ECE 720 – ESL & Physical Design

Project 2 Requirements

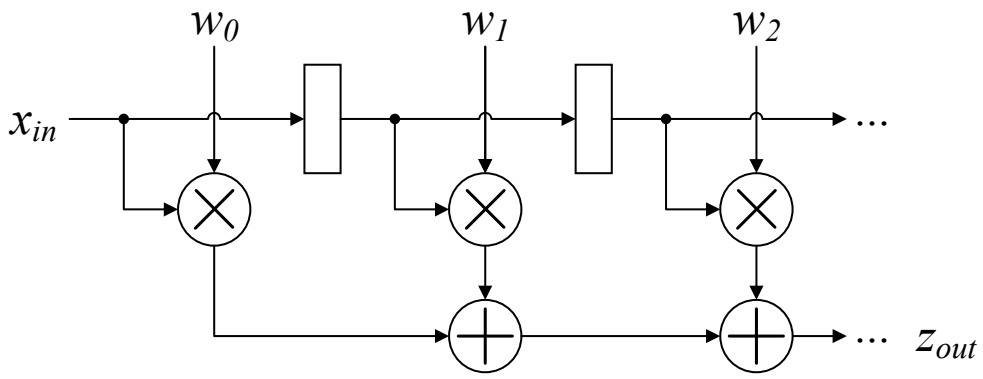
W. Rhett Davis
NC State University

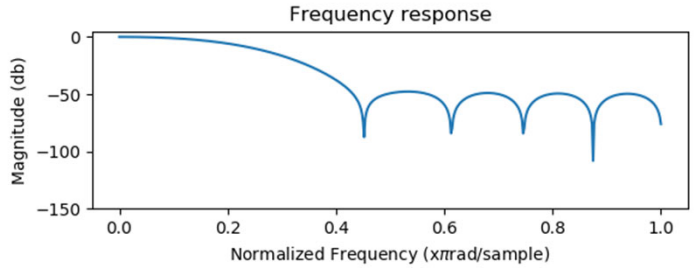
Project Goals

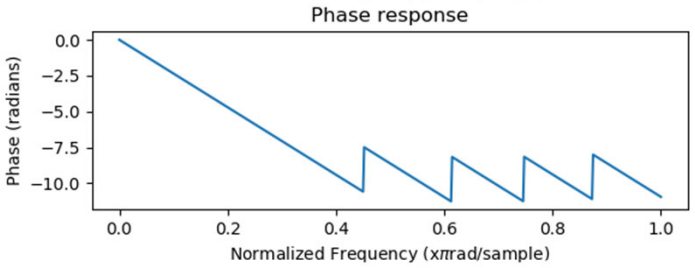
- Design a Finite Impulse Response (FIR) Filter Accelerator for use in an SoC
- Partition the computation between hardware and software
- Show that the Accelerator can be synthesized with HLS
- Examine the speedup and area overhead of the relative to a software-only implementation

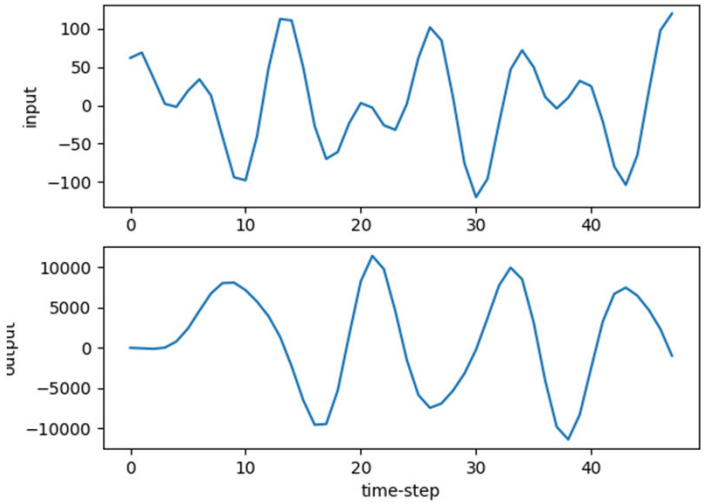
FIR Details

- Test-case for two 16-tap FIR filter operations is provided
- Conceptual diagram below
- Example code given in rocket_sim/fir.c





