Final Project

Shoutout Austin you’re awesome

Cachemeoutside

Our Project

-Super Scalar Processor

-Branch Prediction

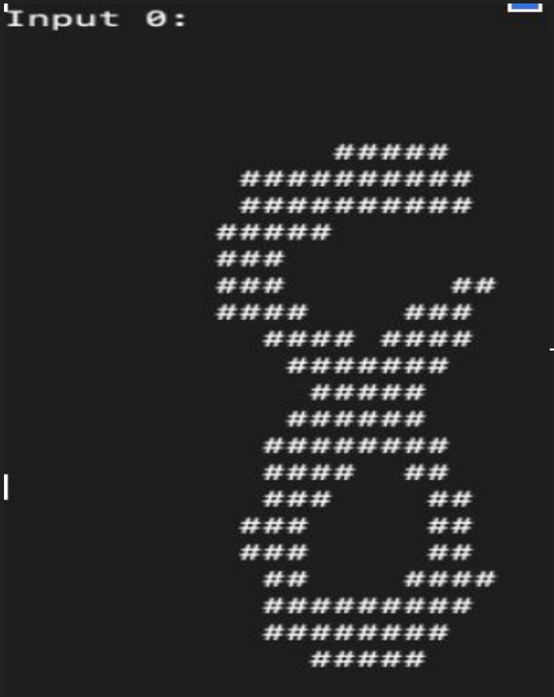
-Ported onto FPGA(NOT SIMULATED)

-We decided to test against running the MNIST-Data set Neural Network(which is the current 6.1910/.004 design project).

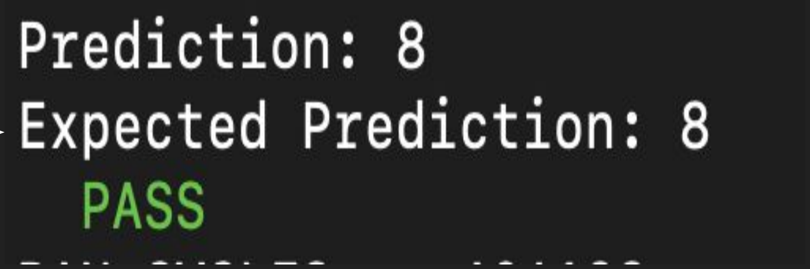
Which:
-converts a 28x28 image of a number into a 784-element input vector

-passes input vector through the network to produce a 10-element output vector

-uses the output vector to predict the number inputted



```
Input 0:



     #####
 ##########
 ##########
#####
###
###       ##
####    ###
  #### ####
   #######
    #####
   ######
  ########
  ####  ##
  ###    ##
 ###     ##
 ###     ##
  ##    ####
  ##########
  ########
    #####
```



```
Prediction: 8
Expected Prediction: 8
  PASS
```

Superscalar Processor (How's it Work)

-2 of Each Stage(Except Fetch), so 7 rules for the processor total

-Create 2 PC’s, and increment PC by 8 each fetch

-We can only process 2 ALU’s at once(so have logic to stall if the two instructions aren’t ALU or are dependent on each other)

-Make it so instruction cacah can potentially fetch 2 instructions if needed

-Changes to Scoreboard and RF(basically doubling it)

Super Scalar Bugs

Some issues we ran into:

-Trying to only use 4 rules

-Adjusting Konata to allow for 2 instructions working at the same time

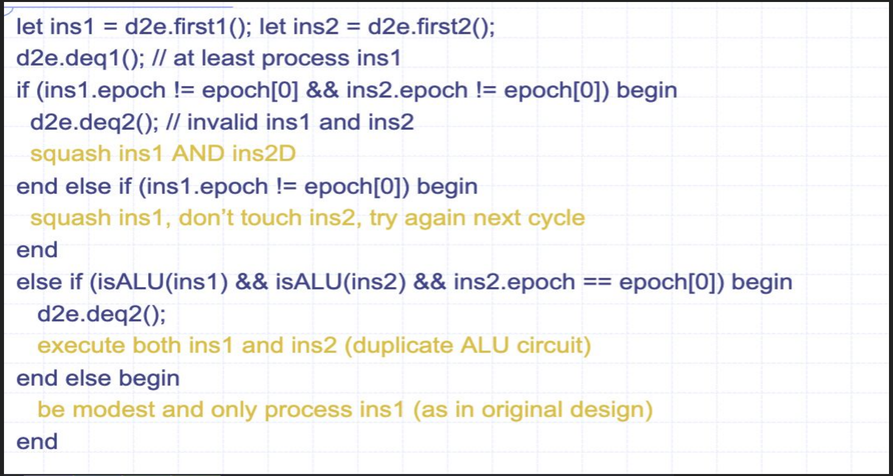
-PC’s being wrong again when we fixed the first Bug

-Forgetting to add additional EPOCHS

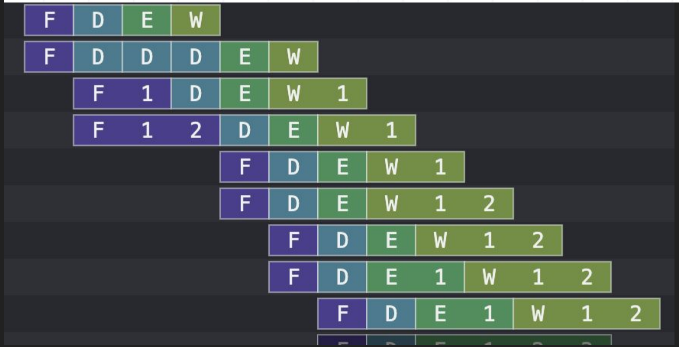
-Why are we enquing 2 instructions when we don’t want to

-why is it just failing

the clickbait pt 1



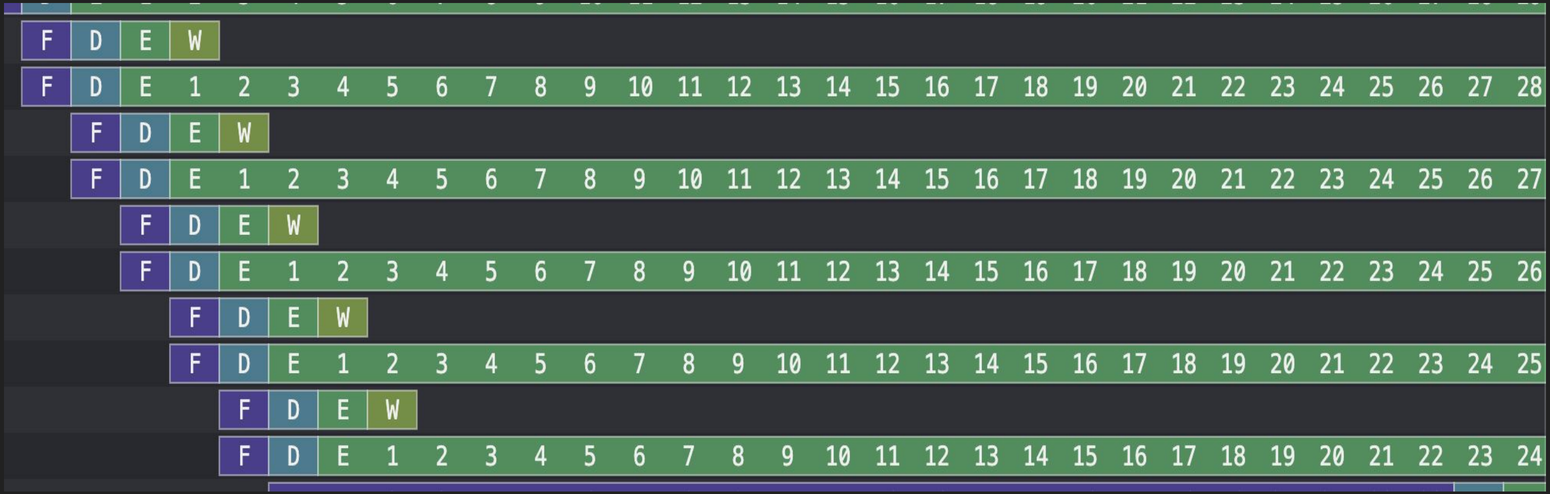
```
let ins1 = d2e.first1(); let ins2 = d2e.first2();
d2e.deq1(); // at least process ins1
if (ins1.epoch != epoch[0] && ins2.epoch != epoch[0]) begin
  d2e.deq2(); // invalid ins1 and ins2
  squash ins1 AND ins2D
end else if (ins1.epoch != epoch[0]) begin
  squash ins1, don't touch ins2, try again next cycle
end
else if (isALU(ins1) && isALU(ins2) && ins2.epoch == epoch[0]) begin
   d2e.deq2();
   execute both ins1 and ins2 (duplicate ALU circuit)
end else begin
   be modest and only process ins1 (as in original design)
end
```



