OsciBear
28nm IoT SoC
Final Design Review

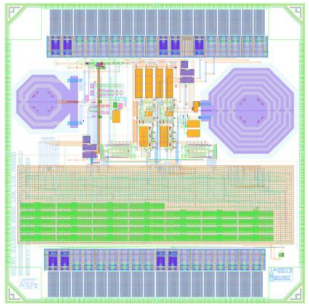
May 17, 2021
UC Berkeley EE290C

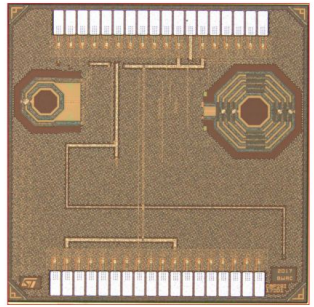
Course History

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

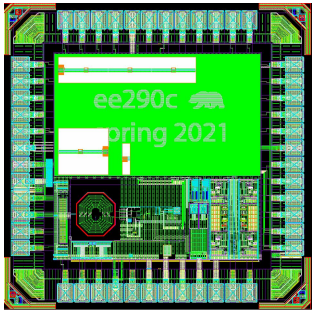
David C. Burnett, Brian Kilberg, Rachel Zoll, Osama Khan, Kristofer S.J. Pister
Department of Electrical Engineering and Computer Sciences, University of California Berkeley, Berkeley, CA 94720
Email: {db, bkilberg, rachelzoll, oukhan, pister}@eecs.berkeley.edu

- 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



Spring 2021 Staff

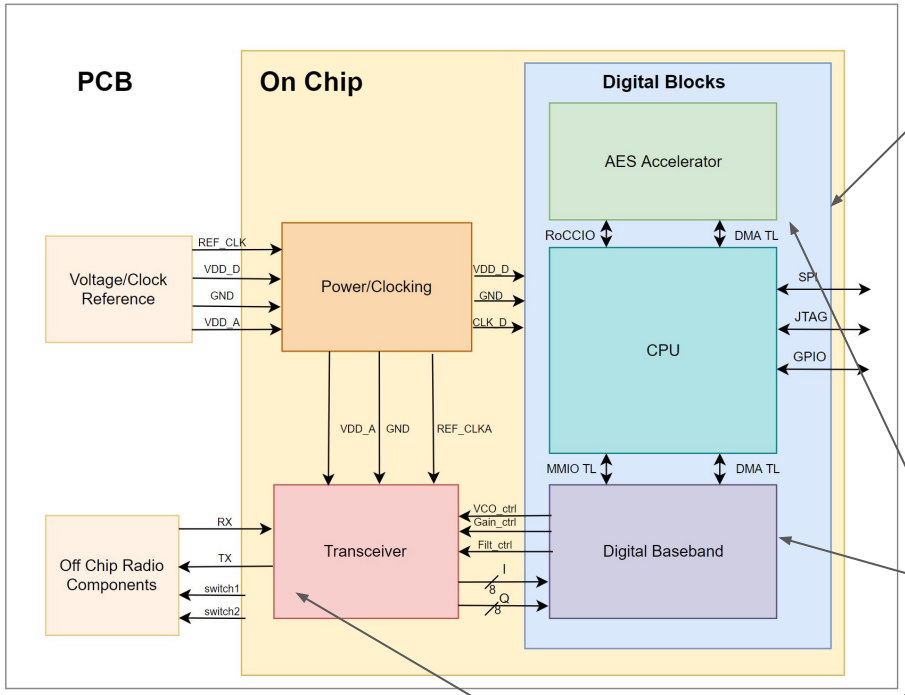
- Kris Pister
- Borivoje Nikolic
- Ali Niknejad

- Avi Pandey
- Dan Fritchman

Research Infrastructure

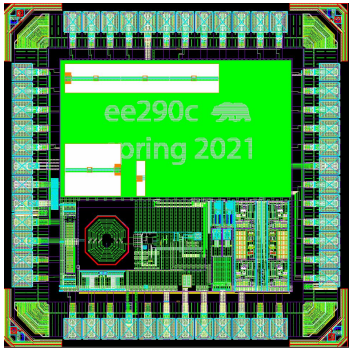
Bora Nikolić
28th-year grad student

Leveraging Research Infrastructure



CHIPYARD

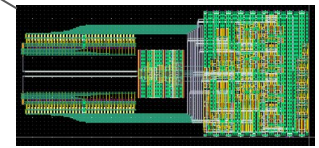
https://github.com/ucb-bar/chipyard
Processor core, interfaces
Software tools



SoC taped out on May 6 in TSMC 28nm HPC (14 weeks)

CHISEL

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



Berkeley Analog Generator (BAG) - generated SAR ADC

Chip Intro & Overview

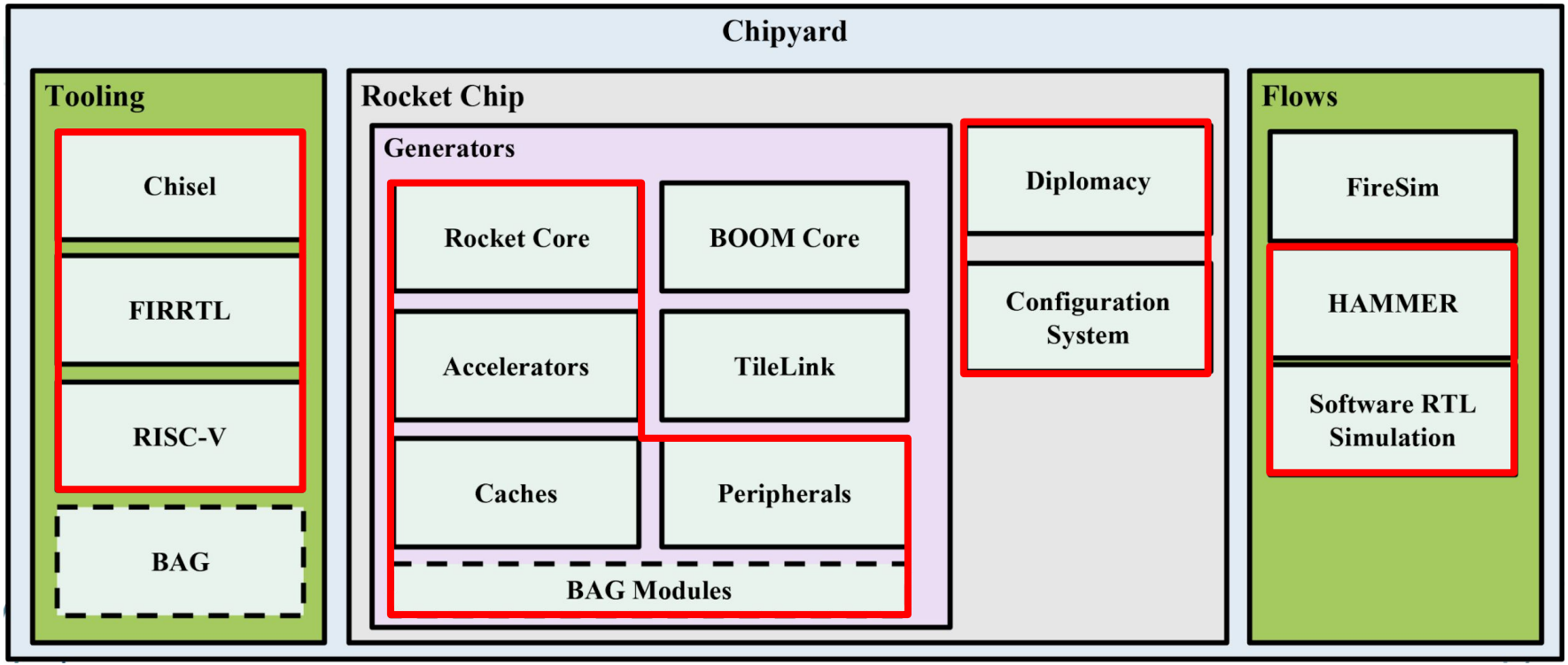
OSCI BEAR

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

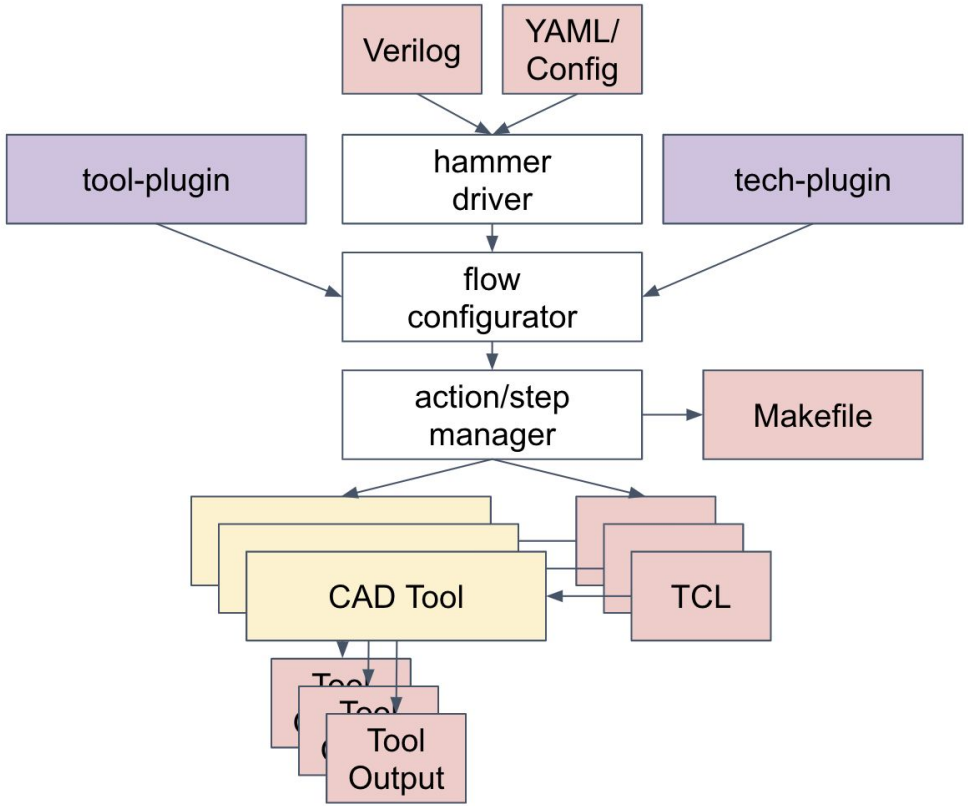
Project Goals

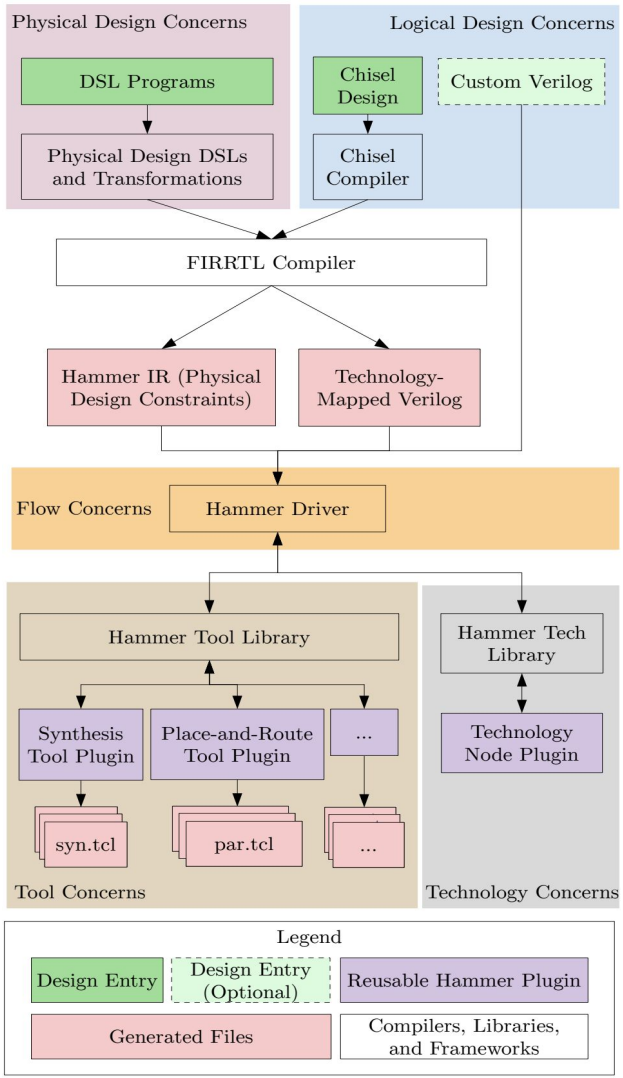
- ✓ Lightweight BLE compatible SoC for IoT applications
  - Wearables, localized sensor networks, smart keys, etc
- ✓ Demonstrate agile hardware development flow
  - Using Berkeley designed tools
- ✓ Tape-out in TSMC 28nm technology
  - 1.00 mm^2 total die area
  - Off chip radio and clock components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

Infrastructure: Chipyard

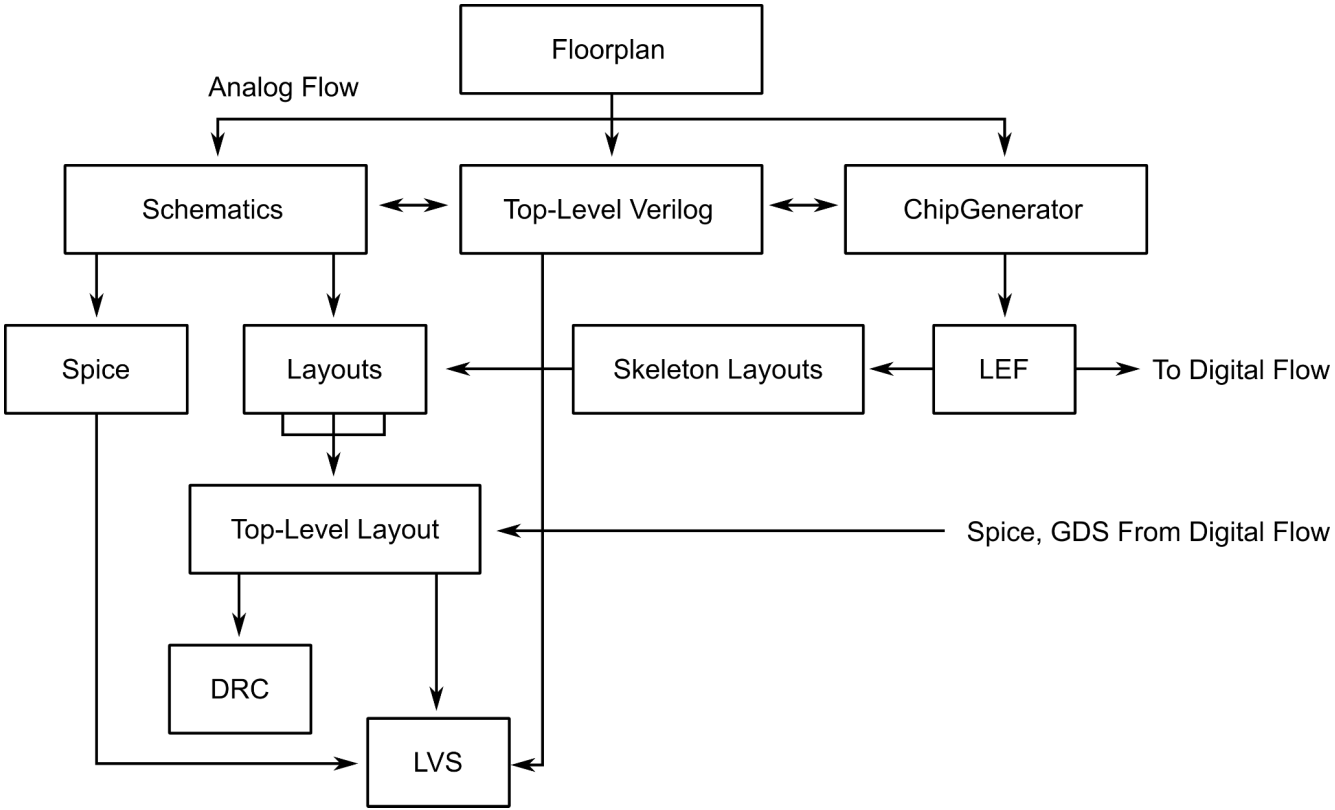


Infrastructure: Hammer VLSI Flow

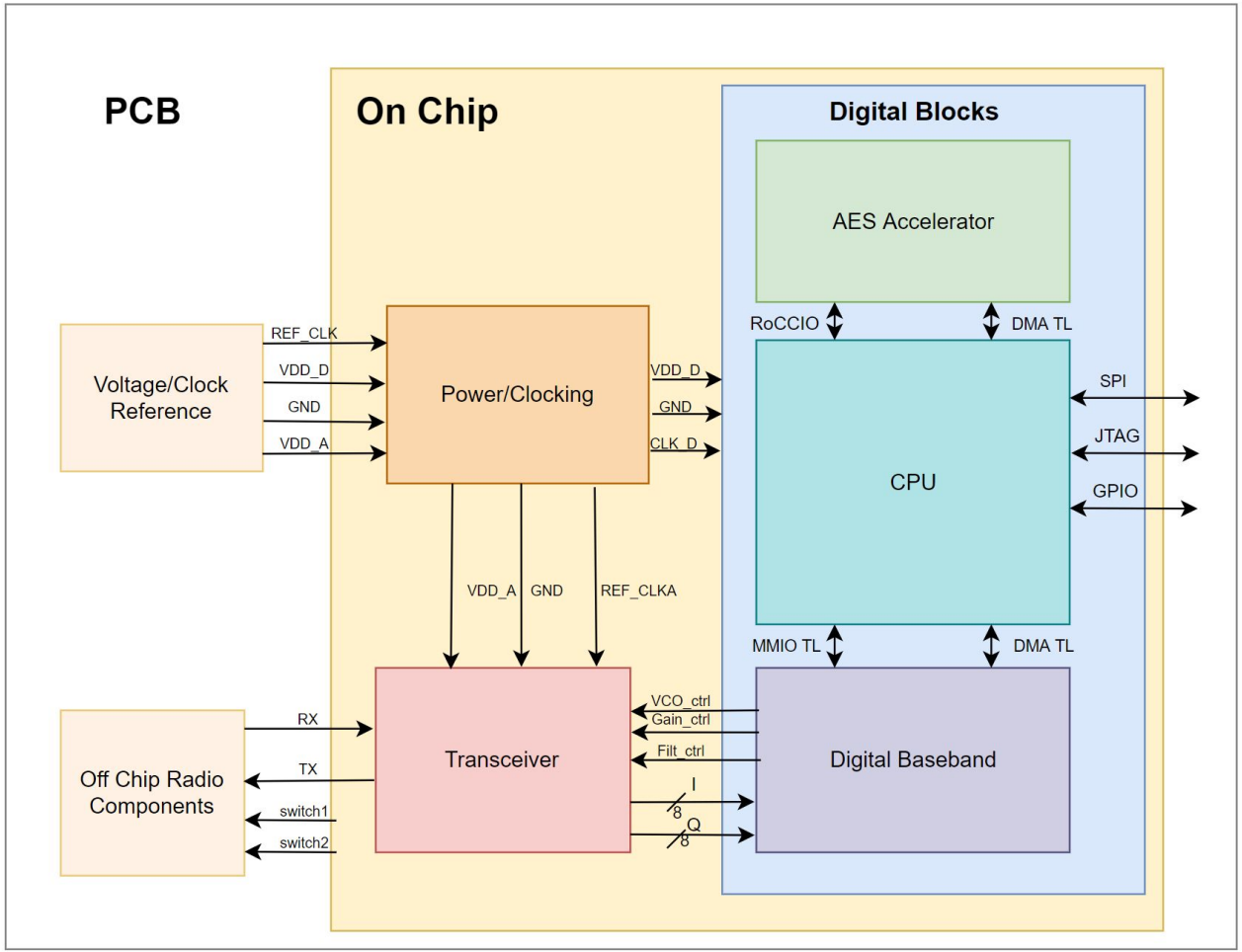




Infrastructure: Integration

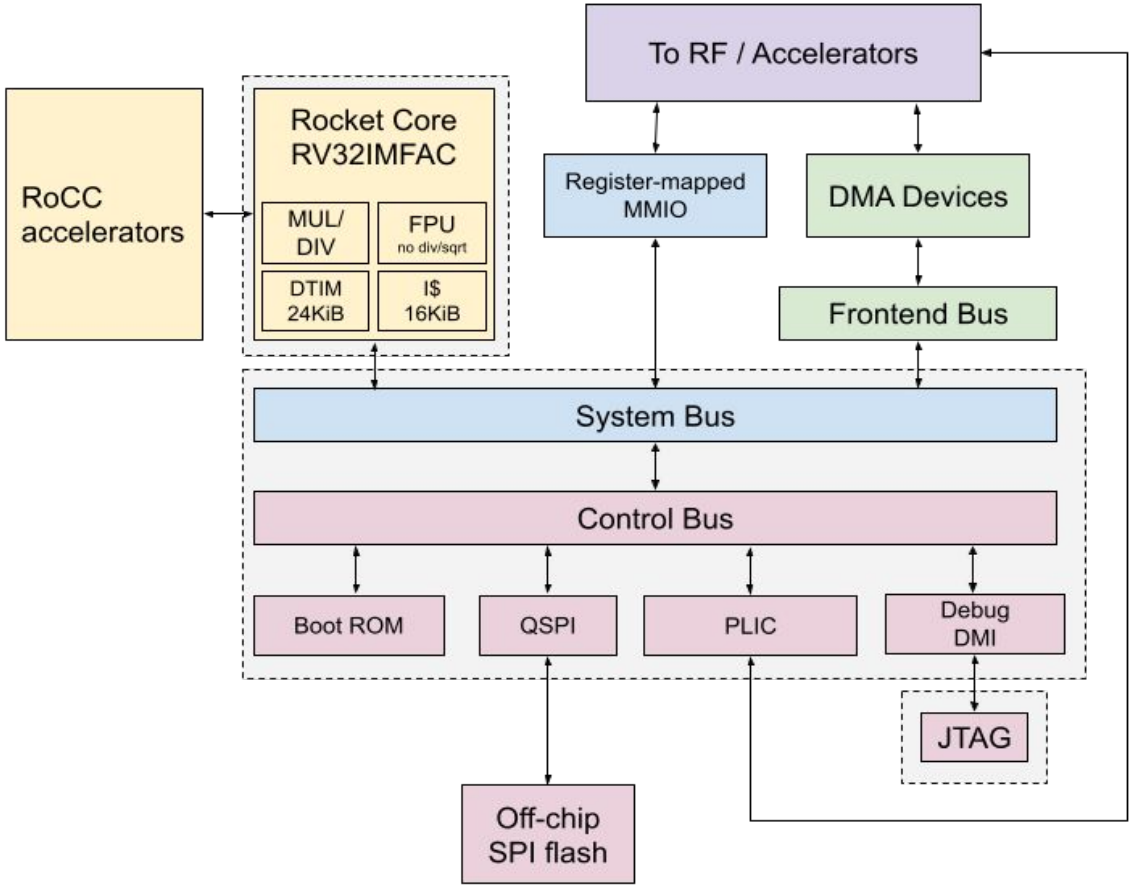


Chip Overview



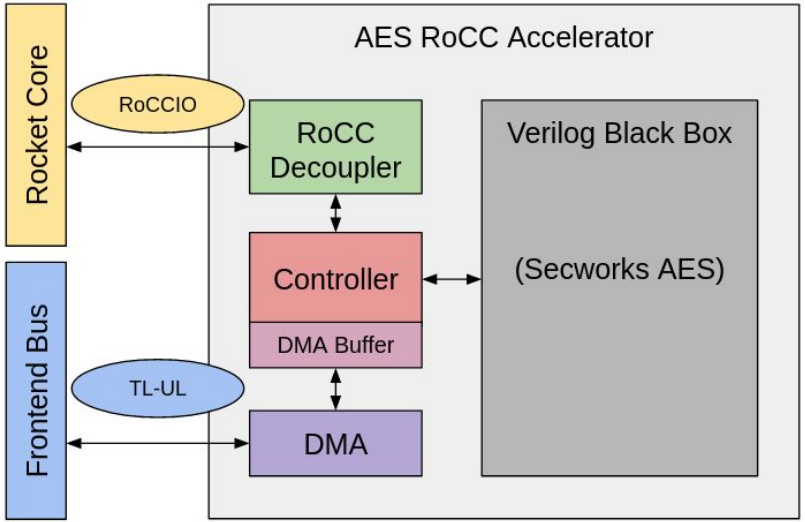
Compute Complex

- RV32IMAFC Core
- 16KB I$
- 24KB Data Memory (DTIM)
- TSI, JTAG
- UART
- GPIO
- QSPI flash
- Interrupt Controller
  - PLIC (Platform level)
  - CLINT (Core level)



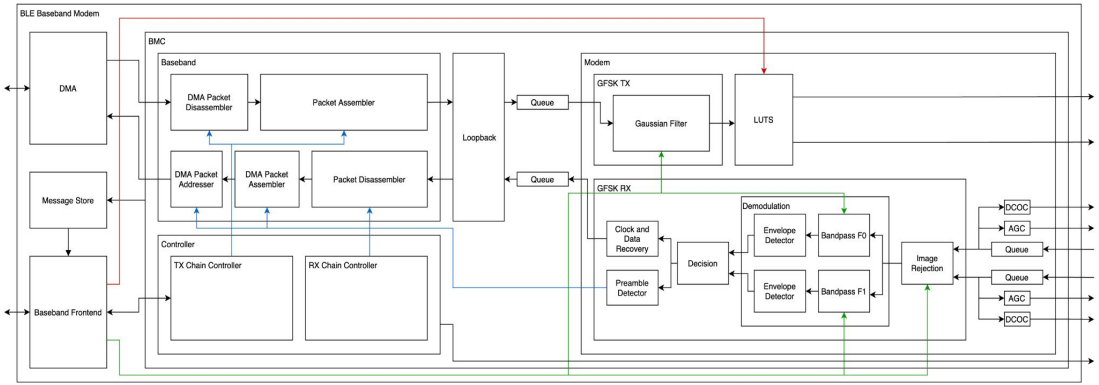
AES Accelerator/Co-processor

- 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

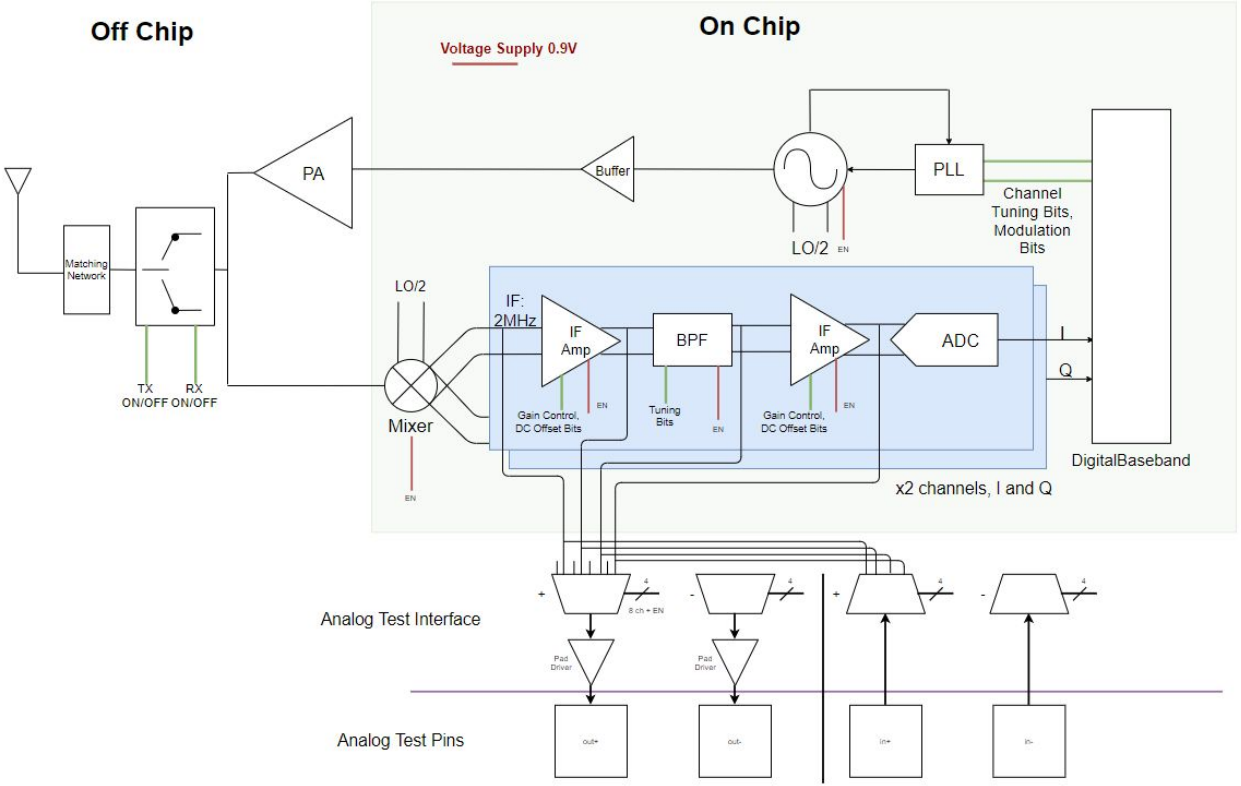


BLE Baseband and Modem

- 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
- Designed to be expandible and highly configurable



BLE Transceiver



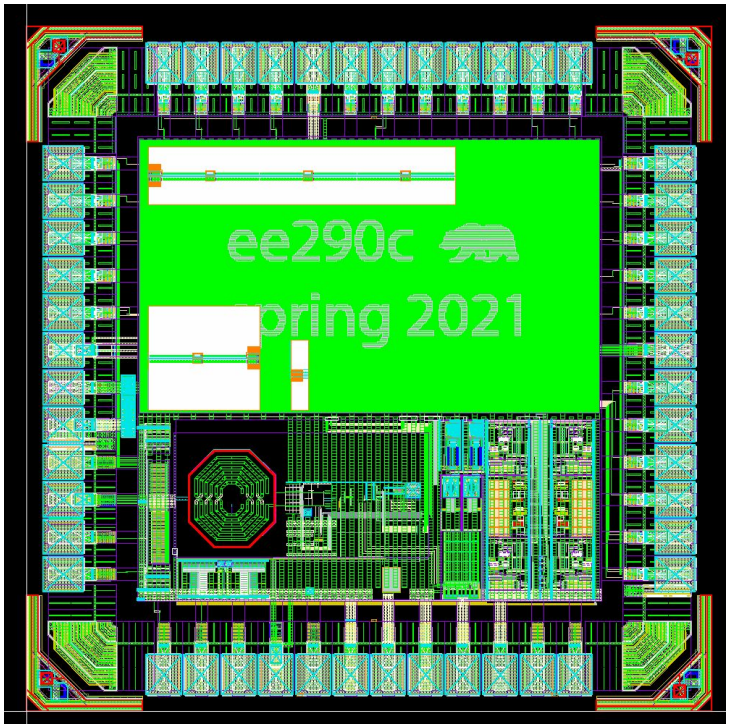
- Direct modulation receiver architecture with tunable LO
- Analog PLL
- Digital control for GFSK modulation

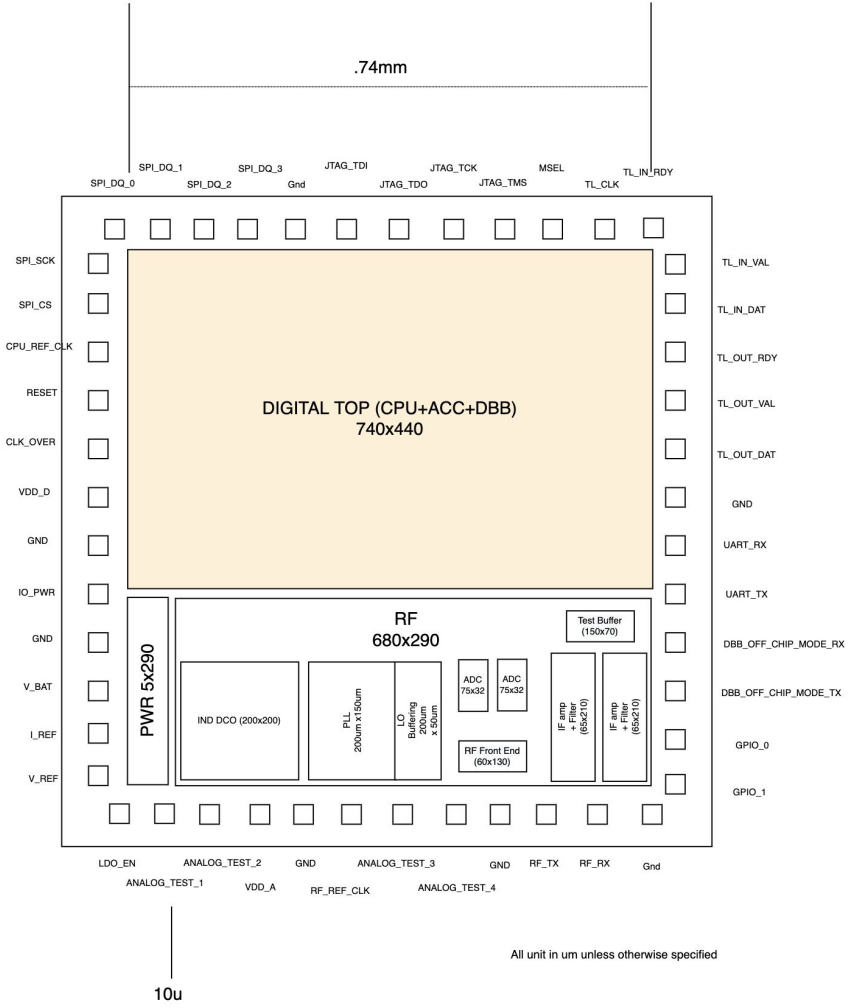
Specs

Digital	
Core Clock	20 MHz
Core Power	16 mW
AES Speedup	~35x speedup from non-accelerated
BLE Transmit Latency	20 μS

Analog	
Power	5.4 mW
BER	0.1%

Final Layout and Floorplan



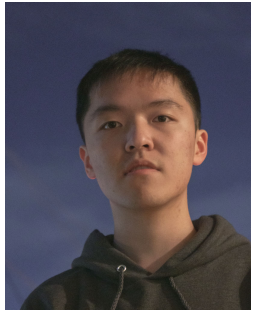


CPU

Team Introduction

Nayiri Krzysztofowicz
1st Yr PhD Student, advised by Bora

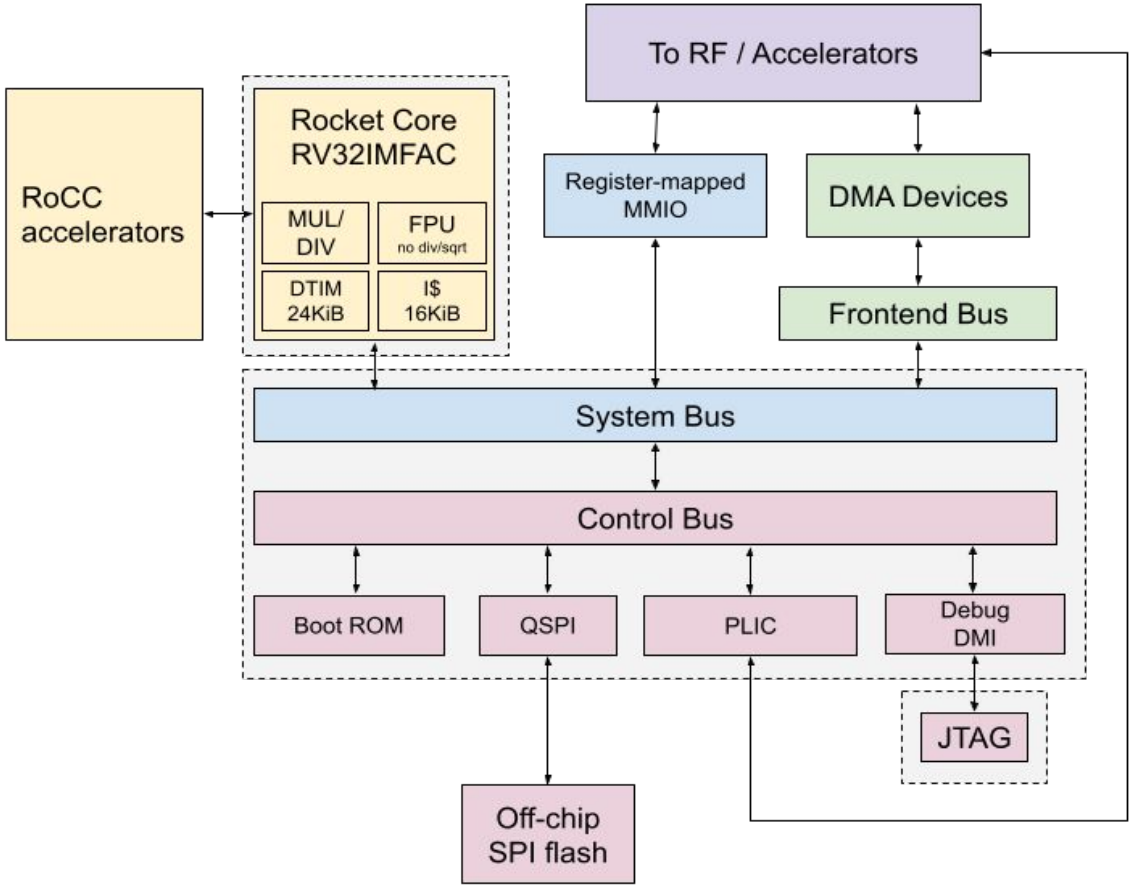
Josh Alexander
4th Yr EECS Undergrad



Cheng Cao
3rd Yr EECS Undergrad

Overview

- RV32IMAFC Core
- 16KB I$
- 24KB Data Memory (DTIM)
- JTAG
- UART
- GPIO
- QSPI flash
- Interrupt Controller
  - PLIC (Platform level)
  - CLINT (Core level)



