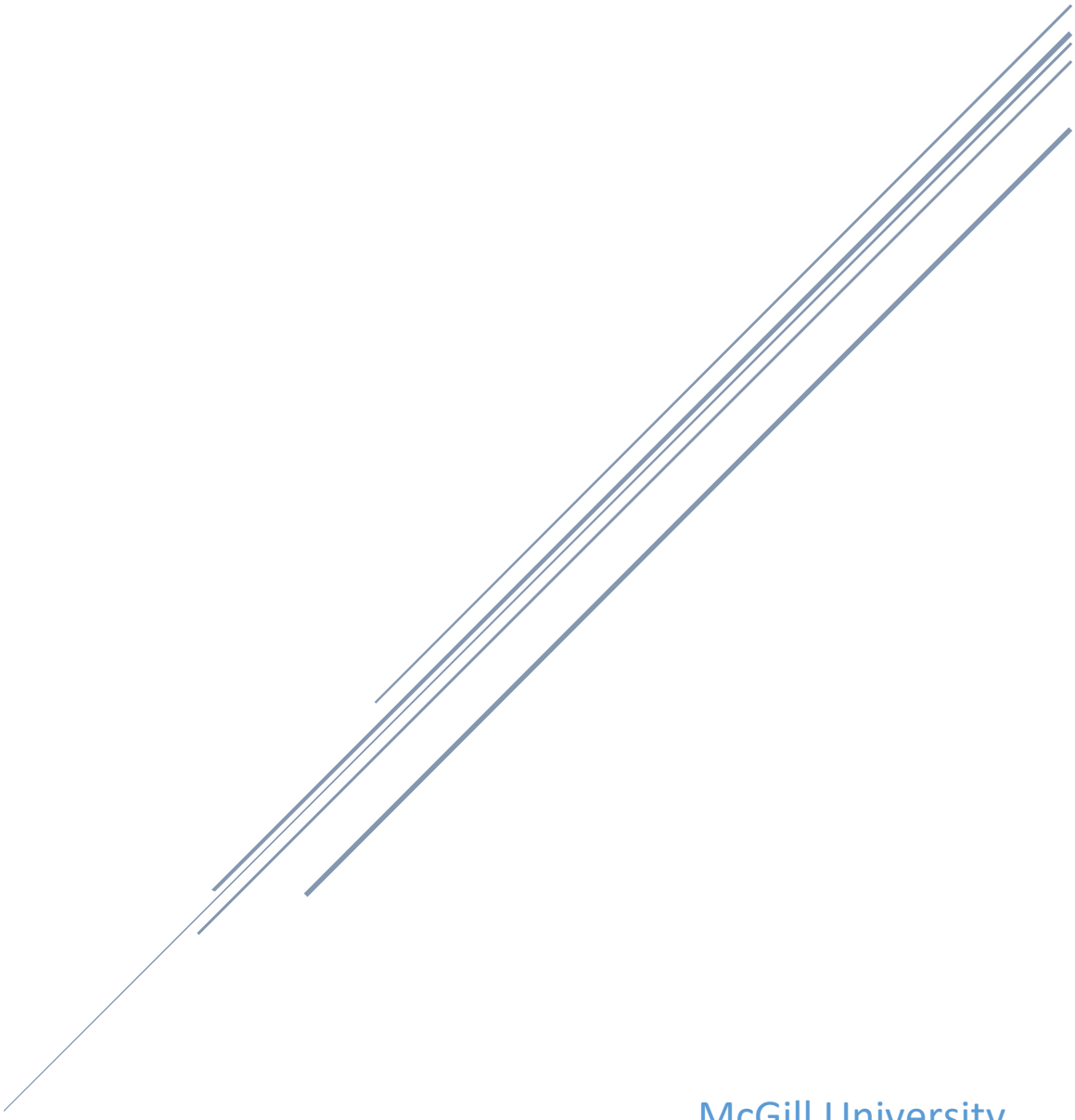
LAB #1 REPORT

October 13, 2017



McGill University
ECSE 323

Group #27
Matthew Lesko 26069352; Romain Nith 260571471

Contents

Group #27
Matthew Lesko 26069352; Romain Nith 260571471

Modulo 13 Circuit

Gate Level Schematic Circuit Diagram

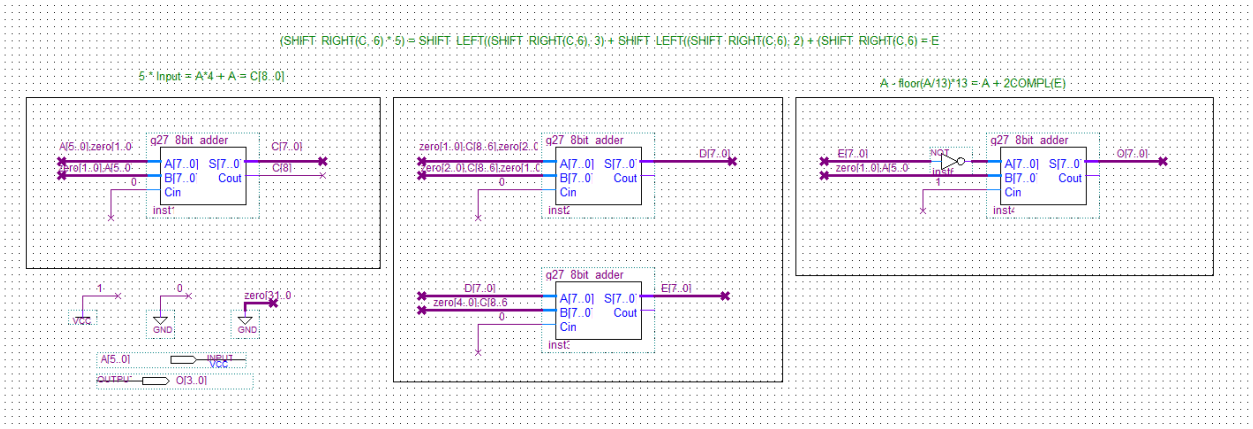


Figure 1 - Modulo-13 Circuit Diagram Implementation in Quartus II

Description of Circuit's Function

The function of the modulo-13 circuit is to take a 6-bit number as input and output as a 4-bit number the answer to the modulo-13 operation of the input. The first part of the circuit computes "floor(A/13)" where A is the user's 6-bit input. Since divisions use large resources, 1/13 can be approximated to 5/64. Furthermore, multiplying by 5 can be simplified by adding the input to itself shifted 2 times to the left. As for the decimals, the sum is shifted to the right 6 times. Conveniently, shifting to the right floors the result so no additional operation is necessary.