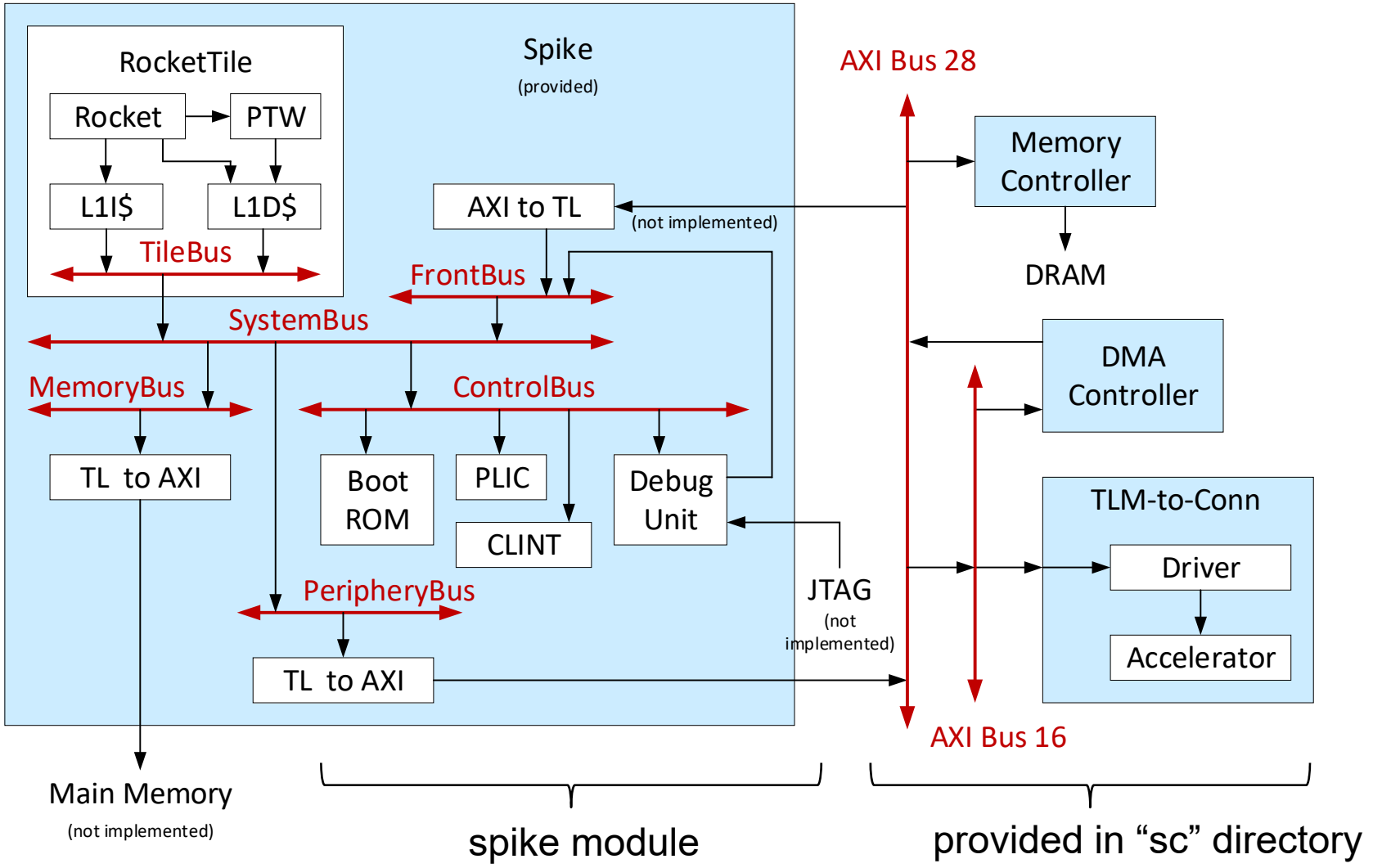




Project Goals (continued)