Memory subsystem

- Single core, uses TileLink protocol to connect peripheral devices
- The Data Tightly Integrated Memory in place of the L1D$
  - Backs the L1I$ through the System Bus
- 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 & providing self-boot capabilities
- The debug module has a small SRAM to provide backing in case of other memory blocks are inaccessible

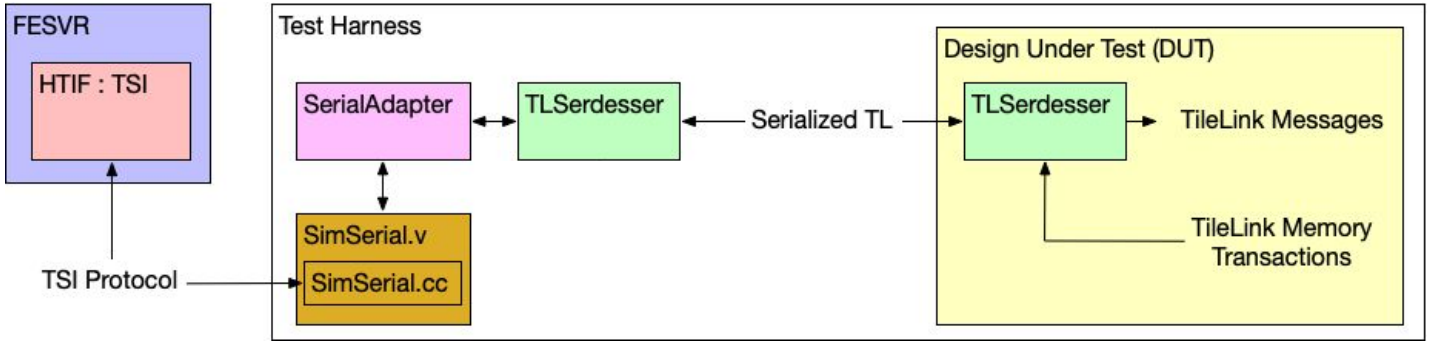
Boot Process (Boot ROM)

Two options:

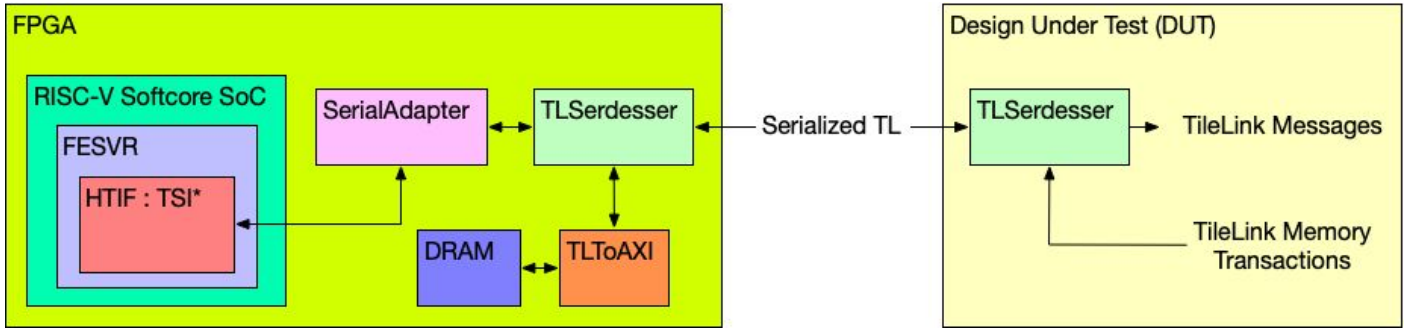
1. Boot select pin is high (Self-boot)
    a. First few kilobytes of SPI is copied to DRAM base address
    b. Jump to DRAM base
2. Boot select pin is low (Tethered debug)
    a. Setup a trap handler, and enter a wait for interrupt loop
    b. Use JTAG, TSI, or other external debugging tools to program the on-chip memory
    c. Use JTAG, TSI, or other external tools to trigger an software interrupt (MSIP)
    d. Boot ROM receives exception, jumps to the programmed memory

TSI Debug

simulation:

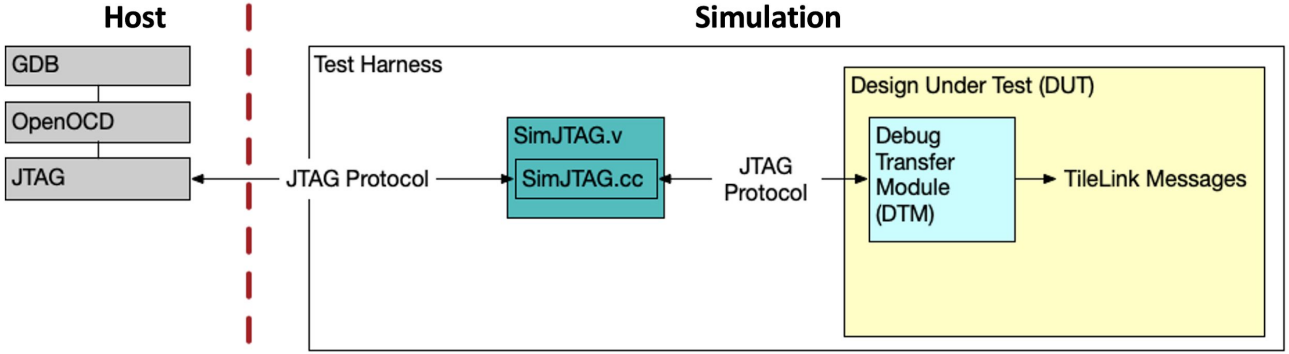


chip bringup:

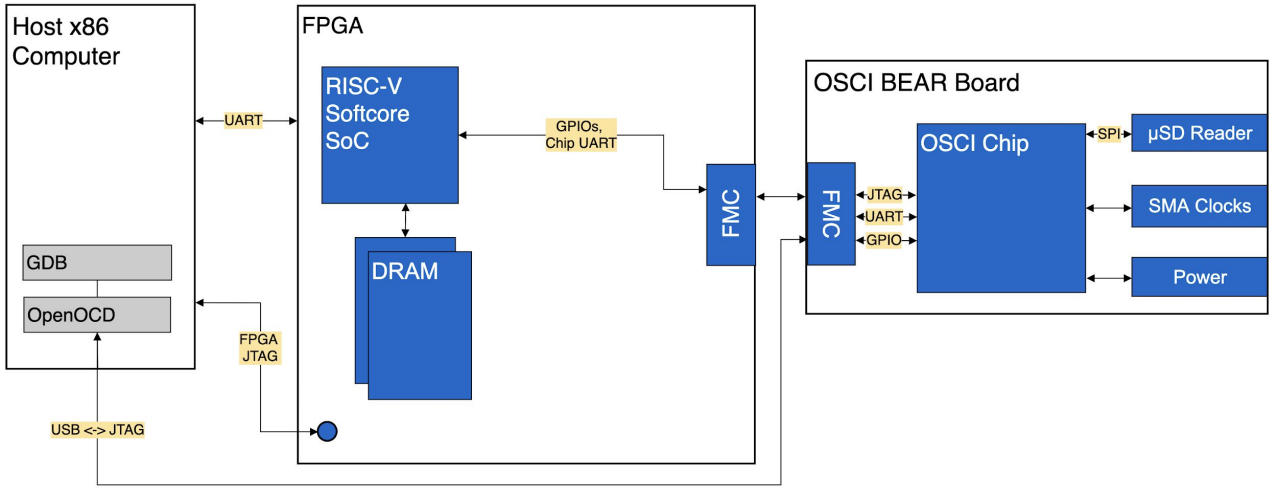


JTAG Debug

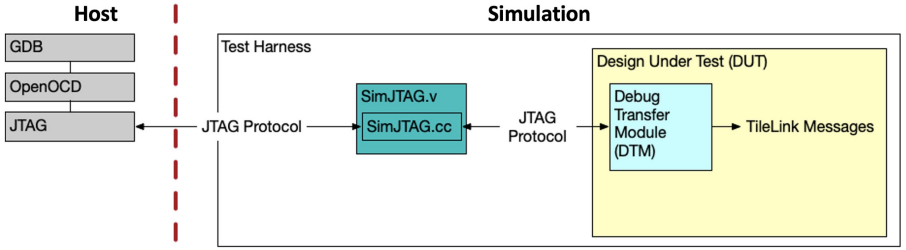
simulation:



chip bringup:
(**preliminary diagram)



JTAG Simulation



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(chipyard) nayiri@bwrcr740-8 /scratch/nayiri/tstech28/chipyard3/sims/vcs (v1p4-digital)$ ./3command.sh
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 200000
(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 0x40 lma 0x80001000
Loading section .text, size 0x12c4 lma 0x80002000
Loading section .text.startup, size 0x7c lma 0x800032c4
Loading section .rodata, size 0x258 lma 0x80003340
Loading section .rodata.str1.4, size 0x40 lma 0x80003598
Loading section .eh_frame, size 0x50 lma 0x800035d8
Loading section .data, size 0x22 lma 0x80003628
Loading section .sdata, size 0x4 lma 0x8000364c
Start address 0x80000000, load size 6232
Transfer rate: 20 bytes/sec, 692 bytes/write.
(gdb) p (int) wait
$1 = 1
(gdb) p (char*) &text
$2 = 0x80003628 <text> "Vafgehpgvba frgf jnag gb or serr!"
(gdb) p (int) wait=0
Left operand of assignment is not an lvalue.
(gdb) print wait=0
'wait' has unknown type; cast it to its declared type
(gdb) p (volatile int) wait=0
Left operand of assignment is not an lvalue.
(gdb) set wait =0
'wait' has unknown type; cast it to its declared type
(gdb) set (volatile int) wait =0
Left operand of assignment is not an lvalue.
(gdb) p (int*) &wait
$3 = (int *) 0x8000364c <wait>
(gdb) set *((int*)&wait)=0
(gdb) p (int) wait
$4 = 0
(gdb) c
Continuing.
keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (1845). Workaround: incr
ease "set remotetimeout" in GDB
keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (3837). Workaround: incr
ease "set remotetimeout" in GDB
keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2514). Workaround: incr
ease "set remotetimeout" in GDB
^C
Program received signal SIGINT, Interrupt.
0x80003322 in main ()
(gdb) p (char*) &text
$5 = 0x80003628 <text> "Instruction sets want to be free!"
(gdb)
```

```
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:46037
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! (21985). Workaround
: increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2179). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2367). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2251). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2208). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2296). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2333). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2351). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2350). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2483). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2338). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2350). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2341). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2367). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2319). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2351). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2350). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2407). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2326). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2319). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2298). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2294). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2576). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2307). Workaround:
 increase "set remotetimeout" in GDB
Warn : keep_alive() was not invoked in the 1000ms timelimit. GDB alive packet not sent! (2338). Workaround:
 increase "set remotetimeout" in GDB
```

1: bash, bash, bash

```
(chipyard) nayiri@bwrcr740-8 /scratch/nayiri/tstech28/chipyard3/sims/vcs (v1p4-digital)$ ./1command.sh
Running with RISCV=/scratch/nayiri/tstech28/chipyard3/riscv-tools-install
(set -o pipefail &&  /scratch/nayiri/tstech28/chipyard3/sims/vcs/simv-chipyard-EE290CBLEConfig +permissiv
e +spiflash0=spi.bin +ee290c_bsel=1 +jtag_rbb_enable=1 -V +dramsim +dramsim_ini_dir=/scratch/nayiri/tstec
h28/chipyard3/generators/testchipip/src/main/resources/dramsim2_ini +max-cycles=100000000000  +ntb_random
_seed_automatic +verbose +permissive-off hellodebug.riscv </dev/null 2> >(spike-dasm > /scratch/nayiri/ts
tech28/chipyard3/sims/vcs/output/chipyard.TestHarness.EE290CBLEConfig/hellodebug.out) | tee /scratch/nayi
ri/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;  Apr  4 18:49 2021
VCS Build Date = May 26 2020 20:23:14
Start run at Apr  4 18:49 2021

NOTE: automatic random seed used: 362664913
[UART] UART0 is here (stdin/stdout).
```

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)
        ;
}
```

Simulation & Software

- Simulation platforms
- Toolchain & Software
- Tests

Simulation platforms

- Digital elaborated RTL in VCS
- Digital elaborated RTL + Digital model of analog toplevel
- Post-syn & post-par with timing information

- Supported mode
  - Self-booting from SPI, terminal console provided through TSI or UART
  - Tethered booting with TSI
  - Tethered debugging with JTAG + openOCD + GDB

Toolchain & Support libraries

- Raw assembly program
- Bare-metal C toolchain & support libraries
  - Custom linker scripts for correct code / stack / data placement
  - Routines / libraries for
    - Interrupts (core level & platform)
    - IO peripherals
    - Baseband and modem control
- Basic benchmarks & ISA unit tests
  - 0.8 Dhrystone MIPS / MHz (Through TSI, might include debugging interface polling overhead)

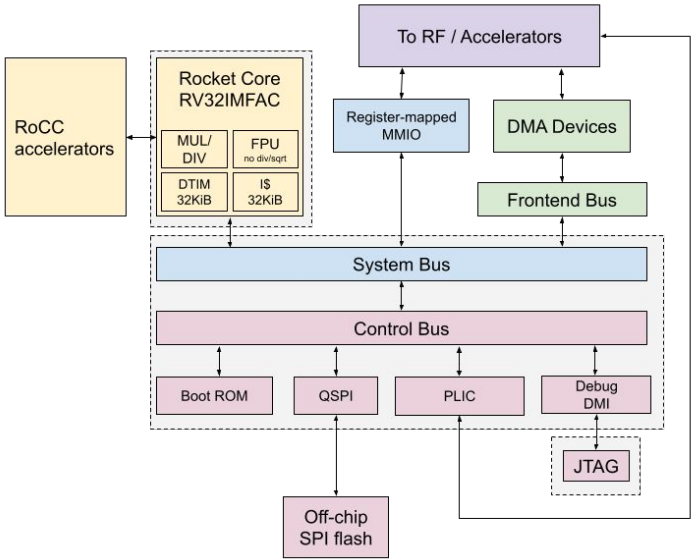
Interface with RF/Accelerators

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

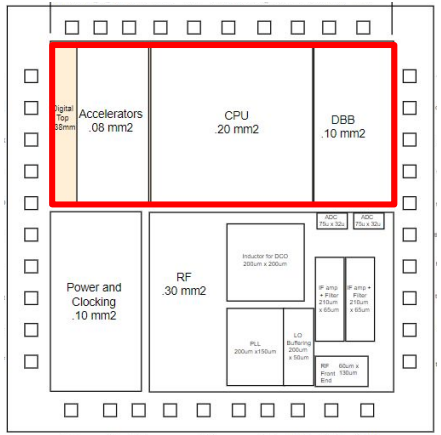
AES

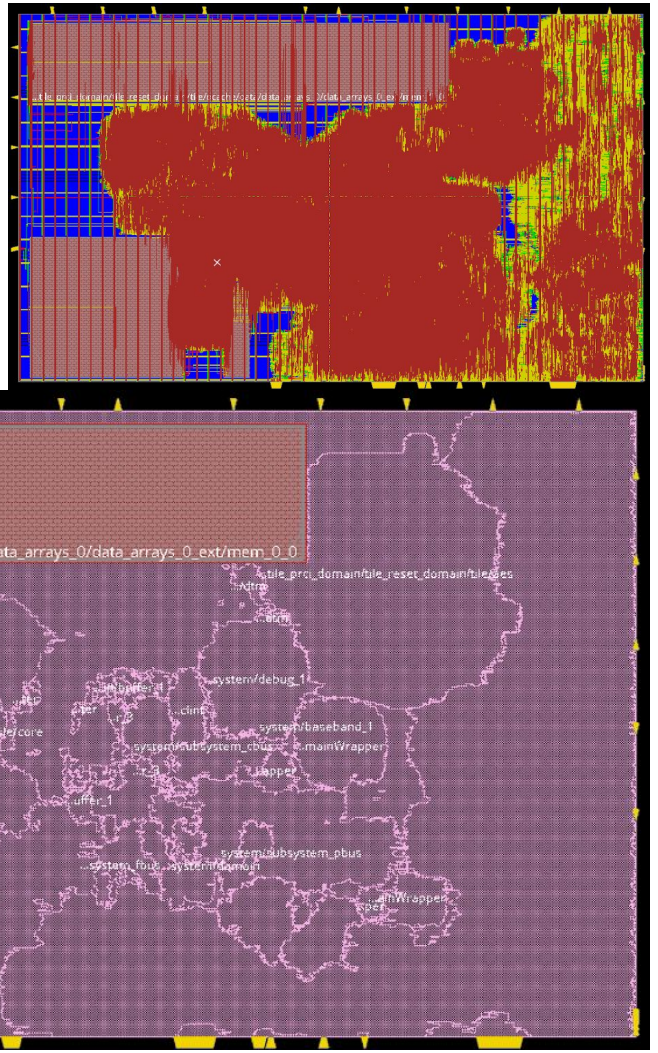
- Interacts directly with Rocket Core
- Uses DMA along with RF
- Space for text to decrypt/encrypt



Physical Design

- Digital
  - Core, accelerators, digital baseband, peripherals
  - VDD = 0.9 V
  - clock freq = 50MHz
  - M3-M6 power straps
- Area = 0.342mm^2, Density = 74%





Timing

- hold time and max cap violations due to slow clock
- constrained clock tree

```
------------------------------------------------------------
     opt_design Final SI Timing Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+
|     Setup mode     |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.209  |  0.209  |  9.469  |  7.211  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |
|          All Paths:|  43879  |  41896  |  1830   |  8885   |
+--------------------+---------+---------+---------+---------+

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

Density: 73.650%
       (100.000% with Fillers)
Total number of glitch violations: 0
------------------------------------------------------------
```

```
------------------------------------------------------------
     opt_design Final SI Timing Summary
------------------------------------------------------------

+--------------------+---------+---------+---------+---------+
|     Hold mode      |   all   | reg2reg |reg2cgate| default |
+--------------------+---------+---------+---------+---------+
|           WNS (ns):|  0.088  |  0.088  |  0.110  |  0.114  |
|           TNS (ns):|  0.000  |  0.000  |  0.000  |  0.000  |
|    Violating Paths:|    0    |    0    |    0    |    0    |
|          All Paths:|  44564  |  42581  |  1830   |  8885   |
+--------------------+---------+---------+---------+---------+

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

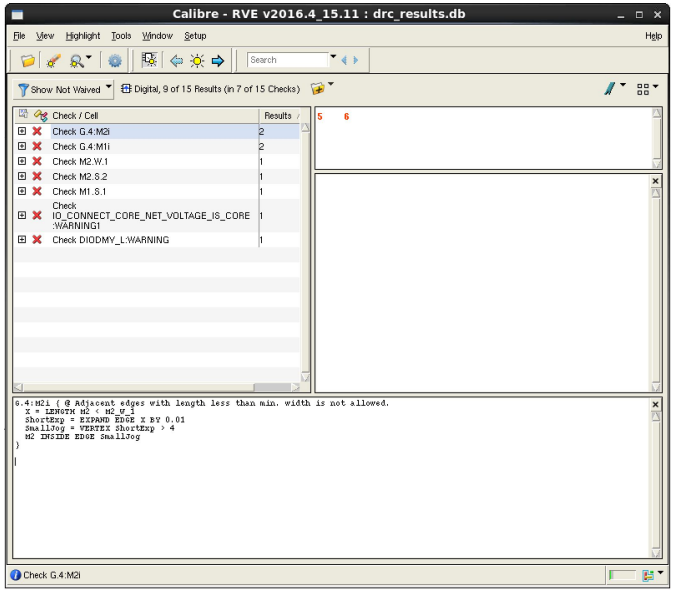
Power

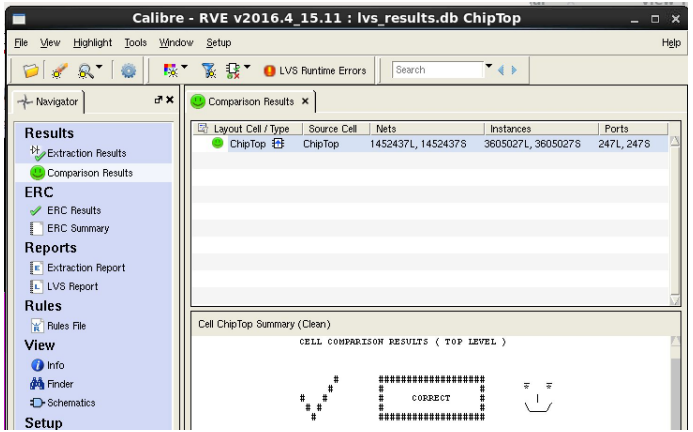
- 16.9mW total @ 50MHz, 0.9V
- after switching to HVT devices and SVT SRAMs
- if switching power scales linearly with f,V: 11.8mW @ 20MHz, 0.5V

```
Total Power
-----------------------------------------------------------
Total Internal Power:        7.19172061      42.5915%
Total Switching Power:       6.46540188      38.2900%
Total Leakage Power:         3.22823840      19.1186%
Total Power:                16.88536091
-----------------------------------------------------------
```

DRC/LVS Cleaning

- DRC:
  - removed 1-site wide filler gaps
  - scaled down M3-M1 vias on power straps and M5-M4 vias over SRAMs
- LVS:
  - set which cells were marked as “physical only”
- Both:
  - improved SRAM placement, allowing better placement of peripherals / baseband by the tools
  - power straps: lower density on some layers to allow routing in same layer
  - reduce design density: 32Kb I$/D$ → 16K I$, 24K D$



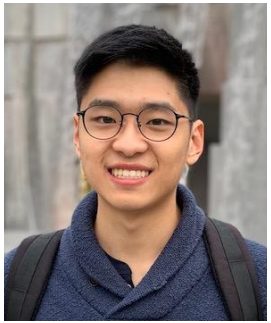


Lessons Learned

- The chip design environment is difficult to set up
  - working on servers, installing toolchains, running large EDA tools
- Digital design flow is all about properly running & constraining the EDA tools
  - One small fix could solve 100s of errors
  - The tools do weird things when not properly constrained
- Asking more experienced grad students for help was crucial

AES Accelerator

Team Introduction



Anson Tsai

5th Yr M.S., advised by Bora

Eric Wu

EECS M.Eng., advised by Kris

Daniel Fan

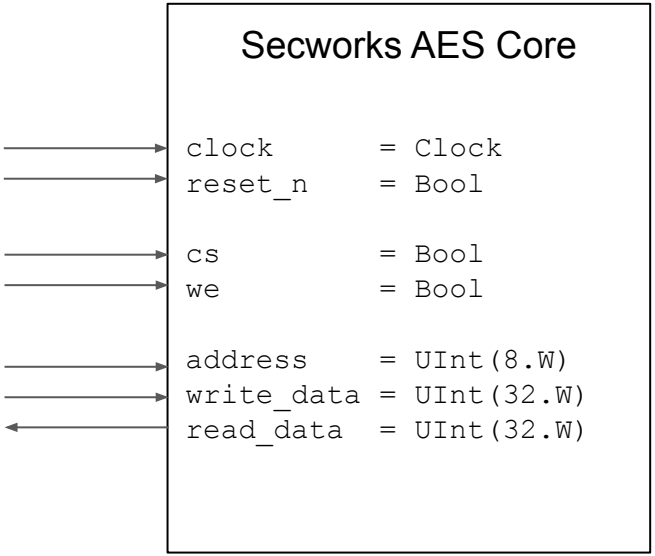
3rd Yr EECS undergrad

Overview Design

- RoCC (Rocket Custom Co-processor) AES Accelerator
  - Special RISC-V instructions for accelerator
- Based on open-source Verilog Secworks AES core
  - Supports 128b and 256b AES encryption and decryption
- Custom HW wrapper logic for Rocket integration
  - Optimizations: non-blocking instructions and sequential operations
- Custom software stack to compile accelerated C code

Secworks AES Core

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



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

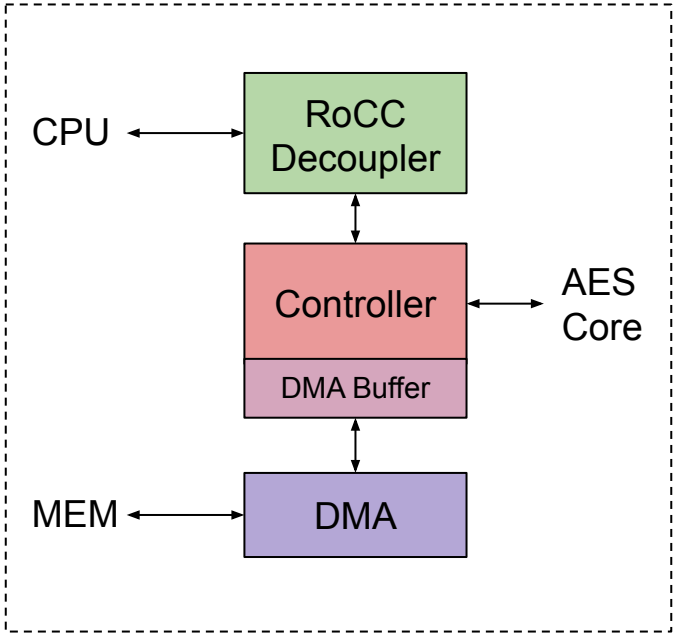
6. Specify encryption/decryption by writing to the config register
7. Start block processing by writing a one to the next bit in the control register.
8. Wait for the ready bit in the status register to be cleared and then to be set again. This means that the data block has been processed.

DATA WRITE

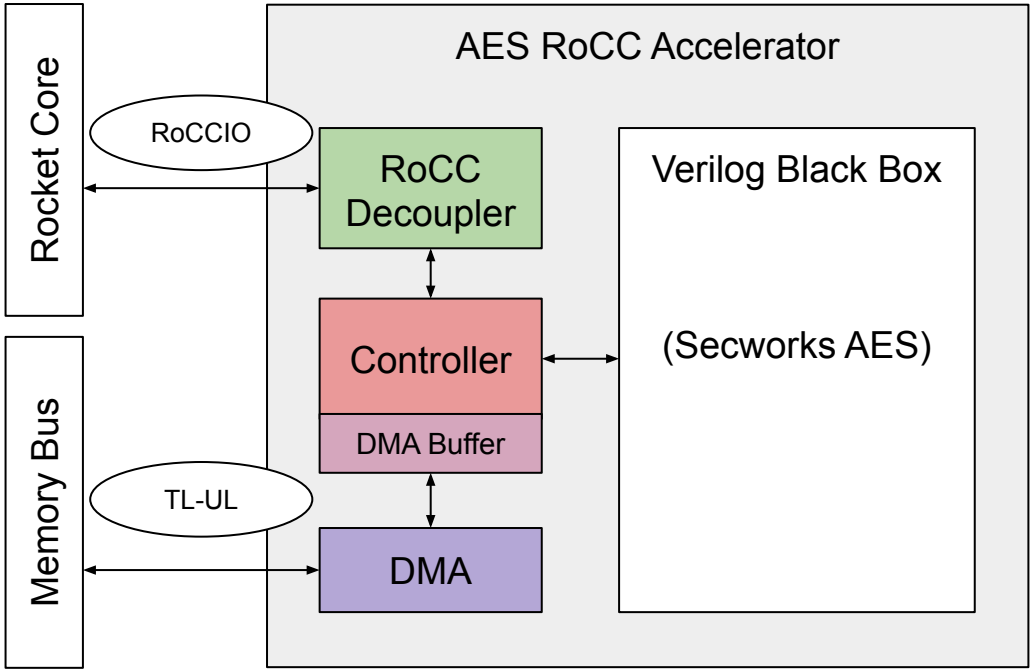
9. Read out the ciphertext block from the result registers.

What we implemented:

- 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

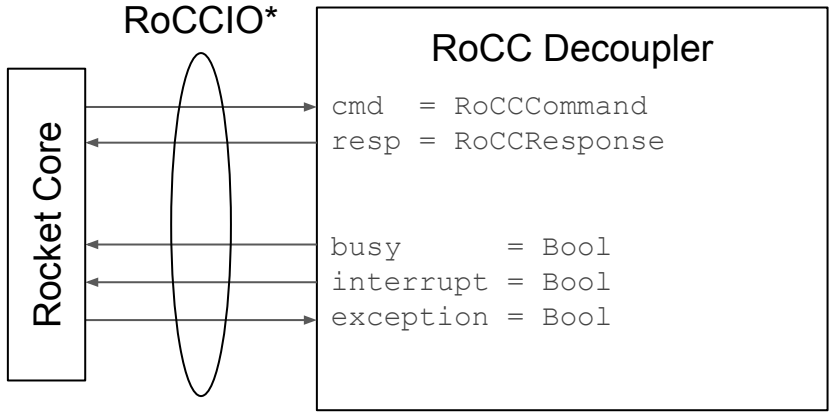


AES Accelerator Top-Level Diagram



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



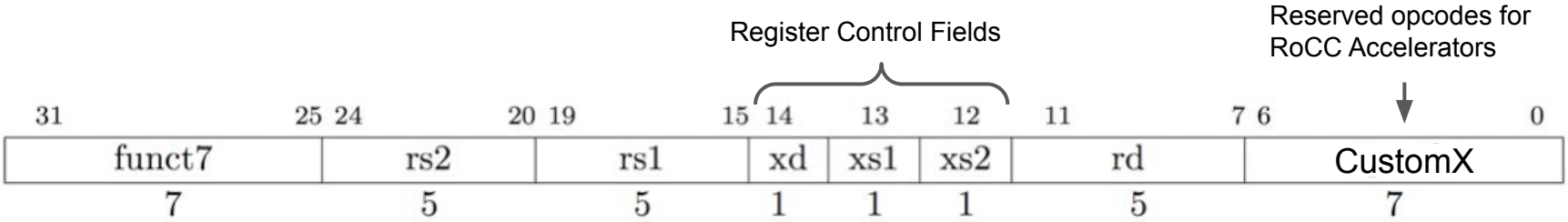
RoCCCommand

