Stefan Sigurdsson
B01189238

ENGN 1640
LAB 6

In this lab you are required to substantially improve your original single-cycle processor design of lab05 by using pipelining.

The processor reads instructions from a register array on a program counter (increments by 1 instead of 4, same for the memory ram) and executes the desired instructions. The processor follows the ARM instruction set and architecture with the exception of the MUL function, which uses Rd as its destination register instead of Rn to match that of other instructions. The outputs of two multipliers (32 bit and 10 bit) are muxed together to speed up the factorial program while preserving full function of the 32 bit ARM processor. The processor is pipelined in 5 stages; F (Fetch), D (Decode), E1 (Execute 1), E2 (Execute 2), and MW (Memory read + Writeback). It employs full forwarding and hazard detection. The total logic and routing resources used for the factorial program are as follows:

Total logic elements 1,336 / 33,216 (4 %)
Total combinational functions 1,245 / 33,216 (4 %)
Dedicated logic registers 369 / 33,216 (1 %)
Total registers 369
Total pins 1 / 475 (< 1 %)
Total virtual pins 0
Total memory bits 4,096 / 483,840 (< 1 %)
Embedded Multiplier 9-bit elements 6 / 70 (9 %)
Total PLLs 1 / 4 (25 %)

The factorial program codes (assembly, binary) and modified palindrome checker codes:

FACTORIAL 6 (Store into 0)

```
MOV R0, #6
BL FACT
MOV R1, #0
STR R0, [R1]
STR R12, [R1, #1]
SWI 0x11
FACT: CMP R0, #1
MOVEQ PC, LR
```

```
SUB SP, SP, #2
STR LR, [SP, #1]
STR R0, [SP]
SUB R0, R0, #1
BL FACT
LDR R1, [SP]
LDR LR, [SP, #1]
ADD SP, SP, #2
MUL R0, R1, R0
MOV PC, LR
MOV R1, R0
SWI 0x11

Instructions[0] = 32'hE3A00006;
Instructions[1] = 32'hEB000003;
Instructions[2] = 32'hE3A01000;
Instructions[3] = 32'hE5810000;
Instructions[4] = 32'hE581C001;
Instructions[5] = 32'hFFFFFFFF;
Instructions[6] = 32'hE3500001;
Instructions[7] = 32'h01A0F00E;
Instructions[8] = 32'hE24DD002;
Instructions[9] = 32'hE58DE001;
Instructions[10] = 32'hE58D0000;
Instructions[11] = 32'hE2400001;
Instructions[12] = 32'hEBFFFFF8;
Instructions[13] = 32'hE59D1000;
Instructions[14] = 32'hE59DE001;
Instructions[15] = 32'hE28DD002;
Instructions[16] = 32'hE0000091;
Instructions[17] = 32'hE1A0F00E;
Instructions[18] = 32'hFFFFFFFF;
```

Palindrome checker (racecar, Replace ORR “|” with EOR “^” in the ALU for this to work)

