ELEC373

Assignment 4

Synthesising the NIOS II Processor

Junhao Zhang
201377244

25 May 2020

Declaration

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment (Appendix L).

1 Introduction

In this assignment, it is required to implement a custom instruction which is Count Leading Ones on the NIOS II system. An additional module was developed in Verilog in order to implement the custom instruction. The custom instruction was connected to the NIOS II CPU as an additional component where the NIOS II CPU can make use of it. ModelSim was used to test if the custom instruction function properly. Therefore, a program was developed in C utilising the NIOS II Software Build Tool for Eclipse. Also developed was a software implementation which is used to compare the speed with the hardware implement on NIOS II CPU.

2 ASM of the Custom Instruction

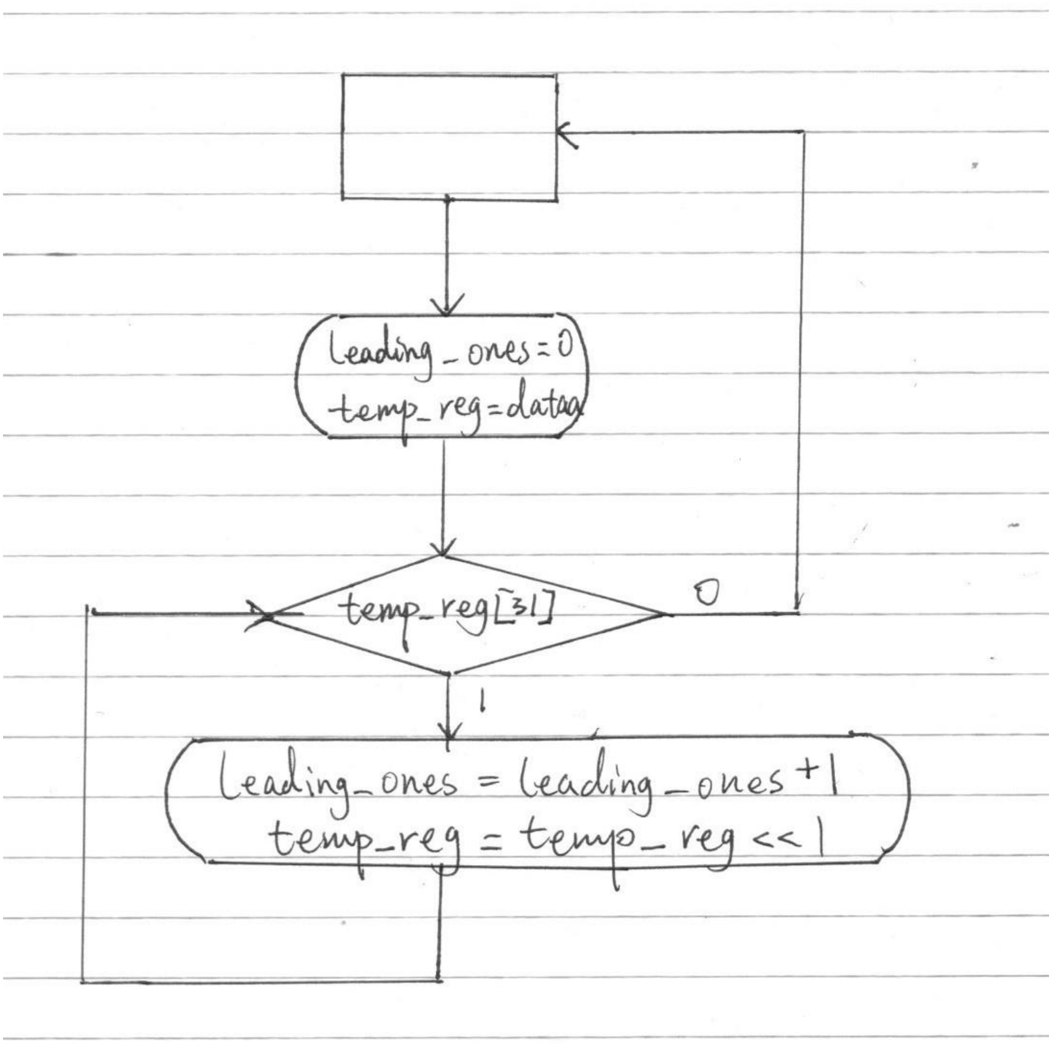

Figure 1: ASM

As shown in Figure 1, to implement the custom instruction count leading ones, a loop is used in order to accumulate the counting number. The entry condition is the MSB of the input data. Each time a 1 is detected at the MSB, the counter will increment by 1, and the data will be left shifted by one so that the bits after the MSB becomes the MSB. Once a bit of 0 is detected at the MSB, the loop will be broken and the counter can give the correct number of the leading ones.