```
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)
```

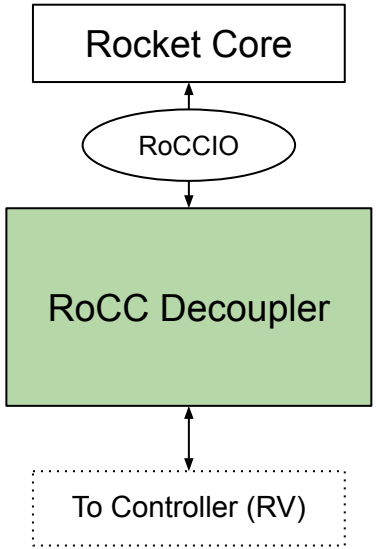
Custom RISC-V Instructions



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

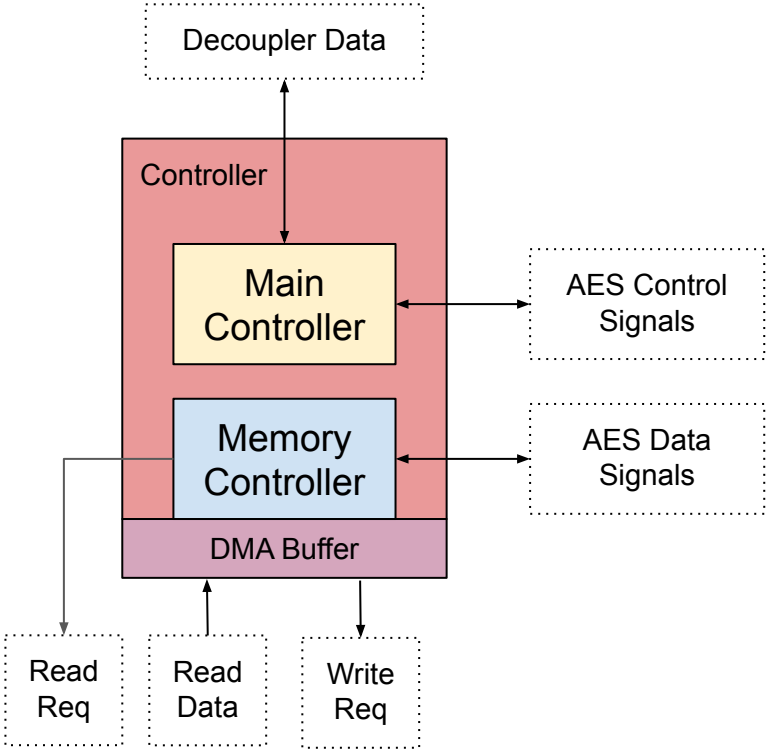
RoCC Decoupler

- Communicates with CPU via RoCCIO
  - Custom co-processor interface for Rocket
  - Wrapper for RISC-V Instructions
- Processes instructions from CPU
  - Non-blocking operation
  - Enables CPU to query accelerator status anytime
- Buffers instruction data for controller
  - Data presented to controller via ready-valid interface

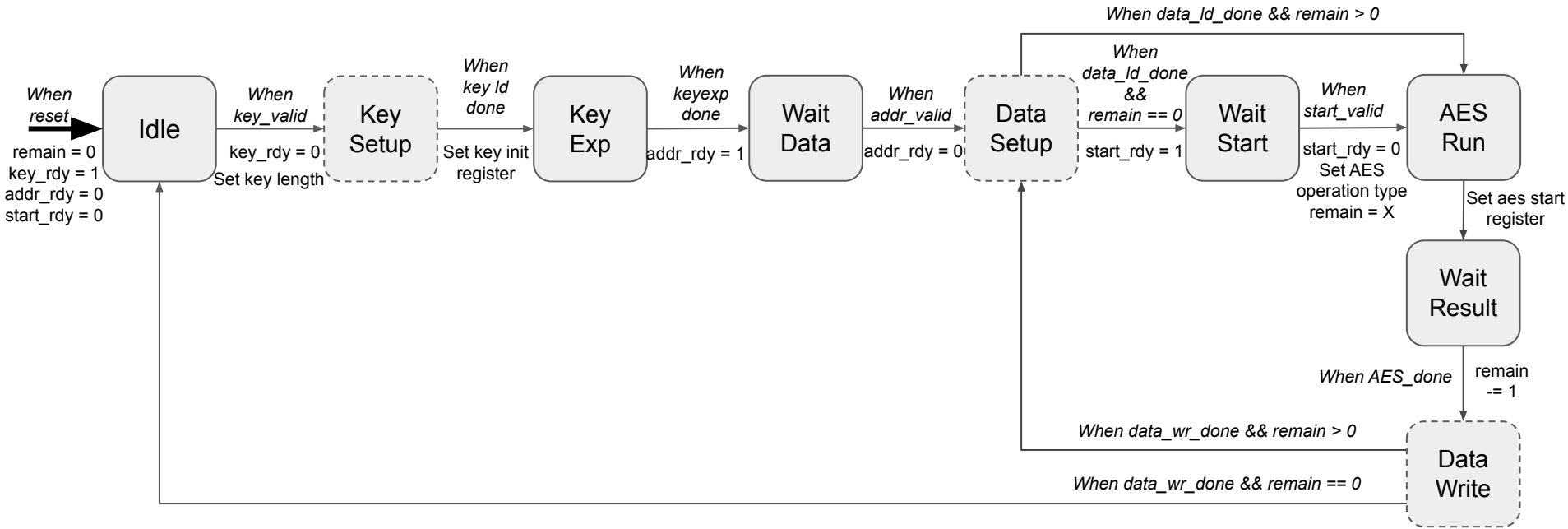


Controller

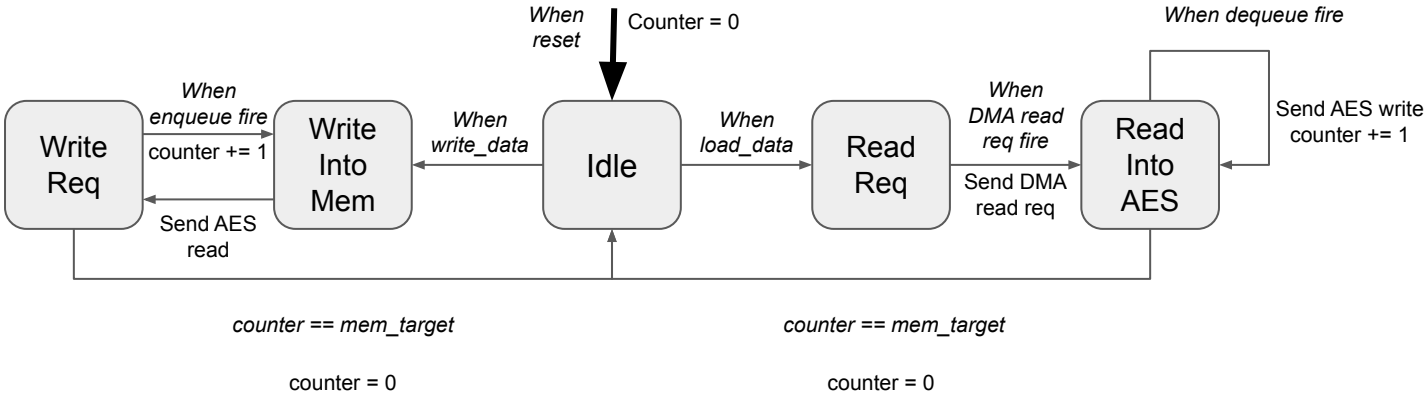
- Main Controller
  - Receive instructions from decoupler
  - Operates the AES Core
- Memory Controller
  - Handles data transfer to/from AES Core via DMA
- DMA Buffer
  - Buffers and processes data to and from DMA (e.g. packing/unpacking data packets)



Main Controller - FSM

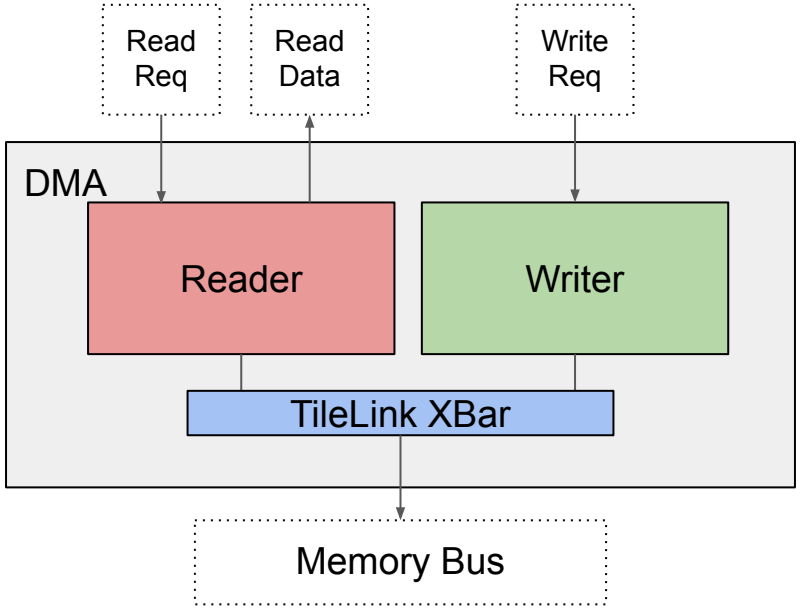


Memory Controller - FSM

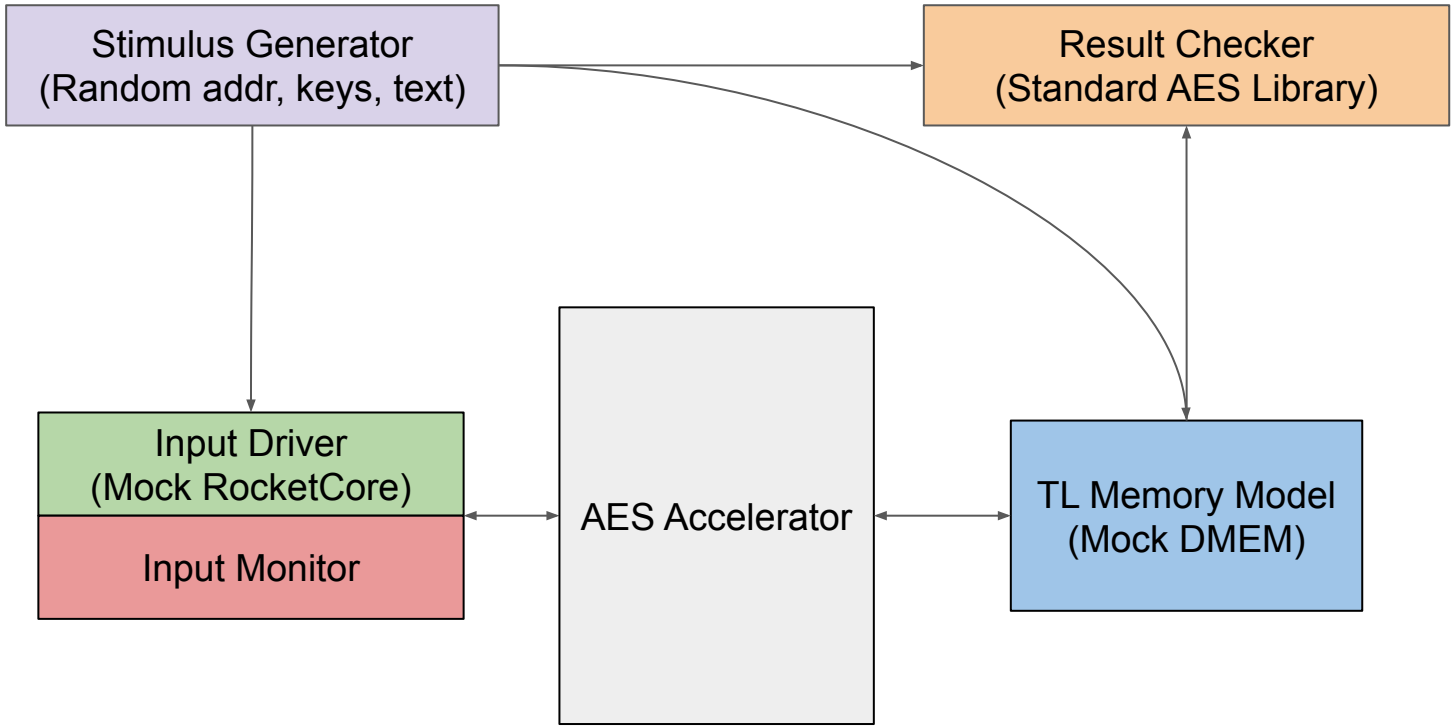


DMA

- Services memory requests from controller
- Interfaces with memory bus via TileLink
  - TileLink is a chip-scale interconnect in Rocket
- Contains reader and writer sub-blocks
  - Concurrent read + write operations



Verification: Accelerator Testbench



Verification Results

- Minor bugs found
  - Top-level busy signal was not properly implemented
  - Controller FSM had an off-by-one error
- RTL coverage
  - Line: 97.32%, Condition: 95.98%, Branch: 95.52%, Toggle: 77.83%
- Accelerator performance (excluding memory latency)
  - 128b enc/dec:
    - Single block: 101 cycles
    - Multiple blocks: 64 cycles per additional block
  - 256b enc/dec
    - Single block: 130 cycles
    - Multiple blocks: 87 cycles per additional block

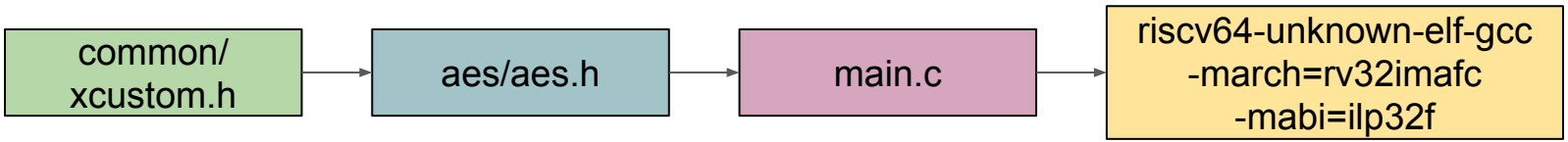
Software - C Library

- Accelerated AES functions supplied via “aes.h” header file
- Primitives created by using C macros to wrap custom RoCC instructions
- End-to-end block cipher routines for ease of use
    - Combinations of ECB/CBC/CTR cipher modes and polling/interrupts available in addition to support for different AES key sizes

Software

- 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")
```



Software

- Example C code
  - Implementation of AES_ECB_ENC_POLL

```
1 void AES_ECB_ENC_POLL(uint8_t *key, int keylen, uint8_t *src, uint8_t *dest, int length) {
2   if (keylen == 128) {
3     aes_extended_keysetup128(key);
4   } else {
5     aes_extended_keysetup256(key);
6   }
7   aes_extended_addressload(src, dest);
8   aes_extended_encryptblocks((int) (length / AES_BLOCKLEN), 0);
9   int status = 1;
10  while (status) {
11      aes_extended_querystatus(status);
12  }
13 }
```

Software

- Example C code
  - More involved implementation of AES_CTR_XCRYPT_POLL

```
1  void AES_CTR_XCRYPT_POLL(uint8_t *IV, uint8_t *key, int keylen, uint8_t *src, uint8_t *dest, int length) {
2    uint8_t buffer[AES_BLOCKLEN];
3    cpy(IV, buffer, AES_BLOCKLEN);
4
5    for (int i = 0, bi = AES_BLOCKLEN; i < length; ++i, ++bi)
6    {
7      if (bi == AES_BLOCKLEN)
8      {
9        cpy(buffer, dest + i, AES_BLOCKLEN);
10
11       /* Increment Iv and handle overflow */
12       for (bi = (AES_BLOCKLEN - 1); bi >= 0; --bi)
13       {
14         /* inc will overflow */
15         if (buffer[bi] == 255)
16         {
17           buffer[bi] = 0;
18           continue;
19         }
20         buffer[bi] += 1;
21         break;
22       }
23       bi = 0;
24     }
25   }
26   AES_ECB_ENC_POLL(key, keylen, dest, dest, length);
27   for (int i = 0; i < length; ++i)
28   {
29     dest[i] ^= src[i];
30   }
31 }
```

Software

- Example C code
  - Using AES_ECB_ENC_POLL in a software test

```
1   // ----------------- Single Block 128b Enc/Dec Loop -----------------
2   printf("AES CBC E/D Loop, 4 blocks, 128b Key, Polling:\n");
3
4   // Encrypting all four blocks using single function call
5   printf("Encrypting plaintext and checking output...\n");
6   AES_CBC_ENC_POLL(iv, keyone, 128, plaintext, fillerspace, 64);
7
8   // Comparing actual results to expected
9   checkBytes(fillerspace, ciphertextone, 64);
10
11  // Decrypting all four blocks using single function call (using expected as source)
12  printf("Decrypting ciphertext and checking output...\n");
13  AES_CBC_DEC_POLL(iv, keyone, 128, ciphertextone, fillerspace, 64);
14
15  // Comparing actual results to expected
16  checkBytes(fillerspace, plaintext, 64);
17
```

SoC Testing Results

- Minor data storage discrepancy found
  - C stores byte arrays in reverse to what accelerator expects
  - Fix: have DMABuffer reverse data from DMA
- Accelerator performance in SoC tests (includes memory latency)
  - 128b enc/dec:
    - Single block: ~200 cycles
    - Multiple blocks: ~135 cycles per additional block
  - 256b enc/dec:
    - Single block: ~264 cycles
    - Multiple blocks: ~155 cycles per additional block

AES <> Digital Baseband

- Digital Baseband supports 1M BLE Spec
- 1 Megasymbol per second, or 10^6 bits per second
  - Given 20MHz clock freq → AES accel needs to process 0.05 bits per cycle
  - Worst case: Single block 256b enc/dec: 128b over 2 * 264 cycles → 0.24 bits per cycle
  - AES Accelerator can enc/dec faster than DBB can receive
- Spec allows a maximum of 150us between request and response
  - 150us → 3000 cycles (20MHz clock)
  - largest packet size is 256B, or 16 128b blocks
  - Worst case: 256b enc/dec: 2 * (264 + 15 * 156) = 5,208 cycles
    - Cannot have dynamic responses
    - 64B packet size is more reasonable, takes 1,464 cycles
  - Limited by clock frequency
    - Originally operating at 100MHz, but ADC only verified at 20MHz

RF Baseband

Team Introduction

Ryan Lund

5th Year M.S. Student

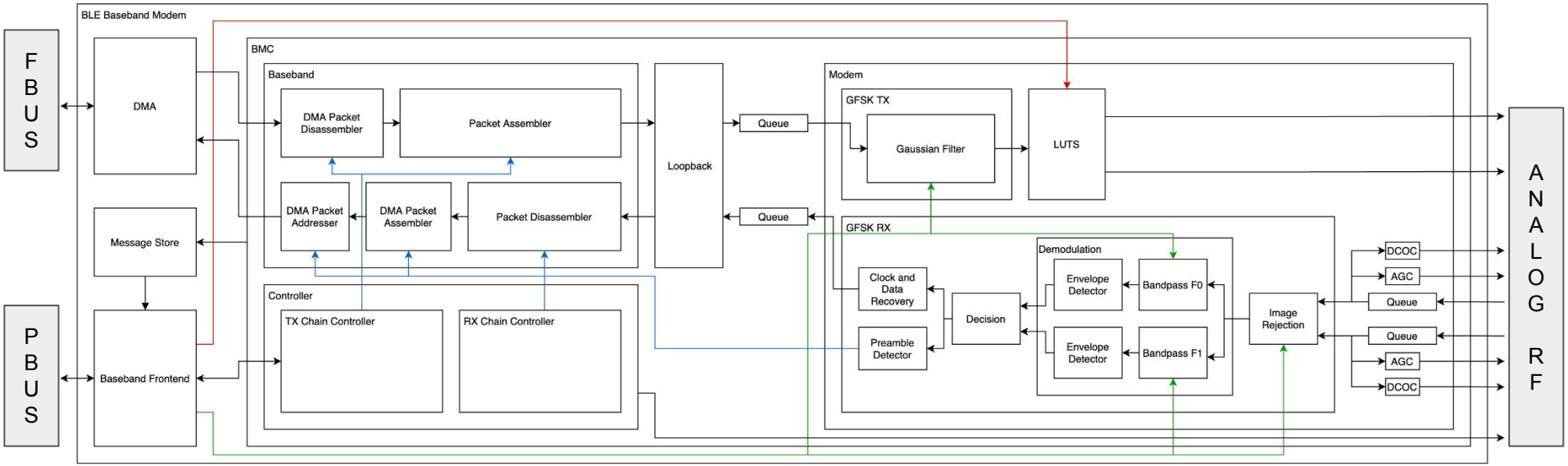
Griffin Prechter

5th Year M.S. Student

Design Characteristics

- Sends and receives 1M Uncoded Link Layer Packets
  - Chisel-based design can be expanded for more packet types (2M, LE Coded) in the future
- Manages RF tuning constants
  - Combination of CPU set and dynamically adjusted (DCO, AGC)
- Resilient to real world circumstances
  - Graceful error and noise handling
- Adjustable post-tapeout
  - Key outputs to RF run through programmable LUTs
  - FIR taps are similarly programmable
- Provides status outputs for post-tapeout debugging
  - Key values are sent to status registers (ADC values, FSM states)
- “In-house” design made fully in Chisel
- We made our cake by starting with the icing
  - Focused early on the parts that would take our block to the “next level” (DMA, interrupts, high configurability, etc.)

High Level Block Diagram

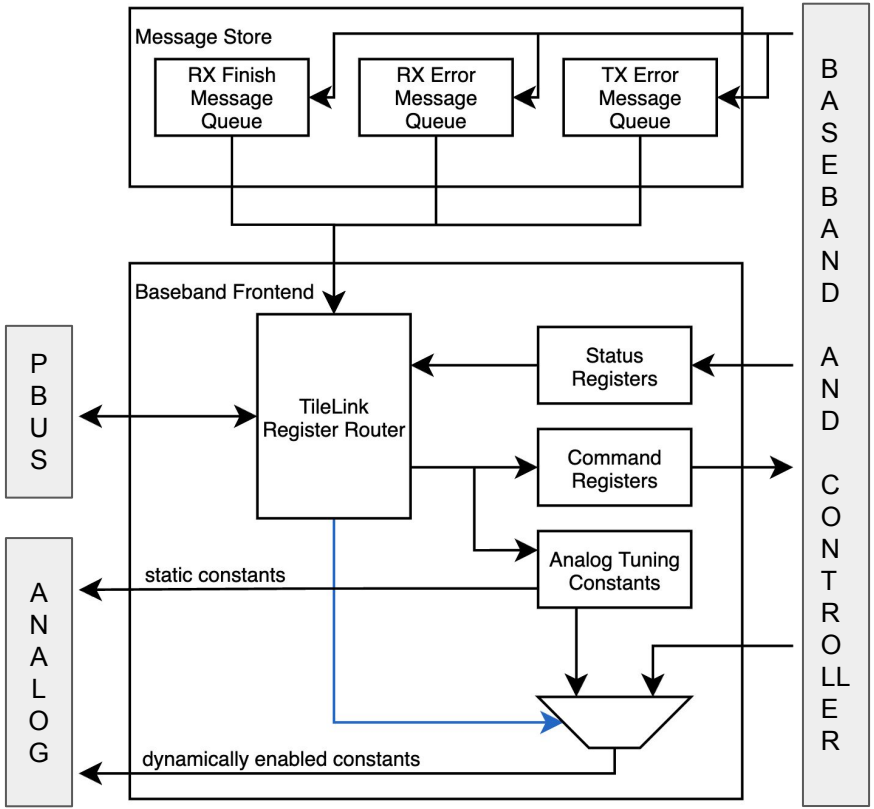


Functional API

1. Main Command: Primary controls for TX and RX
    - CONFIG(*constant, value*): Set *constant* (access address, channel index, etc) to *value*
    - SEND(*n, addr*) : Send *n* bytes stored starting at address *addr* in memory
    - RECEIVE_START(*addr*): Attempt to receive data and store it at address *addr* in memory
    - RECEIVE_EXIT: Exit receive mode
    - DEBUG(*n, addr*): Send *n* bytes stored at address *addr* in memory through the packet assembler and then back through the packet disassembler. The results are stored back into memory at address *word_aligned(addr + n)*.
2. LUT Command: Program LUT entries
    - SET_LUT(lut, index, value): Set the specified *index* in the specified LUT to *value*.
3. FIR Command: Reprogram FIR taps
    - SET_FIR(fir, index, value): Set the indexed tap in the specified FIR to *value*.

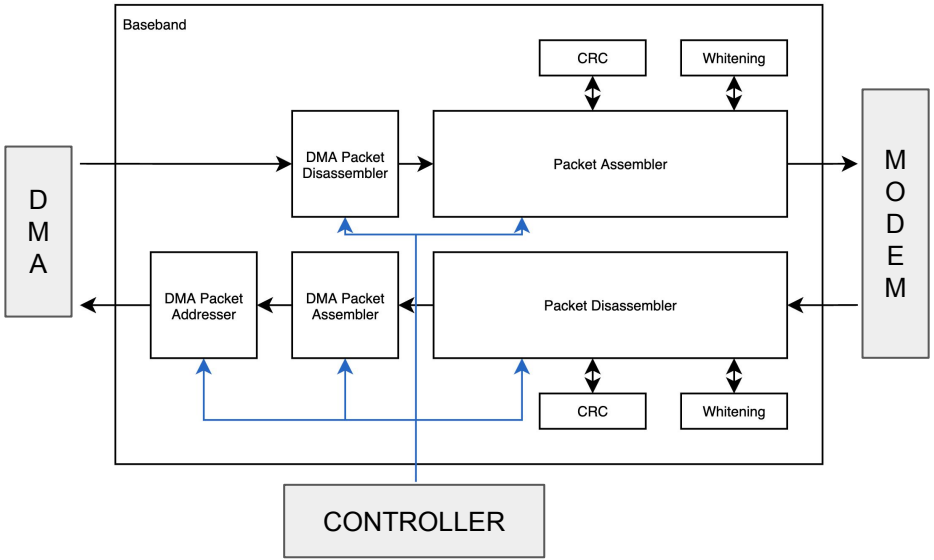
Baseband Frontend & Message Store

- TL register router component exposes registers as memory mapped devices
  - Command registers from CPU
  - Status registers for CPU
  - Analog tuning parameters (e.g. I/Q filter resistors)
- Some tuning constants can be switched between static and dynamic control
  - Ex: Use AGC provided gain code or static code
- Houses connection to PLIC
  - Some interrupts accompanied by error / status messages
- Parameterizable to allow for attachment at different base addresses



Baseband

- Packet assembler (PA) and disassembler (PDA) are FSMs
  - PA: Create valid packet bitstream for some header and data bytes
  - PDA: Extract header and data bytes from packet bitstream
- DMA blocks perform conversions
  - DMA PDA: breaks wide packets into single bytes
  - DMA PA and addresser: Turn single bytes into wide packets and format into valid write requests
- Operations orchestrated by Controller
- PA and PDA send interrupts Frontend and Message Store

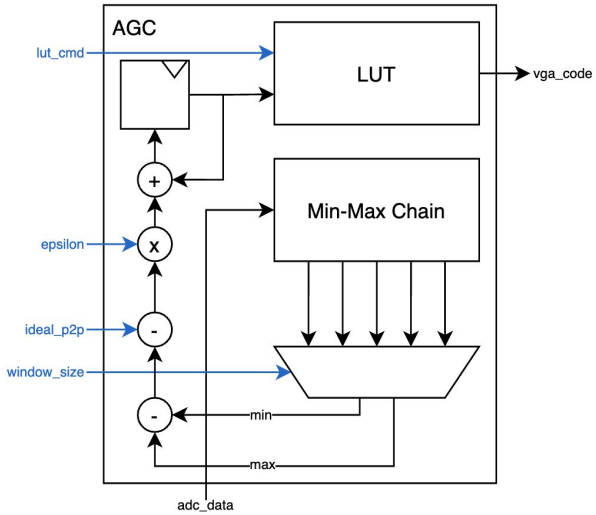


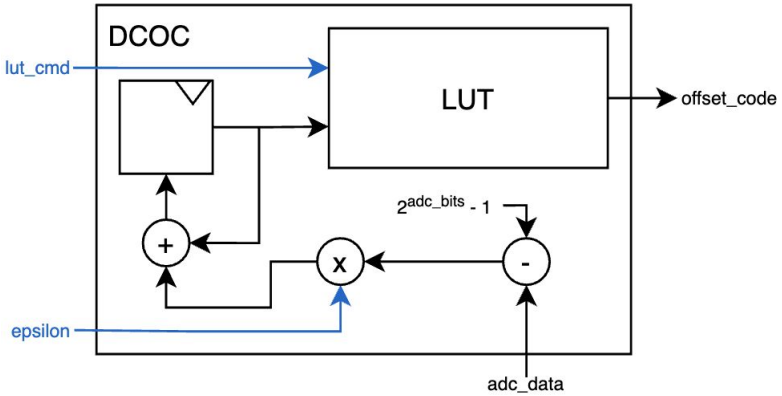
Controller

- Main controller delegates management to sub-controllers for each operation type
  - In debug state, both sub-controllers become active
- Sub-controllers stage the order in which blocks in each chain execute operations
  - Controls are sent throughout the system on R/V interfaces to ensure previous operations have finished execution
- Creates interrupts and interrupt messages based on key points in control orchestration
  - Interrupts with messages: RX start, TX finish
  - Interrupts without messages: RX finish, RX error, TX error

Automatic Gain Control & DC Offset Correction

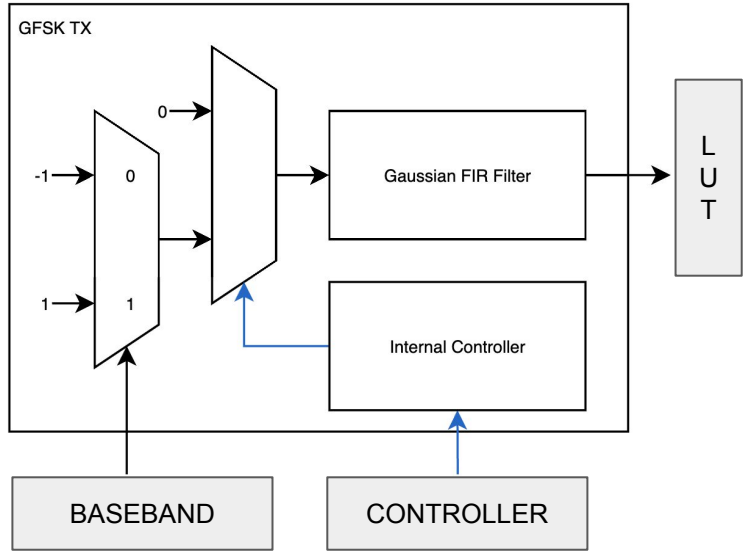
- Highly configurable due to key position in mixed-signal feedback loop
  - Internal LUTs help avoid possible polarity issues on RF interface
- AGC operates by integrating peak to peak deviation from ideal value
- DCO operates by integrating sample deviation from center value

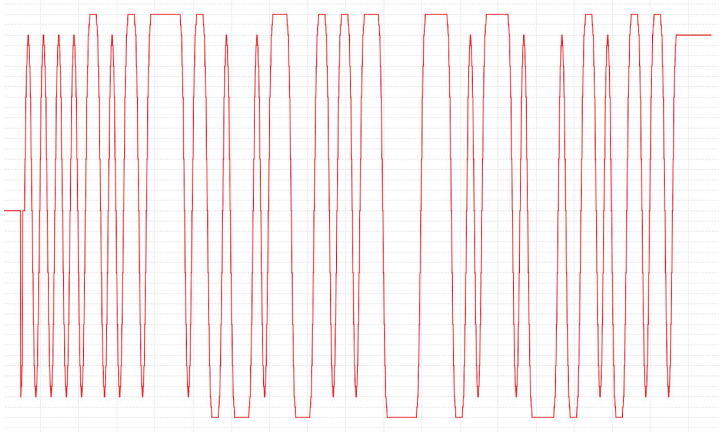




GFSK TX

- Converts incoming bitstream into indices for frequency offset LUT
- Internal controller receives command with number of bits to transmit
  - Each bit is transmitted for 20 cycles during which it is sampled for the FIR 10x (-1 for 0, 1 for 1)
  - 10 empty samples are pushed into the FIR after the last bit to ensure that it is fully transmitted
- Gaussian filter is re-programmable





GFSK RX

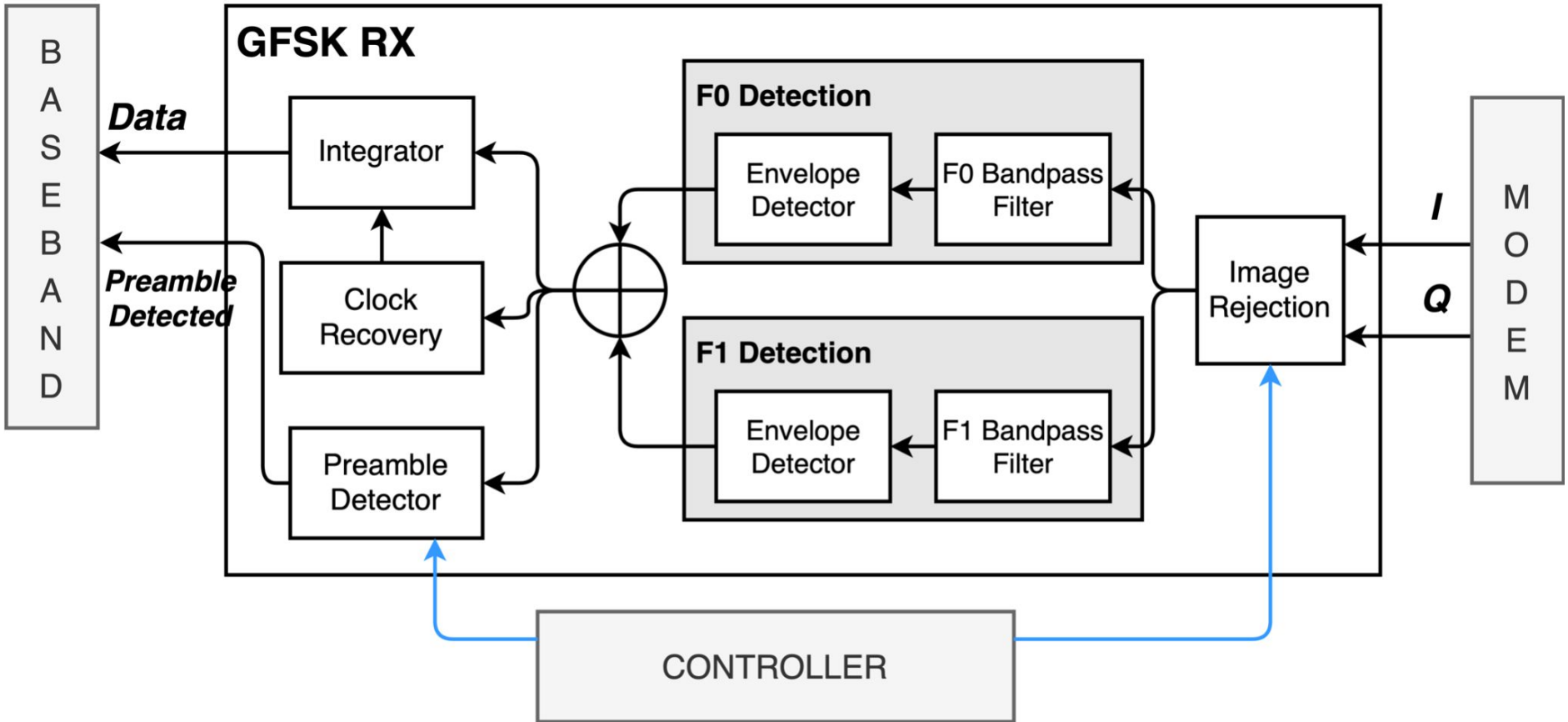
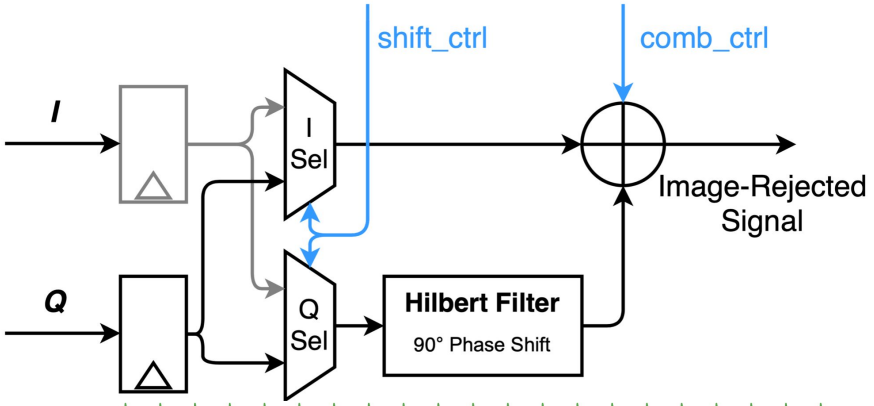

BASEBAND
GFSK RX
Data
Preamble Detected
Integrator
Clock Recovery
Preamble Detector
F0 Detection
Envelope Detector
F0 Bandpass Filter
F1 Detection
Envelope Detector
F1 Bandpass Filter
Image Rejection
I
Q
MODEM
CONTROLLER

Image Rejection

- All-digital; implemented using a Hilbert Filter to perform 90° phase shift
- Highly configurable and controllable post-tapeout
  - Change I and Q signals into digital, configure which signal gets shifted, how signals are combined
  - Fully-programmable 31 tap FIR filter
- Testing demonstrates adequate rejection of image



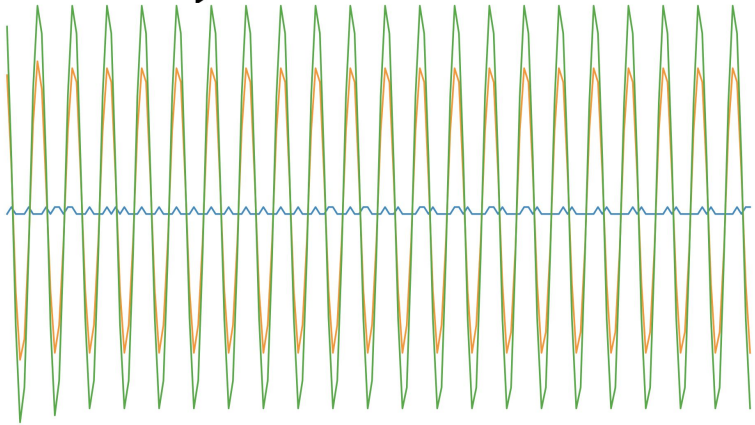
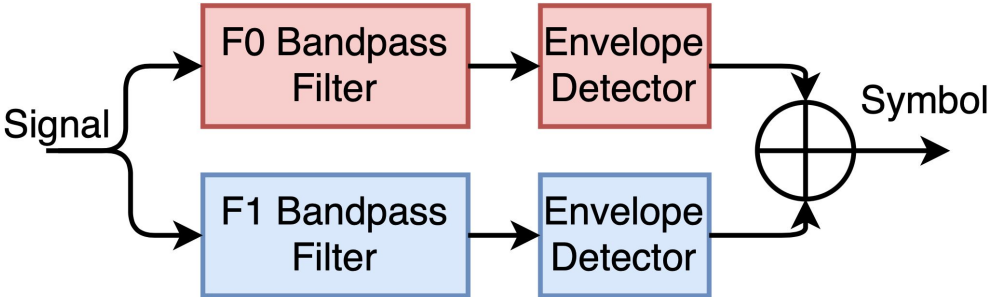
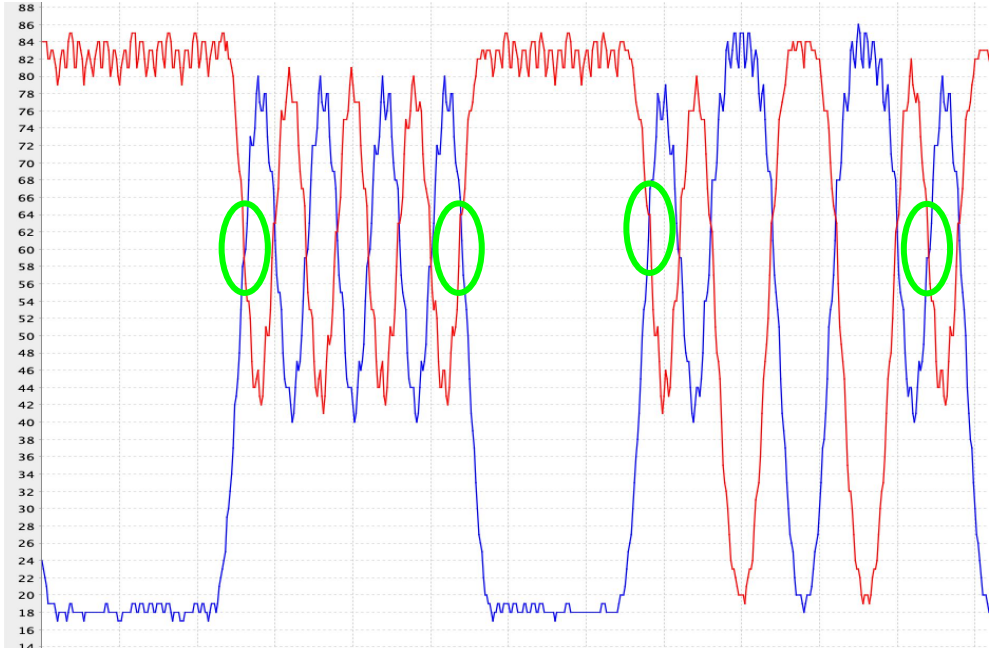


Image Rejection Output: RF, Image, Mixed

Demodulation and Symbol Detection

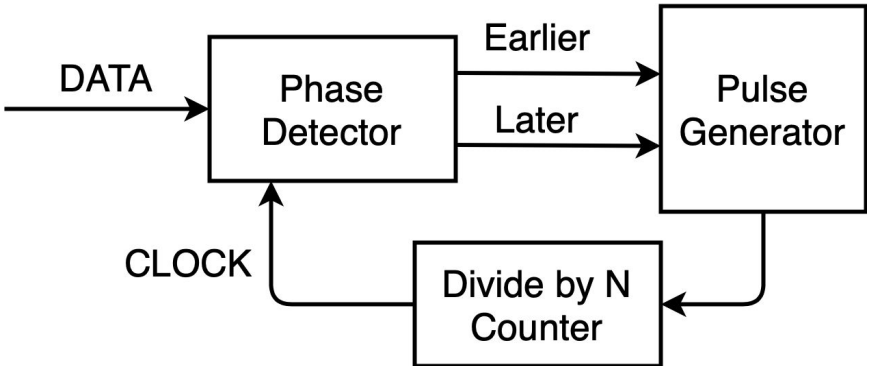
- Bandpass filters around frequency for 0 and frequency for 1
  - Fully programmable FIRs
- Envelope Detectors extract amplitude of bandpassed signal
- Difference between amplitudes determines symbol that is being received.
- Tracks symbol momentum to improve data quality.