- Create a Simulation of an SoC with an Accelerator processing 16 time-steps of an FIR filter at a time
  - Assume that inputs (X) and weights (W) are stored in DRAM before the simulation begins
  - C-program running on CPU tells a DMA unit where to find the inputs and where to write the outputs, then uses burst transactions to read values from memory and write to the Accelerator
  - After completing two filter operations, C-program running on CPU checks the error of the final values

ISS+TLM+HLS Example System



fir.c Starting Point

- Example program is provided that performs this entire operation in a single C-program
- Inputs & weights read from DRAM model in SystemC (memctl module)
  - » Commented code in C-program also has these values, but do not use them
- Nested for-loops implement the FIR filter computation
  - » You must move this into your Accelerator module
- Final error calculation remains in the C-program
- Example code shows transfers to & from Accelerator module with & without using the dma module

fir.c main filter ops (1/2)

```
int main( int argc, char* argv[] )
{

  int n,m;
  volatile short *coef=(short *)0x60004000;
  volatile short *input=(short *)0x60002000;
  // Uncomment the next line to avoid memory controller accesses
  // short output[TSTEP1+TSTEP2];
  volatile short *output=(short *)0x60001000;
  short error,total_error;

  // First FIR filter operation
  for (n=0; n<TSTEP1; n++) {
    output[n]=0;
    for (m=0; m<TAPS; m++) {
      if (n+m-TAPS+1 >= 0) {
        output[n]+=coef[m]*input[n+m-TAPS+1];
      }
    }
  }
```

fir.c main filter ops (2/2)

```
// Second FIR filter operation
 for (n=0; n<TSTEP2; n++) {
   output[TSTEP1+n]=0;
   for (m=0; m<TAPS; m++) {
     if (n+m-TAPS+1 >= 0) {
       output[TSTEP1+n]+=coef[TAPS+m]*input[TSTEP1+n+m-TAPS+1];
     }
   }
 }

 // Error check for both FIR filter operations
 total_error=0;
 for (n=0; n<(TSTEP1+TSTEP2); n++) {
   error=expected[n]-output[n];     // Error for this time-step
   total_error+=(error<0)?(-error):(error);  // Absolute value
 }
 printf("cpu main FIR total error: %d\n",total_error);
```

Memory Organization

Addr (dec)	Addr (hex)	Size	Name	j	Filter
4096	1000	2	OUT	0	1
4098	1002	2	OUT	1	1
4156	103C	2	OUT	30	1
4158	103E	2	OUT	31	1
4160	1040	2	OUT	0	2
4162	1042	2	OUT	1	2
4252	109C	2	OUT	46	2
4254	109E	2	OUT	47	2
8192	2000	2	IN	0	1
8194	2002	2	IN	1	1
8252	203C	2	IN	30	1
8254	203E	2	IN	31	1
8256	2040	2	IN	0	2
8258	2042	2	IN	1	2
8348	209C	2	IN	46	2
8350	209E	2	IN	47	2
16384	4000	2	COEF	0	1
16386	4002	2	COEF	1	1
16412	401C	2	COEF	14	1
16414	401E	2	COEF	15	1
16416	4020	2	COEF	0	2
16418	4022	2	COEF	1	2
16444	403C	2	COEF	14	2
16446	403E	2	COEF	15	2

- memctl base address is 0x60000000 (cpu) or 0x0 (sc)
- Input (x) and Coefficient (w) values loaded at start of simulation
- Store your output in the specified locations
- This table also available in Memory Map spreadsheet, posted on web-page

DMA Controller

- DMA base address is 0x70000000 (cpu) or 0x10000000 (sc)