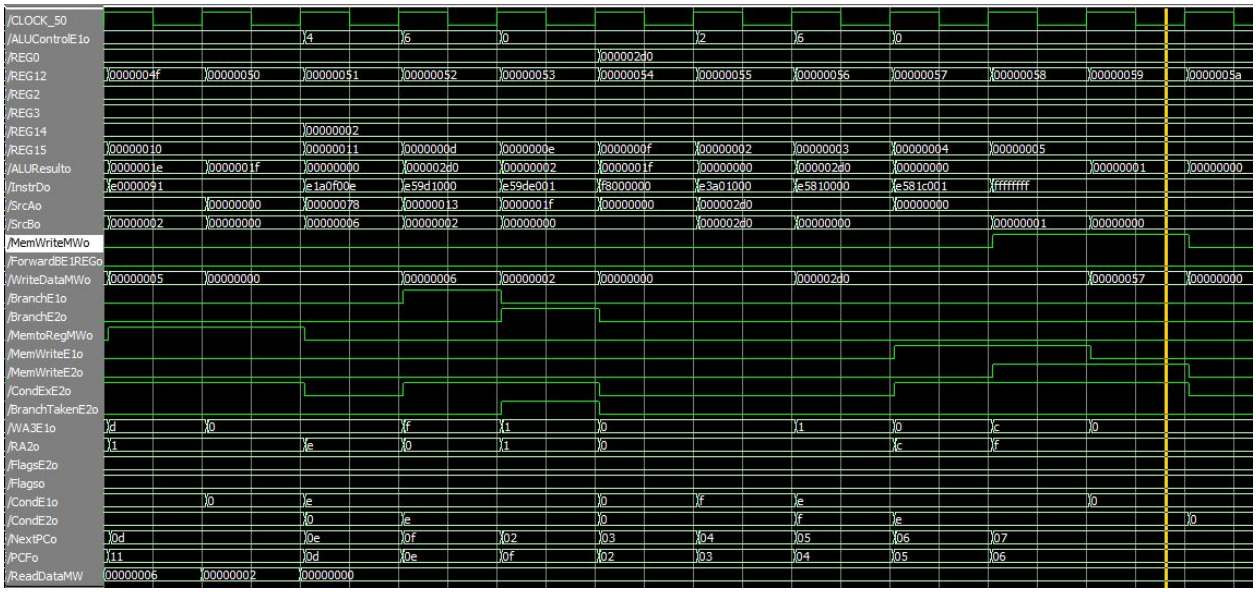
```
MOV R0, #0x00
STR R0, [SP, #-7]
MOV R0, #0x72
STR R0, [SP, #-6]
MOV R0, #0x61
STR R0, [SP, #-5]
MOV R0, #0x63
STR R0, [SP, #-4]
MOV R0, #0x65
STR R0, [SP, #-3]
MOV R0, #0x63
```

```
STR R0, [SP, #-2]
MOV R0, #0x61
STR R0, [SP, #-1]
MOV R0, #0x72
STR R0, [SP]

ADD R2, SP, #1

WHILE:
SUB R2, R2, #1
LDR R3, [R2]
CMP R3, #0
BNE WHILE

CONTINUE:
ADD R2, R2, #1
LDR R3, [R2]
CMP SP, R2
LDR R4, [SP]
SUB SP, SP, #1
EORPLS R5, R3, R4
BEQ CONTINUE

MOV R11, #0
MOVMI R11, #1
STR R11, [R11]
STR R12, [R11, #1]
SWI 0x11

Instructions[0] = 32'hE3A00000;
Instructions[1] = 32'hE50D0007;
Instructions[2] = 32'hE3A00071;
Instructions[3] = 32'hE50D0006;
Instructions[4] = 32'hE3A00061;
Instructions[5] = 32'hE50D0005;
Instructions[6] = 32'hE3A00063;
Instructions[7] = 32'hE50D0004;
Instructions[8] = 32'hE3A00065;
Instructions[9] = 32'hE50D0003;
Instructions[10] = 32'hE3A00063;
Instructions[11] = 32'hE50D0002;
Instructions[12] = 32'hE3A00061;
Instructions[13] = 32'hE50D0001;
Instructions[14] = 32'hE3A00072;
Instructions[15] = 32'hE58D0000;
Instructions[16] = 32'hE28D2001;
Instructions[17] = 32'hE2422001;
Instructions[18] = 32'hE5923000;
```

```
Instructions[19] = 32'hE3530000;
Instructions[20] = 32'h1AFFFFFB;
Instructions[21] = 32'hE2822001;
Instructions[22] = 32'hE5923000;
Instructions[23] = 32'hE15D0002;
Instructions[24] = 32'hE59D4000;
Instructions[25] = 32'hE24DD001;
Instructions[26] = 32'h51935004;
Instructions[27] = 32'h0AFFFFF8;
Instructions[28] = 32'hE3A0B000;
Instructions[29] = 32'h43A0B001;
Instructions[30] = 32'hE58BB000;
Instructions[31] = 32'hE58BC001;
Instructions[32] = 32'hFFFFFFFF;
```

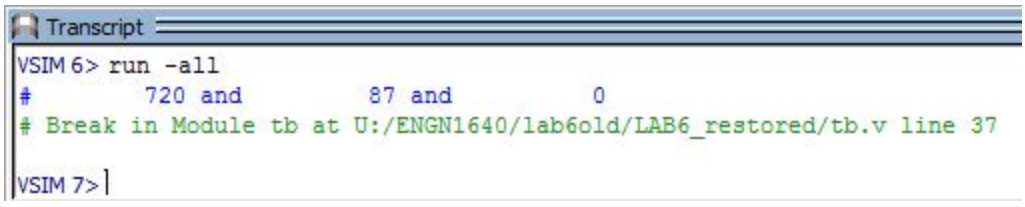
Screen capture of the factorial program timing simulation:

I confirmed that the number of cycles matches what is stored in R12.