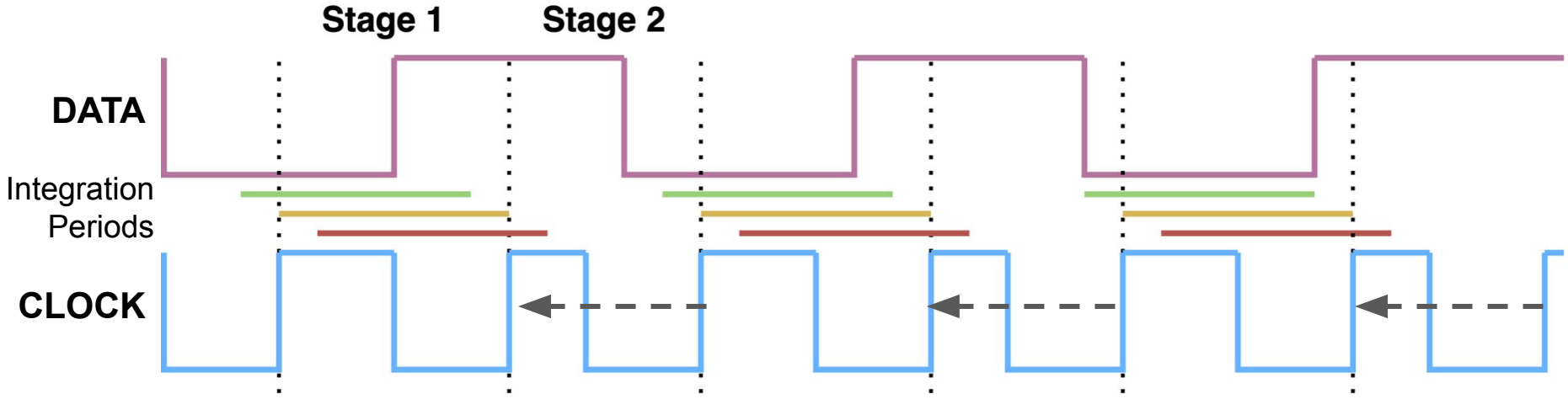




Clock Recovery

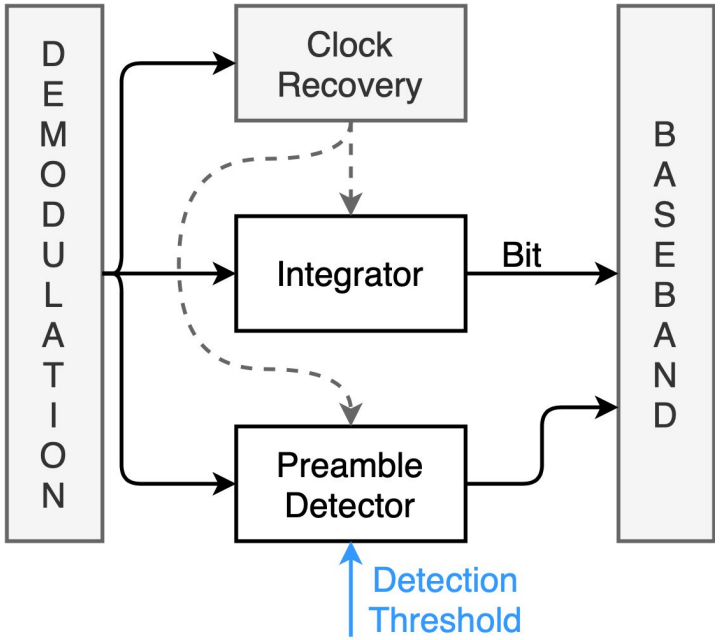
- Early, Prompt, Late phase detector design.
- Phase offset can be adjusted by 10% of clock period every 2 symbols.
  - Chosen to catch up before preamble is over





Data Recovery

- Output of demodulation is integrated across recovered clock period to produce a received bit
- Preamble detector is a correlator that compares output from demodulation with mask depending on BLE access address
  - Number of matches configurable
- Preamble detector tells baseband when to look for the access address



Software

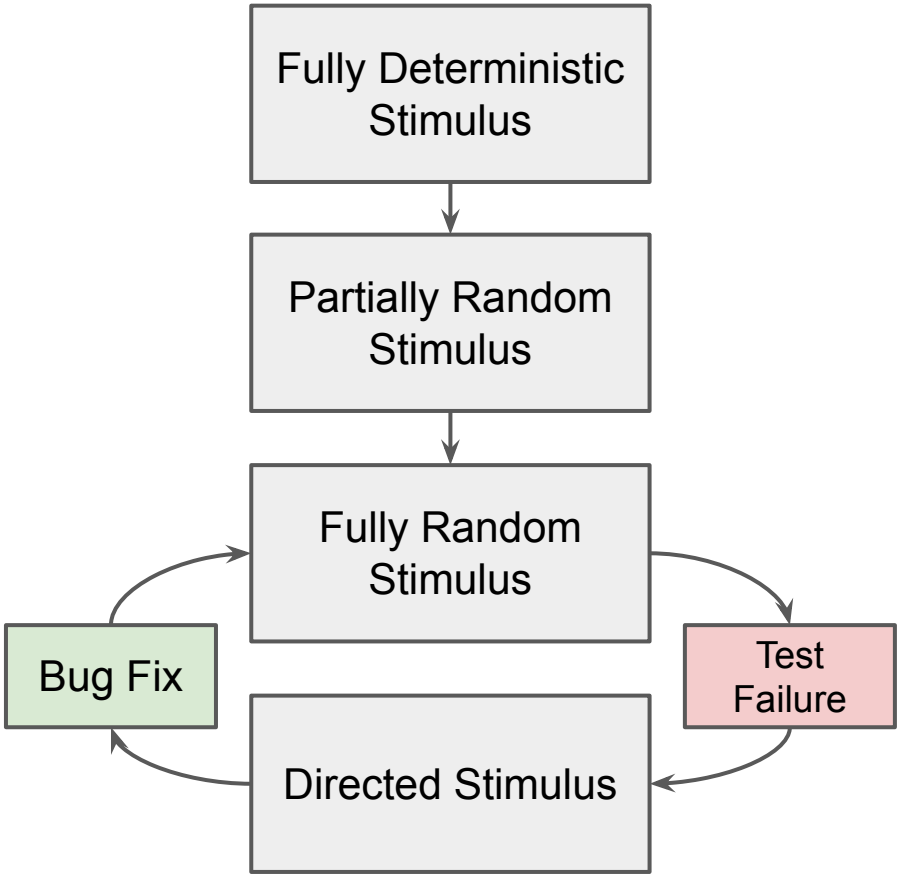
- MMIO controls are easy to interface with in C
  - No special ISA extensions required
  - Generic mmio.h file allows for register reads and writes from 8 to 64-bits
- Series of macros defined for configuring registers and sending baseband instructions to streamline interface

```
// 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));
```

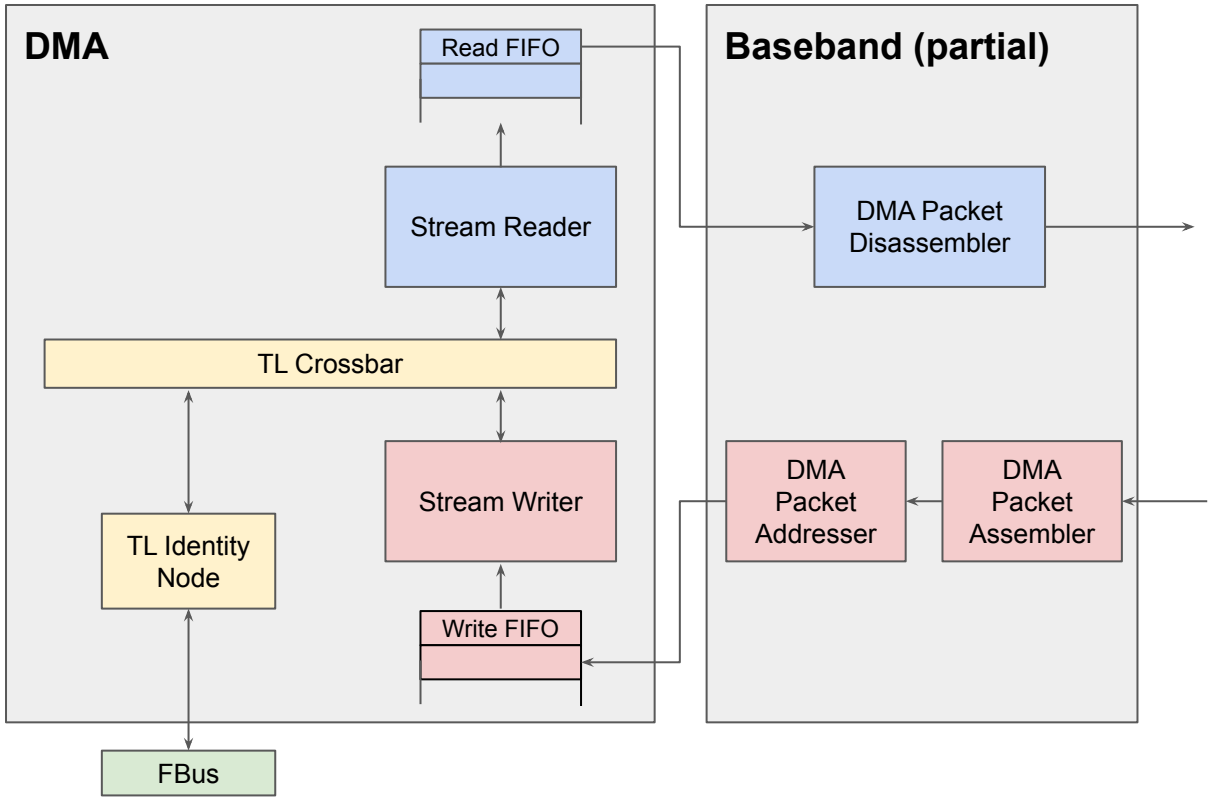
Verification

- Design tested at varying levels of complexity:
  - Block Level
  - Sub-module Level
  - SoC System Level
- Analog RF modeled in software and verilog model (courtesy of Eric and Daniel)
  - Design tested with various non-idealities, unstable modulation index, high SNR, etc.
- Classmate developed verification tool for chisel helped us adopt an agile development cycle, and quickly discover and fix bugs



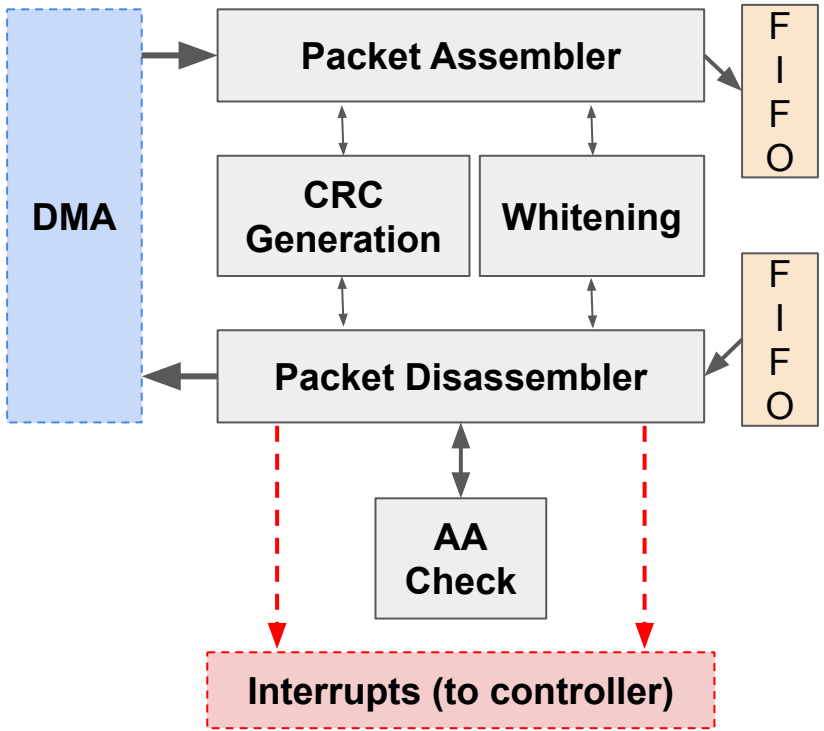
Design: DMA Interface

- 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

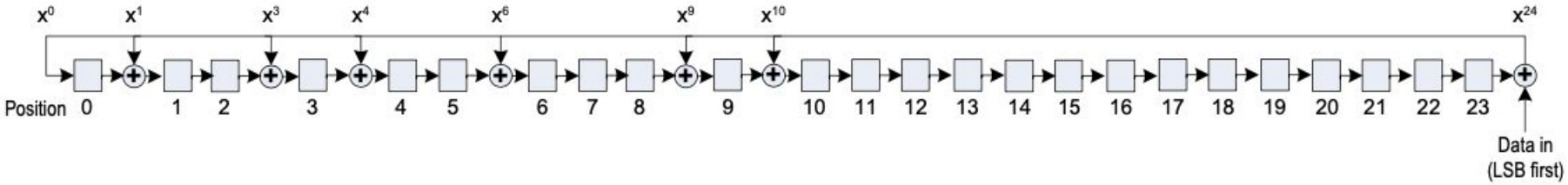
- Send Command:
  - Packet Assembler begins the bit-streaming process, pulling data from specified address via DMA.
- Receive Command:
  - Packet Disassembler begins to wait for preamble; will write to specified address via DMA.
- Debug Command:
  - Both TX and RX chains active, loopback enabled to allow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



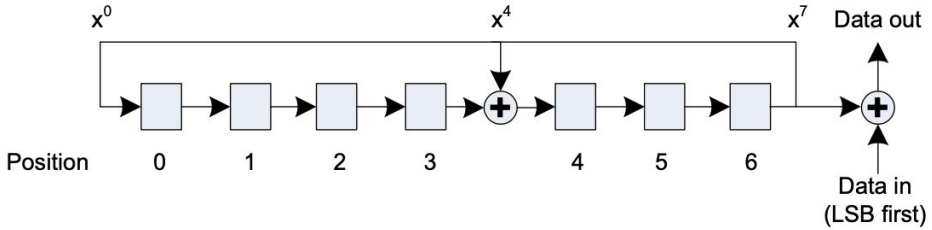
Design: Baseband

Bit Stream Processing

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

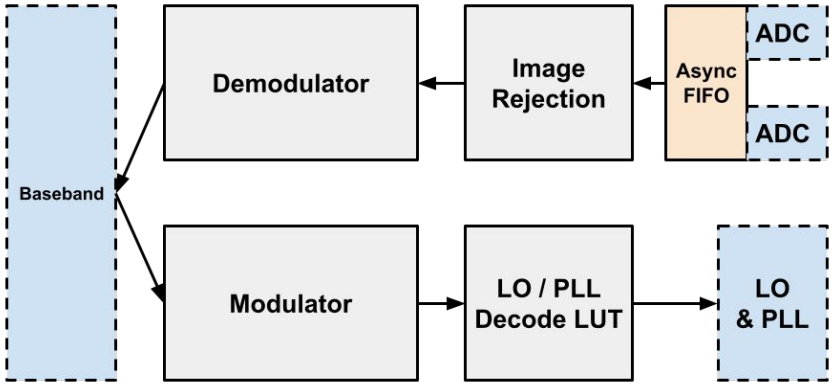


- Data Whitening: intended to avoid continuous streams of all 1s or all 0s, implemented as an LFSR



Design: Modem

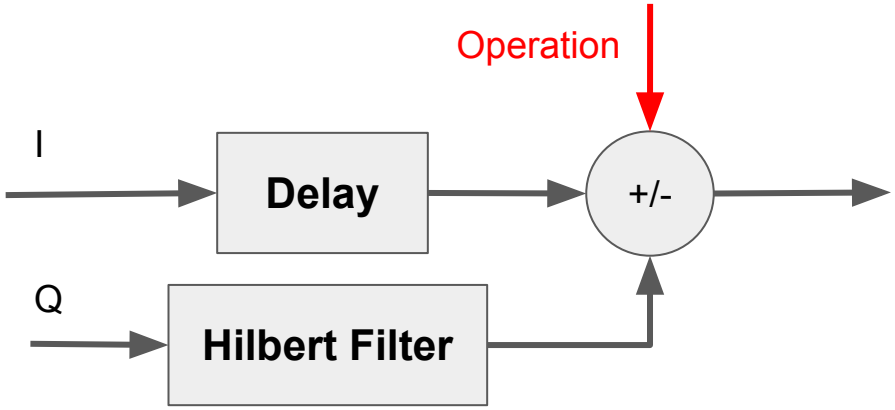
- Designed for GFSK
- TX side
  - Gaussian filter on bitstream input from baseband (with per bit over-sampling)
  - Resulting value mapped to LO / PLL control signals via programmable LUT
- RX side
  - Async FIFO accounts for non-sync ADC logic
  - AGC performed on FIFO output
  - Image rejection uses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



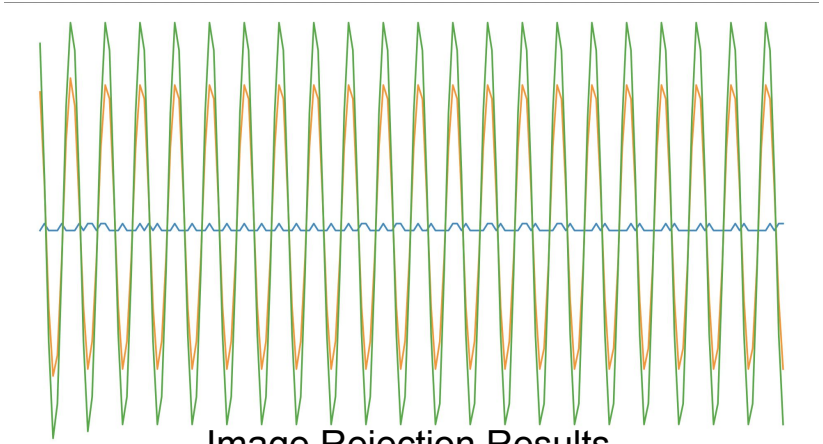
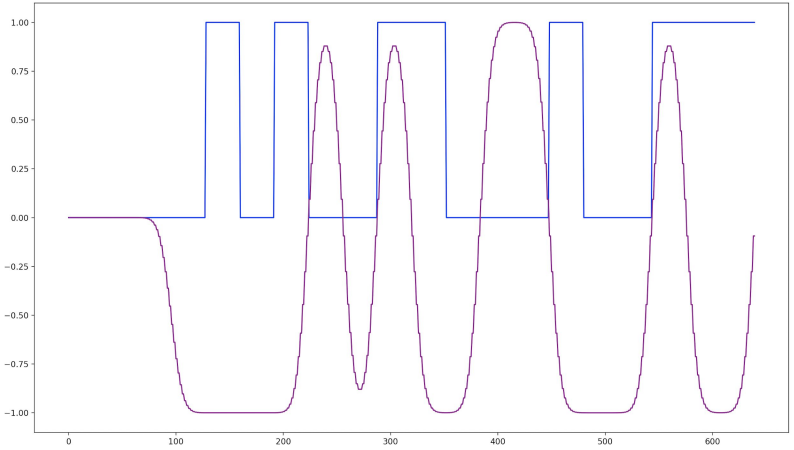


Image Rejection Results

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

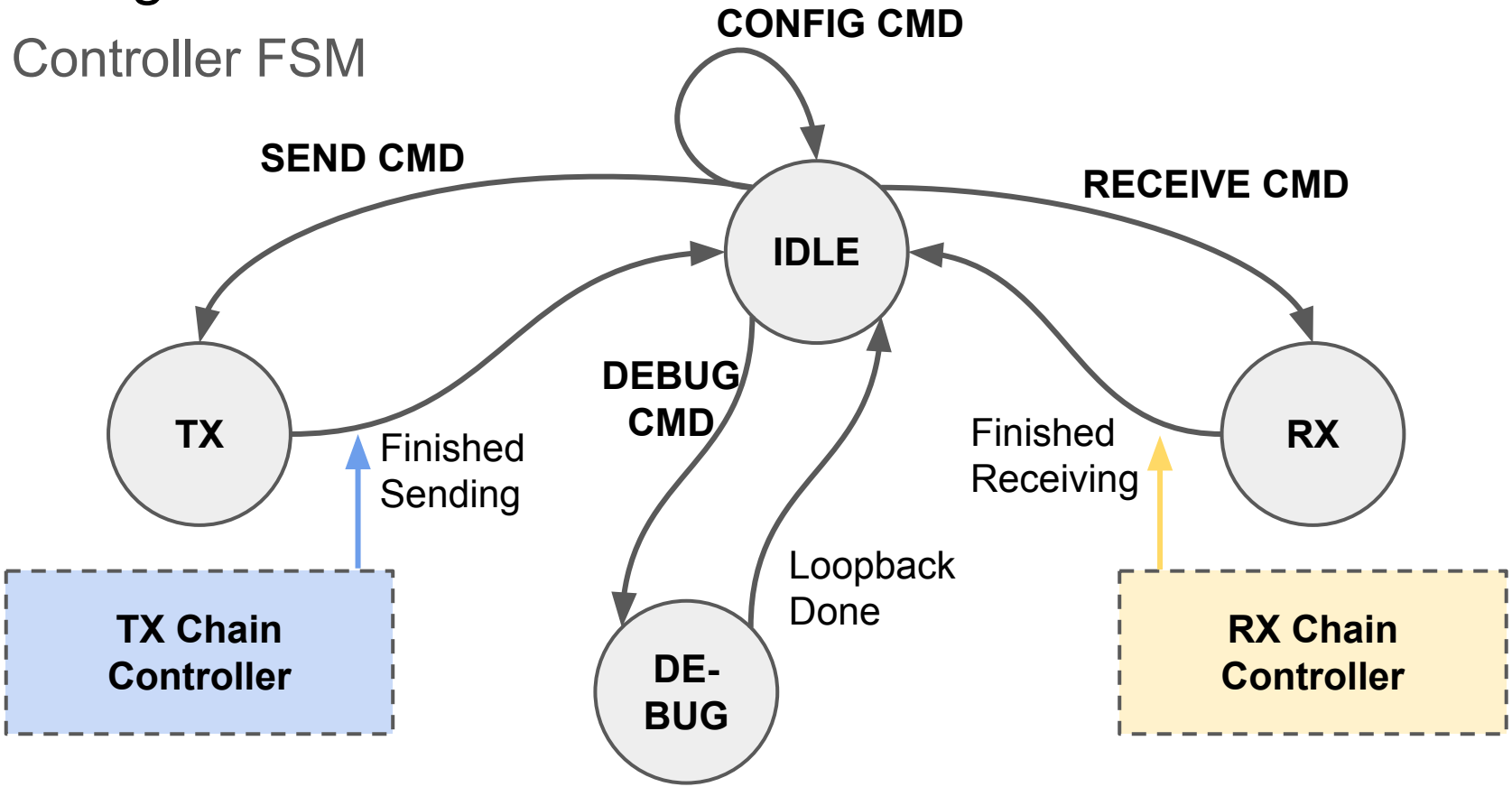
- Commands contain 4 fields
  - Inst. 1: Primary function
  - Inst. 2: Sub-function
  - Data: Small values
  - Additional data: Wide values
- Synchronizes DMA interface, baseband, and modem for command execution
  - Configures loopbacks for debug execution
- Manages interrupts and status registers

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

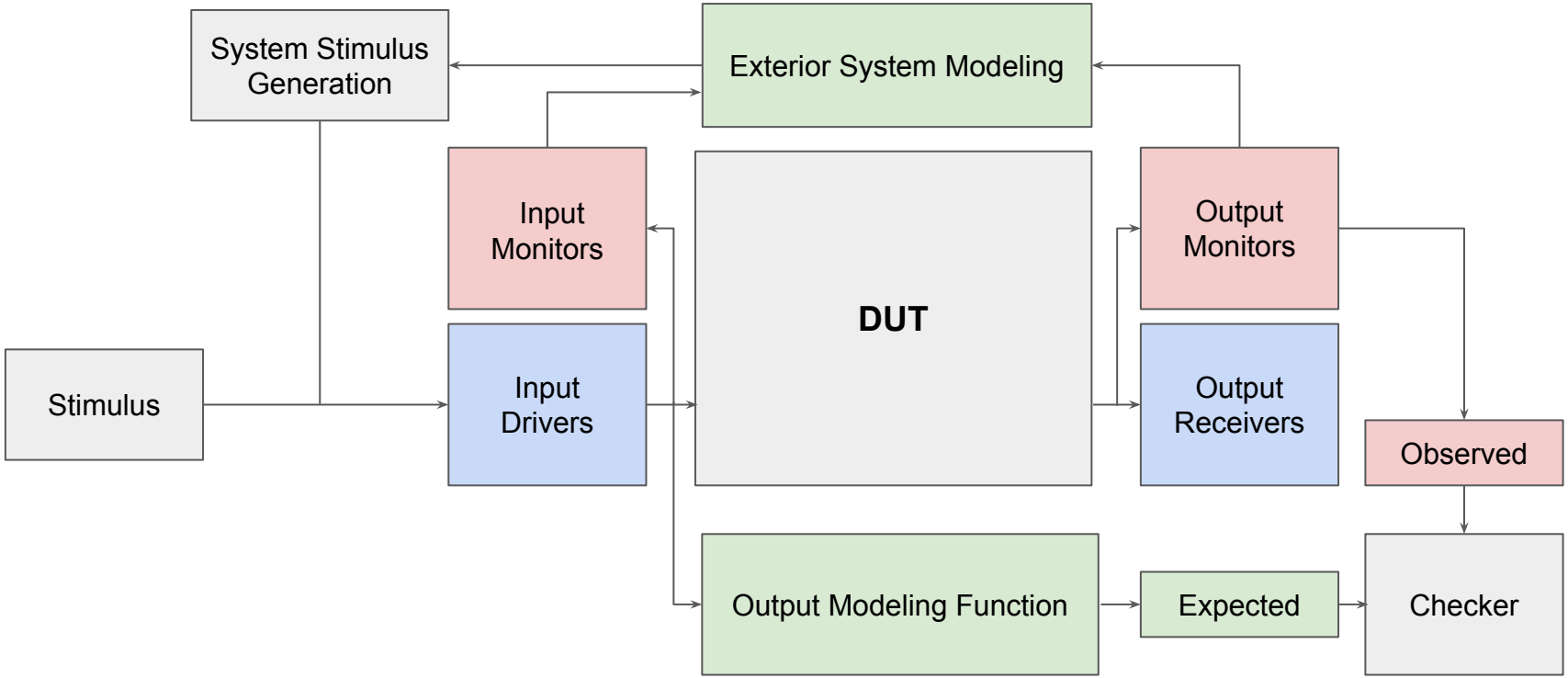
Bits	31-0
Field	Additional Data

Design: Controller

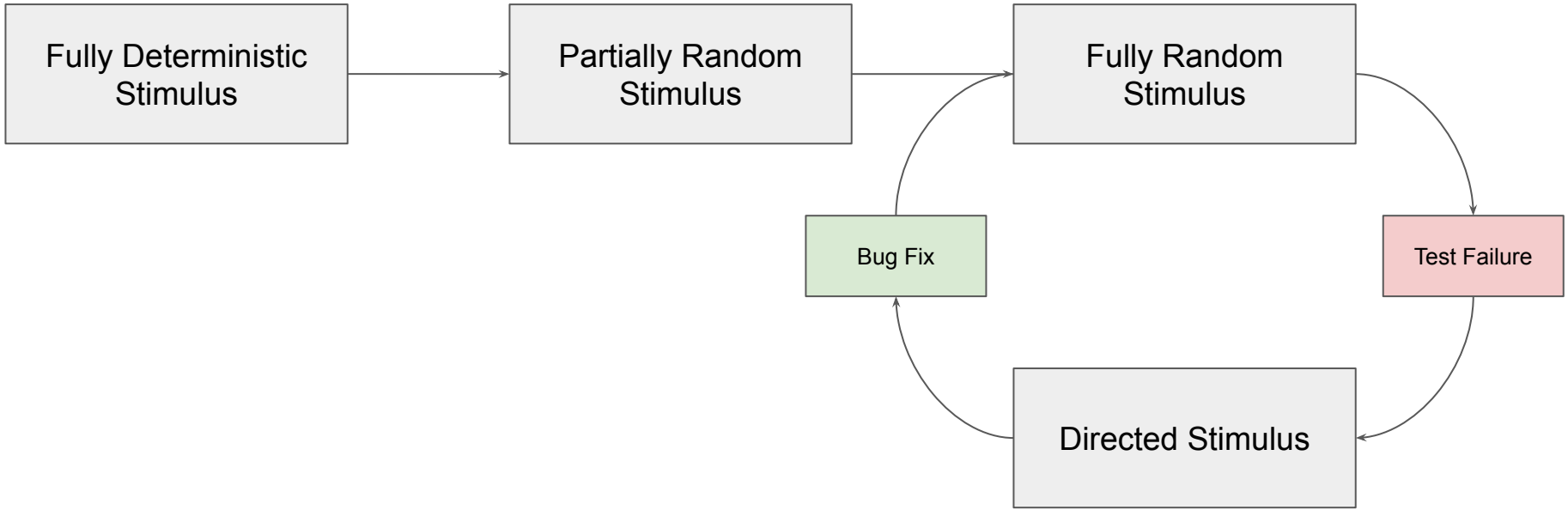
Controller FSM



Verification: Testbench Design



Verification: Stimulus Generation



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

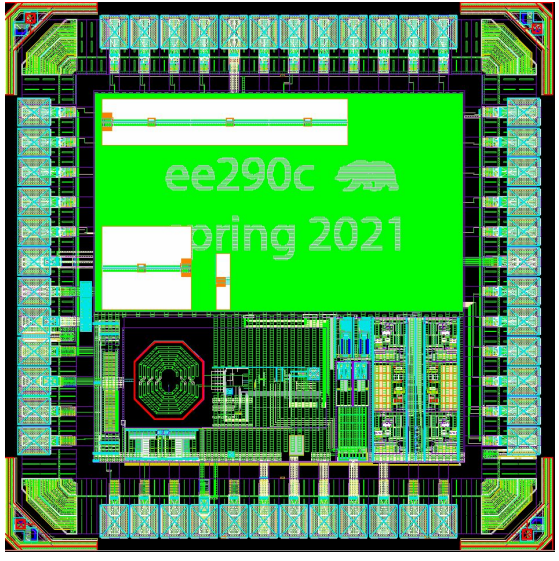
Top Level Summary (Planning)

- Hand floorplan for area/pad assignments
- Top level verilog
- Chip_generator.blend
  - Top-level verilog parsing
  - Block and subblock placement
  - Pin placement
  - Lef generation

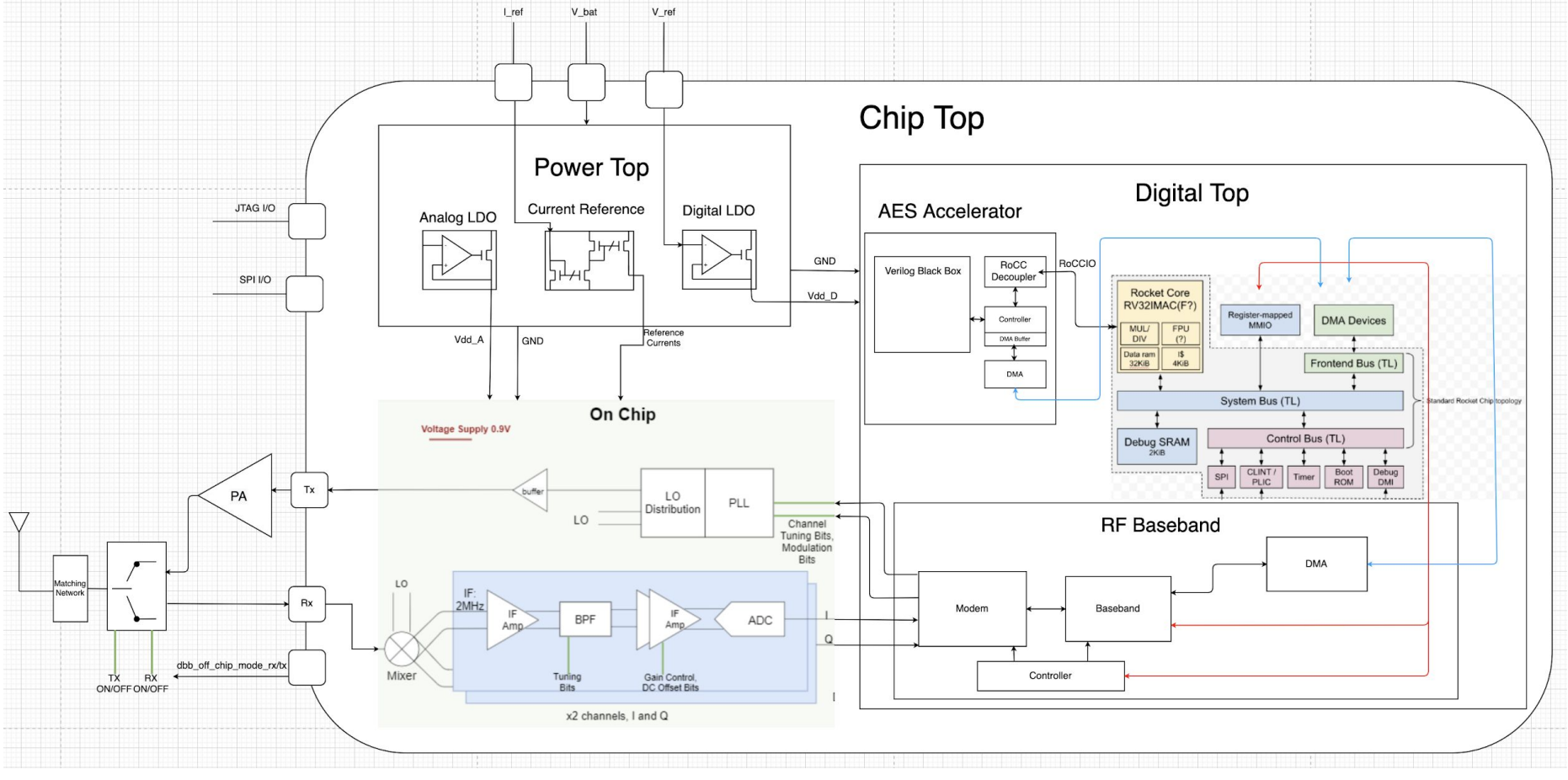
Top Level Summary (Layout)

- Core
  - Digital Top
  - Power Top
  - RF Top
  - Interblock wiring
- Chip
  - IO Ring
  - Core
  - Pad wiring