-this code really makes it look like you can decode 2 instructions in a single rule (which you probably can, but it's quite annoying and you have to catch a bunch of bugs)

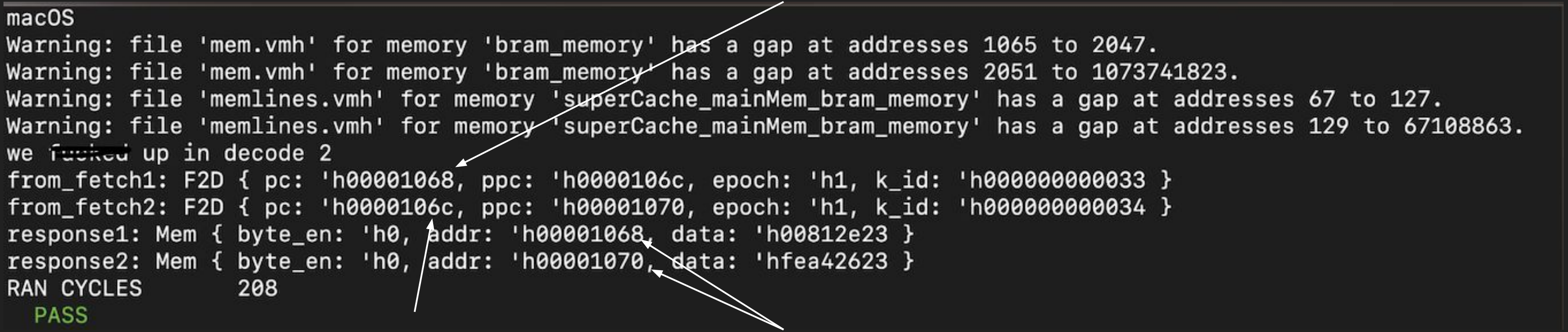
- the bug that set us off was it not stalling properly on the calls to the SuperFIFO

Konata Acting Up


F D E W
F D E 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
F D E W
F D E 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
F D E W
F D E 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
F D E W
F D E 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
F D E W
F D E 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

1 Fifo $\longrightarrow$ 1 Super-fifo $\longrightarrow$ 2 Fifos(This took too long to think off)

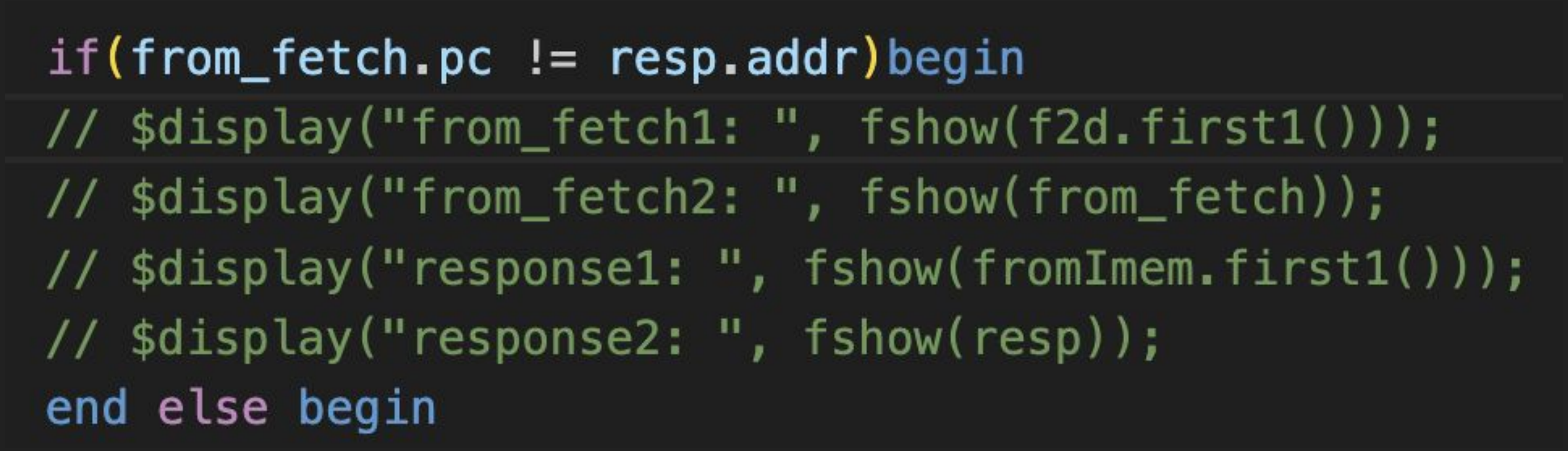
PC Just Not Matching



```
macOS
Warning: file 'mem.vmh' for memory 'bram_memory' has a gap at addresses 1065 to 2047.
Warning: file 'mem.vmh' for memory 'bram_memory' has a gap at addresses 2051 to 1073741823.
Warning: file 'memlines.vmh' for memory 'superCache_mainMem_bram_memory' has a gap at addresses 67 to 127.
Warning: file 'memlines.vmh' for memory 'superCache_mainMem_bram_memory' has a gap at addresses 129 to 67108863.
we [redacted] up in decode 2
from_fetch1: F2D { pc: 'h00001068, ppc: 'h0000106c, epoch: 'h1, k_id: 'h000000000033 }
from_fetch2: F2D { pc: 'h0000106c, ppc: 'h00001070, epoch: 'h1, k_id: 'h000000000034 }
response1: Mem { byte_en: 'h0, addr: 'h00001068, data: 'h00812e23 }
response2: Mem { byte_en: 'h0, addr: 'h00001070, data: 'hfea42623 }
RAN CYCLES        208
  PASS
```