Addr (dec)	Addr (hex)	Size	Name	Description
0	0	8	ST	Status Register
8	8	8	CTRL	Control Register
16	10	8	SR	Source Register
24	18	8	DR	Destination Register
32	20	8	LEN	Length Register - Number of bytes to copy

- Write source/desination pointers to SR/DR
- Writing length to LEN begins transfer
- CPU execution will stall until transfer is complete
- ST & CTRL currently unused

Accelerator Interface (1/2)

- Accelerator base address is 0x70010000 (cpu) or 0x10010000 (sc)
- 0x0 ST Status Register
  - sc_out<sc_uint<8>> port (writes immediately visible)
  - Writes not supported
  - Use to indicate status of your Accelerator, as needed
- 0x1 CTRL Register
  - Use for commands to your Accelerator, as needed
  - Connections::In<sc_uint<8>> port, 1-entry FIFO
  - Must be popped before next write, or next write will stall
  - Writing 0x0F ends the simulation immediately
  - Reads not supported

Addr (dec)	Addr (hex)	Size	Name
0	0	8	ST
8	8	8	CTRL
16	10	32	COEF/W
48	30	32	IN/X
80	50	32	OUT/Z

Accelerator Interface (2/2)

Addr (dec)	Addr (hex)	Size	Name
0	0	8	ST
8	8	8	CTRL
16	10	32	COEF/W
48	30	32	IN/X
80	50	32	OUT/Z

- 0x10 COEF/W
- 0x30 IN/X
  - Connections::In<sc_uint<64>> ports named w_in & x_in, 4-entry FIFOs
  - Writes of 8, 16, 24, or 32 bytes supported
  - Must be popped before additional writes, or next write will stall
  - Reads not supported
- 0x50 OUT/Z
  - Connections::Out<sc_uint<64>> port z_out, 4-entry FIFO
  - Reads of 8, 16, 24, or 32 bytes supported
  - Must be filled, or read will stall
  - Writes not supported

Example Accelerator

```
void run()  {
  ...
  while (1)
  {
    if (!ctrl_in.Empty()) {
      ctrl=ctrl_in.Pop();
      st_out.write(ctrl);
    }
    if (!w_in.Empty()) {
      data=w_in.Pop();
      z_out.Push(data);
    }
    if (!x_in.Empty()) {
      data=x_in.Pop();
      z_out.Push(data);
    }
    wait();
  }
}
```

- Writes to CTRL looped-back to ST
  - No need to read these values
- Writes to either COEF/W or IN/X looped-back to OUT/Z
  - Must be read after 32 bytes are written, or next write will stall
- Because multiple writes to z_out in one while-loop iteration, PIPELINE_INIT_INTERVAL directive must be at least 2
  - Otherwise, will get error "could not schedule even with unlimited resources"

Example Transactions (1/5)

- Spike will send all transactions to addresses 0x60000000- 0x7FFFFFFF to SystemC TLM Simulation
- Note these addresses are mapped to 0x00000000- 0x1FFFFFFF
- fir.c defines includes several example transactions
- variables defined for more convenient reads & writes

```
volatile long long *dma_st     = (volatile long long*)  0x70000000;
volatile long long **dma_sr    = (volatile long long**) 0x70000010;
volatile long long **dma_dr    = (volatile long long**) 0x70000018;
volatile long long *dma_len    = (volatile long long*)  0x70000020;
volatile long long *accel_st   = (volatile long long*)  0x70010000;
volatile long long *accel_ctrl = (volatile long long*)  0x70010008;
volatile long long *accel_w    = (volatile long long*)  0x70010010;
volatile long long *accel_x    = (volatile long long*)  0x70010030;
volatile long long *accel_z    = (volatile long long*)  0x70010050;
```

Example Transactions (2/5)

- First DMA Transfer into Accelerator

write xacts

```
*dma_sr=(volatile long long*)((long)input & 0x1fffffff);
*dma_dr=(volatile long long*)((long)accel_x & 0x1fffffff);
*dma_len=32; // starts transfer
clobber();
```