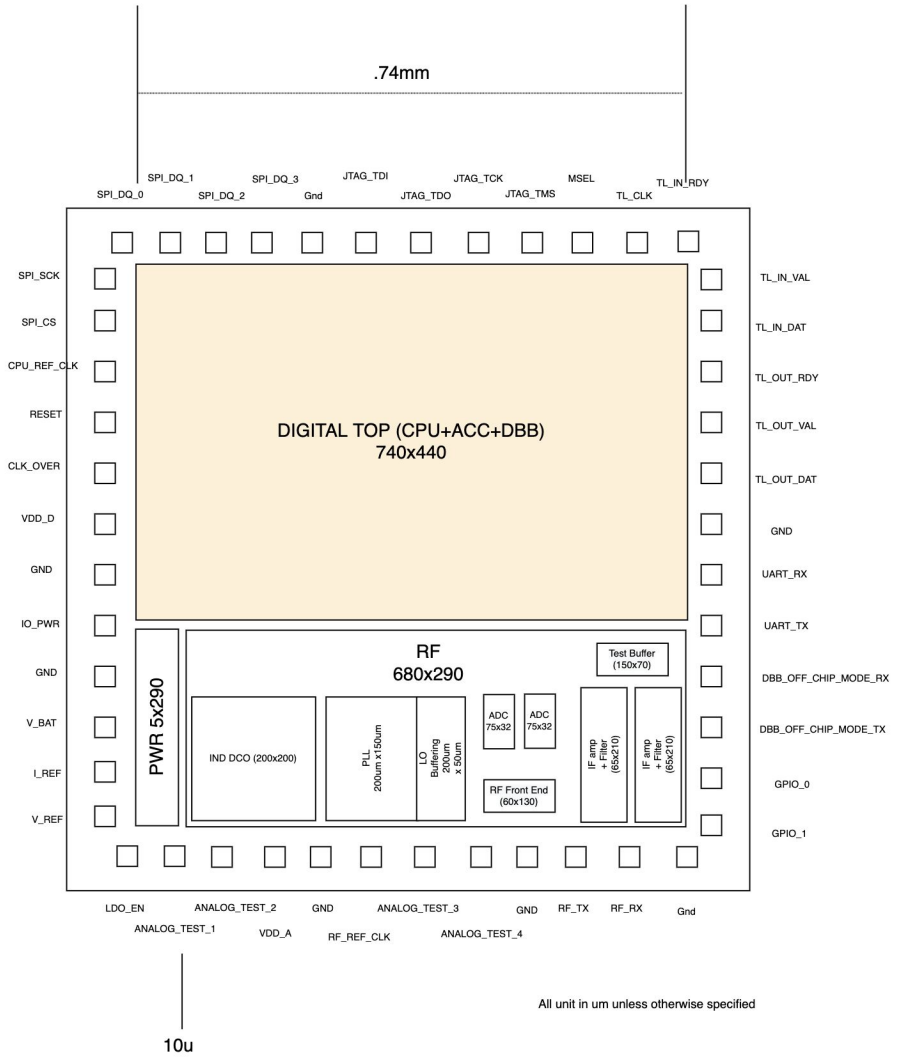
- Generate pin skeletons from LEFs
- Core-level wiring against skeletons
- IO Ring
- Chip-level wiring against IO Ring and Core-skeleton
- DRC, LVS



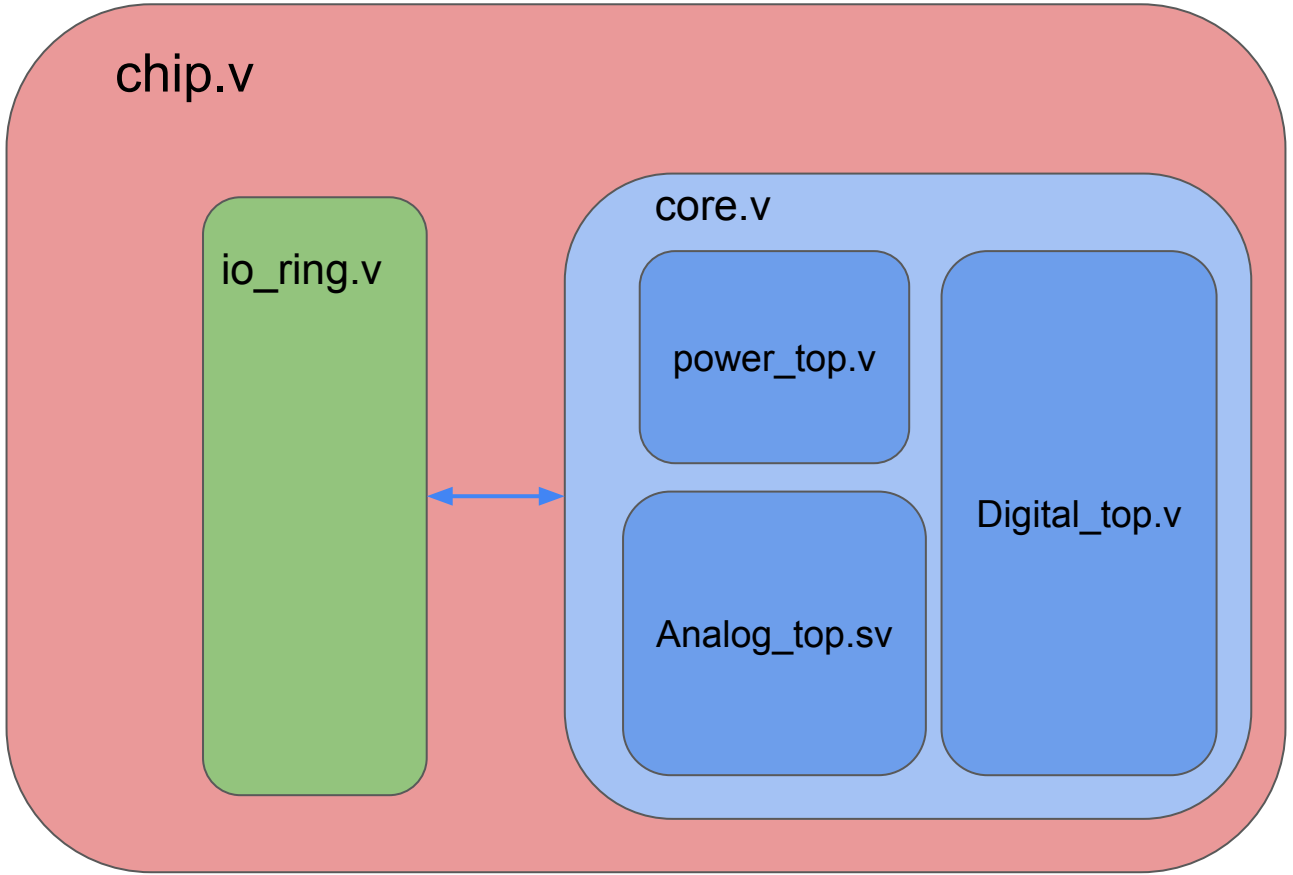
Top Level Diagram



Floorplan Sketch, Area Allocation



Top-Level Verilog



Top-Level Verilog

- Maintenance
  - 90% of the work for maintaining the top level verilog went into updating chip-level and block-level IOs
  - Lots of communication with radio and digital BB
  - Lots of meetings, spamming people on slack etc.
  - Used script in blender (more on this later) to verify syntax + connections
- Chip-Level IO spreadsheet
  - Full list of chip level IOs to be referenced/updated
  - Mappings of various baseband signals such as tuning trim

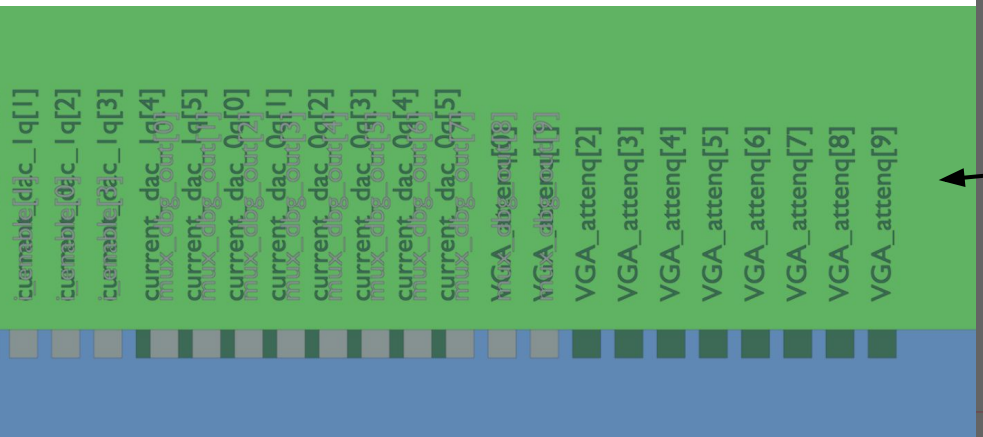
name	7	6	5	4	3	2	1	0
tuning_trim_bits_g0			rf_clk_sel	analog_test2_sel	pll_sign	ref_sel	cc_enable	lo_enable
tuning_trim_bits_g1			pll_idac[5]	pll_idac[4]	pll_idac[3]	pll_idac[2]	pll_idac[1]	pll_idac[0]
tuning_trim_bits_g2								
tuning_trim_bits_g3								
tuning_trim_bits_g4								
tuning_trim_bits_g5								
tuning_trim_bits_g6								
tuning_trim_bits_g7								test_en

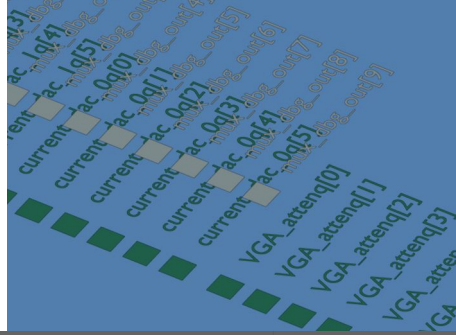
Manual Top Level Layout

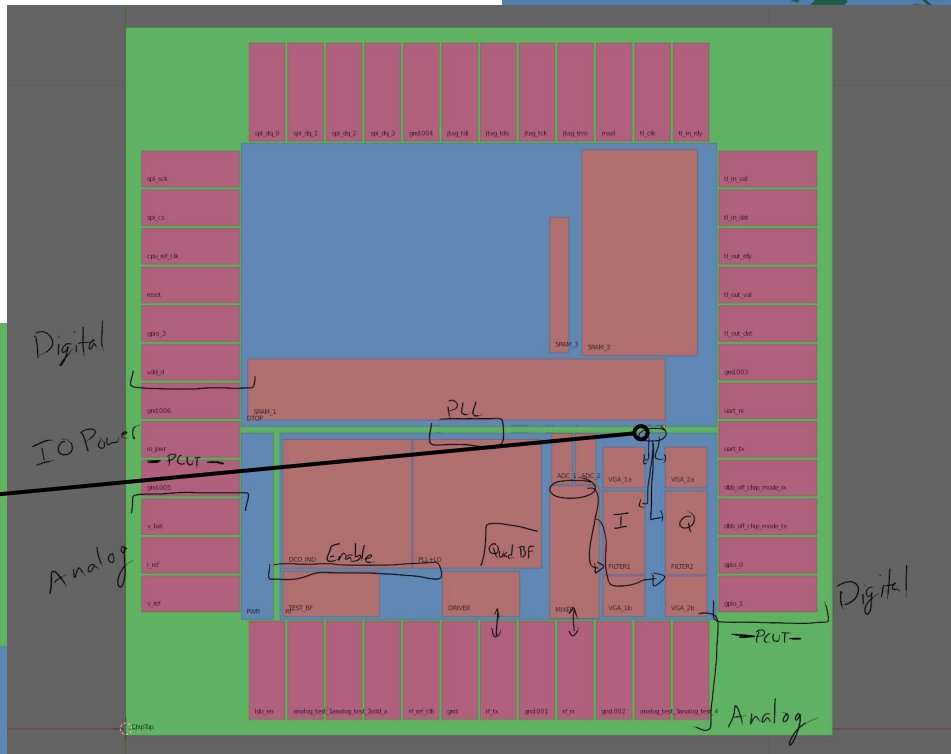
- 9 Metal Layers:
  - M8, M9 Power-grid
  - M6, M7 Typically for routing (flexible)
  - Vertical odds, horizontal evens
  - Sublocks raise local power grids to M7, vias dropped down from M8
- Layout planning starts early on and updates frequently
  - Full layout planning in chip_generator.blend
  - PR-Boundaries requested for rf-subblocks
  - Skeleton Layouts (with pin locations) provided for Power, RF, and Digital Top

Layout Planning (chip_generator.blend)

- Subblock placement for easy planning
- Uses actual units: ruler tool is useful
- Grease Pencil: easy annotations for revisions







Layout Planning (chip_generator.blend)

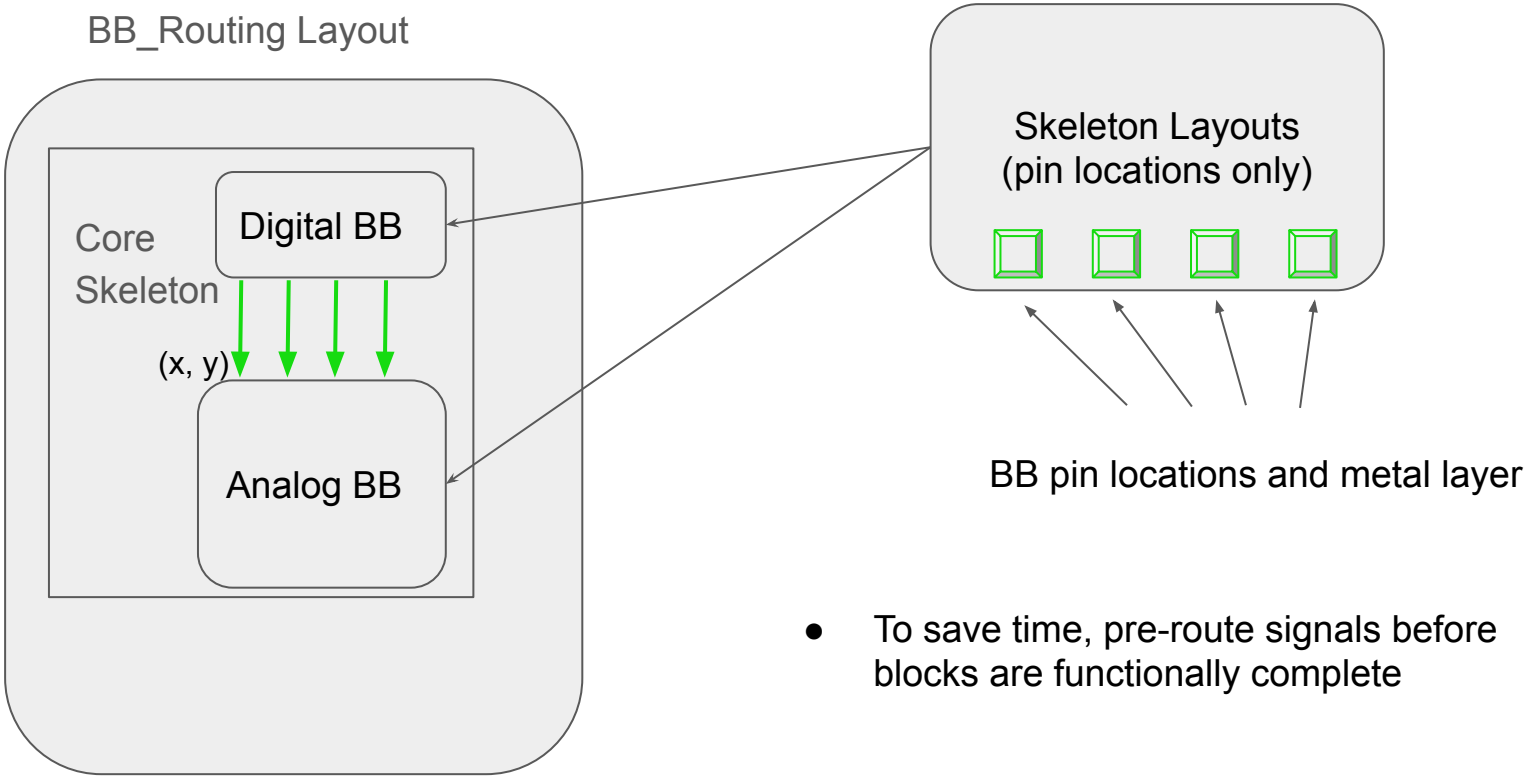
- Parses top-level verilog
  - Infers subblock io and pins
  - Basic verilog checking, warnings
- Manual subblock placement
- RegEx based pin placement
  - Manual placement of pad pins
  - Synchronized placement for easy top-level wiring
- Lef Generation

```
-------------- STARTING --------------
Parsing Verilog
WARNING: tried to connect to nonexistent object P_V_BAT. Inferring wire
WARNING: tried to connect to nonexistent object P_V_REF. Inferring wire
WARNING: tried to connect to nonexistent object P_I_REF. Inferring wire
WARNING: tried to connect to nonexistent object P_GND. Inferring wire
WARNING: tried to connect to nonexistent object P_VDD_A. Inferring wire
WARNING: tried to connect to nonexistent object P_VDD_D. Inferring wire
WARNING: tried to connect to nonexistent object P_LDO_EN. Inferring wire
WARNING: Module DTOP pin gpio_0_0_ie unconnected, SKIPPING
WARNING: Module DTOP pin gpio_0_1_ie unconnected, SKIPPING
WARNING: Module DTOP pin gpio_0_2_ie unconnected, SKIPPING
Checker Started
WARNING: In module (RF) expected pin (tuning_trim_g0) missing from connections
WARNING: In module (RF) expected pin (tuning_trim_g1) missing from connections
WARNING: In module (RF) expected pin (tuning_trim_g2) missing from connections
WARNING: In module (RF) expected pin (tuning_trim_g3) missing from connections
WARNING: In module (RF) expected pin (tuning_trim_g4) missing from connections
```

```
##
## LEF for DTOP ;
## created by chip_generator.blend on 2021-04-26
##

VERSION 5.8 ;

BUSBITCHARS "[]" ;
DIVIDERCHAR "/" ;

MACRO DTOP
  CLASS BLOCK ;
  SIZE 740.0 BY 440.0 ;
  FOREIGN DTOP 0.000000 0.000000 ;
  ORIGIN 0 0 ;
  SYMMETRY X Y R90 ;
  PIN serial_tl_bits_in_valid
    DIRECTION INPUT ;
    USE SIGNAL ;
    PORT
      LAYER M7 ;
        RECT 739.8 400.0 740.0 400.2 ;
    END
  END serial_tl_bits_in_valid
```

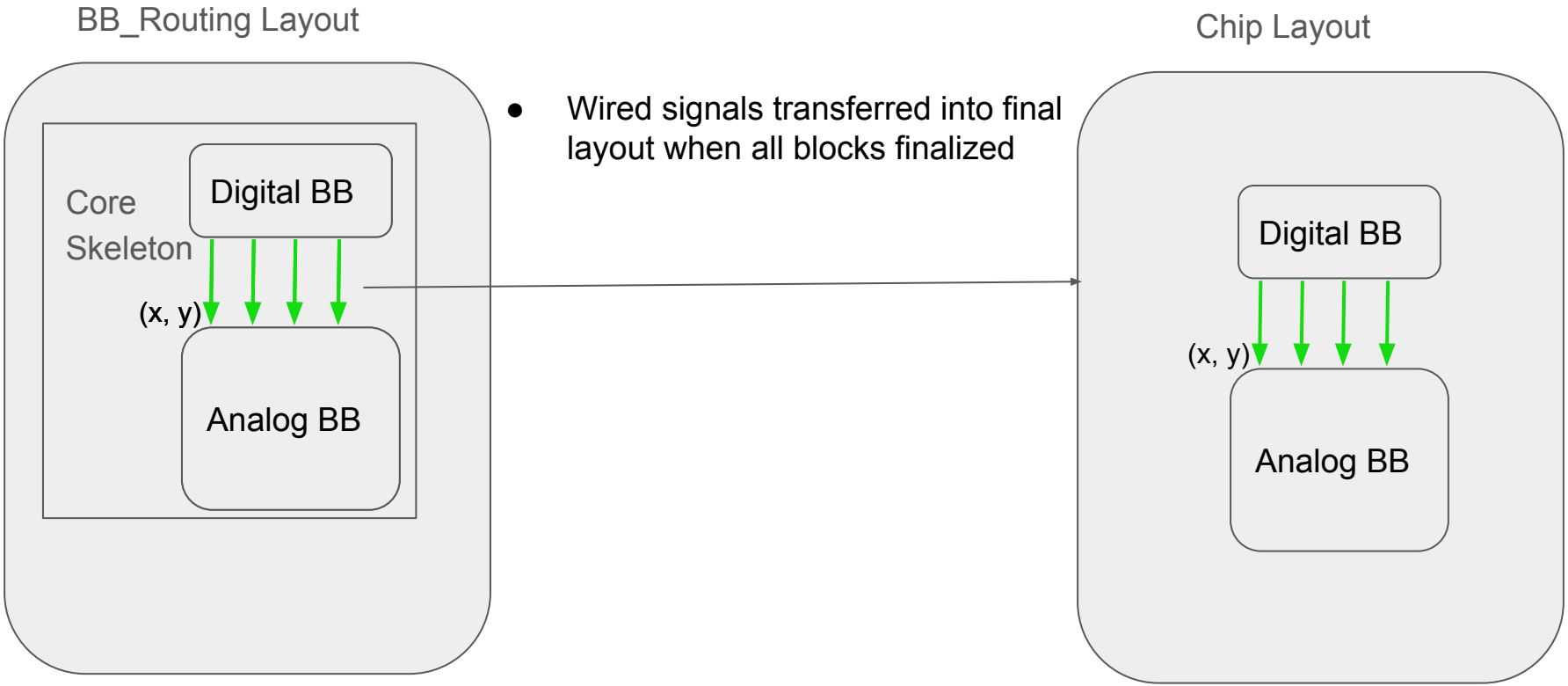
```
# Block Placement
DTOP.place(DTOP_LOC, DTOP_DIM)
RF.place(RF_LOC, RF_DIM)
root = (RF, DTOP, rt_rf, rt_dt)

# Synchronized RegEx Signal Placement (On M5)
classes.place_sync_v2(*root, (.5706, .000), ["mux_dbg_in.*"]     , wires, PIN_SIZE, .0003, 5, reverse=True)
classes.place_sync_v2(*root, (.5738, .000), ["vga_atteni.*"]     , wires, PIN_SIZE, .0003, 5, reverse=True)
classes.place_sync_v2(*root, (.5769, .000), ["current_dac_\di.*"], wires, PIN_SIZE, .0003, 5, reverse=True)
classes.place_sync_v2(*root, (.5806, .000), ["f_r\di.*"]         , wires, PIN_SIZE, .0003, 5, reverse=True)
classes.place_sync_v2(*root, (.5940, .000), ["mux_dbg_out.*"]    , wires, PIN_SIZE, .0003, 5, reverse=True)

# Manual Signal Placement (On M7)
i_enable, q_enable = data.objects["i_enable"], data.objects["q_enable"]
i_enables = [i_enable[i] for i in (4,1,0,3)]
q_enables = [q_enable[i] for i in (4,0,1,3)]
RF.place_pins_connected((.5927, RF.top(PIN_SIZE)), PIN_SIZE, .0003, 7, i_enables)
DTOP.place_pins_connected((rt_dt[0]+ .5927, .000), PIN_SIZE, .0003, 7, i_enables)

# Warnings
print("UNPLACED PINS:")
print(DTOP.unplaced_pins)

# Lef Generation
DTOP.write_lef(pathlib.os.path.join(pathlib.Path.home(),"DTOP.lef"))
```

Top-Level Wiring of Unfinished Blocks



- To save time, pre-route signals before blocks are functionally complete

Top-Level Wiring (continued)



- Wired signals transferred into final layout when all blocks finalized

IO Ring

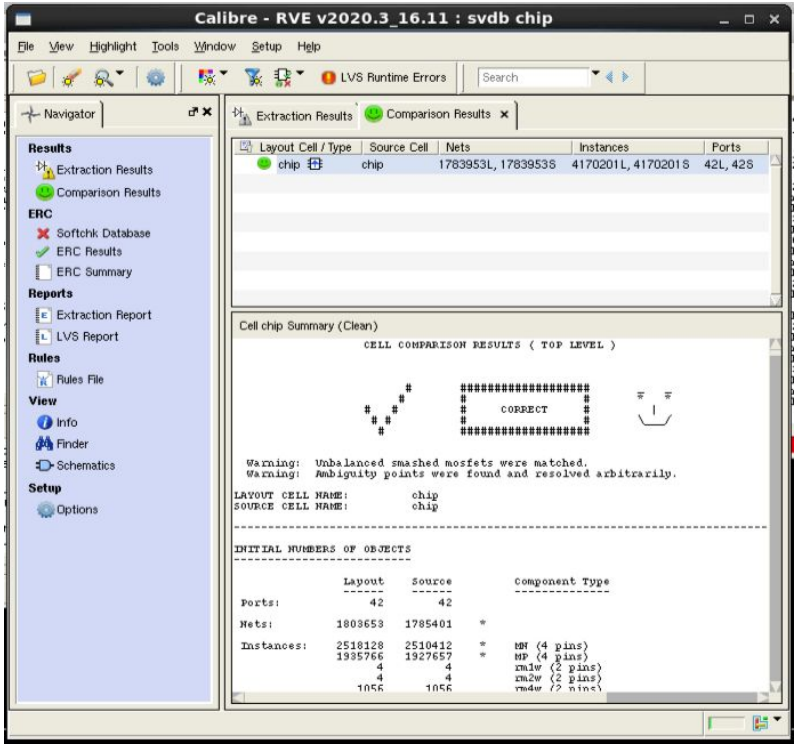
- 2 Domains:
  - Analog (South, Lower West)
  - Digital (North, East, Upper West)
- 1 IO Power Pad:
  - Manually bridged power-cut cells
- ESD Clamp External to ring

DRC and LVS

- Subblock DRC and LVS runs
  - Fix pins, then ports
- Blackbox LVS at Core and Chip levels
  - Core, Chip Verilog
  - Cadence schematics extracted to spice
  - Technology spice files
- Full DRC, LVS

Problems

- Case insensitivity CLAMP == clamp
  - Manual pin renaming
- GPIO[1] != GPIO<1>
  - Export to GDS with replace <> with [] option
- Incorrect netlist extraction
  - Manual fixes



Reflections

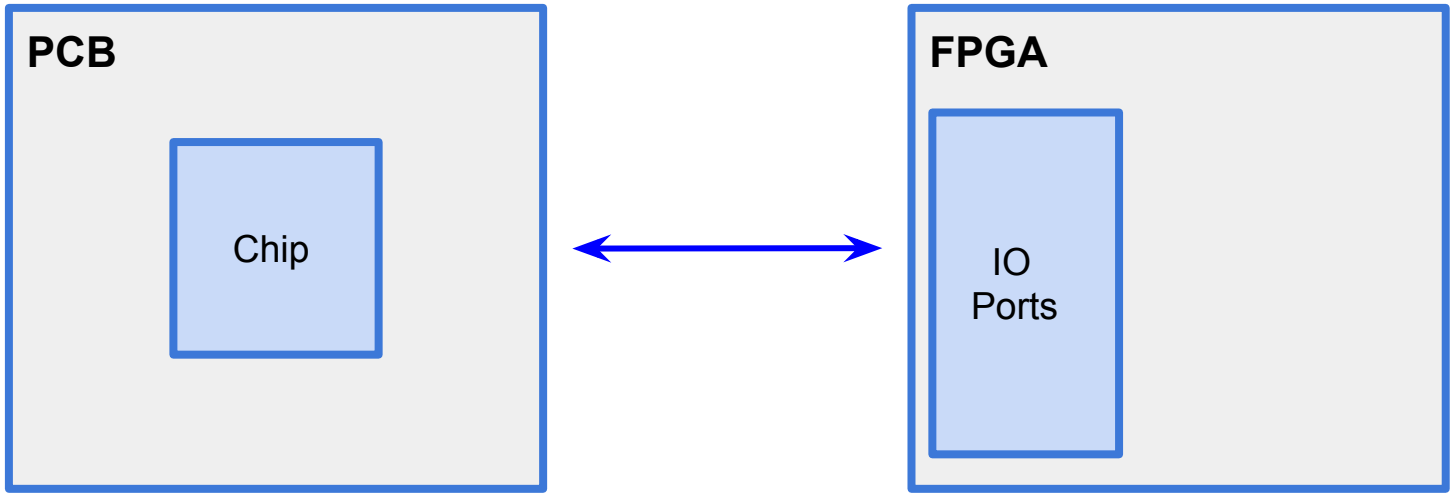
- Freeze pin assignments early, provide a number of spare bits
- Working in parallel rather than waiting on others' outputs in general
  - LVS/DRC as pieces complete rather than at the end
- Open communication channels early, find a point of contact for each piece
- Learned the full flow of how an SoC is created
  - Connection between analog and digital circuits
  - Layout (Cadence Virtuoso, routing, the many sources of DRC errors etc.)
  - Importance of planning and documentation
- Learned the dynamics of being part of a large technical project
  - How to better communicate with different sub-teams
  - How to organize myself within my own subteam

Advice?

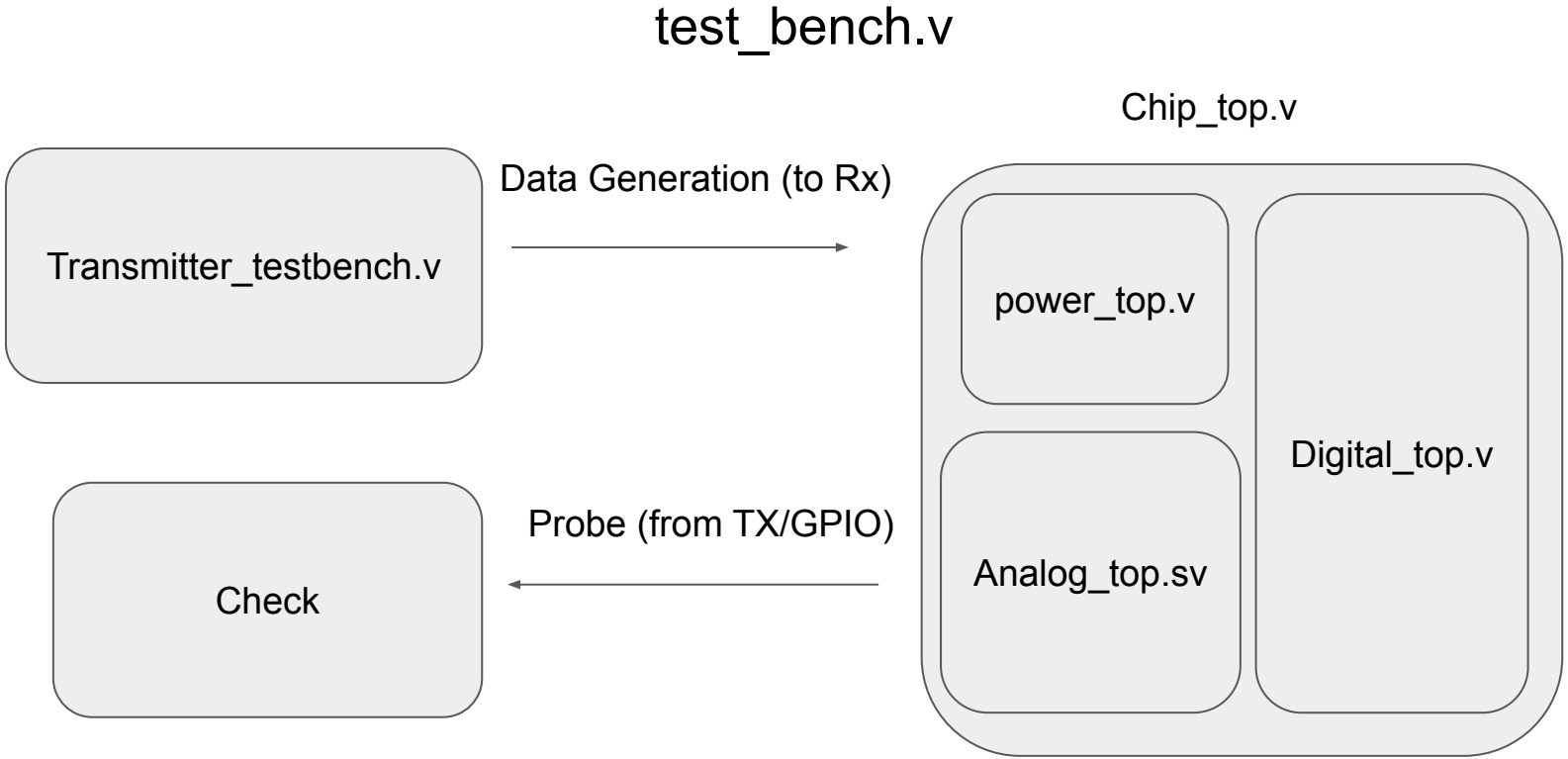
- For integration (and probably everyone in the course), learn how to layout as early as possible!!!
-

What’s Next?