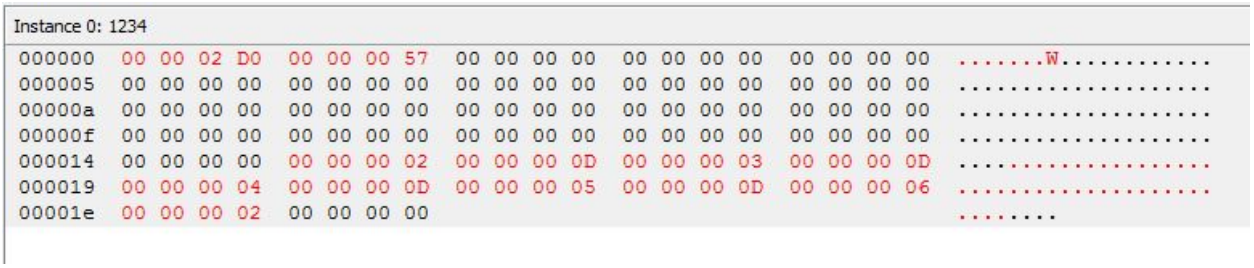


The monitor output:

720 is the result in decimal, and 87 is the number of cycles in decimal

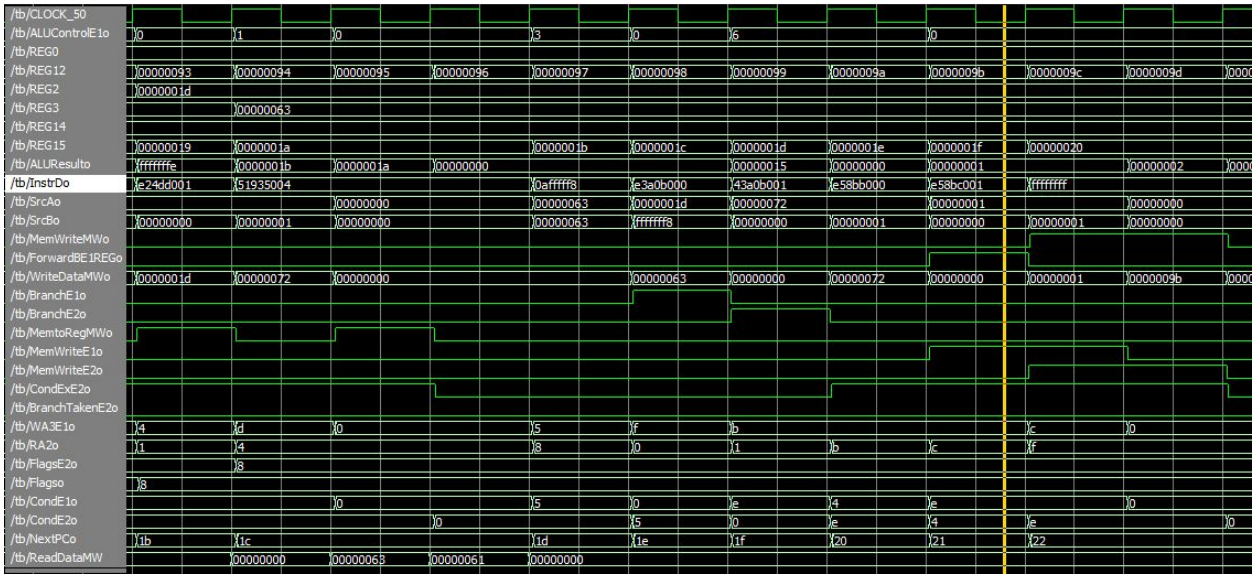


Screen capture of the FPGA memory for the factorial program at 90 MHz:

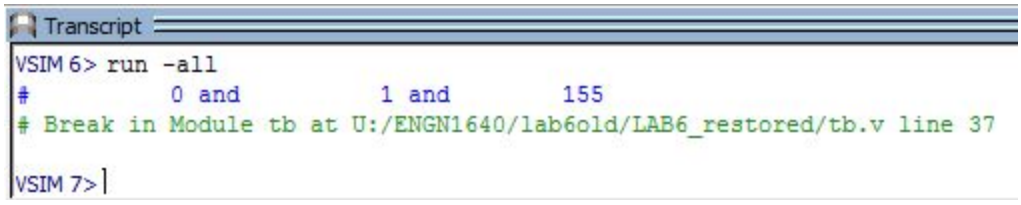


```
Instance 0: 1234
000000  00 00 02 D0  00 00 00 57  00 00 00 00  00 00 00 00  00 00 00 00  .......W........
000005  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000a  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000f  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000014  00 00 00 00  00 00 00 02  00 00 00 0D  00 00 00 03  00 00 00 0D  ................
000019  00 00 00 04  00 00 00 0D  00 00 00 05  00 00 00 0D  00 00 00 06  ................
00001e  00 00 00 02  00 00 00 00                                        ........
```

Screen capture of the palindrome checker timing simulation:

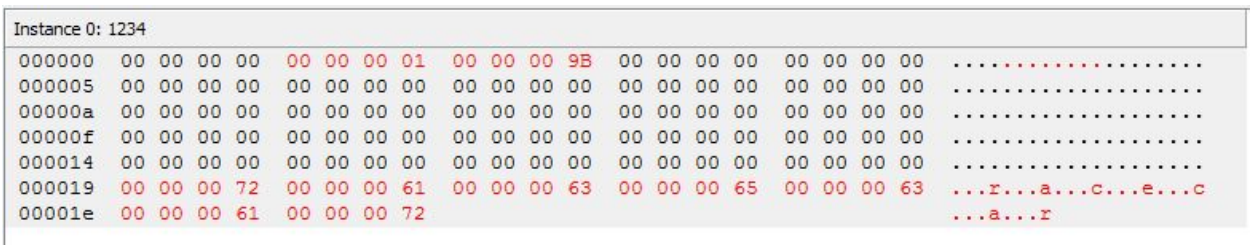


The monitor output: (yields 1 because racecar is a palyndrome, number of cycles is 155 in decimal)



```
Transcript
VSIM 6> run -all
#          0 and          1 and        155
# Break in Module tb at U:/ENGN1640/lab6old/LAB6_restored/tb.v line 37

VSIM 7>
```

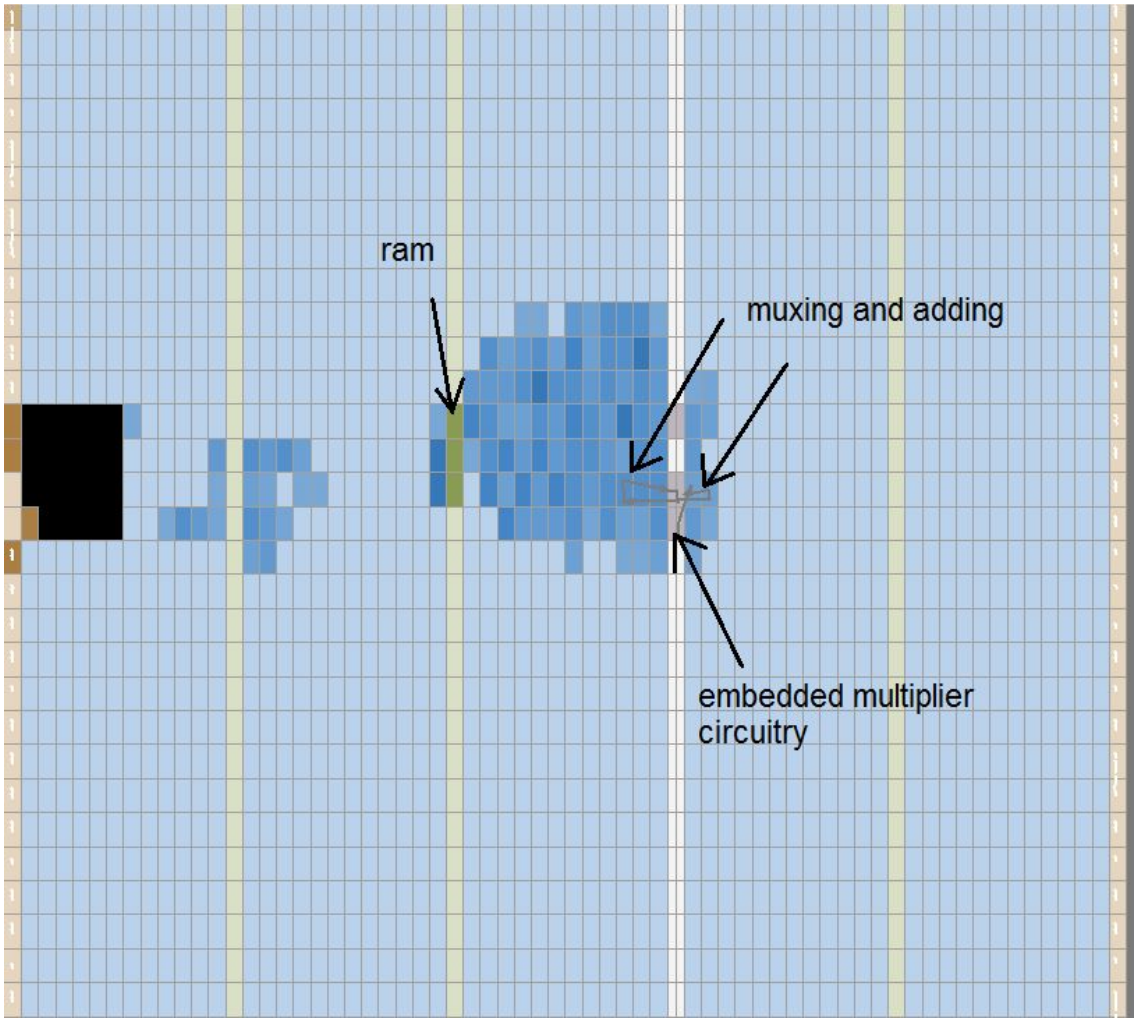
Screen capture of the FPGA memory for the palindrome checker:



```
Instance 0: 1234
000000  00 00 00 00  00 00 00 01  00 00 00 9B  00 00 00 00  00 00 00 00  ................
000005  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000a  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
00000f  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000014  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000019  00 00 00 72  00 00 00 61  00 00 00 63  00 00 00 65  00 00 00 63  ...r...a...c...e...c
00001e  00 00 00 61  00 00 00 72                                        ...a...r
```

