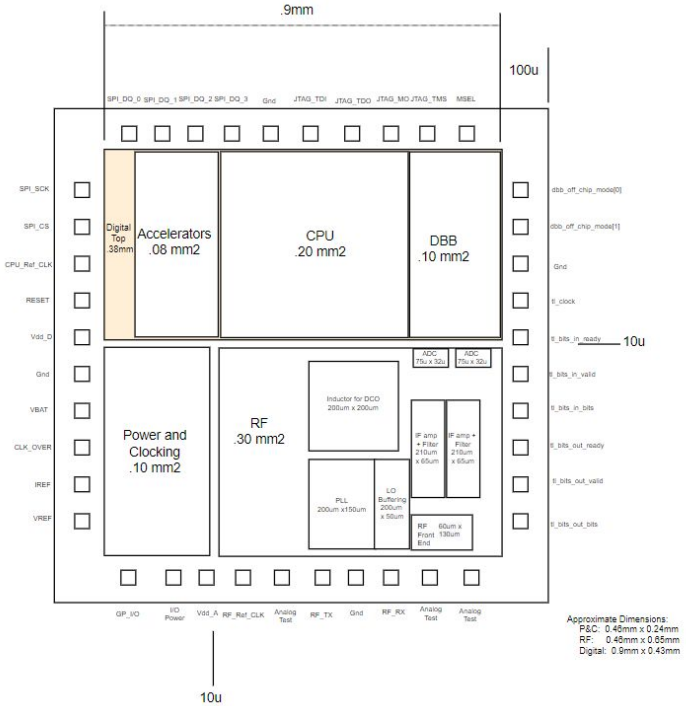
Analog & Power

Introduction

- Jackson Paddock
  - 5th year MS student in Prof. Kristofer Pister's lab
  - Focus on analog circuit design (specifically LDOs)

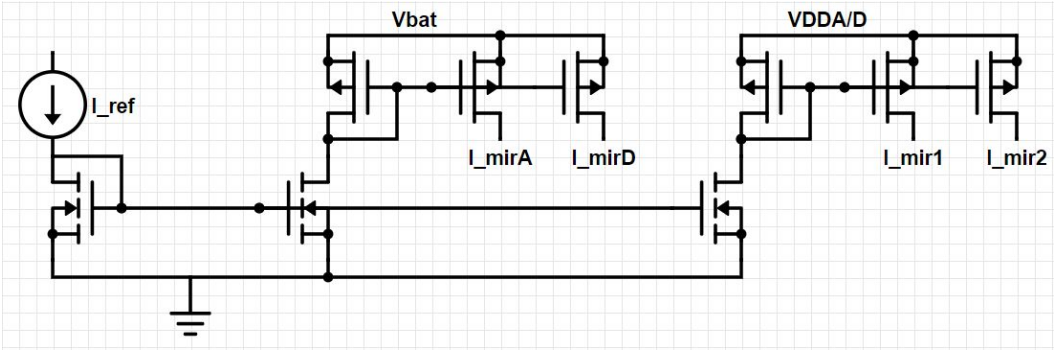
Power Domains

- Three power domains on the chip: analog, digital, and I/O
- Analog:
  - Includes: RF (PLL, LO, mixer, amplifiers, ADCs)
  - Analog LDO (VDDA) supplies 5.75mA at 0.9V
- Digital:
  - Includes: CPU, accelerators, digital baseband
  - Digital LDO (VDDD) supplies 47.2mA At 0.9V
- I/O:
  - Includes: pad ring
  - VDDIO comes from an off-chip supply