The second part takes the output of the first part and multiplies it by 13. This can also be done by adding a shifted 3 times to the left version of the output to a shifted 2 times to the left version and finally by itself.

The final part subtracts the user's input with "floor(A/13)*13". The product of the second part is being passed in a 2's compliment and then added to the user's input to create the subtraction operation. The result is a 4-bit output which represents: INPUT % 13;

Design and Implementation Process

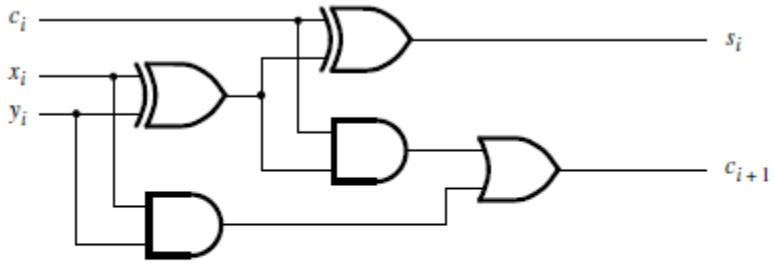
The team designed each circuit with the use of the Quartus II (Version 13.0sp1) software. In-class material and lab material were used as references. StackOverflow and the Altera Quartus forums were used as references. The addition and deletion of wires, gates, input pins, and output pins can be understood and followed in the Quartus II instructions in *lab_1_17F.pdf*. The functionalities of creating new circuit components, saving them under a project and using them in a separate block diagram design can be understood and followed in the Quartus II instructions in *lab_1_17F.pdf*.