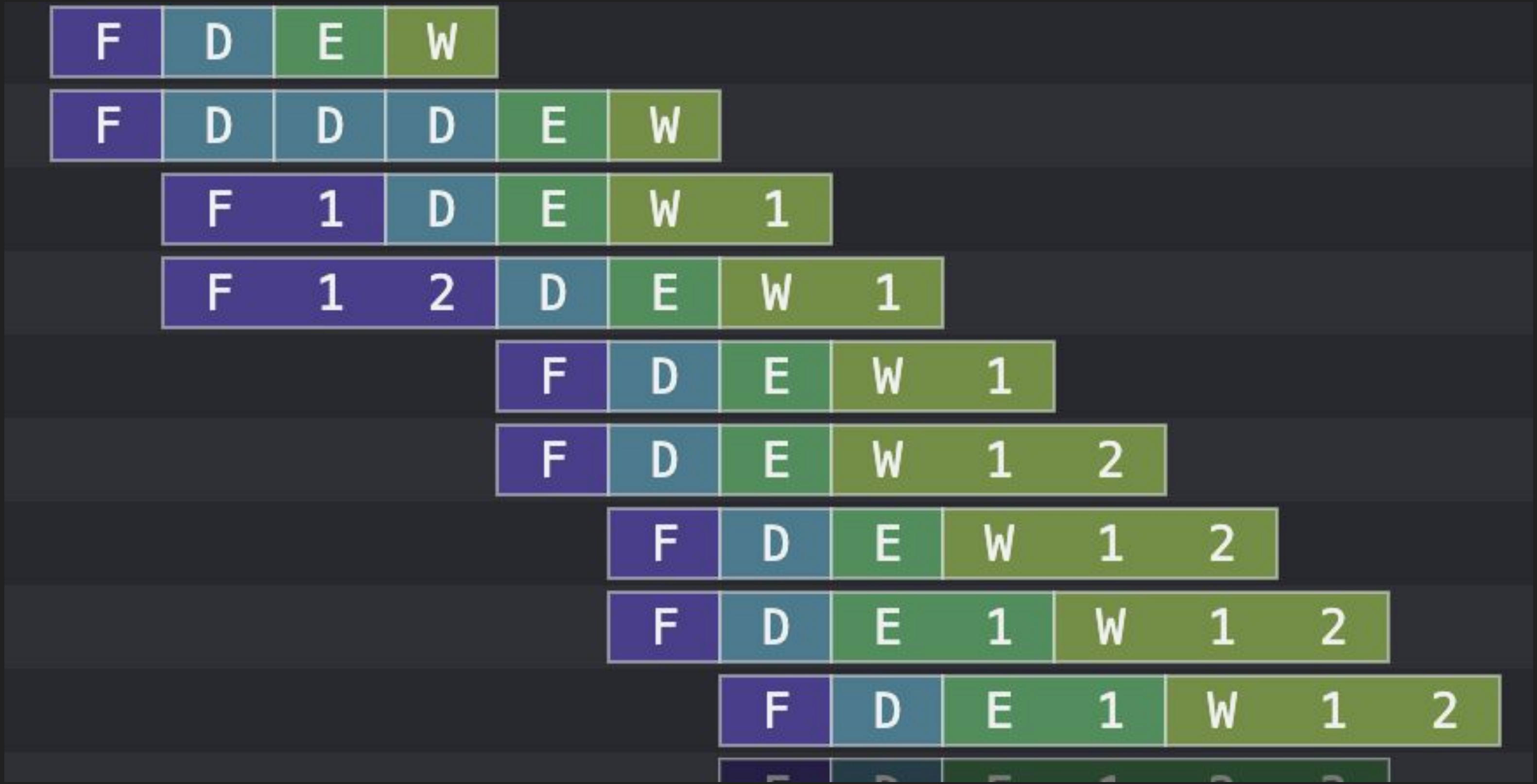
- Still working on instruction associated with PC 106C, but just gets confused and thinks its

How’d we fix it?

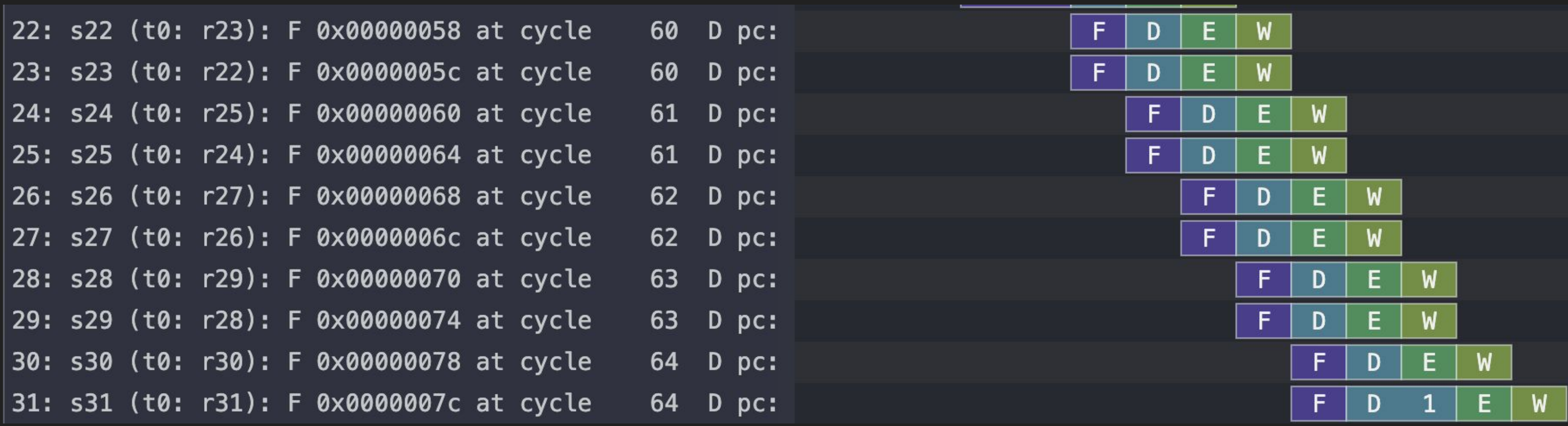


```
if(from_fetch.pc != resp.addr)begin
// $display("from_fetch1: ", fshow(f2d.first1()));
// $display("from_fetch2: ", fshow(from_fetch));
// $display("response1: ", fshow(fromImem.first1()));
// $display("response2: ", fshow(resp));
end else begin
```