- Custom PCB design using Altium over summer
- Testing to follow
  - JTAG / UART to PC for digital control
  - Signal generators, analyzers, and scopes for analog I/O

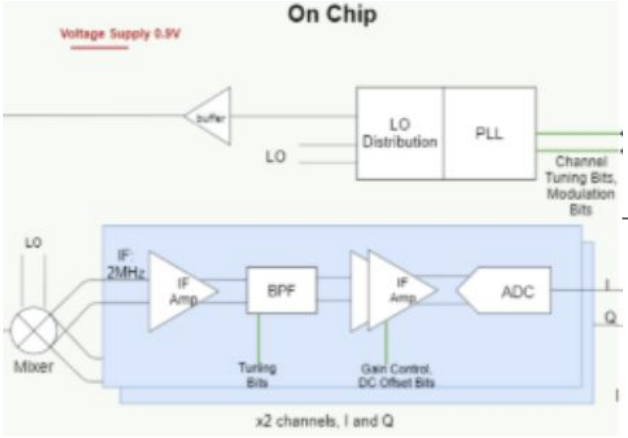


Verification Methodology



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

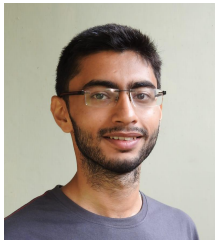
End Goals:

- 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

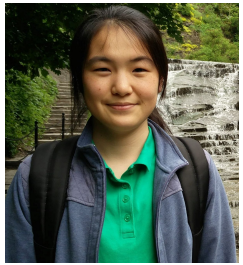


Shreesha S
Ph.D. Student

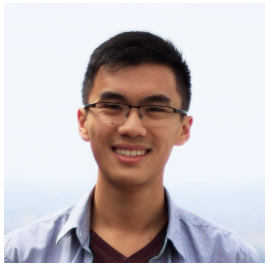
Kerry Yu
4th Year Undergrad

Leon Wu
4th Year Undergrad

Sherwin
4th Year Undergrad

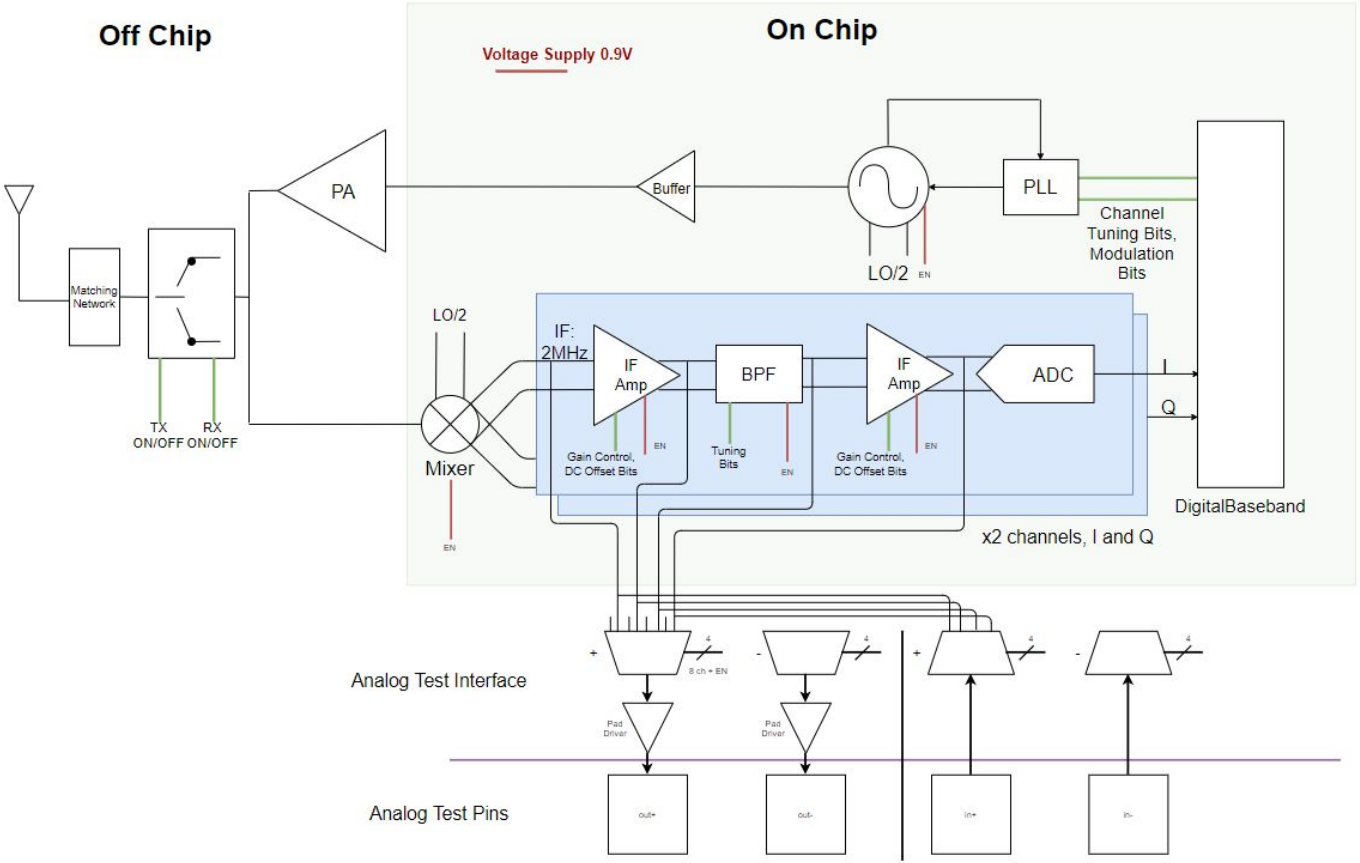


Felicia Guo
Ph.D Student



Jeffrey Ni
4th Year Undergrad

Overview


Off Chip
On Chip
Voltage Supply 0.9V
PA
Buffer
PLL
Channel Tuning Bits, Modulation Bits
LO/2
EN
Matching Network
TX ON/OFF
RX ON/OFF
LO/2
IF: 2MHz
IF Amp
BPF
IF Amp
ADC
I
Q
Gain Control, DC Offset Bits
Tuning Bits
Gain Control, DC Offset Bits
Mixer
EN
DigitalBaseband
x2 channels, I and Q
Analog Test Interface
Analog Test Pins
Pad Driver
Pad Driver
out+
out-
in+
in-

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

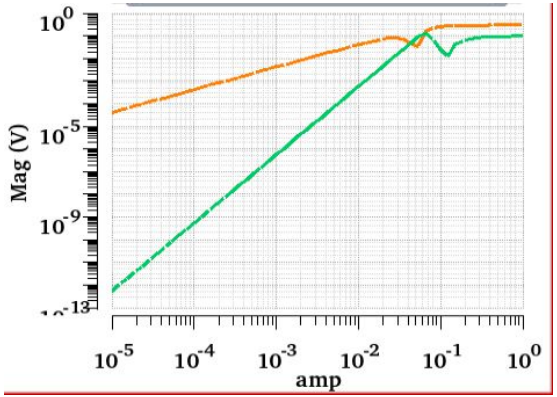
- Symbol period: 1us
- Gaussian Mask - BT=0.5
- Modulation index between 0.45 ~ 0.55
  - 0.495~0.505 for ‘stable modulation’
  - Target 0 = fc-250kHz; 1 = fc+250kHz
- Max center freq. drift: 150kHz
- Maximum freq drift: 50kHz
- Maximum drift rate: 400Hz/us
- Transmission power met using off chip PA

Receiver Link Budget + Modeling

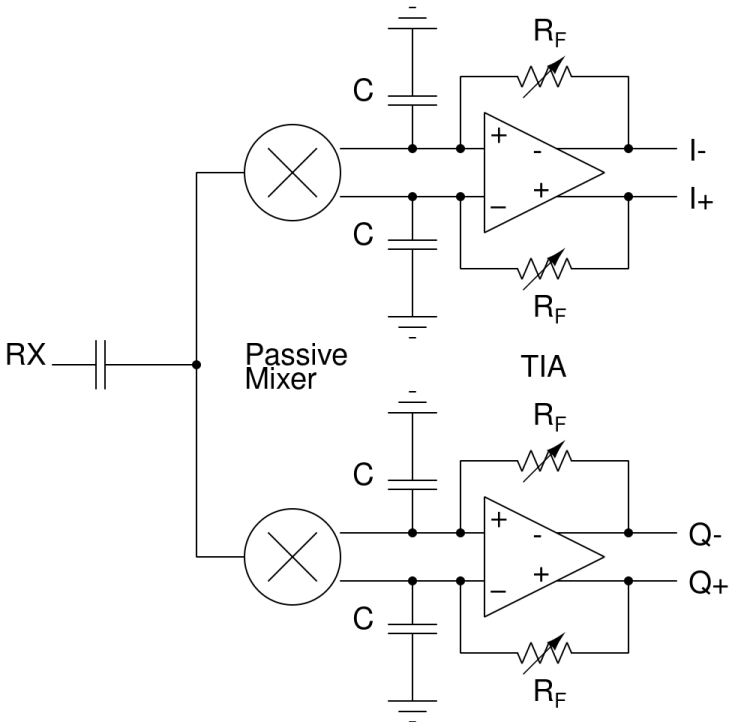
	Rin (ohms)	Rout (ohms)	Gain (dB, 20log)	Max_in (V)	Output Noise (V integrated)	Total Gain (dB)	Total SNR (dB)	input voltage	signal amp
Antenna		50.00	0		-	-6.021			
Matching Network	50.00	225.00	0		-	-12.041			**2.50E-04**
Mixer	225.00	400.00	30	0.007	2.14E-04	17.959	17.919	7.00E-03	7.91E-03
Buffer	1.00E+99	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

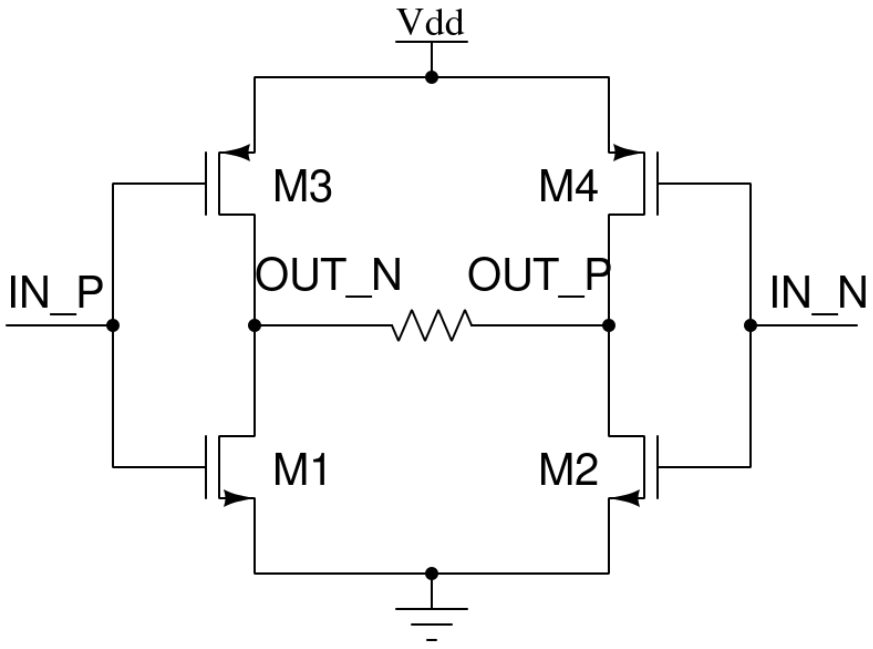
- Modelling also in virtuoso
  - Combination of verilogA and discrete ideal components
  - Similar to link budget in variables that can be adjusted

```
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
```



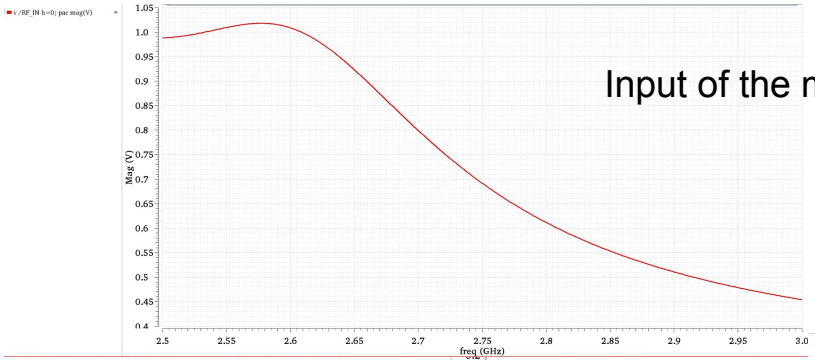
Mixer

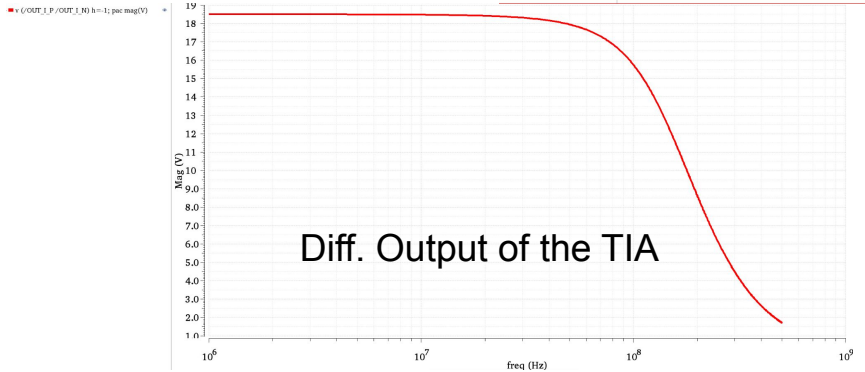


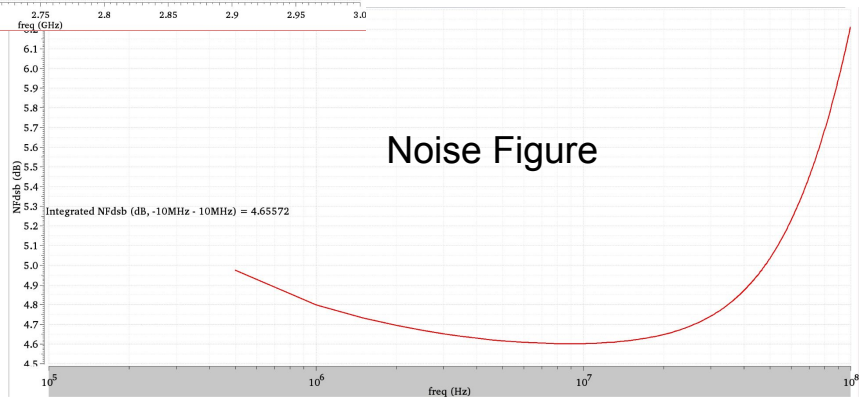


TIA circuit diagram

Schematic Simulation Results





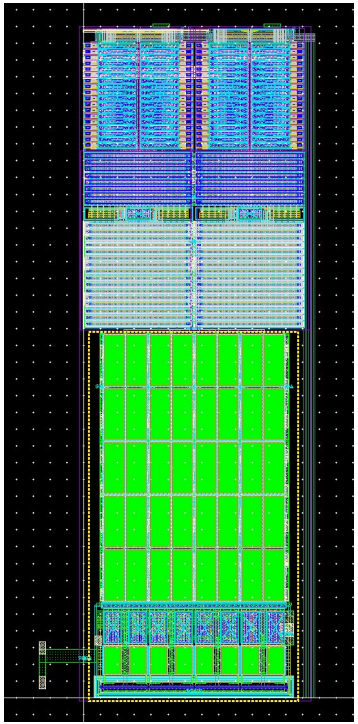


Schematic Simulation Results of the Mixer

	TT	SS	FF	SF	FS	SS HOT	FF COLD
Gain (V/V dB)	18.51	18.16	18.1	18.39	18.71	22.08	13.61
NF (dB)	4.655	5.155	4.305	4.68	4.619	4.923	4.615
S11 (dB)	-38.64	-20.82	-18.79	-47.95	-28.87	-32.76	-25.68
P1dB (mV)	~40	~45	~40	~45	~40	~40	~60
IIP3 (dBm) (in-band)	2.006	3.206	2.702	2.729	1.747	3.172	4.128
Power (uW)	711	495	1045	722	736	915	421

Layout of the mixer

Dimensions: 70 μm * 200 μm

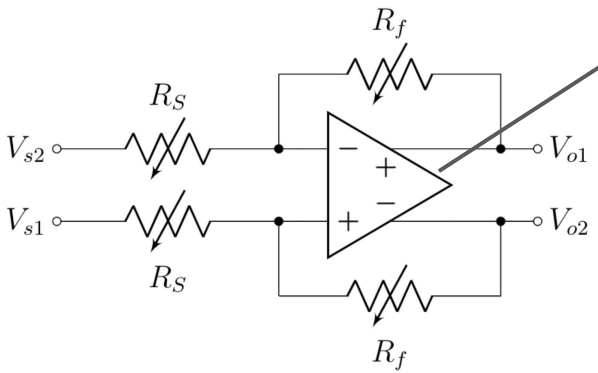


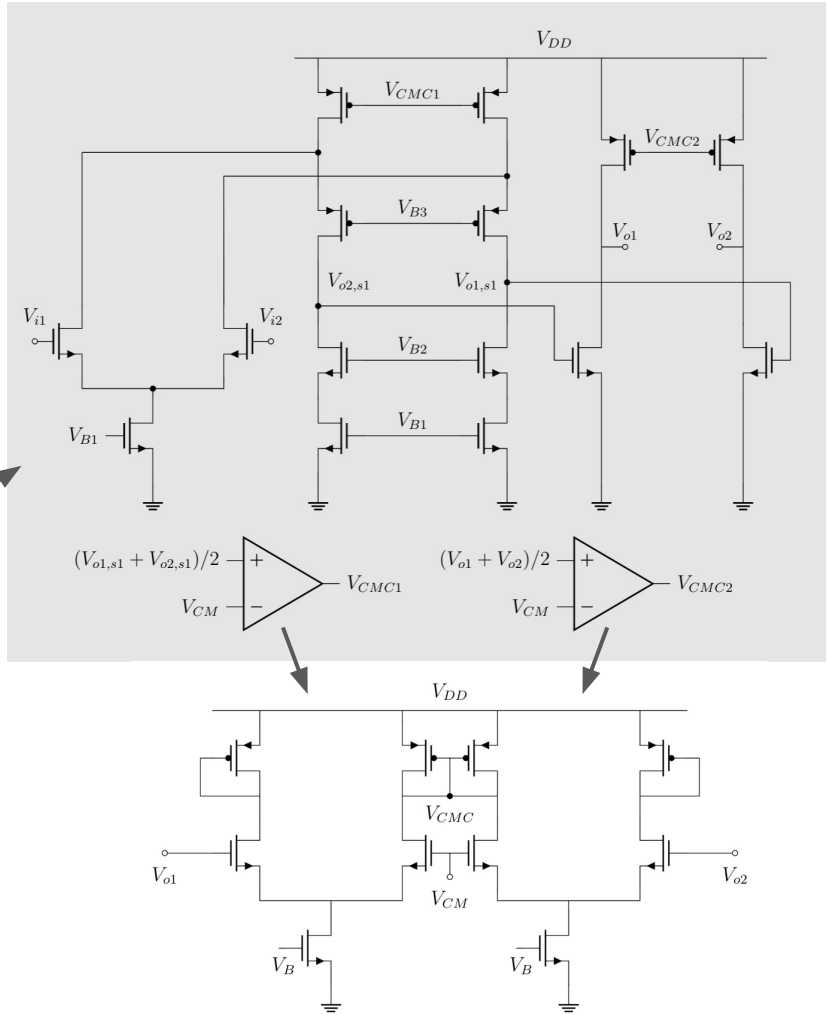
C+CC Simulation Results of the Mixer

	TT	SS	FF	SF	FS	SS HOT	FF COLD
Gain (V/V dB)	19.33	19.22	18.7	19.18	19.56	22.86	14.58
NF (dB)	4.52	4.963	4.22	4.55	4.48	4.82	4.382
S11 (dB)	-27.33	-25.04	-17.14	-31.7	-23.48	-26.56	-39.17
P1dB (mV)	~45	~45	~45	~45	~45	~40	~65
IIP3 (dBm) (in-band)	3.325	6.009	6.973	7.411	4.443	6.45	5.84
Power (uW)	804	565	1165	810	830	1022	489

VGA

- Initial design
  - 0-30 dB gain range for each VGA
  - R_S steps in 6 dB, R_f steps in 2 dB
  - BW(30 dB) = 7 MHz with C_L = 300fF/2.4pF
  - $R_{S,min}$ = 5 kohm, $R_{f,min}$ = 80 kohm
  - 0.6V I/O common mode (use resistive divider to generate VCM for all baseband blocks)





Two-stage Amplifier Design

Monte Carlo @ 30dB gain:

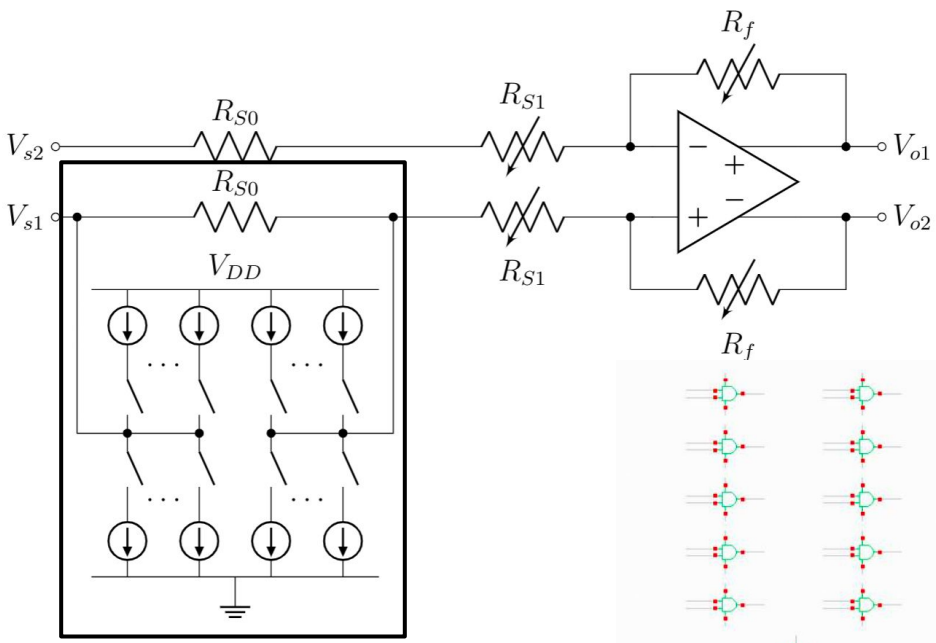
	mean	std dev	max	min
Equivalent input offset (mV)	0.006	**1.25**	3.58	-3.47
PSRR (dB)	60.6	9.69	110	**46.2**
CMRR (dB)	82.0	11.1	130	**56.6**

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

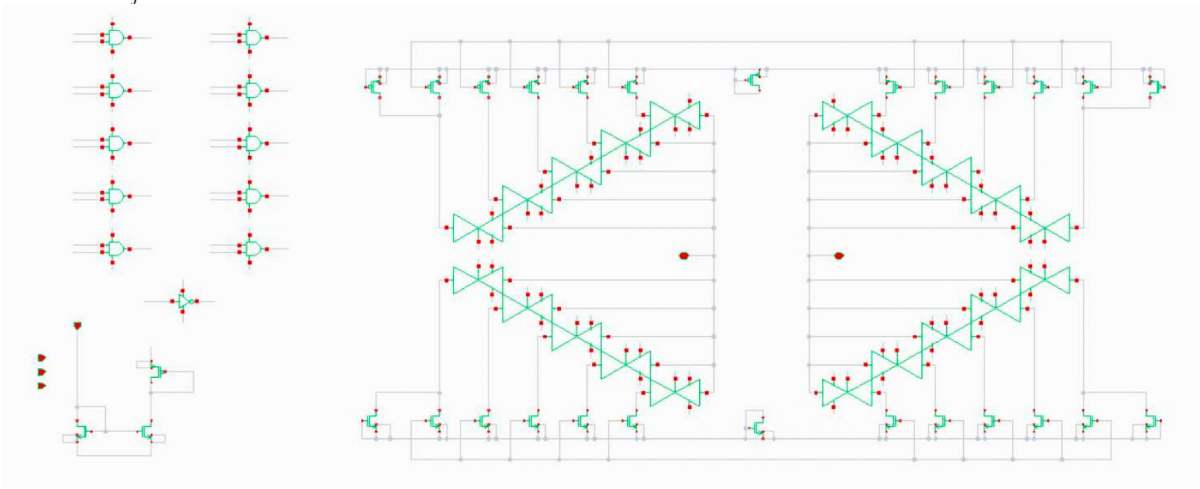
- Fully differential amplifier design
  - $V_{i,CM} = V_{o,CM} = 0.6V$
  - A = 98 dB, BW = 3.15 kHz
  - Current consumption = 270uA
  - Miller compensation, PM = 57
  - Input swing 0.4-0.9 V; output swing 0.15-0.8 V
- Noise @ 30dB gain
  - Input spot(2MHz) = 15 nV/√Hz
  - Integrated output noise (1-3 MHz) = 678 uV
- Linearity
  - (V_{s1}-V_{s2}) 1dB-compression = 17mV @ 29.5dB gain

DC Offset Cancellation



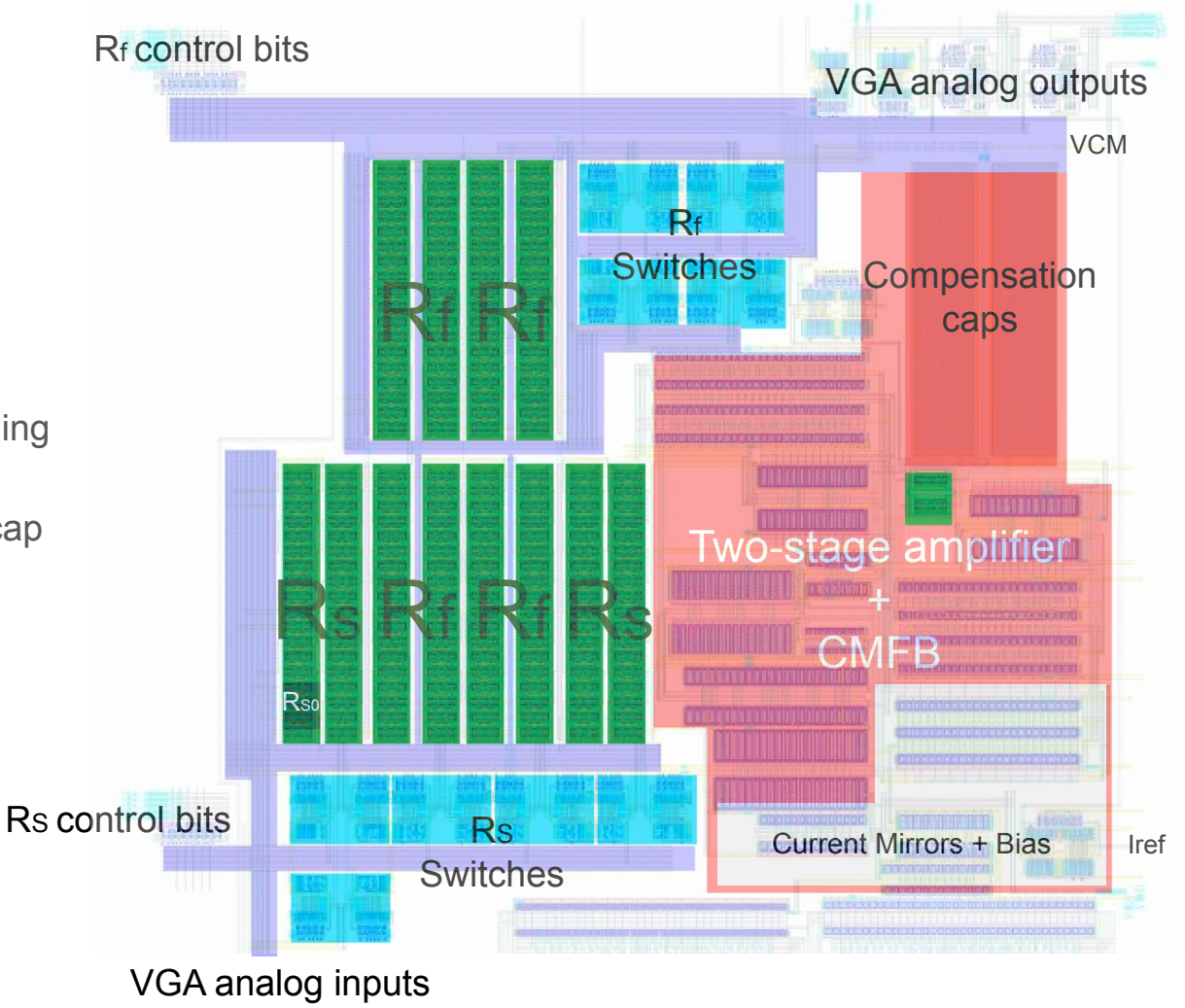
Current DAC

- Use current DACs and R_{s0} to create a offset cancellation voltage V_{dac}
- 6-bit control (5 bits for each polarity)
- $|V_{dac,max}|$ = 6.02 mV
- $|V_{dac,min}|$ = 195 uV; V_{dac} increase linearly with the code
- Move all the tunability of R_s to R_{s1}



VGA Layout

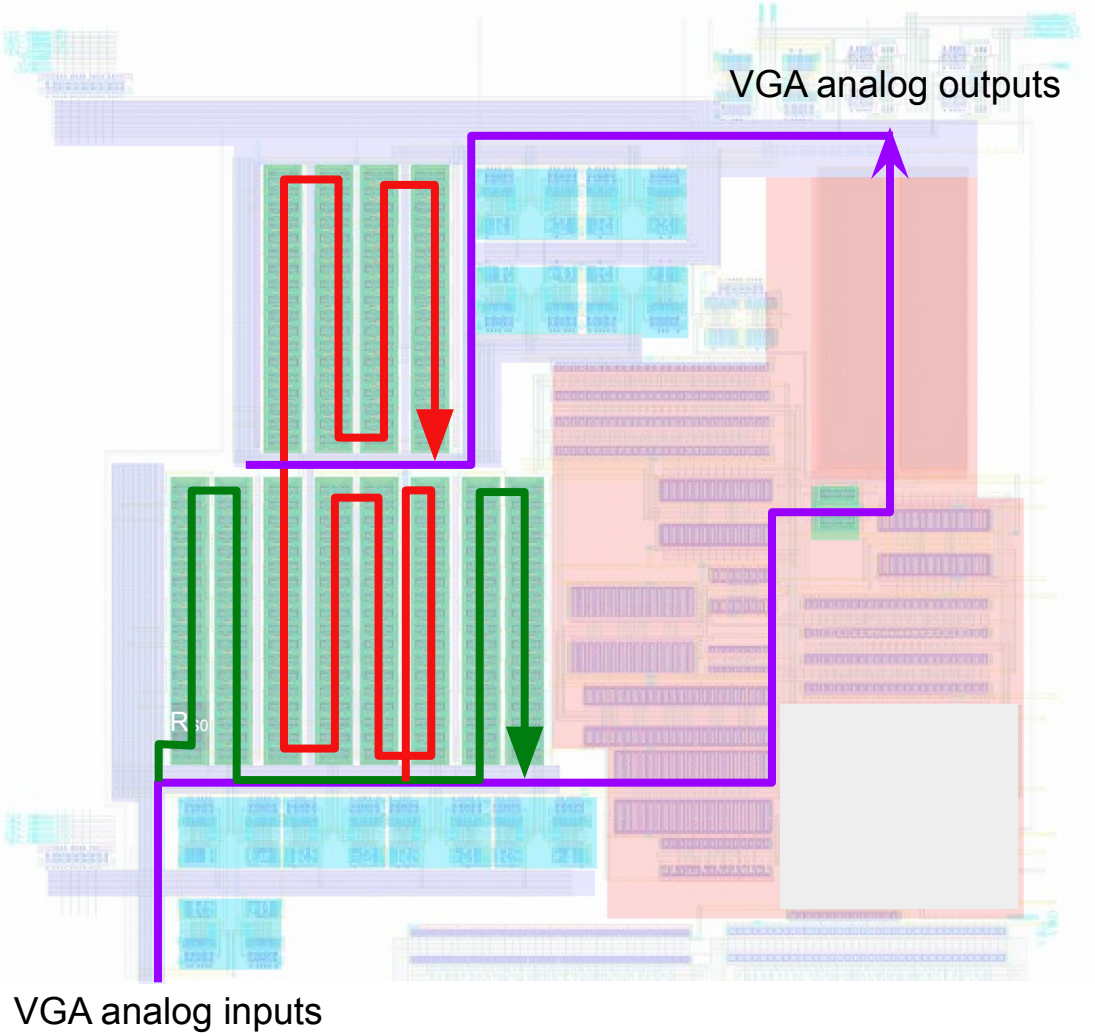
- 2 blocks of R_f in between 2 blocks of R_S for better matching at unity gain.
- Modified the compensation cap based on extraction



VGA Layout & Extraction

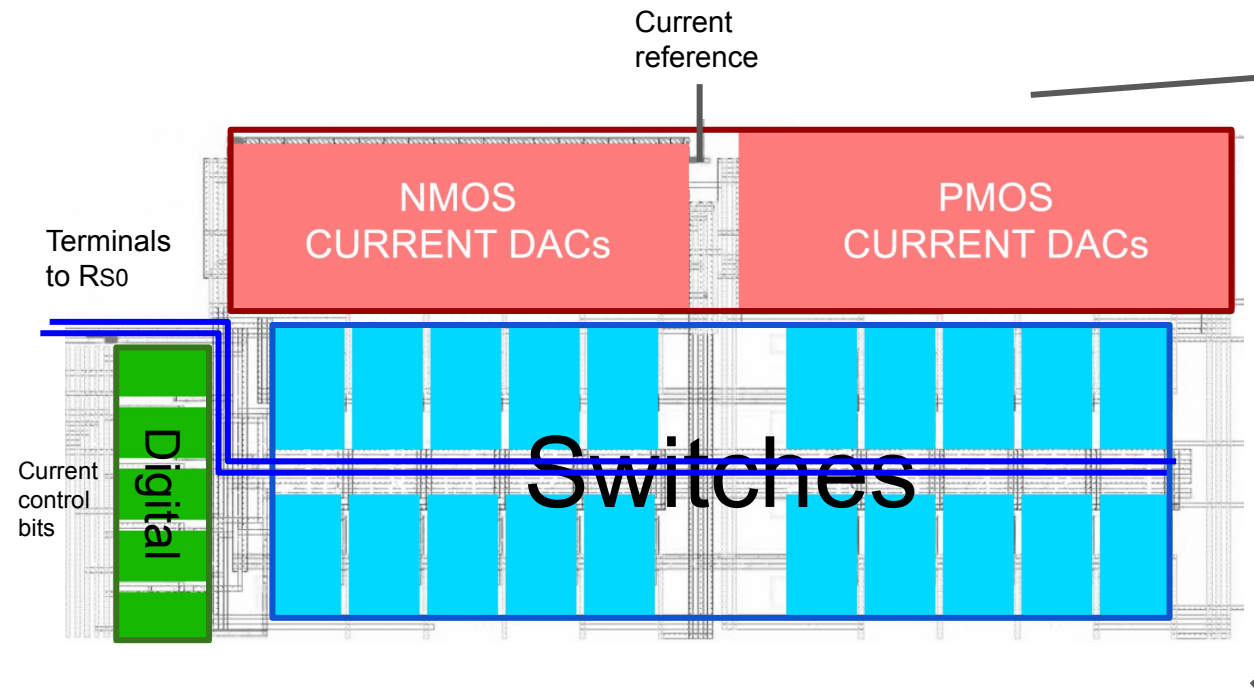
- Purple: Main analog path
- Red: Path traced by R_f
- Green: Path traced by R_S

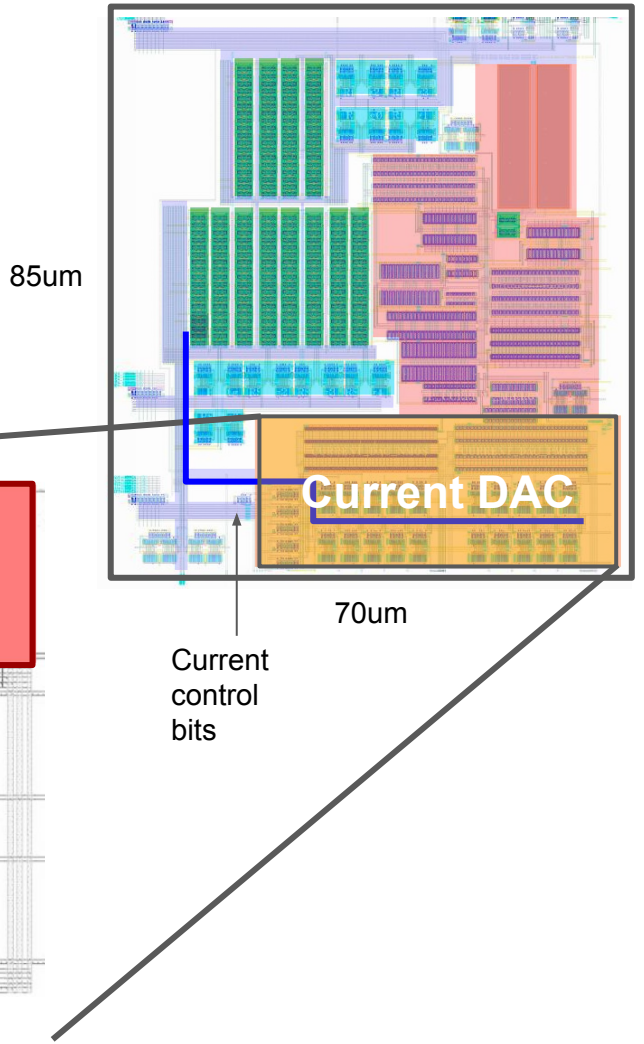
- Gain compression: maximum gain drops to 27.5dB (transient)
- DC offset (with current dac layout): 3σ = 4.9mV



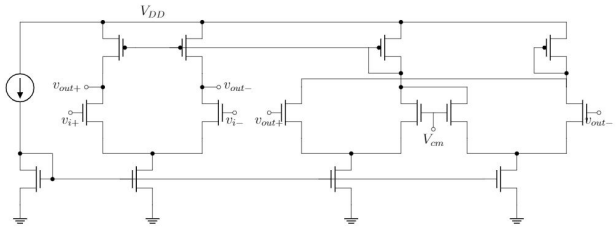
Current DAC Layout & Extraction

- $|V_{dac,max}|$ = 6.3 mV
- $|V_{dac,min}|$ = 200 uV; V_{dac} increase linearly with the code

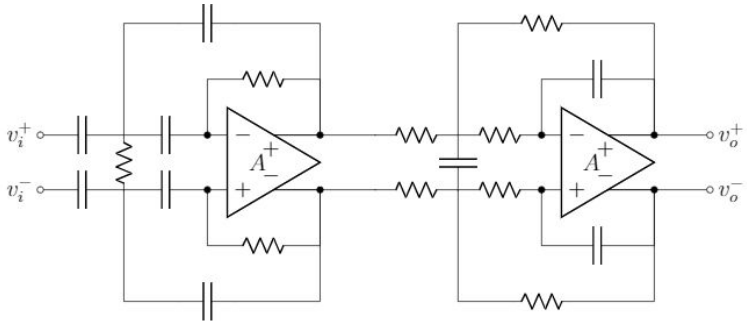




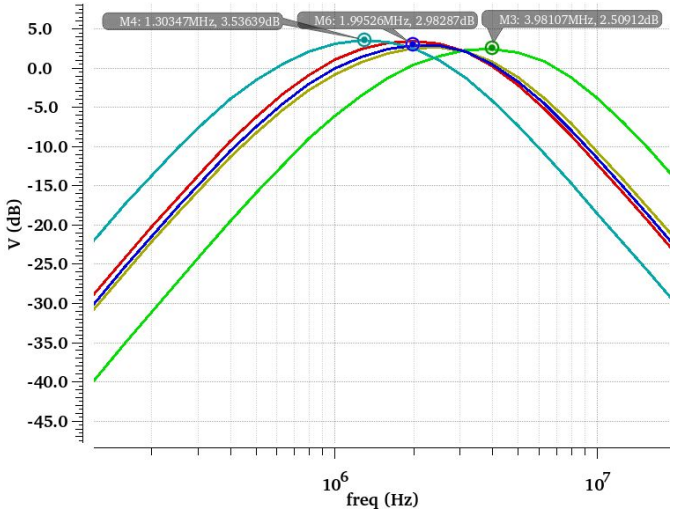
Bandpass Filter



- 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 +-25%* to account for variation



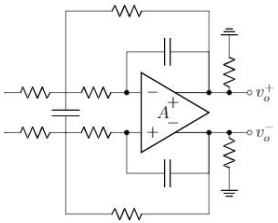
- Noise:
  - Input spot @2MHz: 80nV/√Hz (old)
  - Output integrated (1-3MHz): 170uVrms



No tuning.