The modulo-13 circuit design requires four 8-bit adder components which is comprised of a ripple-carry design of 8 full-bit adder components.

Group #27
Matthew Lesko 26069352; Romain Nith 260571471

Full-Adder

The implementation of the full-adder followed the design of the detailed circuit diagram of a full-adder circuit found in *Fundamentals of Digital Logic with Verilog – Third Edition* class textbook. The design can be found on Figure 3.4 of page 129. This design was chosen because it requires only 5 gates and the team found it easy to implement.



(b) Detailed diagram

Figure 3.4 A decomposed implementation of the full-adder circuit.

Figure 3 – Design of Full-Adder in Textbook

It contains the following primitive gates:

- 2 XNOR Gates
- 2 AND Gates
- 1 OR Gate

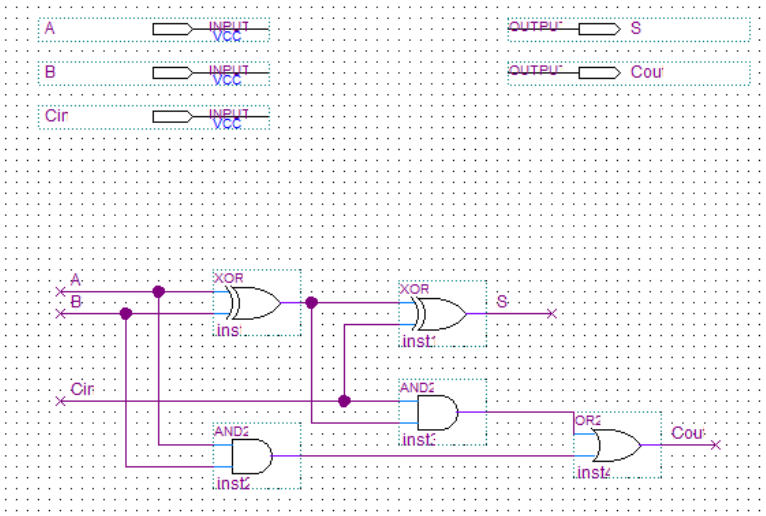


Figure 4 - Block Diagram Implementation of Full Adder in Quartus II

The implementation of the full-bit adder was completed in the Quartus II application. The team followed the instructions of designing a new block diagram in the *lab_1_17F.pdf* file. The implementation of the circuit design requires to be saved as a *Block/Diagram Schematic File*. The three input pins are A, B, and Cin. The output pins are S and Cout.

Group #27
Matthew Lesko 26069352; Romain Nith 260571471

8-Bit Ripple-Carry Adder

The implementation of the 8-bit adder followed the design of the detailed circuit diagram of an n-bit ripple-carry adder circuit found in *Fundamentals of Digital Logic with Verilog – Third Edition* class textbook. The design can be found on Figure 3.5 of page 130.