Not Enough EPOCHS

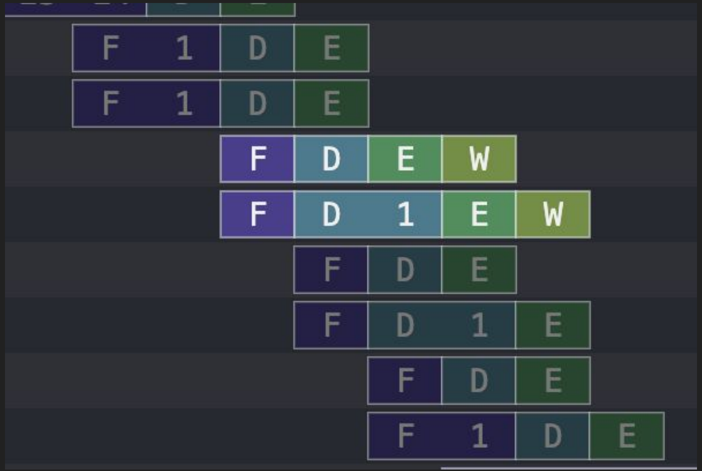


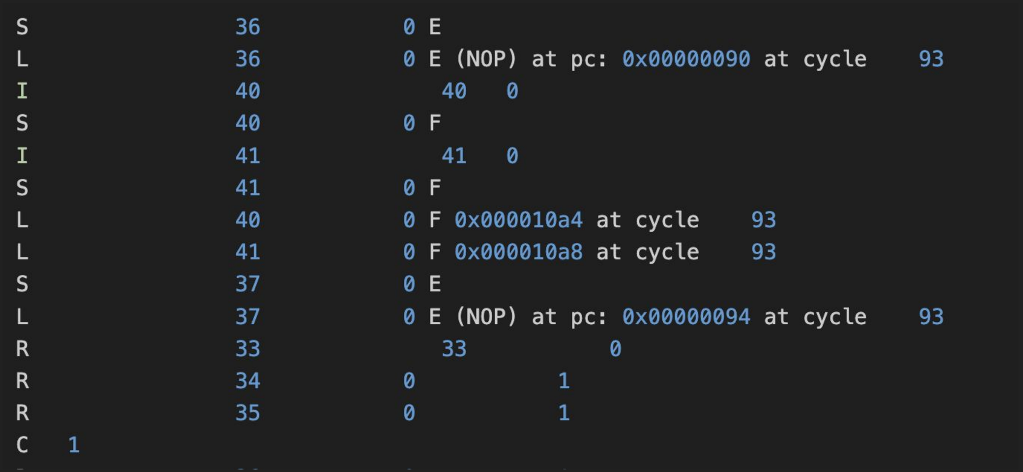
It Working



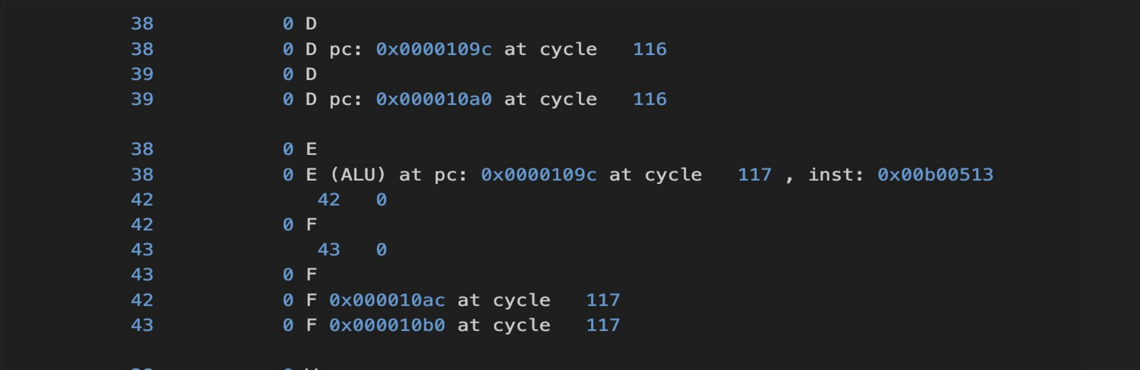
```
22: s22 (t0: r23): F 0x00000058 at cycle    60  D pc:
23: s23 (t0: r22): F 0x0000005c at cycle    60  D pc:
24: s24 (t0: r25): F 0x00000060 at cycle    61  D pc:
25: s25 (t0: r24): F 0x00000064 at cycle    61  D pc:
26: s26 (t0: r27): F 0x00000068 at cycle    62  D pc:
27: s27 (t0: r26): F 0x0000006c at cycle    62  D pc:
28: s28 (t0: r29): F 0x00000070 at cycle    63  D pc:
29: s29 (t0: r28): F 0x00000074 at cycle    63  D pc:
30: s30 (t0: r30): F 0x00000078 at cycle    64  D pc:
31: s31 (t0: r31): F 0x0000007c at cycle    64  D pc:
```

Look at it Working Even more

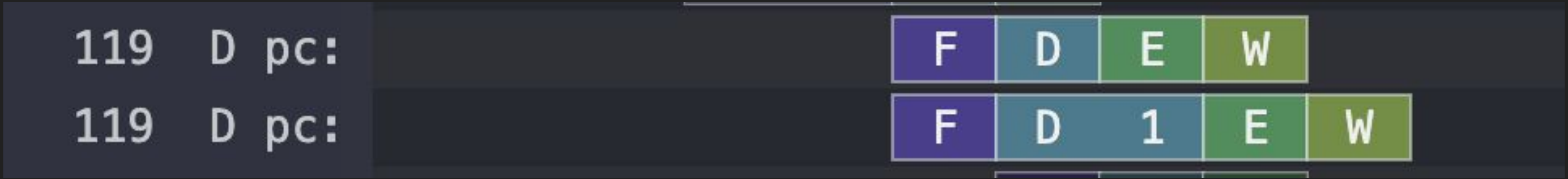




```
S               36          0 E
L               36          0 E (NOP) at pc: 0x00000090 at cycle    93
I               40            40   0
S               40          0 F
I               41            41   0
S               41          0 F
L               40          0 F 0x000010a4 at cycle    93
L               41          0 F 0x000010a8 at cycle    93
S               37          0 E
L               37          0 E (NOP) at pc: 0x00000094 at cycle    93
R               33            33          0
R               34          0           1
R               35          0           1
C   1
```



```
38          0 D
38          0 D pc: 0x0000109c at cycle   116
39          0 D
39          0 D pc: 0x000010a0 at cycle   116

38          0 E
38          0 E (ALU) at pc: 0x0000109c at cycle   117 , inst: 0x00b00513
42            42   0
42          0 F
43            43   0
43          0 F
42          0 F 0x000010ac at cycle   117
43          0 F 0x000010b0 at cycle   117
```



```
0 E
0 E (ALU) at pc: 0x0000109c at cycle
   42   0
0 F
   43   0
0 F
0 F 0x000010ac at cycle   117
0 F 0x000010b0 at cycle   117

0 W
0  W at cycle   118
0 E
0 E (CTRL) at pc: 0x000010a0 at cycle
0 D
0 D pc: 0x000010a4 at cycle   118
0 D
0 D pc: 0x000010a8 at cycle   118
```