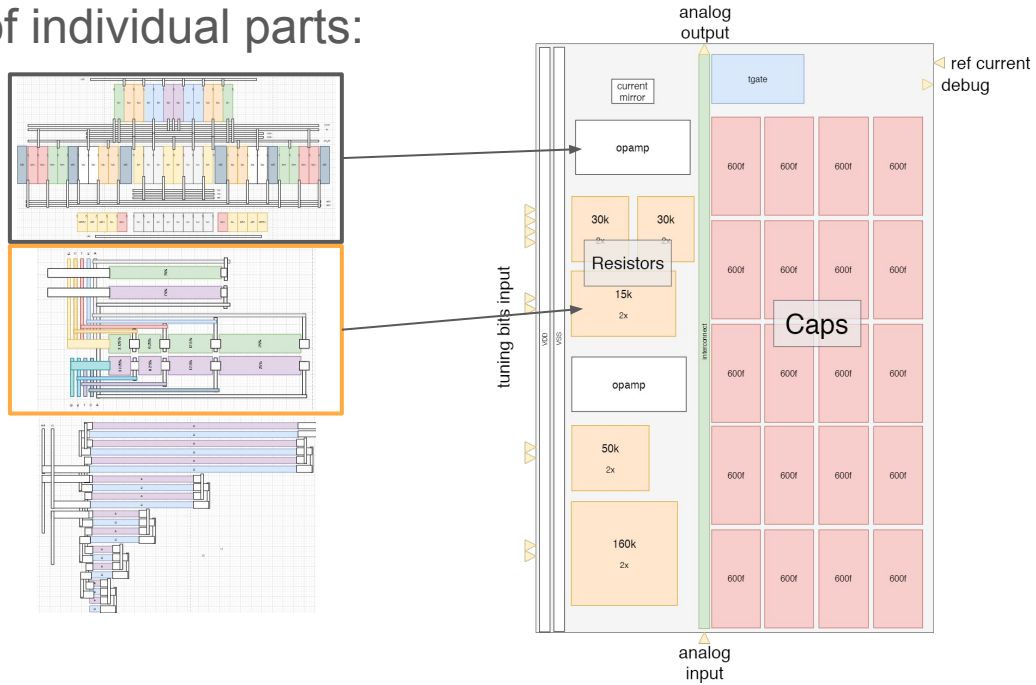
Bandpass Filter

- Changes from last time
  - Output had to be loaded with resistors for gain to not change too much by varying load of next stage (VGA input resistance) and output op-amp required higher current consumption



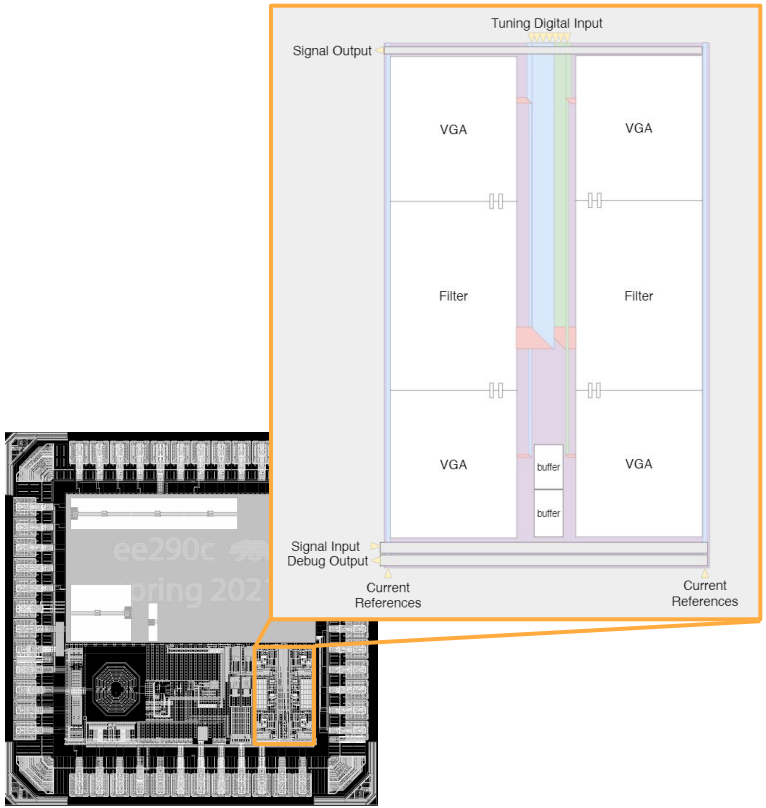
Bandpass Filter Layout

- Planning in draw.io of individual parts:



- Total block layout:
  - Resistors and opamps on the left, capacitance on the right
  - Intermediate signals run in the middle with caps

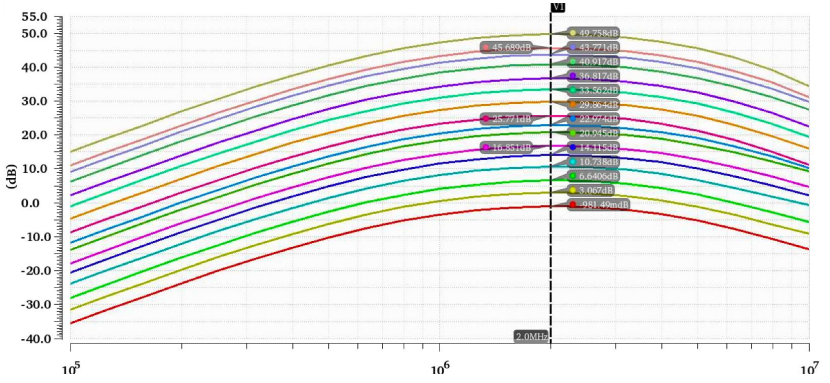
Analog Baseband Layout

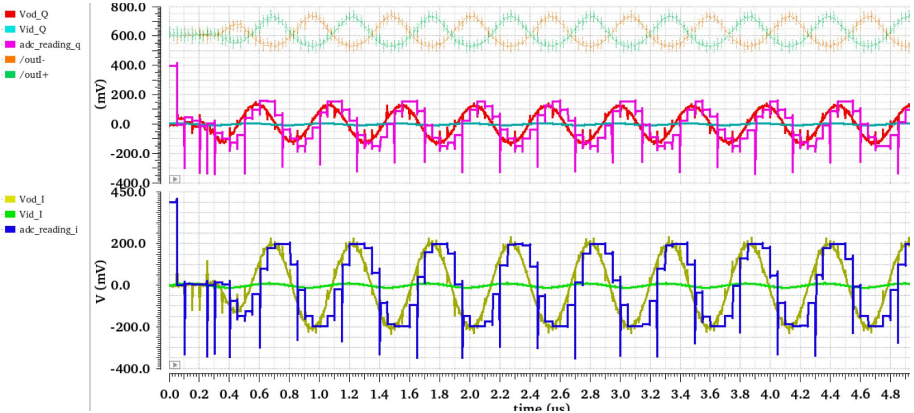


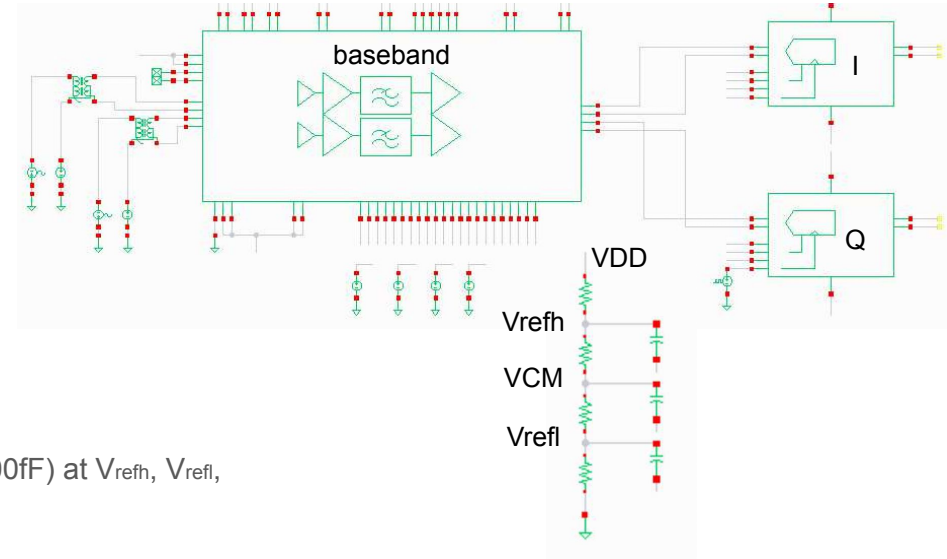
- I/Q Columns
- Digital tuning bits down the middle, analog along the outer edges
- Main signal path routed from bottom to top

Analog Baseband Post-Extraction Simulation

- Gain of the baseband changes monotonically with a nominal step of 4dB; the step size was increased to account for the BPF gain variation and VGA gain compression
- The entire Baseband chain (Mixer-to-baseband buffer+VGA+BPF+VGA) has a gain of -1dB to 50dB



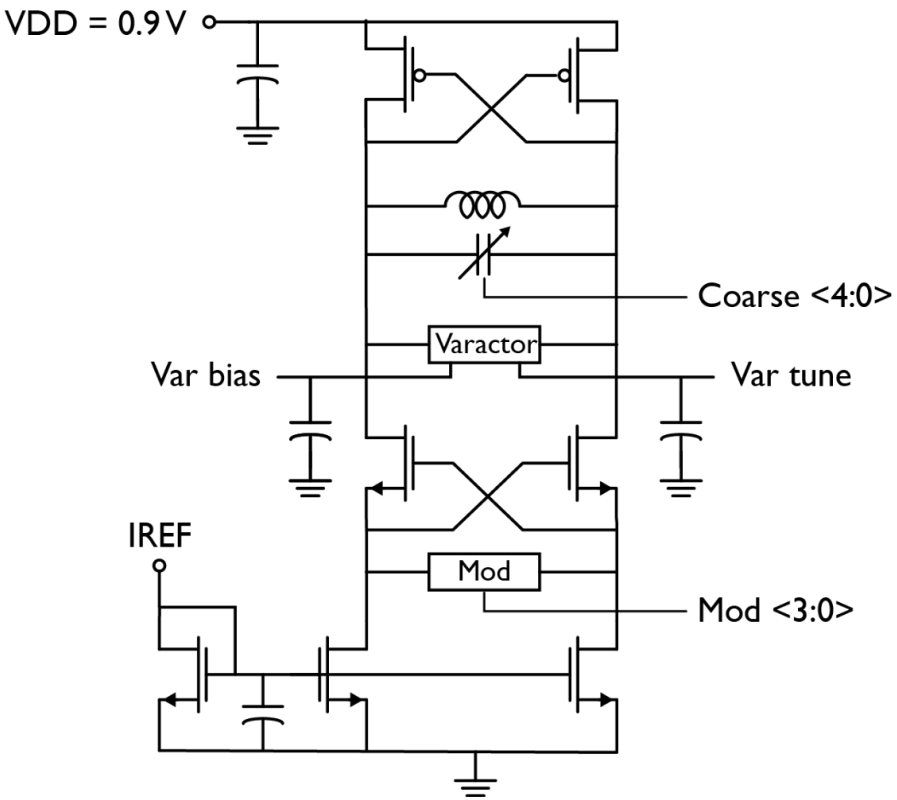




- Simulated with the schematic-level ADC (BAG-generated)
- Resolved large sampling errors by adding a reference filter (~200fF) at V_{refh}, V_{refl}, V_{CM} of the ADC

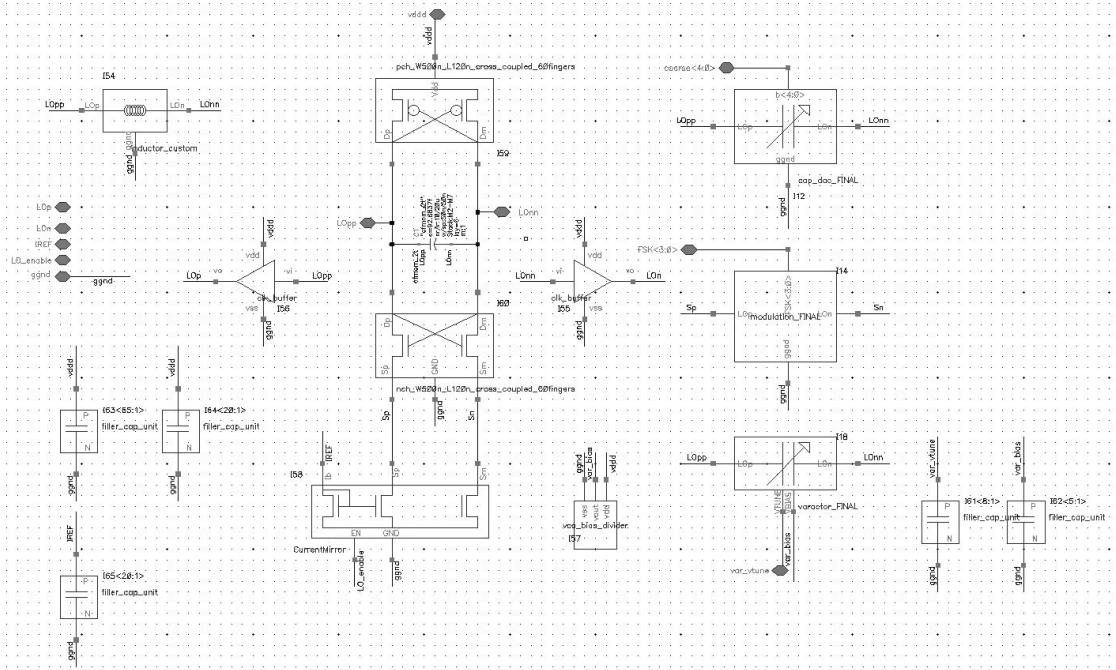
Voltage Controlled Oscillator (VCO): Overview

Spec	Value
F_{center}	4.8 GHz
I_{avg}	1.1 uA
Vdd	0.9 V
Pavg	0.99 mW
Q	10.50
Inductance	2.66 nH

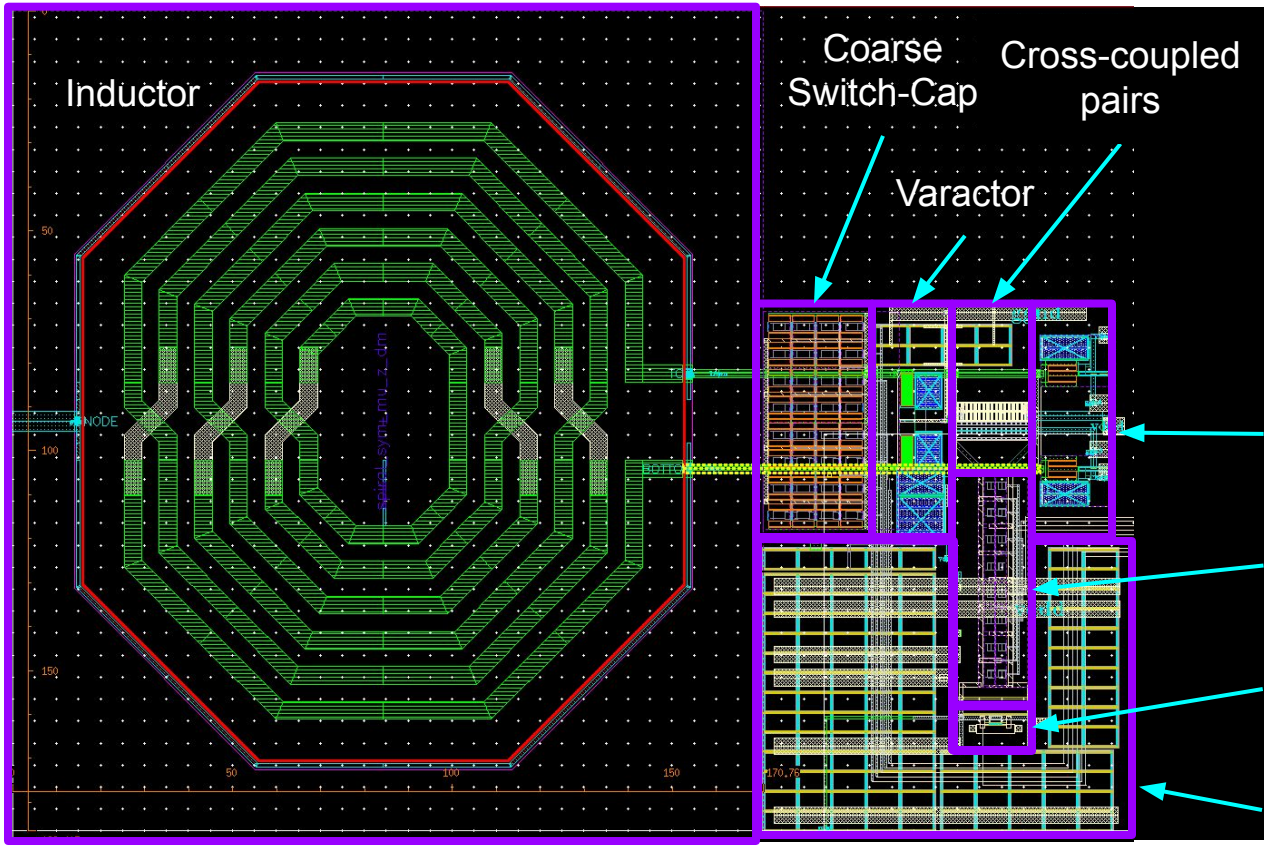


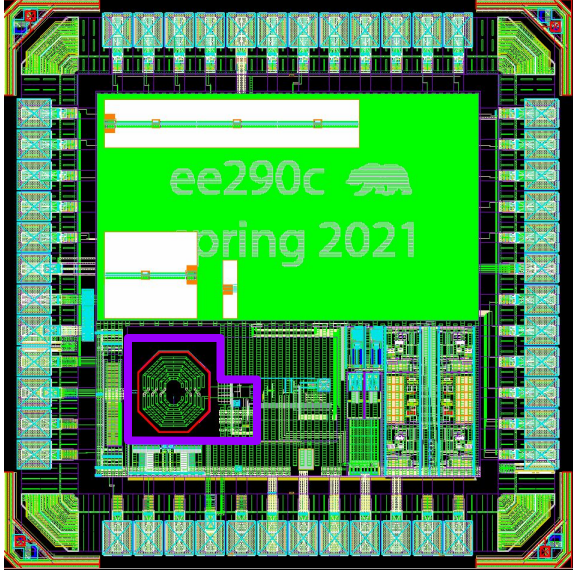
Voltage Controlled Oscillator (VCO): Overview

Spec	Value
F_{center}	4.8 GHz
I_{avg}	1.1 uA
Vdd	0.9 V
Pavg	0.99 mW
Q	10.50
Inductance	2.66 nH



VCO: Layout

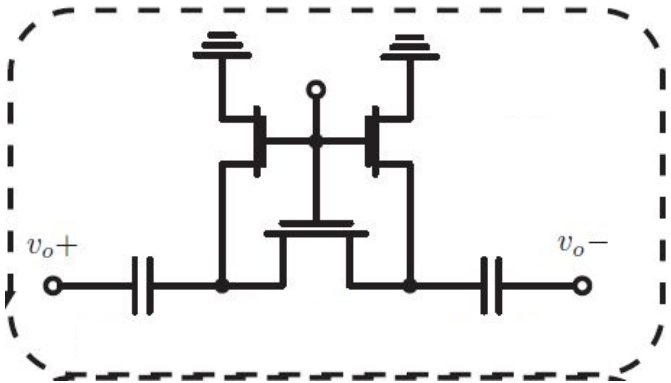


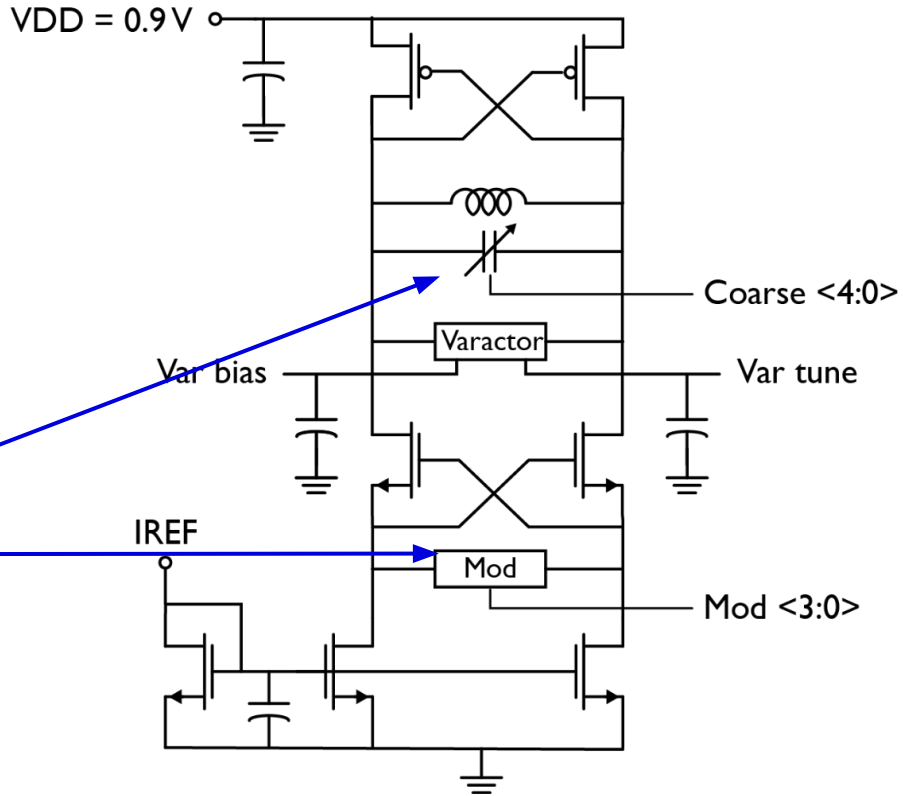


Total Area:
0.0415 mm^2

LO: Cap DAC/Modulation

- Coarse
  - 5 bits Cap Bank
  - ~30 MHz min step
- Modulation
  - 4 bits

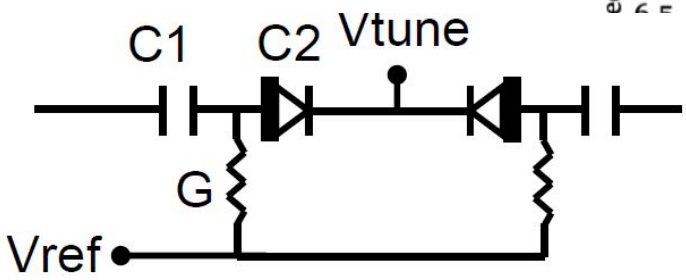




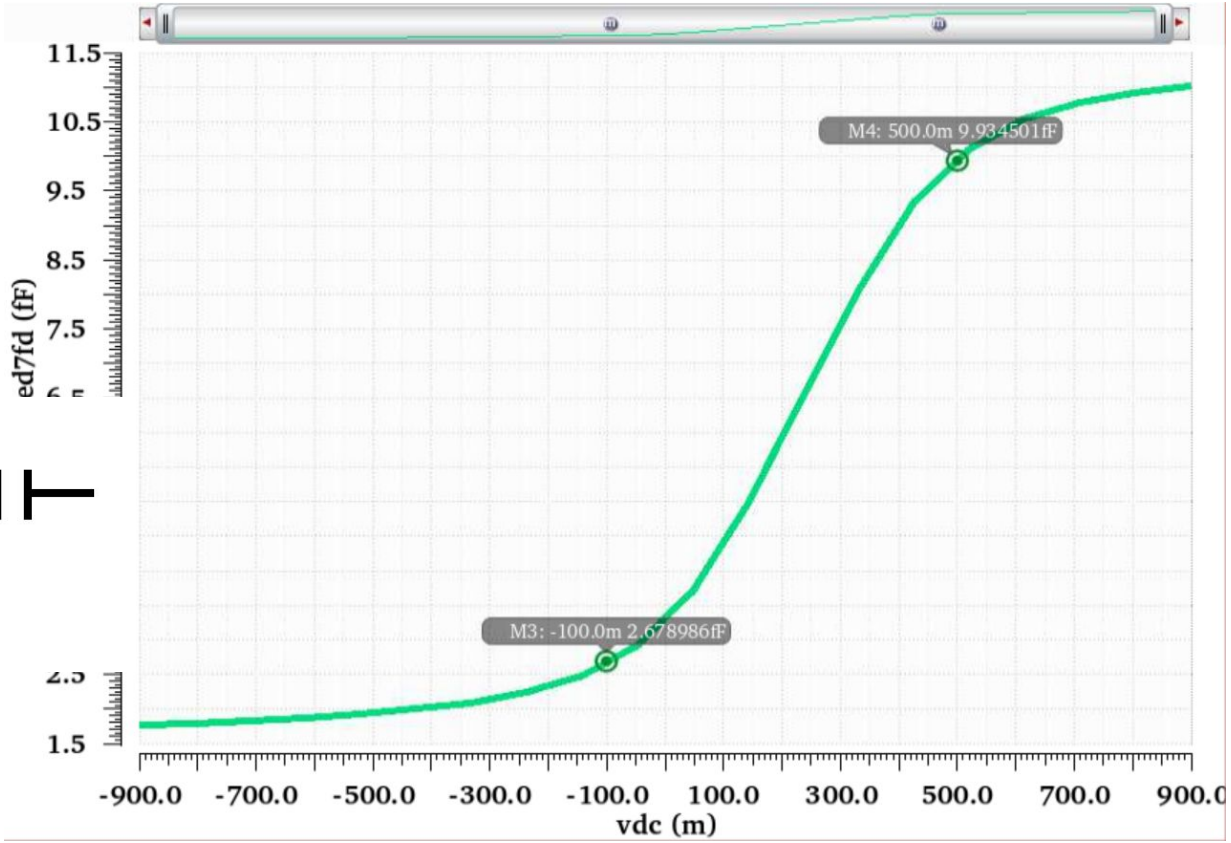
Maksimovic, Monolithic Wireless Transceiver Design, Fall 2018

LO: Varactor

- Range 200mV-800mV
- ~86MHz
- KVCO=143MHz/V

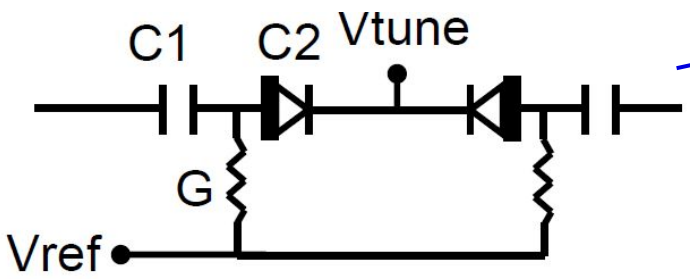


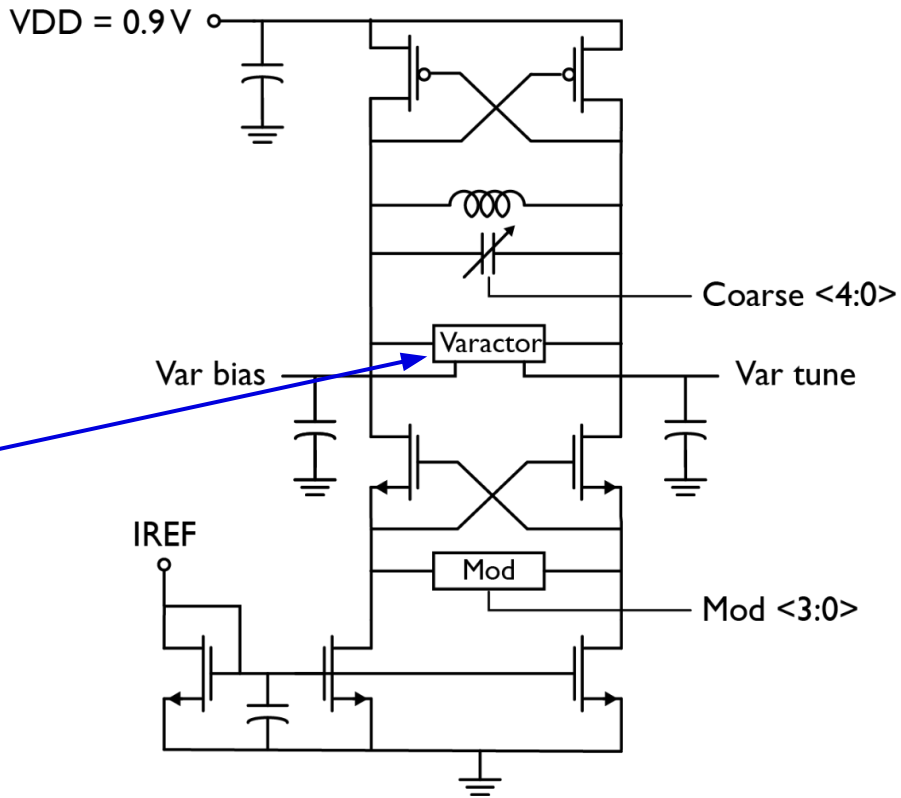
290C, Prof Shana



LO: Varactor

- Range 200mV-700mV
- ~79MHz
- KVCO=143MHz/V





Spring 2019 290C, Prof Shanaa

VCO: Post Extraction Switch-Cap Array only

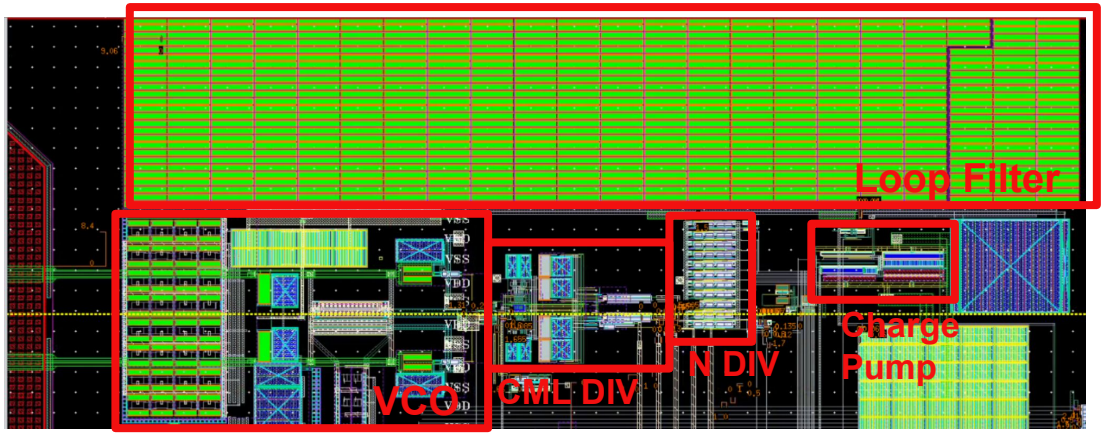
		Parameter						TT	ff_cold	ss_hot
		tcbn28hpcpl...						<unspecified section>	<unspecified section>	<unspecified section>
		temperature						27	-40	80
		toplevel.scs						top_tt	top_ff	top_ss
Point	Test	Output	Spec	Weight	Pass/Fail	Min	Max	TT	ff_cold	ss_hot
Parameters: BUS=0										
1	Alex_4.8GHz_LO:LO_FINAL_tb:1	Amplitude				944m	1.013	978.1m	1.013	944m
1	Alex_4.8GHz_LO:LO_FINAL_tb:1	Frequency				5.072G	5.822G	5.417G	5.822G	5.072G
1	Alex_4.8GHz_LO:LO_FINAL_tb:1	/V0/PLUS								
Parameters: BUS=31										
2	Alex_4.8GHz_LO:LO_FINAL_tb:1	Amplitude				821.8m	967.3m	901.1m	967.3m	821.8m
2	Alex_4.8GHz_LO:LO_FINAL_tb:1	Frequency				3.727G	4.72G	4.16G	4.72G	3.727G
2	Alex_4.8GHz_LO:LO_FINAL_tb:1	/V0/PLUS								

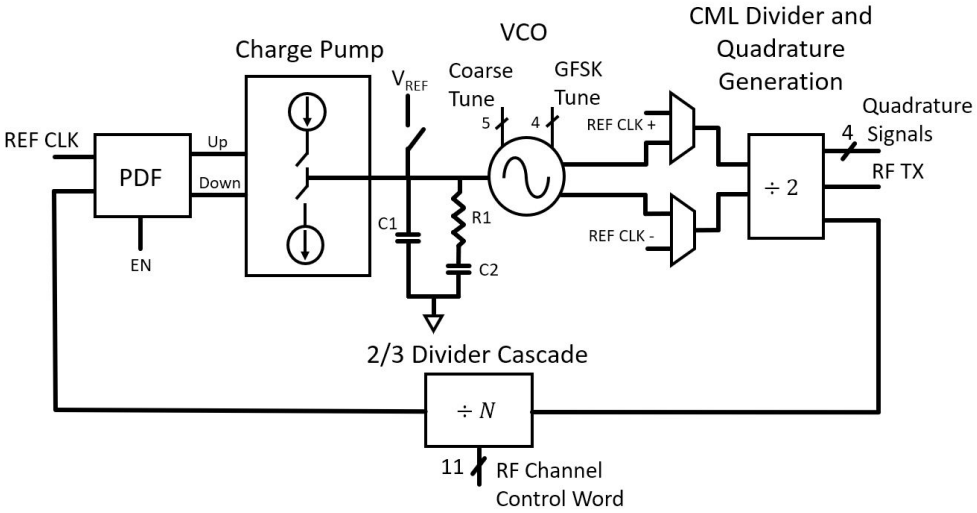
VCO: Post Extraction VCO

		Parameter						TT	ff_cold	ss_hot
		tcbn28hpcplusbwp3...						<unspecified section>	<unspecified section>	<unspecified section>
		temperature						27	-40	80
		toplevel.scs						top_tt	top_ff	top_ss
Point	Test	Output	Spec	Weight	Pass/Fail	Min	Max	TT	ff_cold	ss_hot
Parameters: BUS=0										
1	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	Phase Noise @ 3MHz				-114.8	-111.8	-111.8	-114.8	-113.1
1	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	Amplitude				885.5m	963.7m	923.1m	963.7m	885.5m
1	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	Frequency				4.94G	5.574G	5.234G	5.574G	4.94G
1	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	/net01								
1	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	/net02								
Parameters: BUS=31										
2	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	Phase Noise @ 3MHz				-116.5	-114	-116.5	-114	-116.1
2	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	Amplitude				748.6m	912m	836.4m	912m	748.6m
2	Alex_4.8GHz_LO:LO_FINAL_v2_tb:1	Frequency				3.68G	4.59G	4.08G	4.59G	3.68G

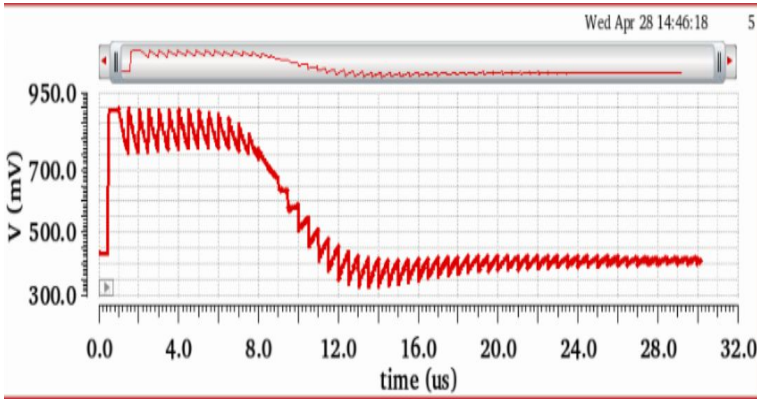
Phase Lock Loop

- Second order integer N PLL
- 2 MHz reference frequency
- Total Area: 0.0203 mm^2
- Power Consumption: 190 uW



