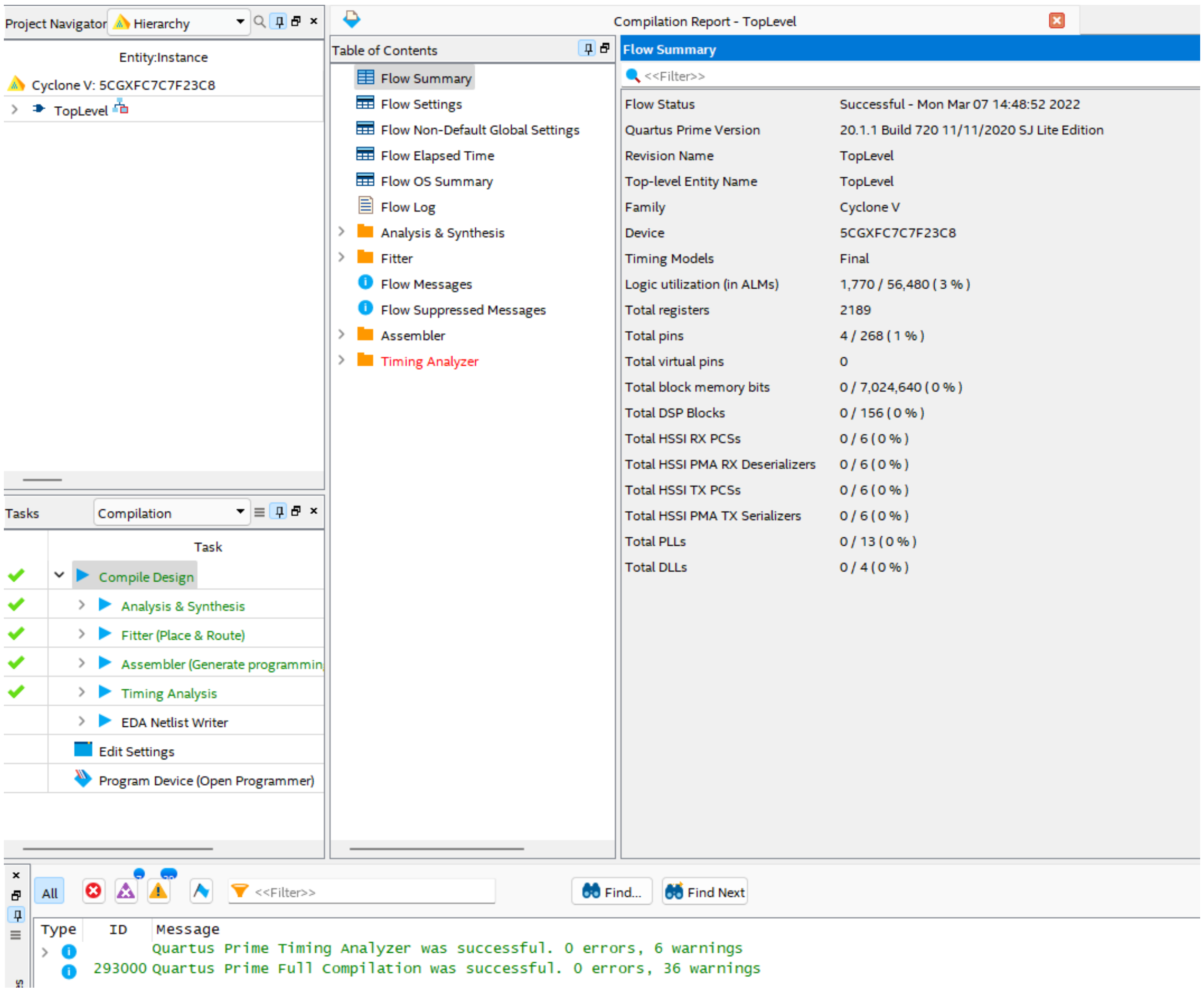
**Team:** William Sun

**Introduction:**

1. Architecture Name: TritonFEC
2. Load-Store: Use fixed-encoding architecture like MIPS

**Architectural Overview:**

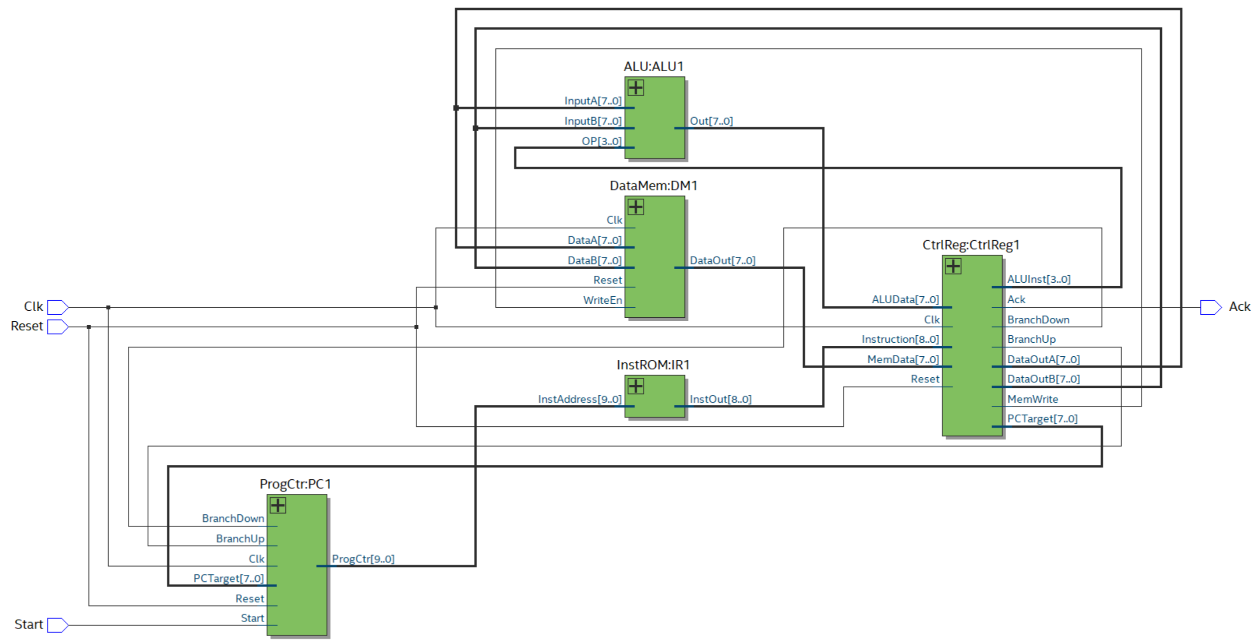


Compilation Report - TopLevel

Flow Summary

| | |
|---|---|
| Flow Status | Successful - Mon Mar 07 14:48:52 2022 |
| Quartus Prime Version | 20.1.1 Build 720 11/11/2020 SJ Lite Edition |
| Revision Name | TopLevel |
| Top-level Entity Name | TopLevel |
| Family | Cyclone V |
| Device | 5CGXFC7C7F23C8 |
| Timing Models | Final |
| Logic utilization (in ALMs) | 1,770 / 56,480 ( 3 % ) |
| Total registers | 2189 |
| Total pins | 4 / 268 ( 1 % ) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 7,024,640 ( 0 % ) |
| Total DSP Blocks | 0 / 156 ( 0 % ) |
| Total HSSI RX PCSs | 0 / 6 ( 0 % ) |
| Total HSSI PMA RX Deserializers | 0 / 6 ( 0 % ) |
| Total HSSI TX PCSs | 0 / 6 ( 0 % ) |
| Total HSSI PMA TX Serializers | 0 / 6 ( 0 % ) |
| Total PLLs | 0 / 13 ( 0 % ) |
| Total DLLs | 0 / 4 ( 0 % ) |

```
Type   ID     Message
              Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
     293000 Quartus Prime Full Compilation was successful. 0 errors, 36 warnings
```