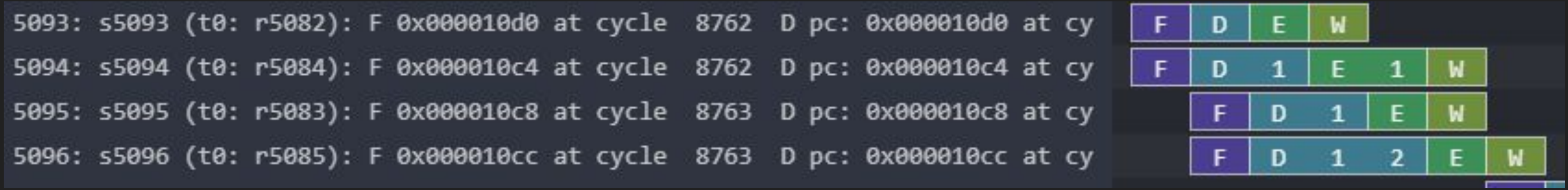
Branch Prediction: BTB

-One BTB for the entire superscalar processor

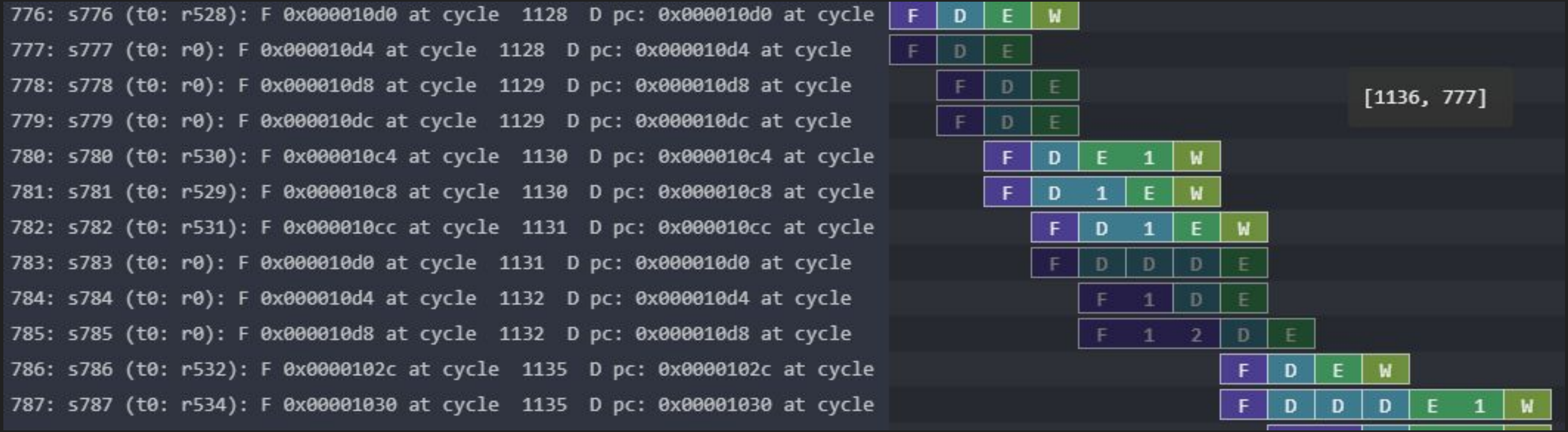
-When pc and BTB-predicted ppc are on the same cache line, access both from the cache

-Otherwise, do single instruction

BTB working



```
5093: s5093 (t0: r5082): F 0x000010d0 at cycle  8762  D pc: 0x000010d0 at cy
5094: s5094 (t0: r5084): F 0x000010c4 at cycle  8762  D pc: 0x000010c4 at cy
5095: s5095 (t0: r5083): F 0x000010c8 at cycle  8763  D pc: 0x000010c8 at cy
5096: s5096 (t0: r5085): F 0x000010cc at cycle  8763  D pc: 0x000010cc at cy
```



```
776: s776 (t0: r528): F 0x000010d0 at cycle  1128  D pc: 0x000010d0 at cycle
777: s777 (t0: r0): F 0x000010d4 at cycle  1128  D pc: 0x000010d4 at cycle
778: s778 (t0: r0): F 0x000010d8 at cycle  1129  D pc: 0x000010d8 at cycle
779: s779 (t0: r0): F 0x000010dc at cycle  1129  D pc: 0x000010dc at cycle
780: s780 (t0: r530): F 0x000010c4 at cycle  1130  D pc: 0x000010c4 at cycle
781: s781 (t0: r529): F 0x000010c8 at cycle  1130  D pc: 0x000010c8 at cycle
782: s782 (t0: r531): F 0x000010cc at cycle  1131  D pc: 0x000010cc at cycle
783: s783 (t0: r0): F 0x000010d0 at cycle  1131  D pc: 0x000010d0 at cycle
784: s784 (t0: r0): F 0x000010d4 at cycle  1132  D pc: 0x000010d4 at cycle
785: s785 (t0: r0): F 0x000010d8 at cycle  1132  D pc: 0x000010d8 at cycle
786: s786 (t0: r532): F 0x0000102c at cycle  1135  D pc: 0x0000102c at cycle
787: s787 (t0: r534): F 0x00001030 at cycle  1135  D pc: 0x00001030 at cycle
```

BTB working, but……?

-Tried superscaling pc and pc+4 if pc is not the last in a line (so BTB superscale -> +4 superscale -> BTB), leads to mostly worse performance than superscalar alone

-Different BTB geometry (128 line, 512 line, 2048 line) does not lead to much change in the cycle count

-Mysterious SuperFIFO anomaly significantly hinders BTB debugging

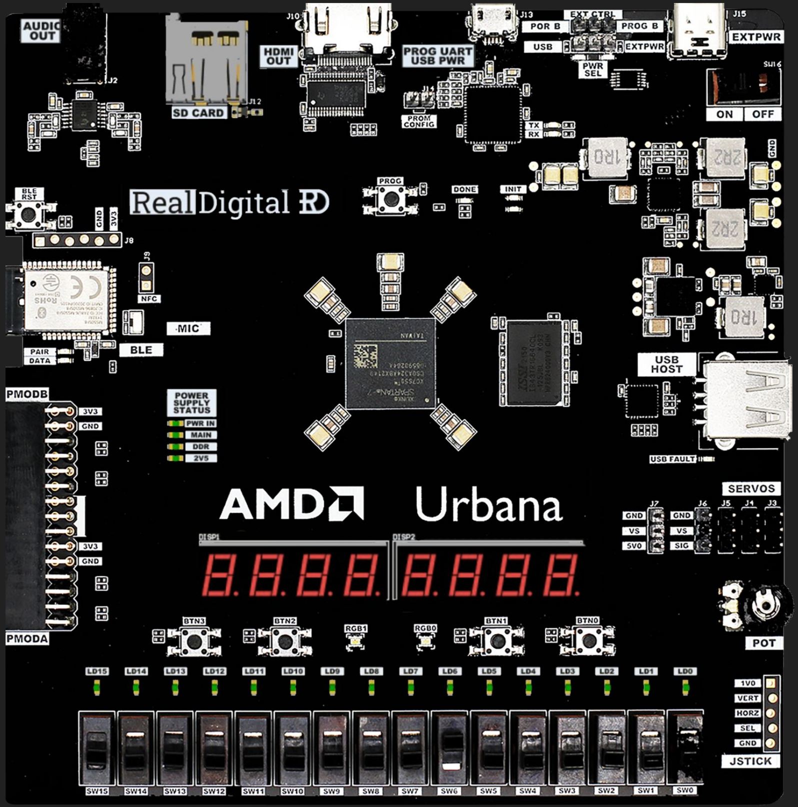
Performance Comparison

	Pipelined	Superscalar	Superscalar + BTB
Add32	226	208	224
The Lie	52445	46690	43779
Morse	19487	19202	21269
Mat Mul	1235074	1084655	1061193
MNIST	646925	630117	648158

Note: Kotana and Debug statements not necessarily removed. Compiling on different platforms lead to different numbers.

FPGA!