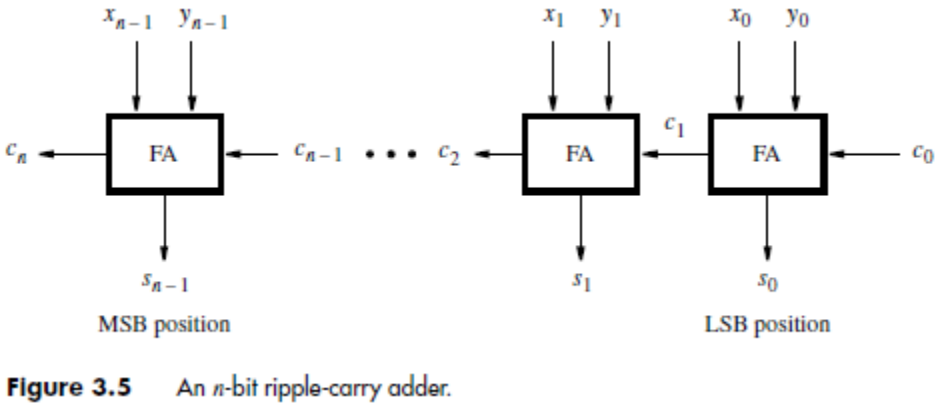


Figure 3.5 An *n*-bit ripple-carry adder.

Figure 5- Design of N-Bit Adder in Textbook

The design can be used for n number of bits. The modulo-13 circuit required 8 bit-adder components. A full-adder can be used for each bit position. The least-significant-bit position is on the left. If a carry is produced in position i, it is carried over to the next component as Cin or as Cout if there is no next component, hence the name of the design philosophy: ripple-carry.

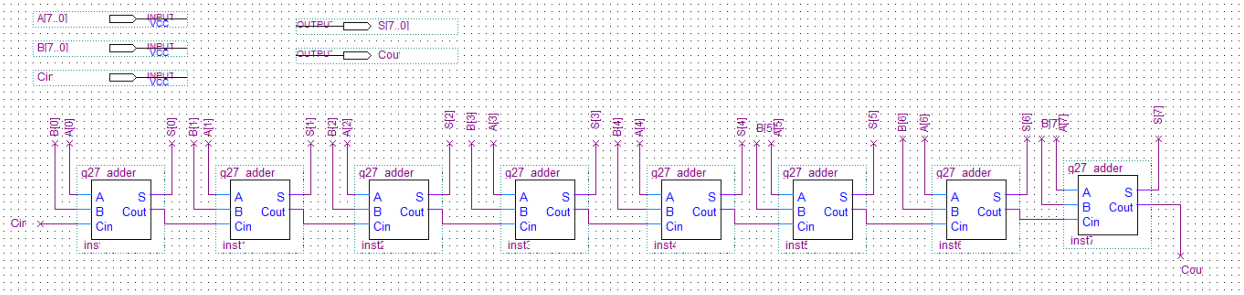


Figure 6 - Block Diagram Implementation of 8-Bit Ripple-Carry Adder in Quartus II

The implementation of the 8-bit ripple-carry adder was completed in the Quartus II application. The team followed the instructions of designing a new block diagram in the *lab_1_17F.pdf* file. The implementation of the circuit design requires to be saved as a *Block/Diagram Schematic File*. The three input pins are A[7..0], B[7..0], and Cin. The output pins are S[7..0] and Cout.

Modulo-13

The design and implementation of the modulo-13 circuit diagram were both completed by the team itself.

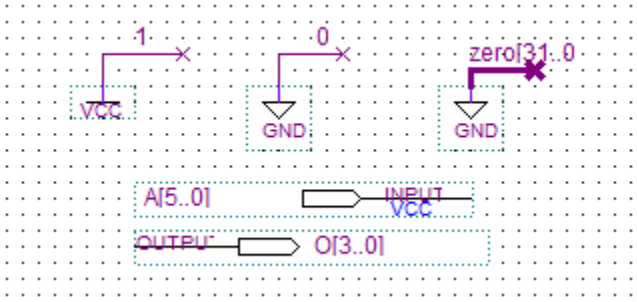


Figure 7 - Signals, Inputs, and Outputs Pins in Modulo-13 Circuit Implementation

The design and implementation require four 8-bit adders. The design of the circuit must functionally behave like this equation: A – floor(A/13) * 13. Instead of creating and using multiplier and divisor components, the design utilizes the concept of shifting and adding for multiplying and dividing. The concept of shifting and adding to multiply or divide inputs can be explained in *lab_1_17F.pdf.* The term A/13 was approximated to 5 * A / 64 in order to use 8 bit-adders and the concept of shifting, since dividing by 64 can be easily represented as shifting to the right by 6 bits.