Stefan Sigurdsson
B01189238

The critical path on the floor plan (First half of 32 bit multiplication in E1):



Looking at critical paths for lab 5 in TimeQuest I was able to break down the worst case times for each stage of execution, as follows, and use that to inform me on how I should pipeline my design for lab 6:

Fetch - Decode - Execution - Memory - Write Back/Conditional Flags
~3.0 ns - ~3.5 ns - ~8.5 ns - ~1 ns - ~1.5 ns

Looking at timequest for lab 6 reveals some changes in stage times, perhaps as a result of the more complex circuitry needed for forwarding and hazard detection + pipelining:

Fetch - Decode - Execution1 - Execution 2 - Memory+Write Back
~6.6 ns - ~6.7 ns - ~10 ns - ~9 ns - ~2.5 ns

The Execution1+2 times drop to around 7 ns each when I take out the 32 bit multiplier and leave only the 10 bit multiplier. These times are larger than the 5 ns cycle time I achieved on the board. Playing around with the compilation settings in Quartus I noticed a big improvement in Fmax when I used timing based compilation and enabled all the settings that were recommended to improve Fmax in the advisor. The one setting that had the opposite effect for me however, which was recommended, was the WYSIWYG resynthesis, which I did not enable as a result.

Unconditional Branch resolution in my design is done by storing the last destination of branches for individual instructions and immediately branching to those locations on subsequent encounters of those instructions, while checking for updates in branch destinations and correcting if needed every time. I did not do any sort of predicting for conditional branches, they were simply resolved in E2.

Fmax on the board:
By incrementing the PLL clock frequency from 95 to 225 MHz I found that the processor was able to run the factorial program producing the correct results up to 200 MHz, as confirmed by compiling and running the archived project on canvas. The factorial program takes 87 cycles to complete which yields a runtime of 87/0.2 = 435 ns. TimeQuest estimates a max frequency of only 102.15 MHz with the 32 bit multiplier, and around 150 MHz with only the 10 bit multiplier. This is because; First, it uses a slow model to simulate the circuit. Second, it accounts for paths within the circuit which the factorial program never uses (and any program may never use), meaning that the critical path found by TimeQuest is never actually used by the factorial program. In general TimeQuest will always give a slower Fmax than the FPGA because it assumes the absolute worst case scenario imaginable which does not occur for our programs.