- Same FPGA from our beloved class 6.205 (digital systems laboratory)
- Attempted to get our processor running on the FPGA, mapping MMIO to the actual MMIO on the FPGA



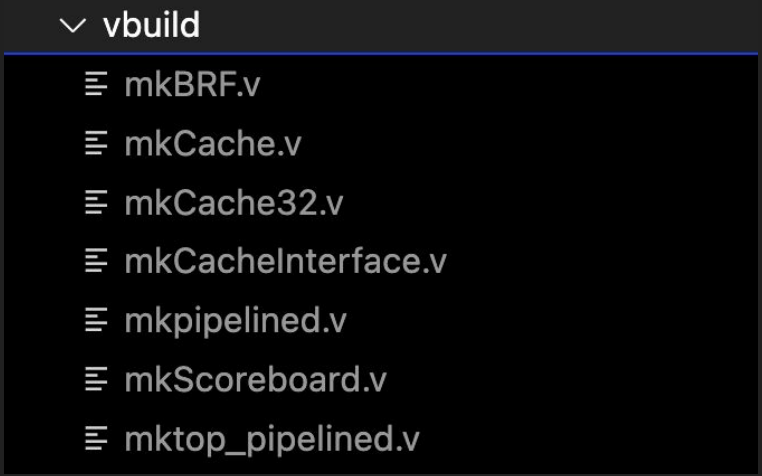
The clickbait pt. 2

“Just tell bsc to output Verilog files and drop it on an FPGA”

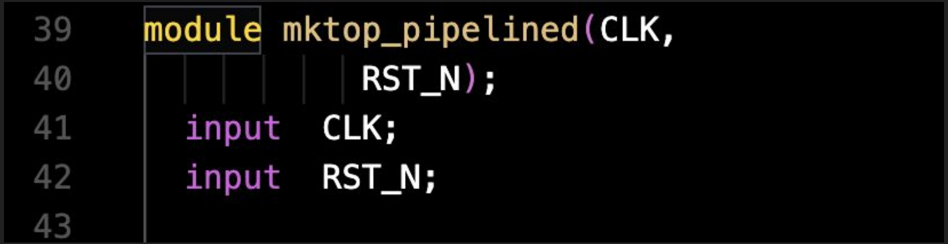
- ….yeah right

Step 1: Add the compiler flag to generate the files

- Just add –verilog
- Realize that mktop_pipelined.v is basically useless cuz there is no I/O
- Make a top_level.sv that wraps around an instance of mktop_pipelined



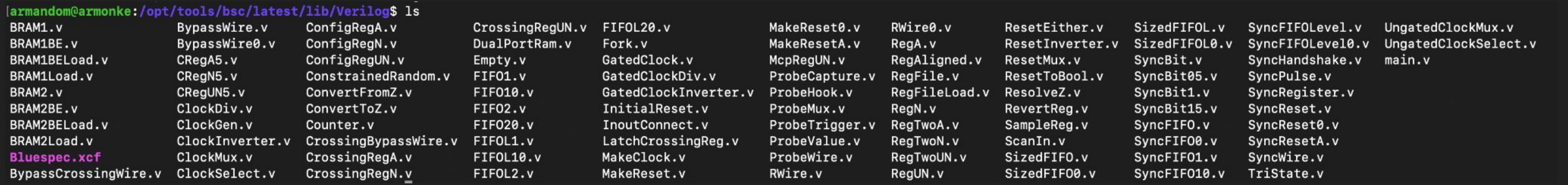
```
vbuild
    mkBRF.v
    mkCache.v
    mkCache32.v
    mkCacheInterface.v
    mkpipelined.v
    mkScoreboard.v
    mktop_pipelined.v
```



```verilog
module mktop_pipelined(CLK,
                       RST_N);
  input  CLK;
  input  RST_N;

```

Step 2: Resolve dependencies

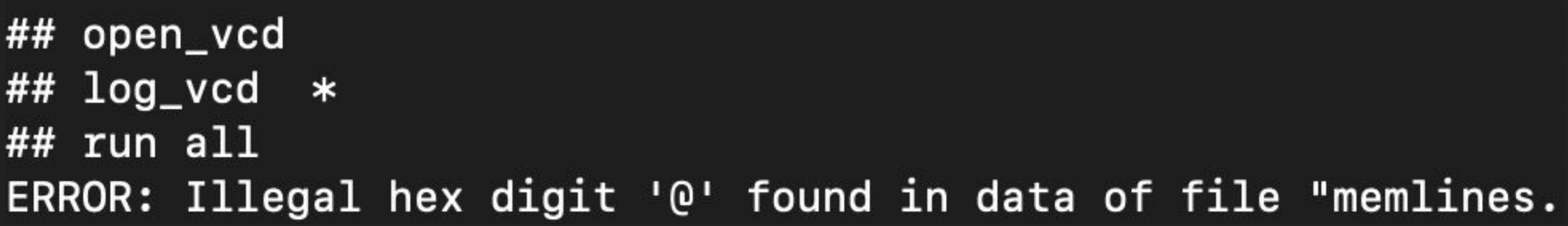
- Steal the tcl script from 6.111 to your bidding (building)
- Get lots of fun errors about all these modules that the bluespec compiler expected you to have already
- Do some unnecessarily long digging to find that they are implemented in the depths of the bluespec compiler directory
- Copy them to your project and claim them as your own (just kidding)



```
armandom@armonke:/opt/tools/bsc/latest/lib/Verilog$ ls
BRAM1.v                BypassWire.v      ConfigRegA.v             CrossingRegUN.v  FIFOL20.v                MakeReset0.v     RWire0.v        ResetEither.v    SizedFIFOL.v   SyncFIFOLevel.v   UngatedClockMux.v
BRAM1BE.v              BypassWire0.v     ConfigRegN.v             DualPortRam.v    Fork.v                   MakeResetA.v     RegA.v          ResetInverter.v  SizedFIFOL0.v  SyncFIFOLevel0.v  UngatedClockSelect.v
BRAM1BELoad.v          CRegA5.v          ConfigRegUN.v            Empty.v          GatedClock.v             McpRegUN.v       RegAligned.v    ResetMux.v       SyncBit.v      SyncHandshake.v   main.v
BRAM1Load.v            CRegN5.v          ConstrainedRandom.v      FIFO1.v          GatedClockDiv.v          ProbeCapture.v   RegFile.v       ResetToBool.v    SyncBit05.v    SyncPulse.v
BRAM2.v                CRegUN5.v         ConvertFromZ.v           FIFO10.v         GatedClockInverter.v     ProbeHook.v      RegFileLoad.v   ResolveZ.v       SyncBit1.v     SyncRegister.v
BRAM2BE.v              ClockDiv.v        ConvertToZ.v             FIFO2.v          InitialReset.v           ProbeMux.v       RegN.v          RevertReg.v      SyncBit15.v    SyncReset.v
BRAM2BELoad.v          ClockGen.v        Counter.v                FIFO20.v         InoutConnect.v           ProbeTrigger.v   RegTwoA.v       SampleReg.v      SyncFIFO.v     SyncReset0.v
BRAM2Load.v            ClockInverter.v   CrossingBypassWire.v     FIFOL1.v         LatchCrossingReg.v       ProbeValue.v     RegTwoN.v       ScanIn.v         SyncFIFO0.v    SyncResetA.v
Bluespec.xcf           ClockMux.v        CrossingRegA.v           FIFOL10.v        MakeClock.v              ProbeWire.v      RegTwoUN.v      SizedFIFO.v      SyncFIFO1.v    SyncWire.v
BypassCrossingWire.v   ClockSelect.v     CrossingRegN.v           FIFOL2.v         MakeReset.v              RWire.v          RegUN.v         SizedFIFO0.v     SyncFIFO10.v   TriState.v
```

Step 3: Try to simulate using the verilog files

- Get disappointed when build completes but nothing happens
- Try simulating using the verilog files, get greeted with tiny compiler errors



```
## open_vcd
## log_vcd  *
## run all
ERROR: Illegal hex digit '@' found in data of file "memlines.
```