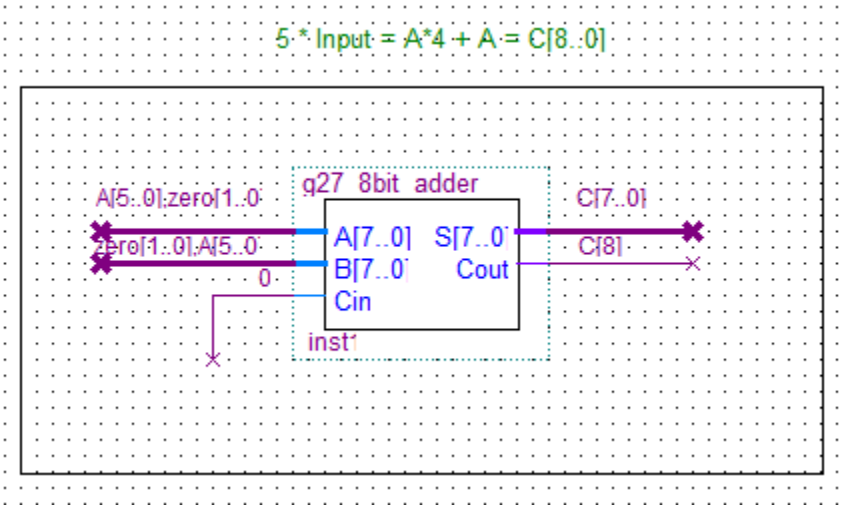


Figure 8 - Component Equivalent to Multiplying Input A[5..0] by 5

Multiplying A by 5 is equivalent to the addition of A, multiplied by 4, and A. Multiplying A by 4 is equivalent to shifting the most-significant-bit by 2 to the left and concatenating two 0 bits on the right. These were the intentions behind the design of the component represented in Figure 8. The first input for the 8-bit adder is the input A[5..0] concatenated with two zero bits on the right, hence shifting the input by two to the left. The second input is A[5..0] concatenated with two zero bits on the left. This makes no change to the value but ensures that the input is 8 bits. Cin is also set to 0 to avoid ambiguity, since the simulation system (ModelSim Altera) may interpret it as 0 or 1 if it hasn't been explicitly set to a value. The output is a 9-bit addition of A * 4 and A. The 9-th bit is saved as the carry-out in order to avoid loss of a bit due to overflow.