3 Verilog Code

```
module clo (dataa, leading_ones);

    input   [31:0]  dataa;
    output  [31:0]  leading_ones;

    reg     [31:0]  temp_reg;
    reg     [31:0]  leading_ones;

    always @(dataa)
    begin
        leading_ones = 0;
        temp_reg = dataa;

        while(temp_reg[31])
        begin
            leading_ones = leading_ones + 1;
            temp_reg = temp_reg << 1;
        end
    end
endmodule
```

4 Test Programme in C

```
#include "system.h"
#include "sys/alt_stdio.h"

int main()
{
  unsigned a = 0x00;
  unsigned z = 0x00;

  z = ALT_CI_CLO_0(a);
  alt_printf("a = %x, z = %x\n", a, z);

  a = 0xE0E00000;
  z = ALT_CI_CLO_0(a);
  alt_printf("a = %x, z = %x\n", a, z);

  a = 0xA0A00000;
  z = ALT_CI_CLO_0(a);
  alt_printf("a = %x, z = %x\n", a, z);

  a = 0xF0000000;
  z = ALT_CI_CLO_0(a);
```

```
  alt_printf("a = %x, z = %x\n", a, z);

  while (1);

  return 0;
}
```

5 Result

5.1 ModelSim Result

```
VSIM 3> run 300 us
# Warning: read_during_write_mode_mixed_ports is assumed as
# Time: 0  Instance: first_nios2_system_tb.first_nios2_system_i
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: ---------------------------------
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: first_nios2_system_tb.first_nios2
#                    0: INFO: ---------------------------------
#                    0: INFO: first_nios2_system_tb.first_nios2
#               990000: INFO: first_nios2_system_tb.first_nios2
# a = 0, z = 0
# a = e0e00000, z = 3
# a = a0a00000, z = 1
# a = f0000000, z = 4
```

Figure 2: ModelSim Simulation Result

As demonstrated in Figure 2, the ModelSim result shows that the custom instruction count leading ones is working properly. When the input data a is 0, the leading ones is 0; when the input data a is 0xE0E0 0000, where the leading E is 1110 in binary, the leading ones is therefore 3; when the input data a is 0xA0A0 0000, where the leading A is 1010 in binary, the leading ones is therefore 1; and when the input data a is 0xF000 0000, where the leading F is 1111 in binary, the leading ones is therefore 4. Hence, the function of the custom instruction is proven to be correct.

5.2 Speed Comparison

```
#include <stdio.h>
#include <chrono>
#include <windows.h>

#define INT_SIZE 32

int main()
{
    int num, count, msb, i;

    num = 0xFFFFFFFF;
```

```
    auto begin = std::chrono::high_resolution_clock::now();
    msb = 1 << (INT_SIZE - 1);

    count = 0;

    /* Iterate over each bit */
    for (i = 0; i < INT_SIZE; i++)
    {
        /* If leading set bit is found */
        if ((~num << i) & msb)
        {
            /* Terminate the loop */
            break;
        }

        count++;
    }

    auto end = std::chrono::high_resolution_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin);

    printf("Total number of leading zeros in %x is %d\n", num, count);
    printf("Time measured: %.12f seconds.\n", elapsed.count() * 1e-9);

    return 0;
}
```

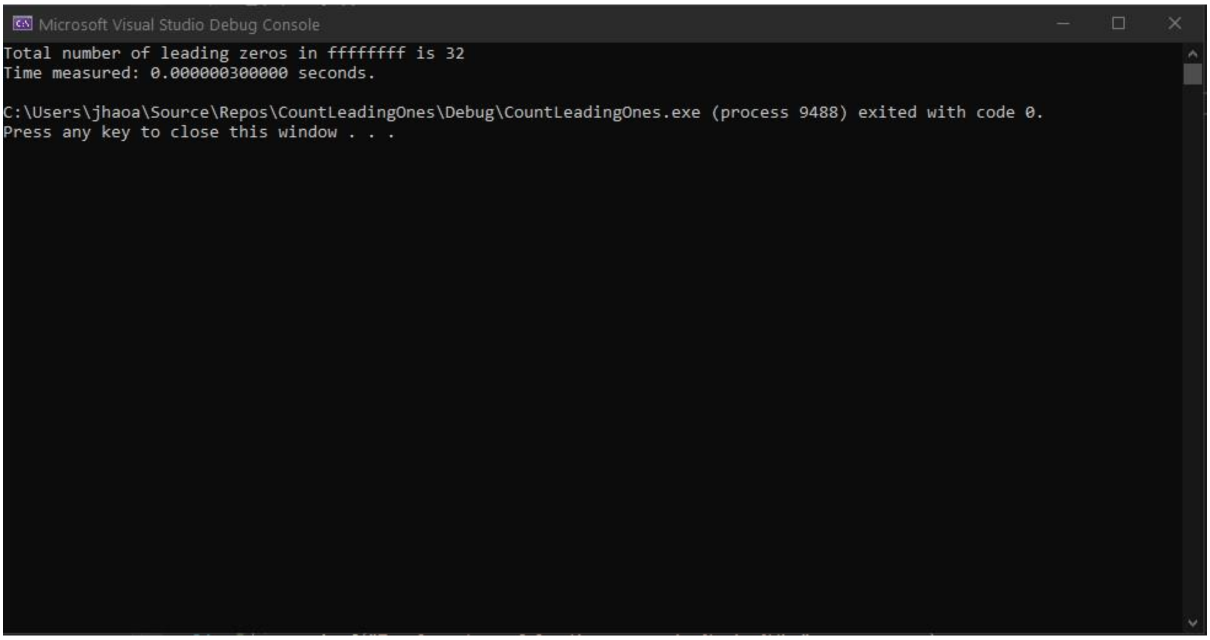

Figure 3: Execution Time

As shown in Figure 3, the execution time for this program which perform the same function as the custom instruction is much faster that the latter. The software implementation spend approximately 0.3us on the function performing counting leading ones, whereas as shown in the result of the ModelSim, the custom instruction simulated on NIOS II takes over estimated 300us. The reason is because the CPU of the PC where the software is running has much higher performance that the NIOS II CPU with respect to the clock frequency, cores number and other CPU parameters. Hence the software implementation only takes a few microseconds to complete this function.