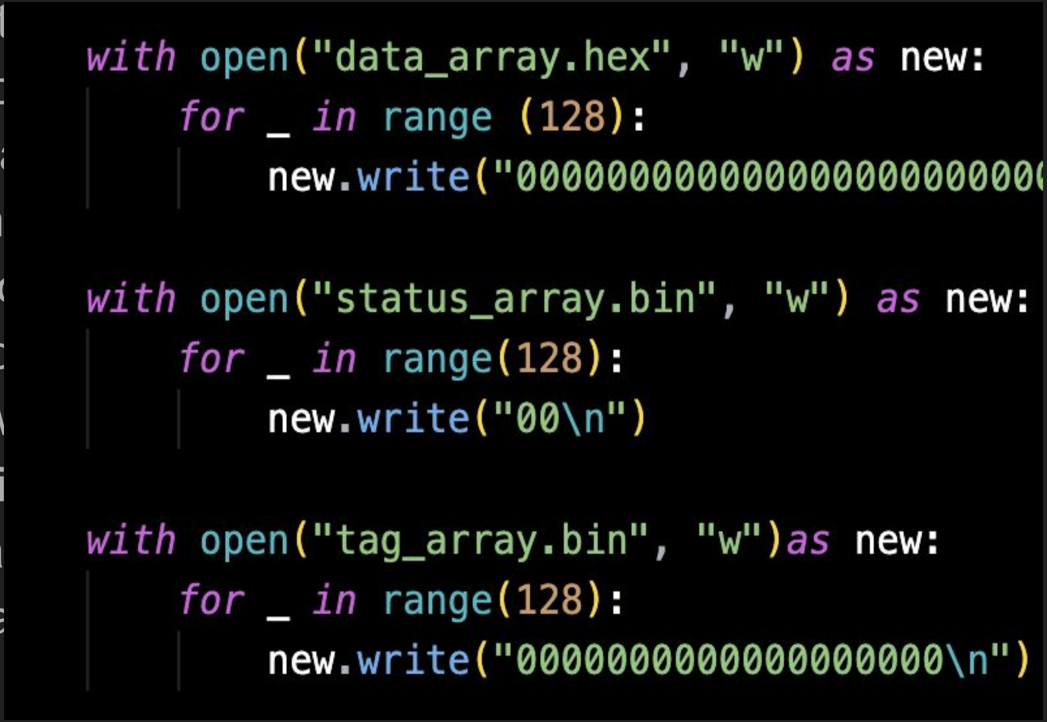
- Make a python script to generate the entire memory files

Step 4: The last 2 fundamental errors

- Still nothing works
- Dig into the generated verilog files…(yuck)
- Find that reset is active LOW despite the rest of the system as active HIGH
- Last, nothing was STILL happening
- Turns out, initializing BRAM with one 0 per line doesn’t really work. Breaks silently as no compiler warnings were given
- State after lookups was “x”
- Generated more .hex and .bin files via python…

Step 4: The last 2 fundamental errors

- Still not
- Dig into
- Find tha
  system
- Last, no
- Turns o
  really w
  were gi
- State a
- Genera

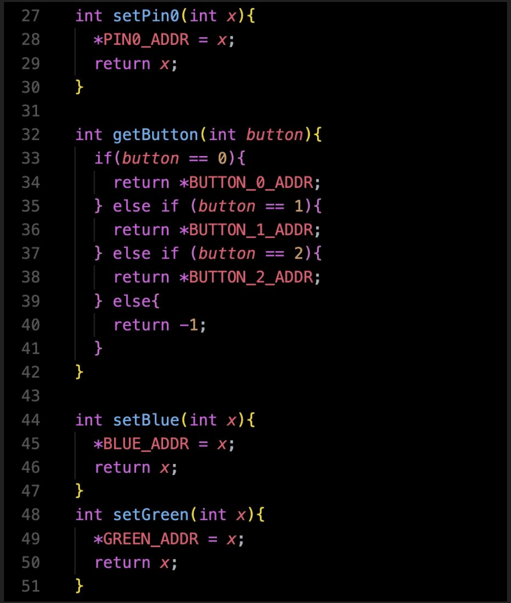


```
with open("data_array.hex", "w") as new:
    for _ in range (128):
        new.write("000000000000000000000000
with open("status_array.bin", "w") as new:
    for _ in range(128):
        new.write("00\n")

with open("tag_array.bin", "w")as new:
    for _ in range(128):
        new.write("0000000000000000000\n")
```

MMIO

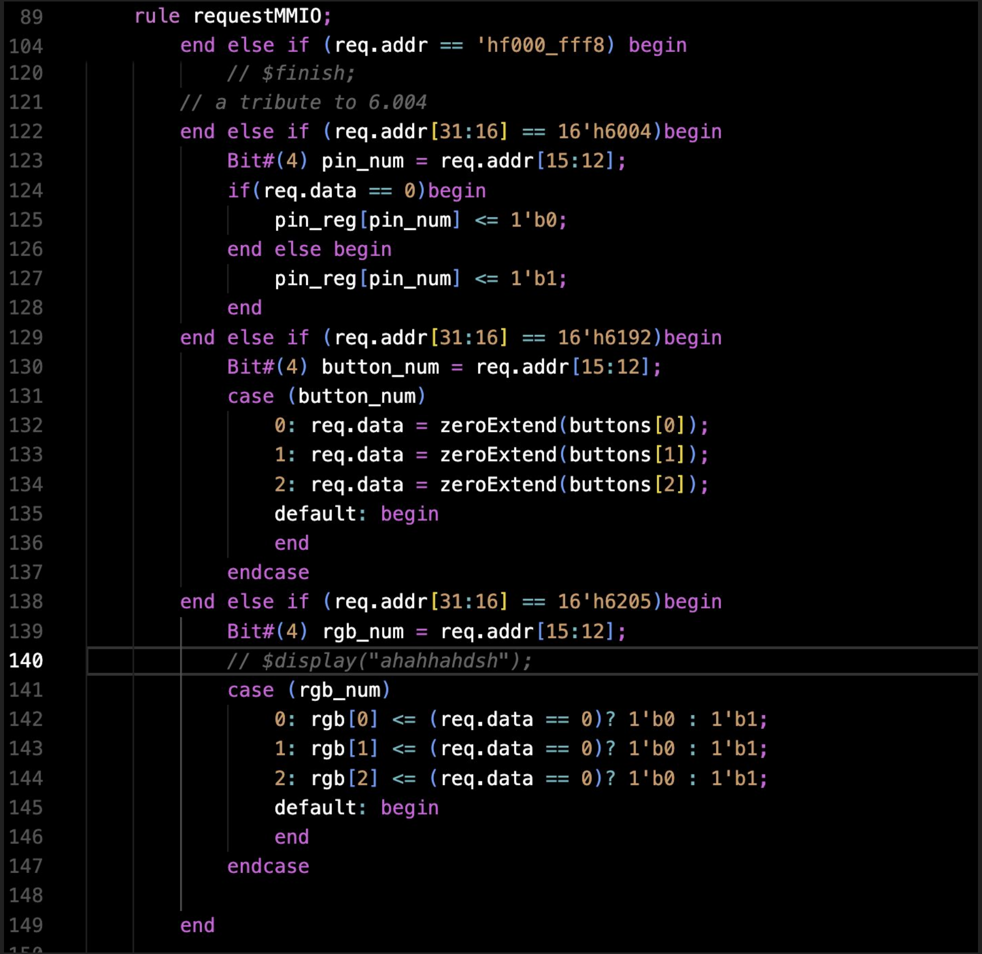
- The structure they gave us was actually pretty good