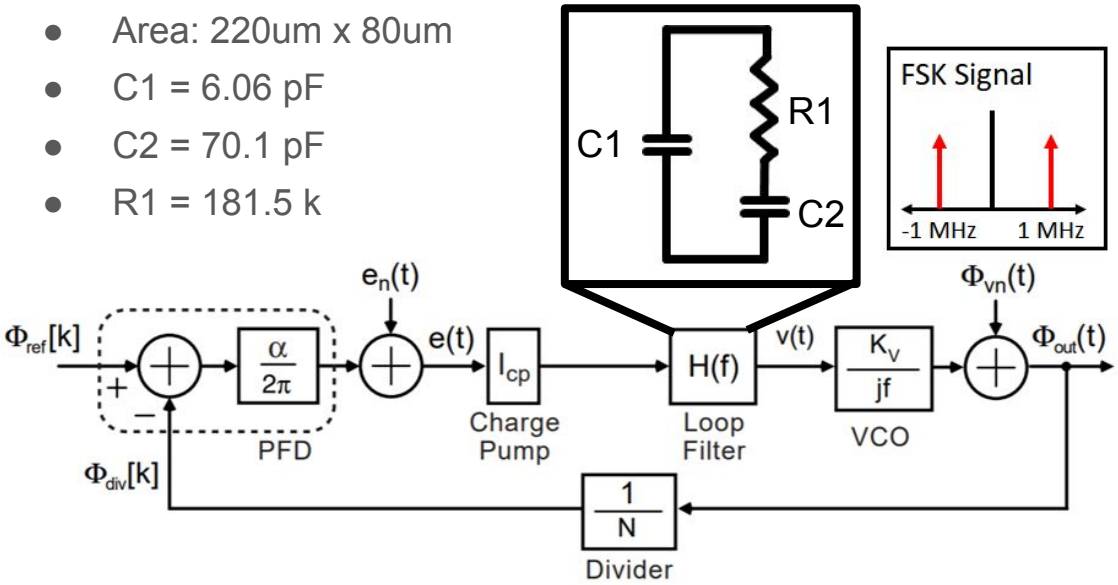


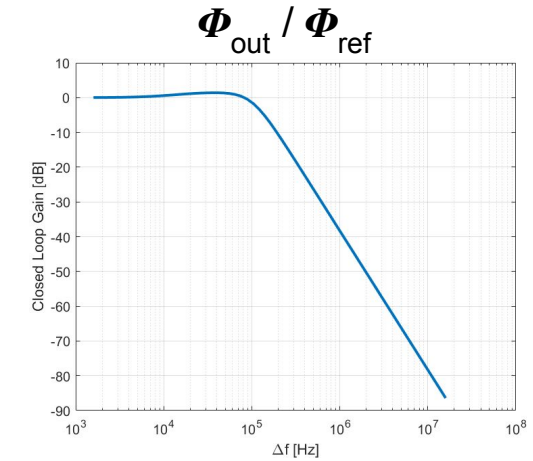
VCO Control Voltage

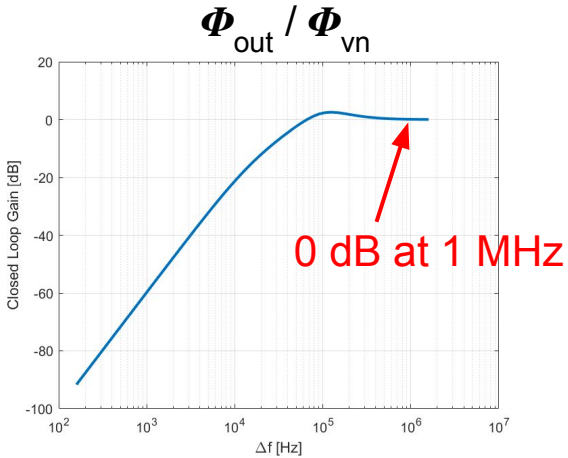


PLL Loop Filter

- 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*
- Area: 220um x 80um
- C1 = 6.06 pF
- C2 = 70.1 pF
- R1 = 181.5 k

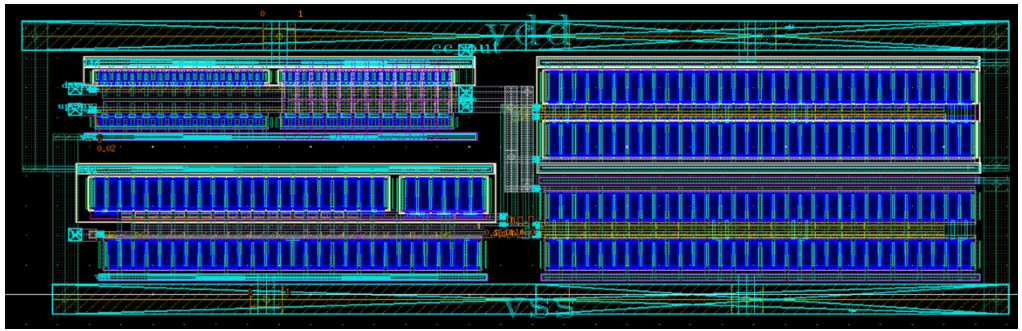


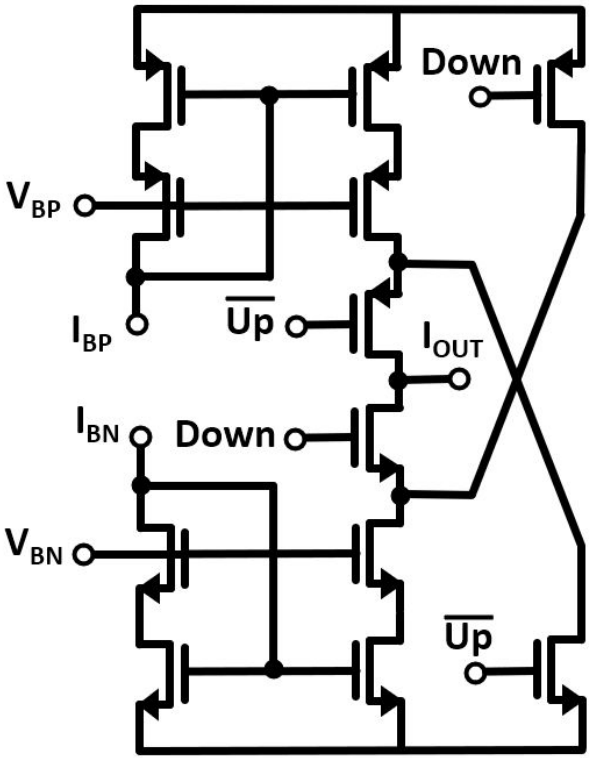




Charge Pump

- Cascode current source with wide swing current mirrors
  - I_{OUT} = 50 uA
  - $\Delta I < 1.1\%$ over the output range (200 mV - 700 mV)
  - Maximum leakage current: 160 pA
  - Estimated Power Consumption: 144 uW
- Dummy loads keep current through “Up” and “Down” branches always on, reducing transient currents when switching

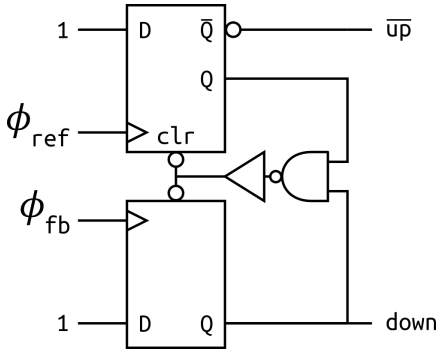


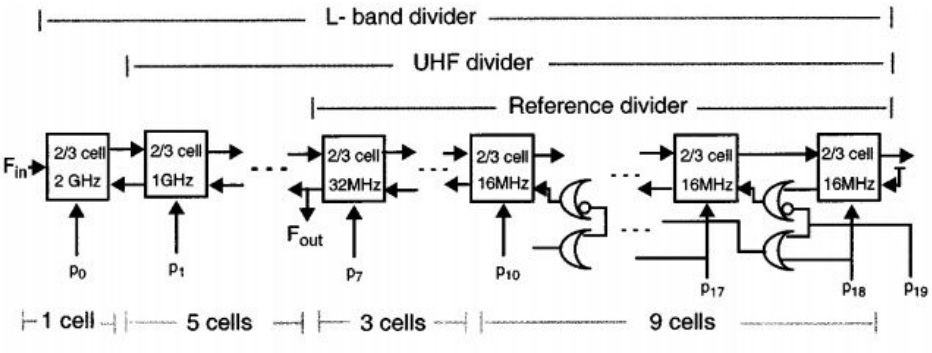


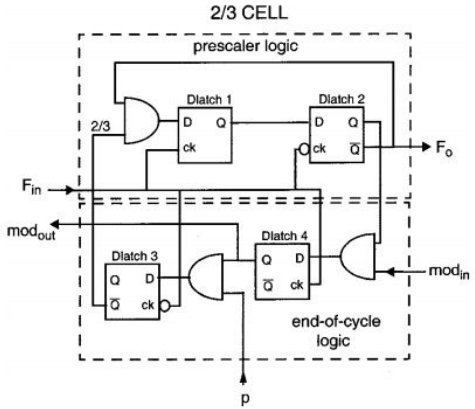
Phase/Frequency Detector and N-Divider

- Channel control word sets divide ratio from 1200-1240 for constant IF receiver, and transmitter
- Estimated Area: 30um x 15um
- Power Consumption: 40 uW

Phase/Frequency Detector and N-Divider

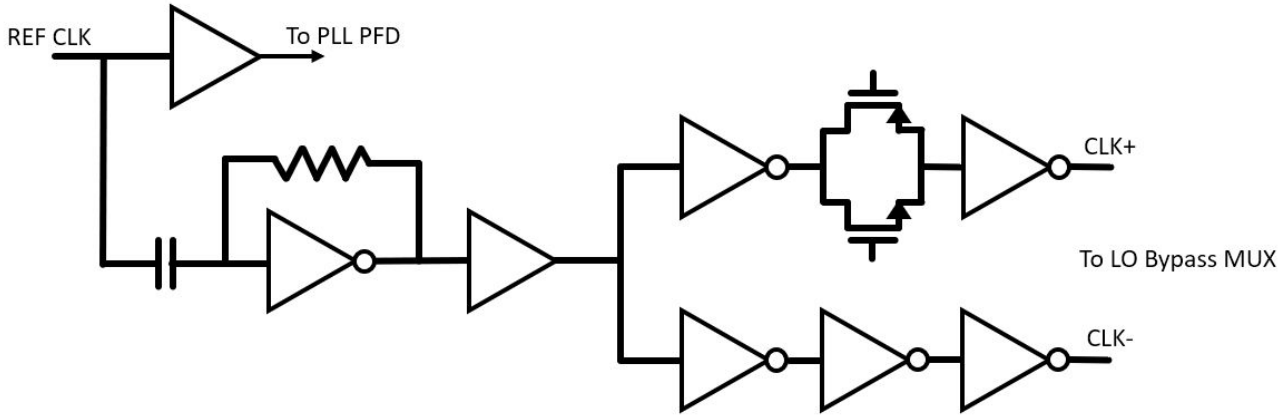




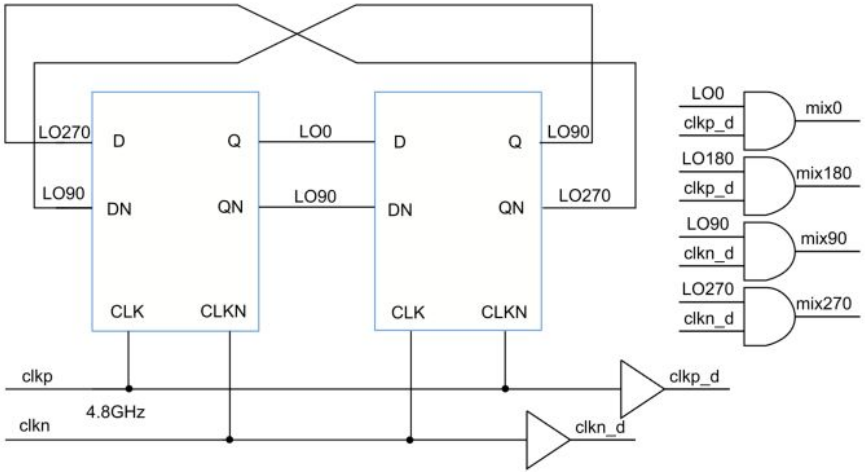


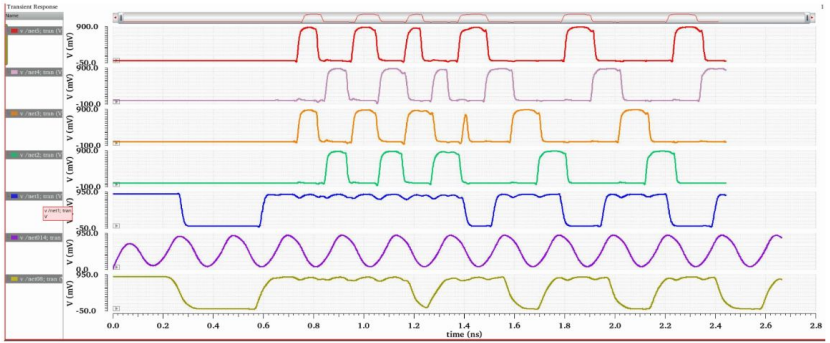
On-Chip LO Bypass

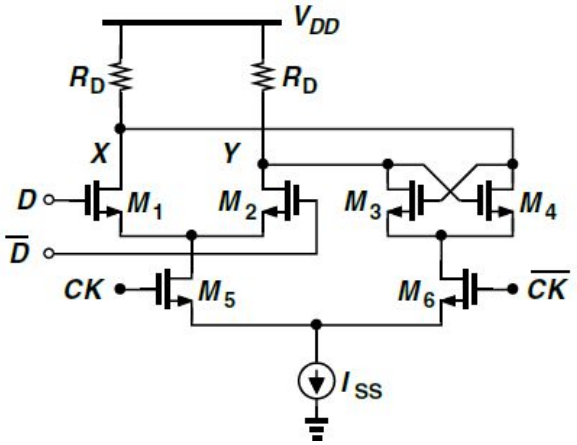
- Back-up plan for the LO is to feed a 4.8 GHz signal off chip directly into quadrature generation.
- First branch goes directly to the PFD, intended for 2 MHz.
- Second branch gets buffered and converted to differential, intended for 4.8 GHz



Pre-scalar/Mixer LO Distribution



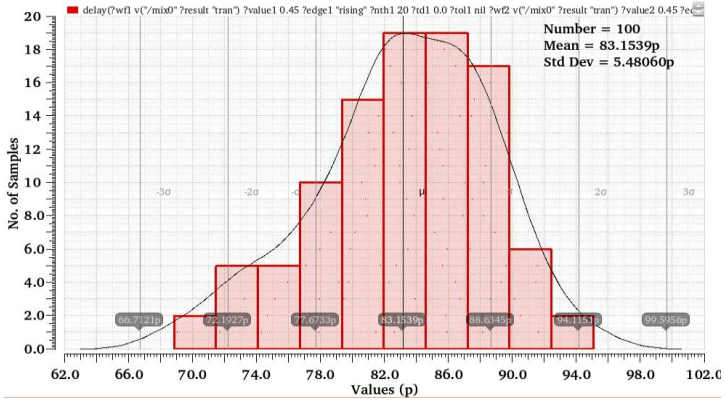


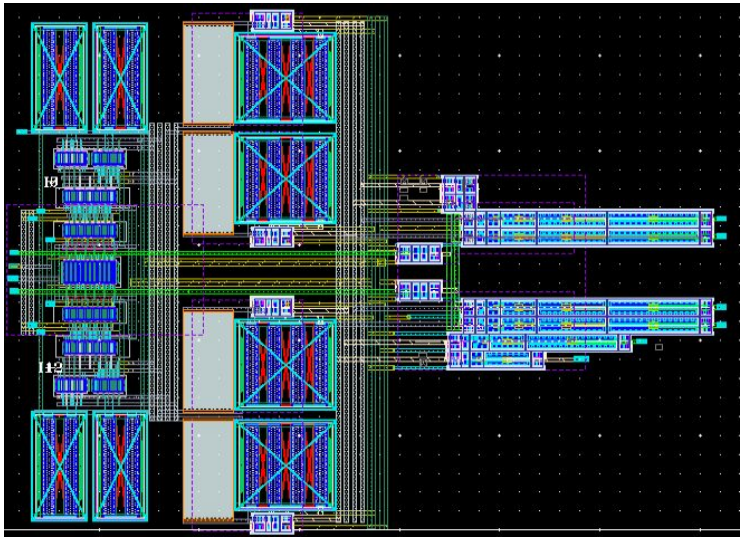


- CML Logic for better differential output balance
  - <4ps output pulse width variation
- Takes 1 clock cycle to settle
- Current consumption average 560uA

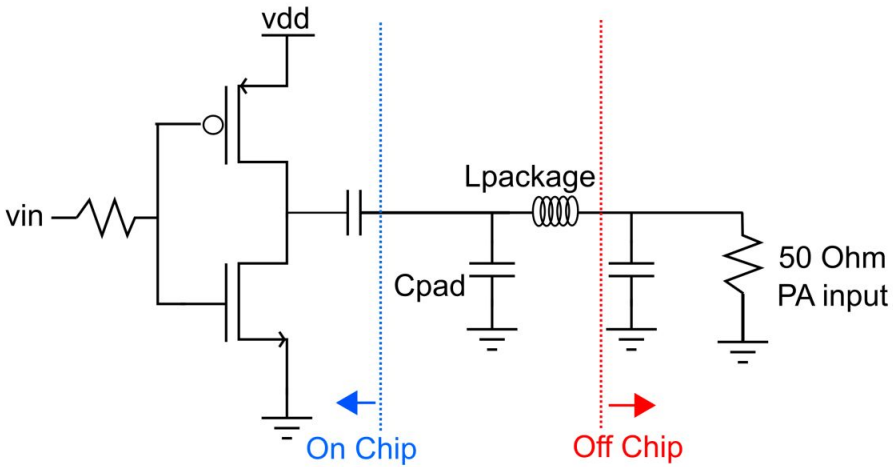
Pre-scalar/Mixer LO Divider Cont.

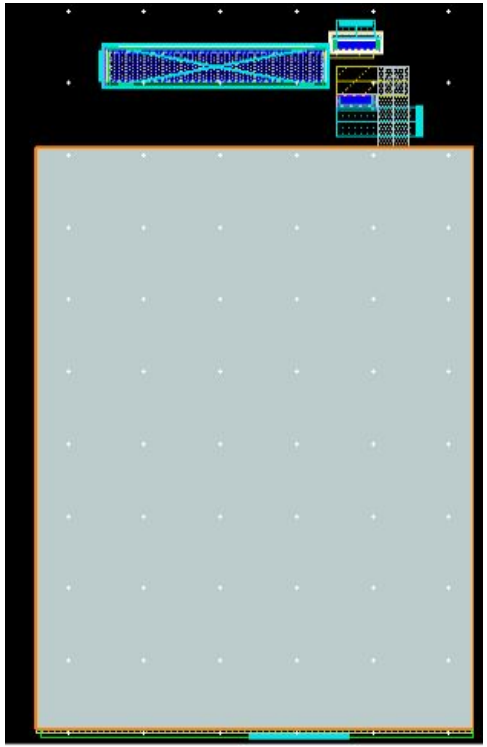
- CML latch delay does not track well with CMOS delay
  - Potential issues for quadrature generation
  - different process corners - might have to debug by varying temperature
- Monte Carlo of quadrature pulse width in TT conditions to demonstrate no failure





LO Driver

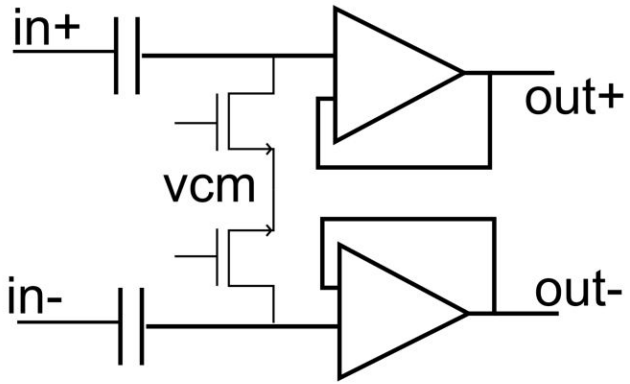


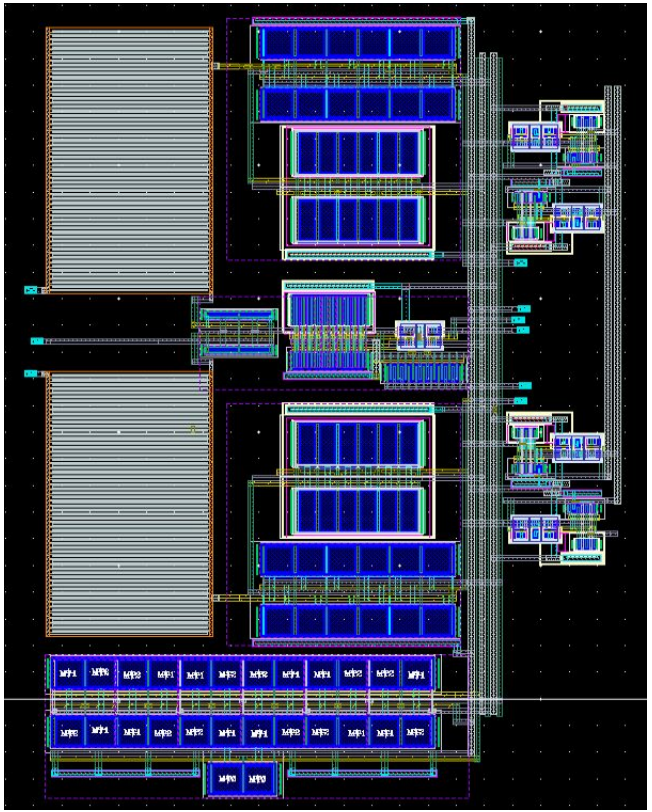


- Post layout sim of min -10dBm output power
- Current consumption of average 1mA

Mixer to Baseband Buffer

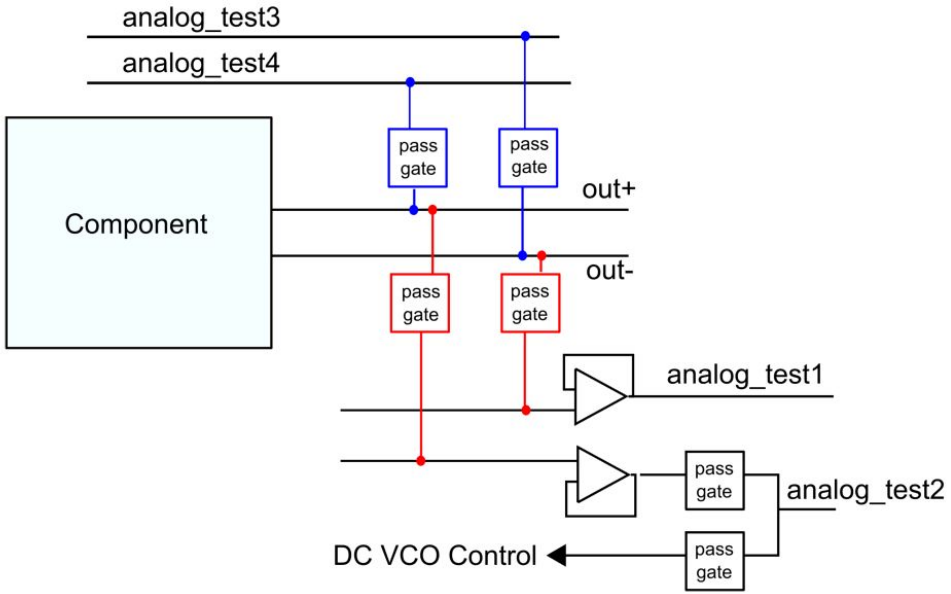
- Pseudo-differential
- DC Offset 3σ difference = 4.5mV
- Current Consumption ~120uA per buffer



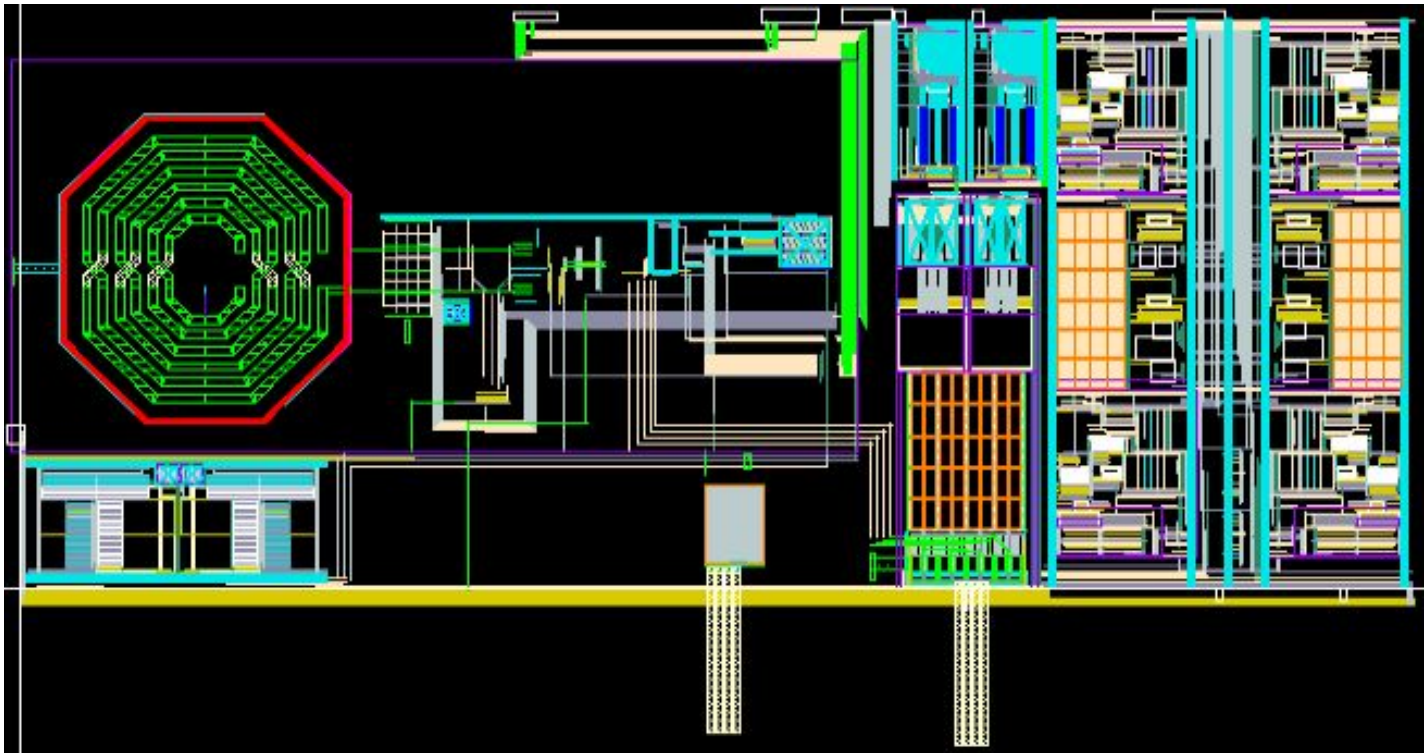


Debug Setup

- 4 test pads
- Baseband components' in+, in-, out+, out- signals share pad
- Analog_test2 pad also muxed to allow off chip VCO control

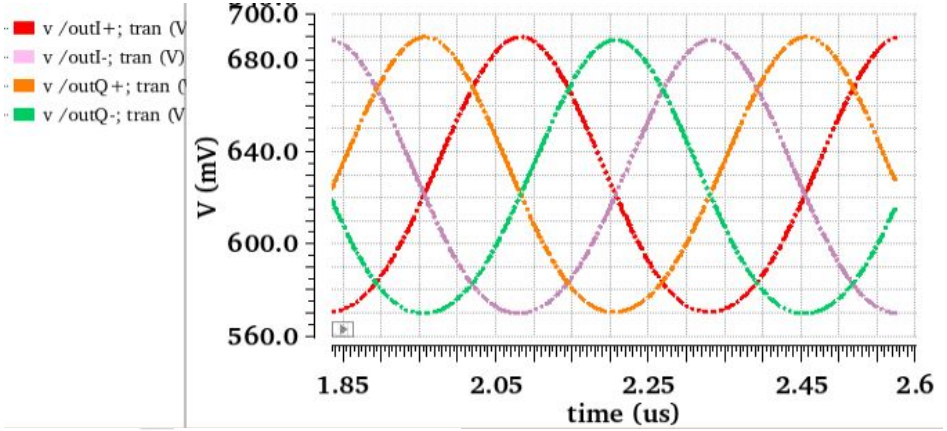


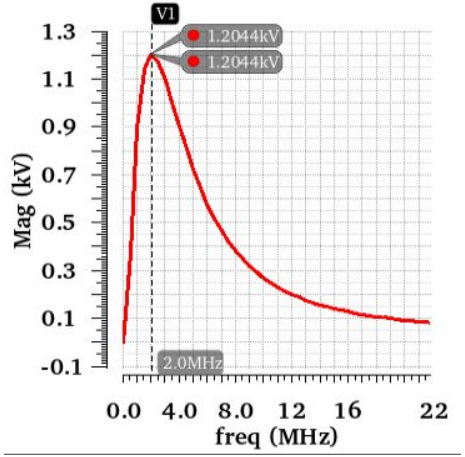
Radio Top Layout

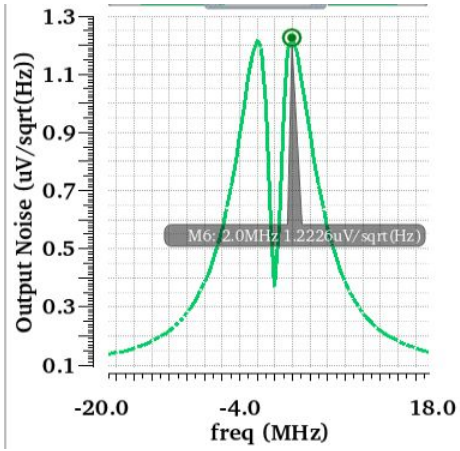


Current Receiver Results

- 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







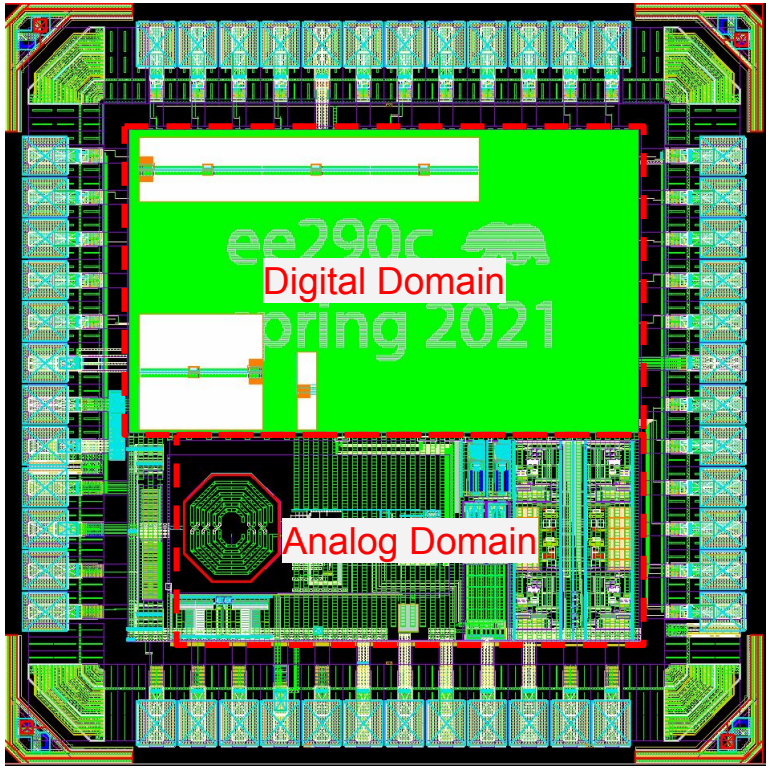
Analog & Power

Introduction

- Jackson Paddock
  - 5th year MS student in Prof. Kristofer Pister's lab
  - Focus on analog circuit design (specifically LDO generation with the Berkeley Analog Generator, BAG)

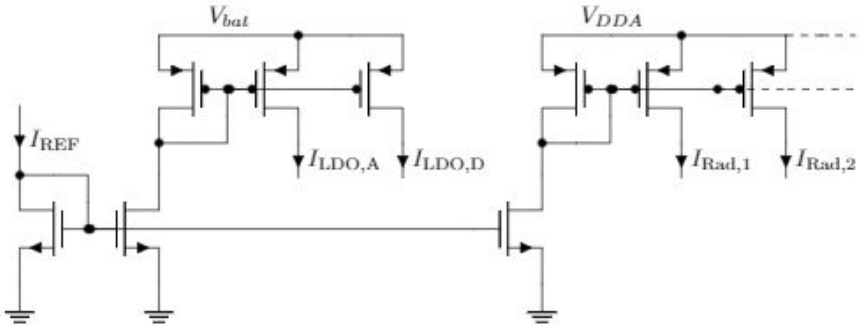
Power Domains

- 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) nominally at 0.9V
  - With the slow corner and temperature, the current is 20.8mA
- I/O:
  - Includes: pad ring
  - VDDIO comes from an off-chip supply



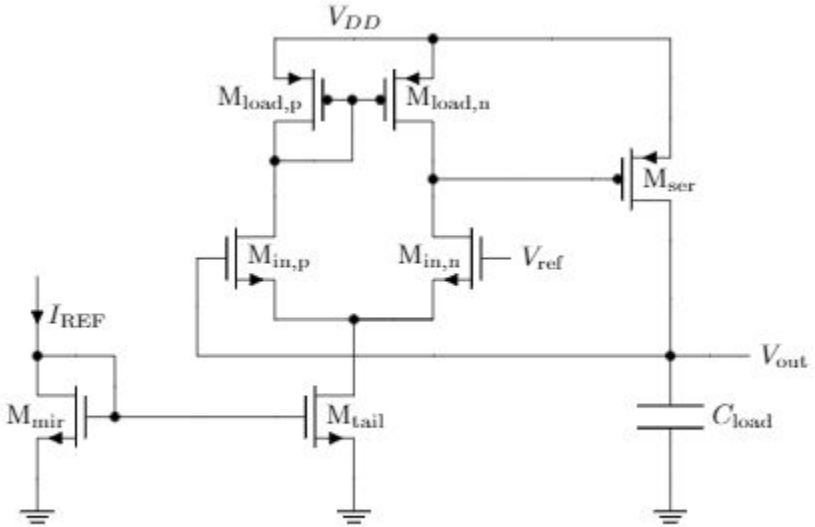
Current Reference Topology

- 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 V_{DS}



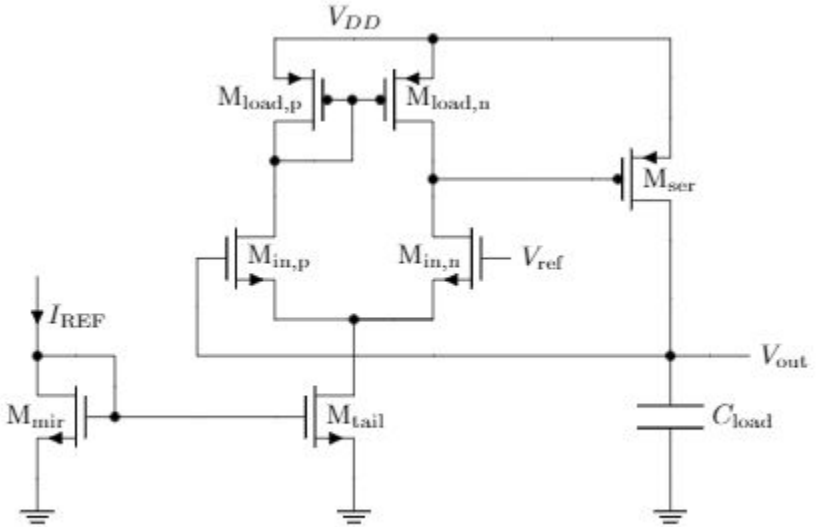
LDO Topology

- Both the digital and analog LDOs use the same topology: a 5T differential pair amplifier with a PMOS series device
- Both LDOs use the same amplifier and have outputs at 0.9V
- Both LDOs are rated for a peak of 10mA, designed at a slightly lower nominal value of 8.3mA
- The dominant pole is at the output with an off chip (due to area) capacitor for stability with C_{load}=100nF



LDO Specs (TT, 27°C)*

Parameter	**Value (I_L=1mA)**	**Value (I_L=10mA)**
Static Error	-6.363mV	666.8uV
Amplifier Current	201.6uA	193.8uA
Phase Margin	84.79°	61.54°
PSRR	57.13dB	49.92dB
PSRR bandwidth	750.7kHz	1.730MHz
Load Regulation*	237.7u	459.7u



*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	728.8uV	666.8uV	518.1uV	-360.8uV	1.630mV	893.2uV
Amp Current	189.9uA	193.8uA	201.2uA	174.3uA	214.9uA	197.3uA
PM	60.54°	61.54°	63.64°	57.89°	67.02°	65.37°
PSRR	49.95dB	49.92dB	49.86dB	47.29dB	52.00dB	47.47dB
PSRR BW	1.801MHz	1.730MHz	1.607MHz	1.425MHz	2.385MHz	1.983MHz
Load Reg	429.7u	459.7u	524.7u	368.8u	544.3u	630.5u

*Simulated with load current at 10mA