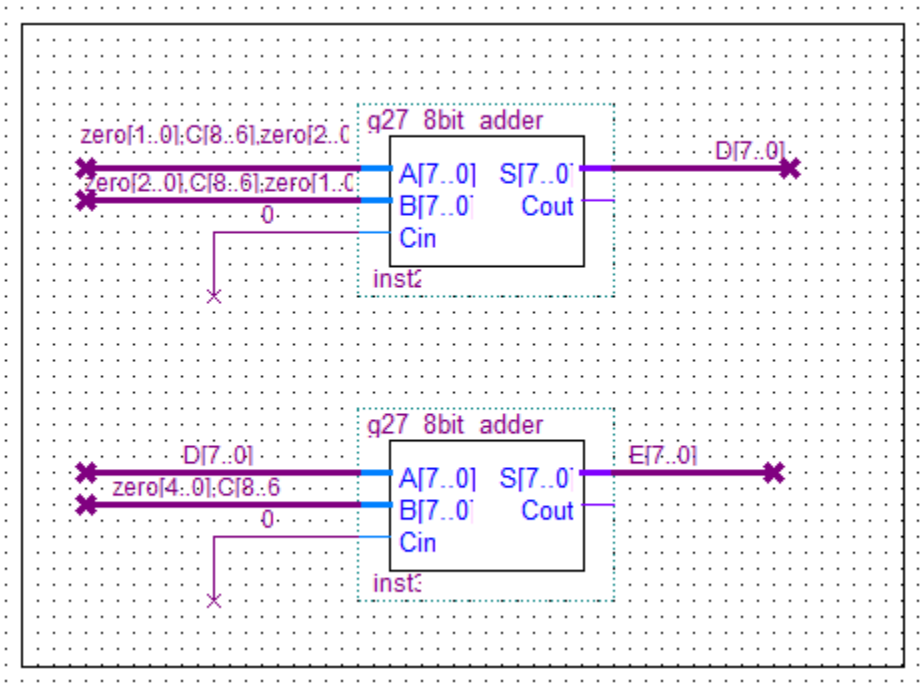


*Figure 9 - Components Equivalent to Shifting the Term (5 * A) to the Right by 6 and Multiplying the Result by 13*

The components above represent the following equation:

(SHIFT_RIGHT(C,6)) * 13 = SHIFT_LEFT((SHIFT_RIGHT(C,6), 3) + SHIFT_LEFT((SHIFT_RIGHT(C,6), 2) + SHIFT_RIGHT(C,6) = E

SHIFT_RIGHT(C,n) shifts C to right by n bits. SHIFT_LEFT(C,n) shifts C to the left by n bits.

The intuition behind the components above is as follows:

- Multiplying by 13 is equivalent to : A * 8 + A * 4 + A. Multiplying by 8 is equivalent to shifting A by 3 to the left. Multiplying by 4 is equivalent to shifting A by 2 to the left.

The inputs for the component *inst2* are as follows:

- zero[1..0]C[8..6]zero[2..0]
  - The output from the previous component C[8..0] is shifted to the right by 6. Leaving bits C[8..6] and 5 zero bits on the left. The result is then shifted to the left by 3.
- zero[2..0]C[8..6]zero[1..0]
  - The output from the previous component C[8..0] is shifted to the right by 6. Leaving bits C[8..6] and 5 zero bits on the left. The result is then shifted to the left by 2.
- Cin = 0

The inputs for the component *inst3* are as follows:

- D[7..0]
  - The output of *inst2*
- zero[4..0]C[8..6]
  - The result from *inst1* shifted to the right by 6. Leaving bits C[8..6] and 5 zero bits on the left.
- Cin = 0

The implementation adds these inputs together and outputs the term floor(A / 13) * 13 which will be used in the next component.

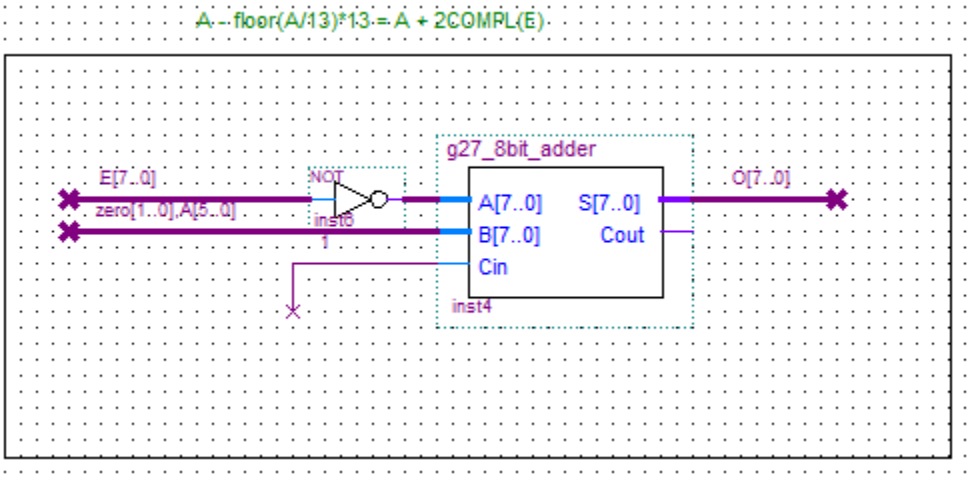


Figure 10 - Components Equivalent to Subtracting input A[5..0] to the previous result E[7..0]