- Output

```
175888 ns dma0 WRITE len:0x8 addr:0x10 data:0x0000000000002000
175890 ns dma0 WRITE len:0x8 addr:0x18 data:0x0000000010010030
175892 ns dma0 WRITE len:0x8 addr:0x20 data:0x0000000000000020
175892 ns dma0 transfer READ addr:0x2000
175992 ns dma0 transfer READ Complete
175992 ns dma0 transfer WRITE addr:0x10010030
175992 ns tlm2conn WRITE len:0x20 addr:0x30
175992 ns tlm2conn.driver WRITE addr=0x30 length=0x20 data=0x200240045003e
175992 ns tlm2conn.driver WRITE addr=0x30 length=0x20 data=0xd00220013fffe
175993 ns tlm2conn.driver WRITE addr=0x30 length=0x20 data=0xffd8ff9effa2ffd7
175994 ns tlm2conn.driver WRITE addr=0x30 length=0x20 data=0x32006f00710031
175995 ns tlm2conn transaction complete
175995 ns dma0 transfer WRITE Complete
```

Example Transactions (3/5)

- First DMA Transfer out of Accelerator + Error Check

write xacts

```
*dma_sr=(volatile long long*)((long)accel_z & 0x1fffffff);
*dma_dr=(volatile long long*)((long)output & 0x1fffffff);
*dma_len=32; // starts transfer
clobber();

total_error=0;
for (n=0; n<TAPS; n++) {
  error=input[n]-output[n];  // Error for this time-step
  total_error+=(error<0)?(-error):(error);  // Absolute value
  //printf("cpu main n: %d input: 0x%x output 0x%x\n",
  //        n,input[n],output[n]);
}
printf("cpu main DMA transfer 1 total error: %d\n",total_error);
```

read xacts

- Should see that “error: 0” in program output

Compiler Barrier

- clobber() is a compiler-barrier function
- Controls compiler instruction re-ordering while allowing high-performance benchmarking
- I learned it from here: https://youtu.be/nXaxk27zwlk

```
static void clobber() {
  asm volatile ("" : : : "memory");
}
```

- Compiler will not insert any code in the binary where this function is called, BUT it will prevent the compiler from moving VOLATILE reads or writes before or after the call.
- Must call this after writing DMA LEN to prevent reordering at the second DMA transfer from corrupting the first
- Could also be needed before writing DMA LEN, but I have not yet found this to be necessary

Example Transactions (4/5)

- 3 tests of writes to Accelerator CTRL (sizes long long, short, and char) to show values appearing in ST register
- Fourth Read shows that there is no stall, even w/o write to CTRL
- Should see that “error: 0” in program output
- Note 2 printf() statements per test, gives time for CTRL to reach ST

```
*accel_ctrl=2;
printf("cpu main accel_ctrl write/read long long test error: ");
printf("%d\n",2-*accel_st);
volatile short *sp=(volatile short *)accel_ctrl;
*sp=3;
printf("cpu main accel_ctrl write/read short test error: ");
printf("%d\n",3-*accel_st);
volatile char *cp=(volatile char *)accel_ctrl;
*cp=4;
printf("cpu main accel_ctrl write/read char test 1 error: ");
printf("%d\n",4-*accel_st);
// Sim does not hang on next line, becauase st is signal, not fifo
printf("cpu main accel_ctrl write/read char test 2 error: %d\n",4-*accel_st);
```

Example Transactions (5/5)