mmio.c



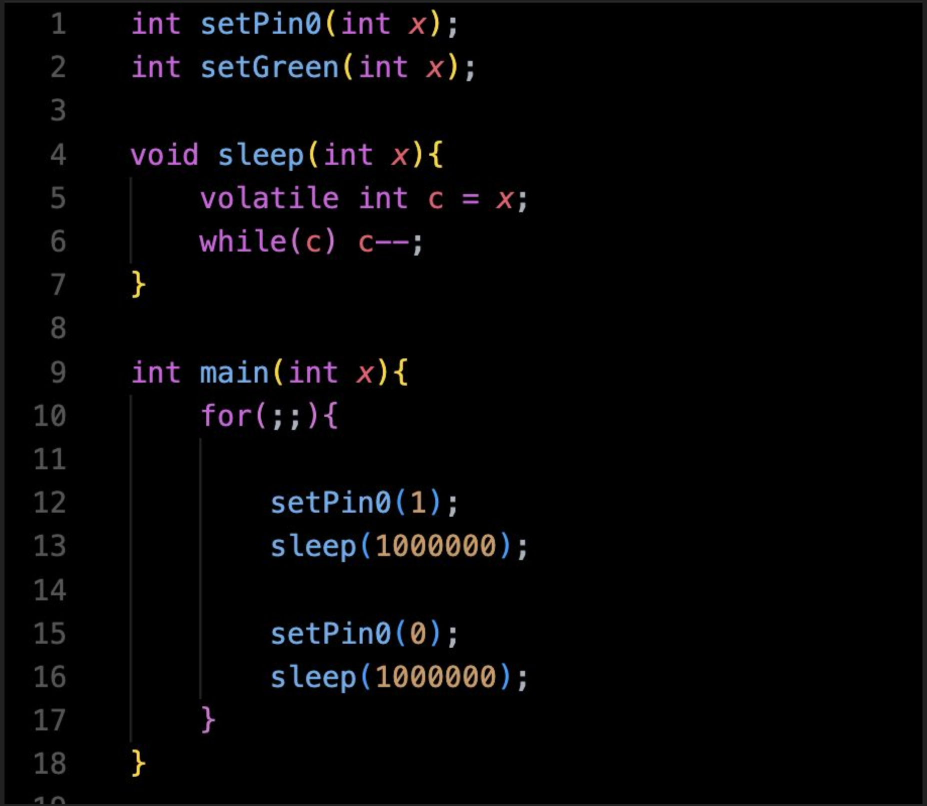
```c
int setPin0(int x){
  *PIN0_ADDR = x;
  return x;
}

int getButton(int button){
  if(button == 0){
    return *BUTTON_0_ADDR;
  } else if (button == 1){
    return *BUTTON_1_ADDR;
  } else if (button == 2){
    return *BUTTON_2_ADDR;
  } else{
    return -1;
  }
}

int setBlue(int x){
  *BLUE_ADDR = x;
  return x;
}
int setGreen(int x){
  *GREEN_ADDR = x;
  return x;
}
```

top_superscalar.bsv



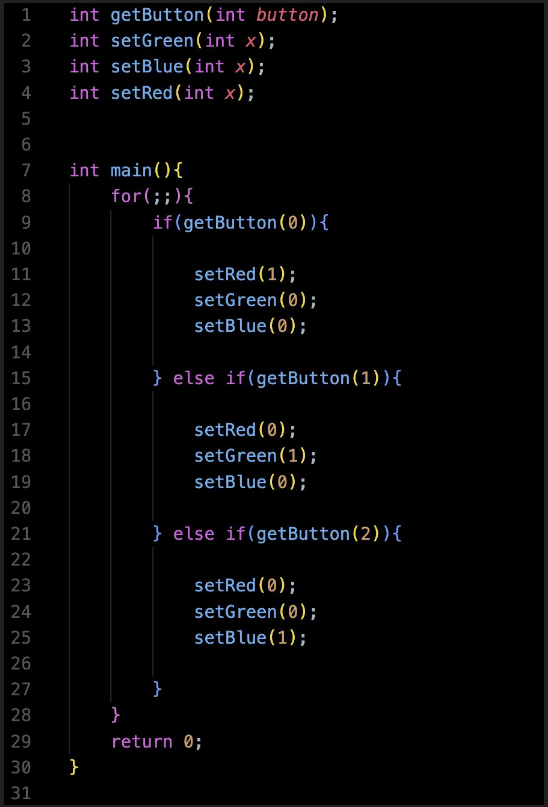
```
    rule requestMMIO;
      end else if (req.addr == 'hf000_fff8) begin
        // $finish;
      // a tribute to 6.004
      end else if (req.addr[31:16] == 16'h6004)begin
        Bit#(4) pin_num = req.addr[15:12];
        if(req.data == 0)begin
          pin_reg[pin_num] <= 1'b0;
        end else begin
          pin_reg[pin_num] <= 1'b1;
        end
      end else if (req.addr[31:16] == 16'h6192)begin
        Bit#(4) button_num = req.addr[15:12];
        case (button_num)
          0: req.data = zeroExtend(buttons[0]);
          1: req.data = zeroExtend(buttons[1]);
          2: req.data = zeroExtend(buttons[2]);
          default: begin
          end
        endcase
      end else if (req.addr[31:16] == 16'h6205)begin
        Bit#(4) rgb_num = req.addr[15:12];
        // $display("ahahhahdsh");
        case (rgb_num)
          0: rgb[0] <= (req.data == 0)? 1'b0 : 1'b1;
          1: rgb[1] <= (req.data == 0)? 1'b0 : 1'b1;
          2: rgb[2] <= (req.data == 0)? 1'b0 : 1'b1;
          default: begin
          end
        endcase

      end
```

Demo #1 - Using the GPIO!



```c
int setPin0(int x);
int setGreen(int x);

void sleep(int x){
    volatile int c = x;
    while(c) c--;
}

int main(int x){
    for(;;){

        setPin0(1);
        sleep(1000000);

        setPin0(0);
        sleep(1000000);
    }
}
```

Demo #2 - input and output



```c
int getButton(int button);
int setGreen(int x);
int setBlue(int x);
int setRed(int x);


int main(){
    for(;;){
        if(getButton(0)){

            setRed(1);
            setGreen(0);
            setBlue(0);

        } else if(getButton(1)){

            setRed(0);
            setGreen(1);
            setBlue(0);

        } else if(getButton(2)){

            setRed(0);
            setGreen(0);
            setBlue(1);

        }
    }
    return 0;
}
```

Demo #3 - MNIST prediction

```
     #####
 ##########
 ##########
#####
###
###       ##
####    ###
  #### ####
   #######
    #####
   ######
  ########
  ####  ##
  ###    ##
 ###     ##
 ###     ##
  ##    ####
  #########
  ########
    #####
```

```
     ######
   ########   #
  ####      ###
 ###       ####
###       ####
 ##     ######
 ############
  ######  ###
          ##
         ###
         ##
         ##
         ##
        ##
        ##
        ##
       ##
       ##
       #
       #
```

Future Improvements

-It seems very doable to do 1 mem then 1 ALU, it seems incredibly doable, but for the sake of meeting a deadline we didn’t

-FIX THE SUPERFIFO: under edge cases when superscaling after a squashed command, the SUPERFIFO may end up dequeuing the later inserted element first

-Investigate a better combination of branch predictors, namely the BHB, and try to find a way to not harm