**Machine Specification:**

1. Instruction Formats (2 bits)
    a. Memory and Comparison Operations (00)
        i. opcode(2'b) specifier(3'b) $rt(2'b) $rd(2'b)
        ii. ex: move $t, $s
    b. 1 Variable Operations (01)
        i. opcode(2'b) specifier(1'b) imm(6'b)
        ii. opcode(2'b) specifier(1'b) $rt(2'b) extra(4'b)
        iii. ex: set i
    c. 2 Variable Operations (10)
        i. opcode(2'b) specifier(3'b) $rt(2'b) $rd(2'b)
        ii. ex: add $s, $t
    d. Branch Operations (11)
        i. opcode(2'b) specifier(1'b) $rs(2'b) $rt(2'b) $ru(2'b)
        ii. ex: beq $s, label
2. Operations
    a. Memory and Comparison Operations (00)
        i. 00_000 → get $t $s; $t = REGISTER[$s]
        ii. 00_001 → put $t $s; REGISTER[$t] = $s
        iii. 00_010 → lw $t $s; $t = MEM[$s]
        iv. 00_011 → sw $t $s; MEM[$t] = $s
        v. 00_100 → seq $t $s; $t = ($t == $s)
        vi. 00_101 → sne $t $s; $t = ($t != $s)
        vii. 00_110 → slt $t $s; $t = ($t < $s)
    b. 1 Variable (01)

        i. 01_0 → set i; $r3 = SE(i) // NOTE: hardcode r3
        ii. 01_1 → not $d; $d = ~($d)
    c. 2 Variable (10)
        i. 10_000 → add $s $t; $s = $s + $t
        ii. 10_001 → sub $s $t; $s = $s - $t
        iii. 10_010 → and $s $t; $s = $s & $t
        iv. 10_011 → or $s $t; $s = $s | $t
        v. 10_100 → xor $s $t; $s = ^($t)
        vi. 10_101 → sleft $t $s; $t = $t << $s
        vii. 10_110 → sright $t $s; $t = $t >> $s
        viii. 10_111 → bxor $s $t; $s = $s ^ $t
    d. Branch (11)
        i. 11_0 → b_up $s $t $u; if ($s == $t) pc = pc - $u
        ii. 11_1 → b_down $s $t $u; if ($s == $t) pc = pc + $u
3. Internal Operands
    a. Total: 16 registers
    b. Temporary Use: 4 registers (r0 - r3)
        i. During operation, use r0 - r3 as operands
        ii. Use mov, lw, sw to move result into general storage
    c. General Storage: 12 registers (r4 - r15)
4. Control Flow
    a. Type
        i. Branch on equals to save bits
    b. Target Address
        i. Load in address from register
    c. Maximum Distance
        i. Relative branching is used. Register data is 8 bits and data is unsigned (use minus/add to go up/down). Thus, relative range is i < 2^8 = 256
        ii. Alternatively, program counter will load in specified program start location
5. Addressing Modes
    a. Register-indirect addressing
        i. Load and store data via contents of register
        ii. lw, sw

**Changelog Milestone 2**

- Reorganize ISA opcode based on instruction format
    - Allows for more instructions
    - Allows for larger immediate values
- Change 2 registers to 4
- Update control flow based on ISA changes
- Fix addressing modes issues

**Changelog Milestone 3**

- Distinguish between moving data from registers to memory (lw, sw) and moving data between working registers and general registers (put, get)
- Removed unnecessary operations in (00)
- Replaced NOR with OR in (10)
- Change XOR into bitwise operation in (10)
- Updated BEQ in (11) with register-indirect addressing
- Added BNE in (11)
- Rename shift operations
- Maximum branch distance updated to 128
- Update addressing mode to register-indirect
- Removed PC-relative addressing

**Changelog Final Report**

- ISA:
    - Added (==) and (!=) operation (for program 2)
    - Added bitwise xor (BXOR) (for program 2)
    - Changed to only relative branching, with unsigned distance

**Programmer's Model:**

1. The programmer should think of this machine as trying to conserve as many bits as possible. With only 9 bits total, we must split up the instruction encoding wisely. We use 2 bits to first split between major types of instructions. That gives us enough bits to set large immediate values (2^6). With variable instructions, we can use up all 9 bits, so we can split those further into specific instructions to increase the total number of instructions we can encode. Additionally, the programmer should abuse the hardware and use NOR as a universal gate to recreate other operations to save bits. Although this will increase CPI, it makes it easier to fit the 9-bit instruction limit.
2. Example
    a. C → \$r0 = \$r0 + \$r1
    b. Assembly → add r0 r1
    c. Machine → 10_000 (add) 00 (r0) 01 (r1)