ECE 2500 Project2

Report

Xiaoyu Hou
Virginia Tech

Summary:

This project involved the MIPS assembly code to write an iterative C function. And use the Verilog to complete the required instruction parts.
GCD algorithm in C:

```c
unsigned int gcd(unsigned int u, unsigned int v)
{
    // Base cases
    //  gcd(n, n) = n
    if (u == v)
        return u;

    //  Identity 1: gcd(0, n) = gcd(n, 0) = n
    if (u == 0)
        return v;
    if (v == 0)
        return u;

    if (u % 2 == 0) { // u is even
        if (v % 2 == 1) // v is odd
            return gcd(u/2, v); // Identity 3
        else // both u and v are even
            return 2 * gcd(u/2, v/2); // Identity 2

    } else { // u is odd
        if (v % 2 == 0) // v is even
            return gcd(u, v/2); // Identity 3

        // Identities 4 and 3 (u and v are odd, so u-v and v-u are known to be even)
        if (u > v)
            return gcd((u - v)/2, v);
        else
            return gcd((v - u)/2, u);
    }
}
```

Assembly Design

In this project, I divide the function into two parts. In the first part, I check if u equal to v; if u or v equal to 0. The first part does not need any recursive. And in the second part, I check if u and v are even or odd. So, I will use the following instructions: addi, beq, andi, andi, slt, srl, sll, sub in my assembly code.

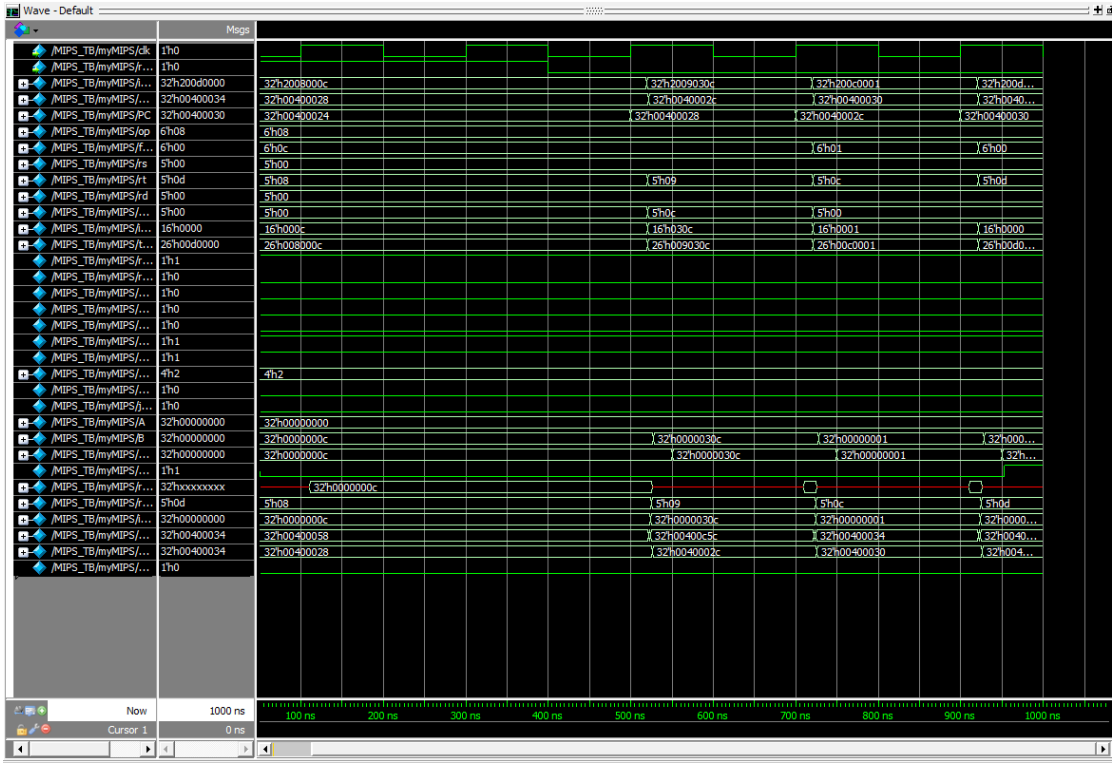
After test my testbench in Qtspim, this is my result:

```
PC         = 4194476
EPC        = 0
Cause      = 0
BadVAddr   = 0
Status     = 805371664

HI         = 0
LO         = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 10
R3  [v1] = 0
R4  [a0] = 0
R5  [a1] = 12345
R6  [a2] = 2147481360
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 1
R10 [t2] = 0
R11 [t3] = 7875
R12 [t4] = 1
R13 [t5] = 0
R14 [t6] = 1
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 12
R18 [s2] = 12
R19 [s3] = 2048
R20 [s4] = 1
R21 [s5] = 50400
R22 [s6] = 131072
R23 [s7] = 12345
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 268468224
R29 [sp] = 2147481324
R30 [s8] = 0
R31 [ra] = 4194468
```