- Last tests show examples of single long long word reads and writes to Accelrator
- Final line shows an example of how simulation can hang if fir.c attempts to read from an empty FIFO

```
*accel_w=0x133;
*accel_w=0x134;
*accel_w=0x135;
*accel_w=0x136;
printf("cpu main accel_w write + accel_z read test 1 error: %d\n",0x133-*accel_z);
printf("cpu main accel_w write + accel_z read test 2 error: %d\n",0x134-*accel_z);
printf("cpu main accel_w write + accel_z read test 3 error: %d\n",0x135-*accel_z);
printf("cpu main accel_w write + accel_z read test 4 error: %d\n",0x136-*accel_z);
// The next line would cause sim to hang, because no data is in z_fifo
//printf("cpu main accel_x write + accel_z read test 5 error: %d\n",
//        0x136-*accel_z);
```

Using Gnu Debugger

- If your simulation hangs, but you can't tell where, try using **gdb**
- Change "all:" target from "rel" to "dbg" in sc/Makefile and rebuild (adds debugging info into object files)
- Use command "make gdb" in rocket_sim directory
- Copy "r …" command into prompt, press "Enter"
- After simulation hangs, press CTRL-C
- Use the "bt" (backtrace) command to print the stack
- May need to try this several times to halt the simulation in your code

```
$ make gdb
echo Use the command \"r --isa=rv64gc -l fir.riscv\" to start gdb simulation
Use the command "r --isa=rv64gc -l fir.riscv" to start gdb simulation
gdb ../sc/main.x
GNU gdb (GDB) Red Hat Enterprise Linux 8.2-20.el8
...
Reading symbols from ../sc/main.x...(no debugging symbols found)...done.
(gdb) r --isa=rv64gc -l fir.riscv
```

Last Lines of fir.c Output

```
198027 ns tlm2conn.driver WRITE addr=0x8 length=0x8 data=0xf
198027 ns tlm2conn transaction complete
198027 ns tlm2conn received exit signal

Info: /OSCI/SystemC: Simulation stopped by user.
Simulation time: 198027 ns
Wall clock time: 1 seconds

real    0m0.296s
user    0m0.137s
sys     0m0.109s
```

- Your submission should show reduced simulation time (not including example transactions)
  - Simulation assumes 1ns of delay per instruction
  - Note that Synthesis of your Accelerator will likely show a critical path delay larger than 1ns, but do not change the simulated cycle time (for simplicity)

Requirements

- Design Objectives
  - Minimize the time needed to complete both filter operations (cycles*critical-path delay)
  - Do not waste area, *i.e.* do not spend area on things that do not improve the time to completion
- System Model Requirements
  - Accelerator is the only module that should be modified
  - Error must be zero
- Report Requirements
  - Compare the simulation time of your (HW-SW) design to the provided (SW-only) base-system
    - Use the Verilog (vsim) time, not the C++ (sc) time
    - Do not include the FIR error-check in this time

Requirements

- Report Requirements (continued)
  - Compare cycles-per-second performance of your design's C++ (sc) simulation to the Verilog (vsim) simulation
    - Do not include the FIR error-check in this time
    - Repeat the filter operations to get more meaningful time-measurements
  - Summarize the resources (func, reg, mem, etc.) used in your Accelerator and compare the total assumed area-score of the RocketTile (~1,500,000)
  - Discuss how you attempted to meet the design objectives
    - Focus on what may be unique about your design
  - Details claimed in report must match submitted code
  - Report should contain an introduction, body, and conclusion. You may use whatever section titles you like for the body

Project Grading

- Due Fri. Dec. 6 (25 days from today)

- (20%) Satisfying Design Objective (minimize cycles*critpath)
- (20%) Successful Execution of the simulation (zero error)
- (20%) Successful High-Level Synthesis & Verilog sim
- (20%) Report Completeness and Consistency
- (20%) Writing Quality – Emphasis on Clarity.
  Good grammar is necessary, but not sufficient

- Recall that Project 2 is 30% of the total grade

A Word About the Report

- Your paper is my only “window” into your project beyond the one simulation you submit.
- Your final conclusions may be excellent, but if the report is poorly written, superficial, and not comprehensive, I will not be able to fully appreciate how good it is.
- Originality is important
  - Do not share your code
  - You may share your results
  - Your results should differ