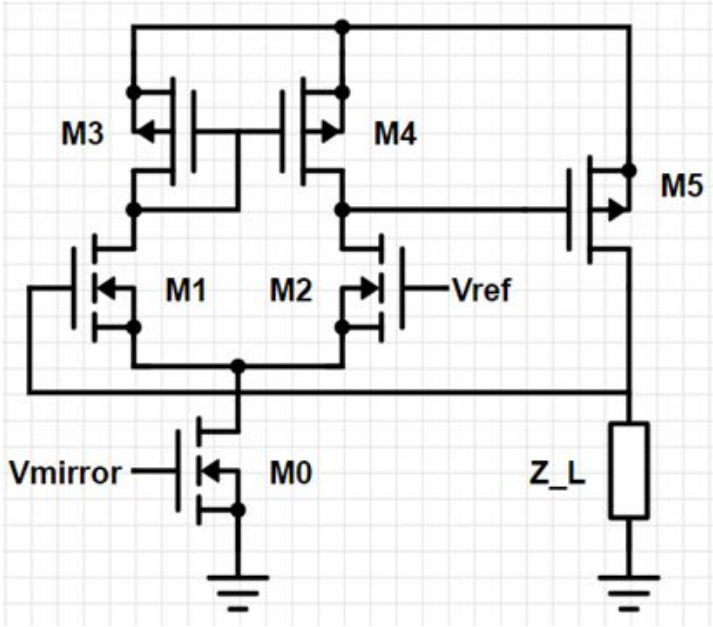
Current Reference Topology

- Currently, there is no bandgap reference on chip so the voltage and current references are from off chip (0.9V and 10uA respectively)
- With an input NMOS device, the 10uA reference is mirrored up to PMOS devices on:
  - Vbat, and then brought down for the tail current mirrors of the analog and digital LDOs
  - VDDA and VDDD for current references in the analog and digital circuits on chip
- All devices shown have max channel length to reduce variation from VDS



LDO Topology

- Both the digital and analog LDOs use the same topology: a simple diff pair amplifier with a PMOS pass device
- Both LDOs use the same amplifier and have outputs at 0.9V
- The dominant pole is at the output with an off chip capacitor (due to area) for stability

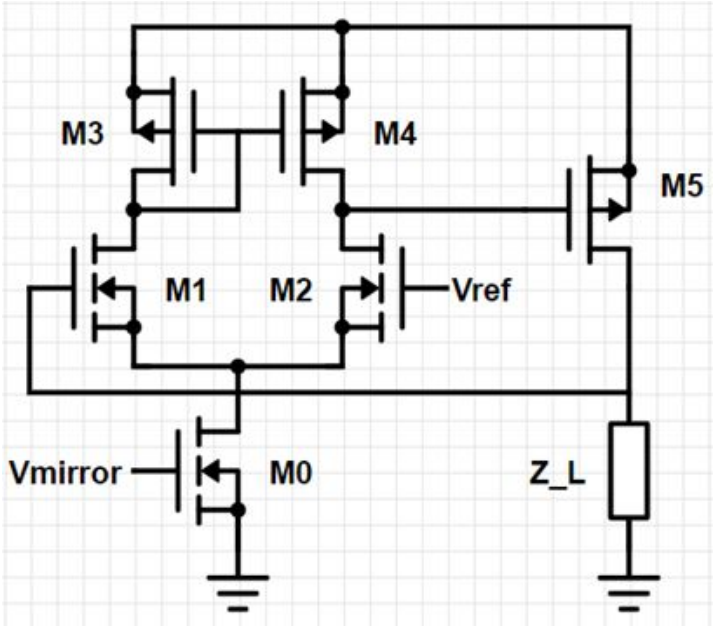


LDO Design Tradeoffs

- Pole location:
  - Not enough information is known about the load capacitance from other blocks at the moment
  - A dominant pole at the gate of M5 reduces PSRR bandwidth significantly and may cause stability issues if the chip’s built-in capacitance is too large
- Power vs Stability:
  - As the relative width of all devices in the amplifier increases, the output resistance decreases and pushes the secondary pole higher (increasing the phase margin)
  - Wider devices mean higher current consumption in the amplifier

LDO Specs (TT, 27°C)*

Parameter	Value
Static Error	35.4uV
Amplifier Current	211.1uA
Phase Margin	75.35°
PSRR	51.88dB
PSRR bandwidth	1.992MHz
Line Regulation*	99.95m
Load Regulation*	412.0u
Load Pole (C_L=100nF)	4.498kHz
Second Pole (Gate of M5)	1.434MHz



*Simulated with output device set for 5.75mA
**Simulated with 20% variance peak to peak of reference voltage and load current

Corner and Temperature Simulations*

	Temperature (TT)			Corners (TT, FF, SS, FS, SF)		(TT, 27°)
Parameter	**0°**	**27°**	**84°**	**Minimum**	**Maximum**	**R_S=10Ω**
Static Error	-69.07uV	35.4uV	275.0uV	-864.0uV	995.0uV	127.7uV
Amp Current	207.4uA	211.1uA	218.0uA	192.0uA	231.1uA	208.4uA
PM	74.66°	75.35°	76.78°	73.05°	78.54°	75.46°
PSRR	51.81dB	51.88dB	51.99dB	49.78dB	54.11dB	50.68dB
PSRR BW	2.112MHz	1.992MHz	1.803MHz	1.621MHz	2.560MHz	2.184MHz
Line Reg	99.96m	99.95m	99.93m	99.89m	100.0m	99.94m
Load Reg	384.7u	412.0u	471.9u	379.1u	474.6u	455.2u

*Simulated with output device set for 5.75mA

Overview

- RV32IMAFC Core
- 32KB I$
- 32KB Data Memory
- BootROM
- TSI
- JTAG
- SPI flash
- Interrupts
- GPIO