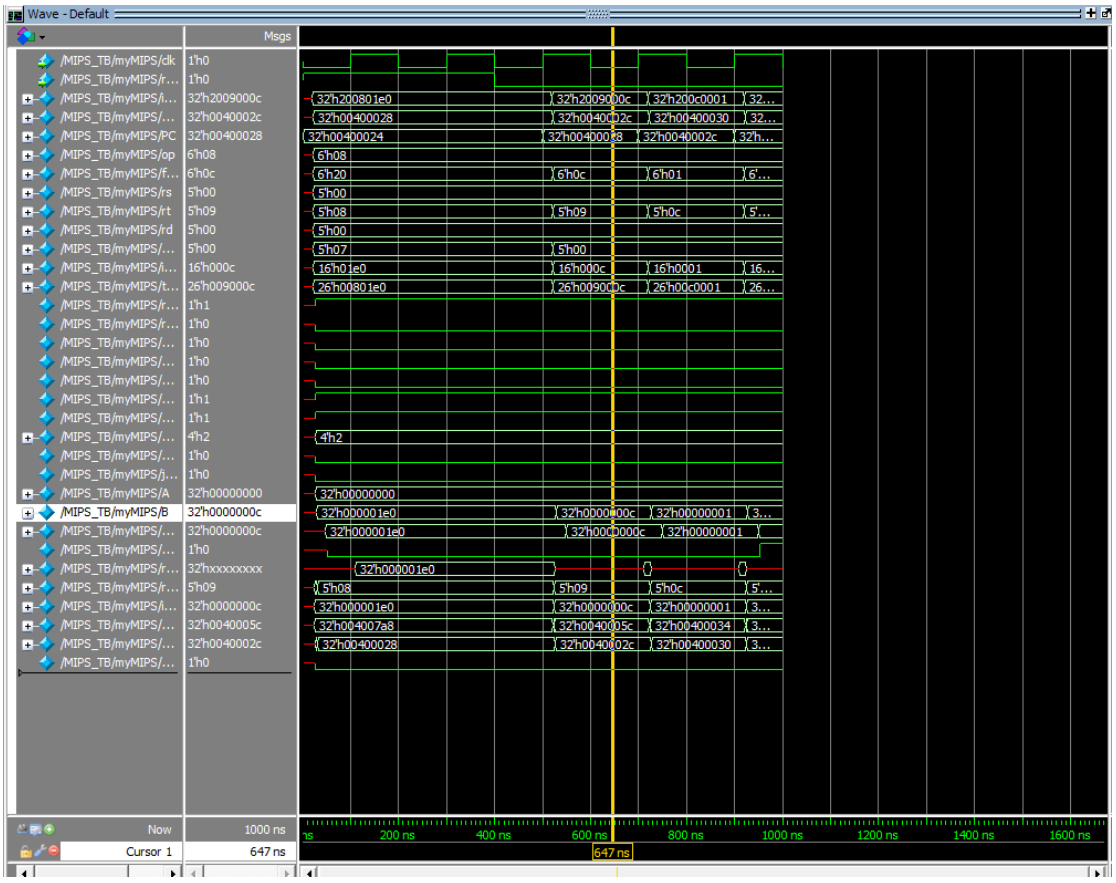
Verilog Design

In this project, we only need to modify mips-control and mips. In the mips, I add the control signals for the new instructions which is srl, beq, andi, slt and sub. Because I use beq in my assembly code, I also add new wires for branch in the mips file.
These is test result in ModelSim:

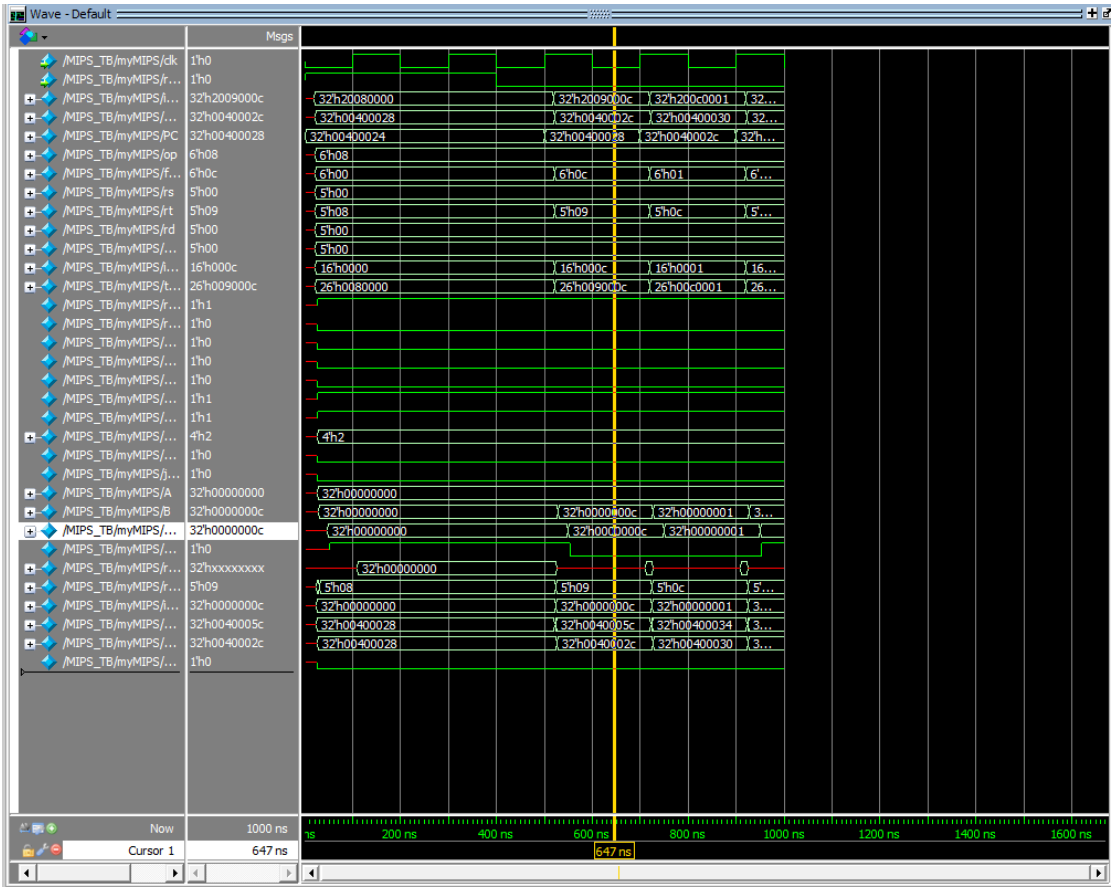
12 and 780:



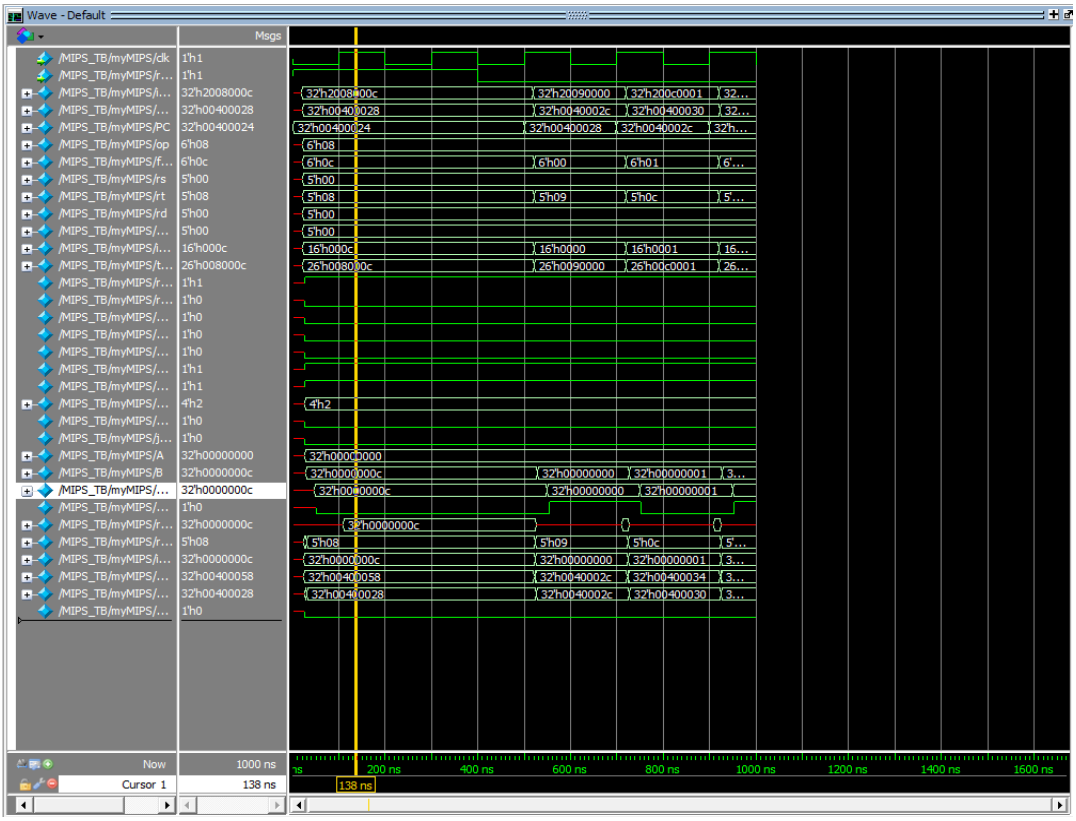
780 and 12:



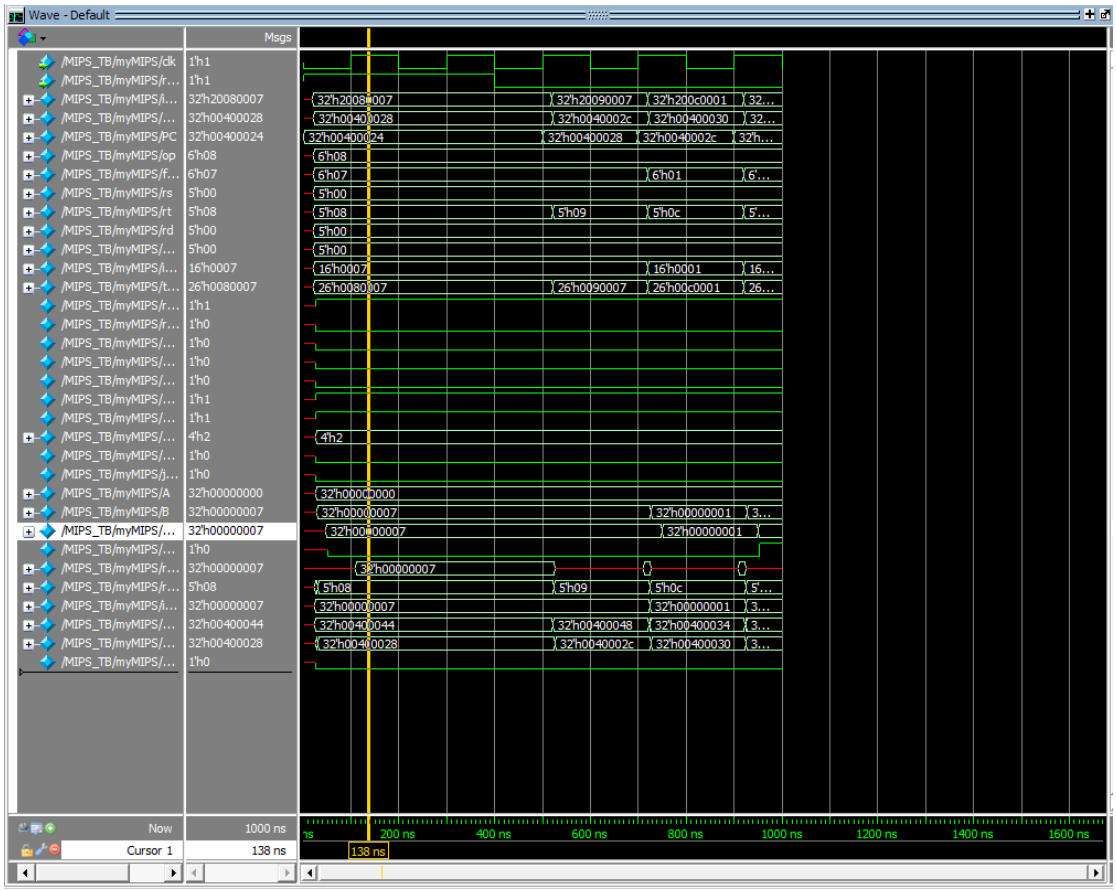
0 and 12:



12 and 0:



7 and 7:


Wave - Default
Msgs
/MIPS_TB/myMIPS/clk
/MIPS_TB/myMIPS/r...
/MIPS_TB/myMIPS/i...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/PC
/MIPS_TB/myMIPS/op
/MIPS_TB/myMIPS/f...
/MIPS_TB/myMIPS/rs
/MIPS_TB/myMIPS/rt
/MIPS_TB/myMIPS/rd
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/i...
/MIPS_TB/myMIPS/t...
/MIPS_TB/myMIPS/r...
/MIPS_TB/myMIPS/r...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/j...
/MIPS_TB/myMIPS/A
/MIPS_TB/myMIPS/B
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/r...
/MIPS_TB/myMIPS/r...
/MIPS_TB/myMIPS/i...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
/MIPS_TB/myMIPS/...
Now
Cursor 1
1000 ns
138 ns
200 ns
400 ns
600 ns
800 ns
1000 ns
1200 ns
1400 ns
1600 ns