The last component of the modulo 13 is represented by the diagram above. It consists of subtracting A with the result of the previous component resulting in the modulo 13 of A. Since an 8-bit adder was already created, the subtraction is done by taking the 2's complement of the previous component and add it to A. Therefore, the previous result E[7..0] needs to be negated using a NOT gate while the 6-bit A[5..0] input needs to be converted to an 8-bit by using the same method as in the other components. Additionally, a 1 carry in bit in Cin of the adder allows to complete the 2's complement. Finally, the addition results in an 8-bit output which is being truncated to an 4-bit adder O[4..0] illustrated in *Figure 7*.

Simulation Files

The report contains the following main files:

g27_Modulo_13.qpf: main project file of Lab 1

g27_adder.bdf: single bit adder schematic

g27_adder.bsf: single bit adder symbol file

g27_adder.vwf: single bit adder simulation file

g27_8bit_adder.bdf: 8-bit adder schematic

g27_8bit_adder.bsf: 8-bit adder symbol file

g27_8bit_adder.vwf: 8-bit adder simulation file for all values but limited by end time (refer to *Testing of 8-bit adder)*

g27_8bit_adder_B.vwf: 8-bit adder simulation file for B oscillating and Cin pulled low

g27_8bit_adder_I.vwf: 8-bit adder simulation file for B oscillating and Cin pulled high

g27_bit_Modulo_13.bdf: modulo 13 schematic

g27_bit_Modulo_13.vwf: modulo 13 simulation file

g27_comp7.bdf: Lab 1 part 1's comparator schematic

g27_comp7.bsf: Lab 1 part 1's comparator symbol file

g27_comp7.vwf: Lab 1 part 1's comparator simulation file

g27_lab1.bdf: Lab 1 part 1's comparator as a component schematic

g27_lab1.vwf: Lab 1 part 1's comparator as a component simulation file (used to cross check with the comparator simulation file)

Group #27
Matthew Lesko 26069352; Romain Nith 260571471

Testing

Testing of 1-bit adder

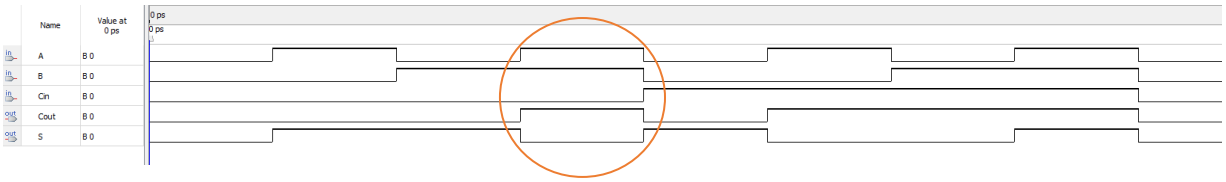


Figure 11 – 1-bit Adder Simulation Test

To test the 1-bit adder, A was set to oscillate every 1ns whereas B and Cin oscillate every 2ns and 4ns respectively. This allows to test all combination of inputs possible for a single bit adder. The total end time is set to $2^3 = 8ns$. In the simulation above, at 3ns, A and B are pulled high while Cin is low. As a result, the adder outputs S as low and Cout as high which corresponds to the expected values.

Testing of 8-bit adder

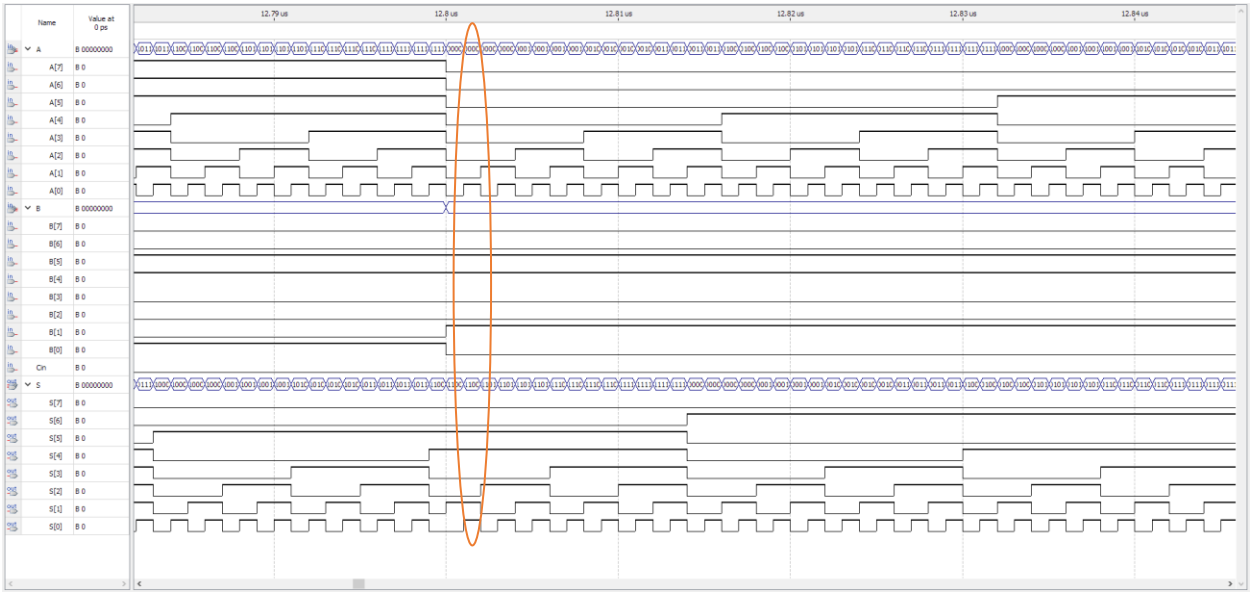


Figure 12 - 8-bit Adder Simulation Test

To test the 8-bit adder, A was set to oscillate every 1ns whereas B and Cin oscillate every 256ns and 512ns respectively. This allows to test all combination of inputs possible for a single bit adder. The total end time is set to $2^8 * 2^8 * 2^1 = 131{,}072ns$. However, since the software can only set 100ms, a first simulation tested the B input with Cin low and the second one with Cin high. In the simulation above, S and Cout are returning the correct values for the adder. From the simulation above running with Cin low, at 12.81ms, A takes the value of 0000 0001, B = 0011 0010 and S = 0011 0011 which is the expected value.

Testing of Modulo 13

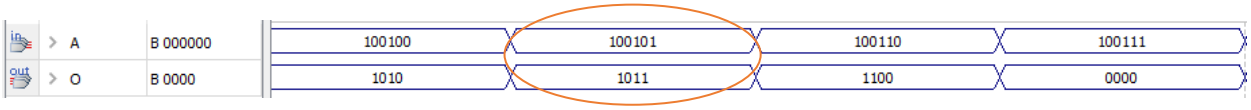


Figure 13 – Modulo 13 Simulation Test

Since the modulo 13 only takes a 6-bit input, the input A was set to oscillate at 1ns for 64ns which corresponds to $2^6 = 64$ possibilities. From the simulation above, for the input A = 37 ("10 0101" in binary), S outputs "1011", 11 in base 10. Algebraically:

$$37 mod 13(37) = 37 - floor\left(\frac{37}{13}\right)13 = 37 - floor(2.846\ldots)13 = 37 - 2 * 13 = 11$$

which validates the test.