622.755 – Introduction to Cybersecurity Winter Term, 2020/21

Project Documentation

Roberto Tonino 12040367

1 Initial chioces

The function that will be computed in this project is the **Sum of a set of values**.

The number of the input bits representing the numbers to sum will be 4. Therefore, the name of the function is "4bit full adder".

In a 4bit full adder one of the two inputs needs to have an additional bit which represents the initial carry. In my implementation, the initial carry input will be inserted by Alice. Therefore, the final input bits are:

- Alice: 5 input bits (4 bits representing the number, 1 initial carry)

$$A_3, A_2, A_1, A_0, C_0$$

- Bob: 4 input bits (4 bits representing the number)

$$B_3, B_2, B_1, B_0$$

The output of the function will have 5 bits (4 bits representing the sum, 1 final carry):

$$R_3, R_2, R_1, R_0, C_3$$

Note: C_3 is the fourth carry of the whole function computation. This happens because the 4bit full adder is a concatenation of four 1bit full adders, which have an output carry each and are concatenated by the 4bit full adder internally in order to correctly perform the sum.

2 Circuit description

In Figure 1 I have represented the 4bit full adder using logic ports. The inputs are represented as squares, while the outputs are represented as circles.

3 Script functionalities

3.1 main.py file

- `init` and `main` are used for the initial setup. They are modified versions of [1].

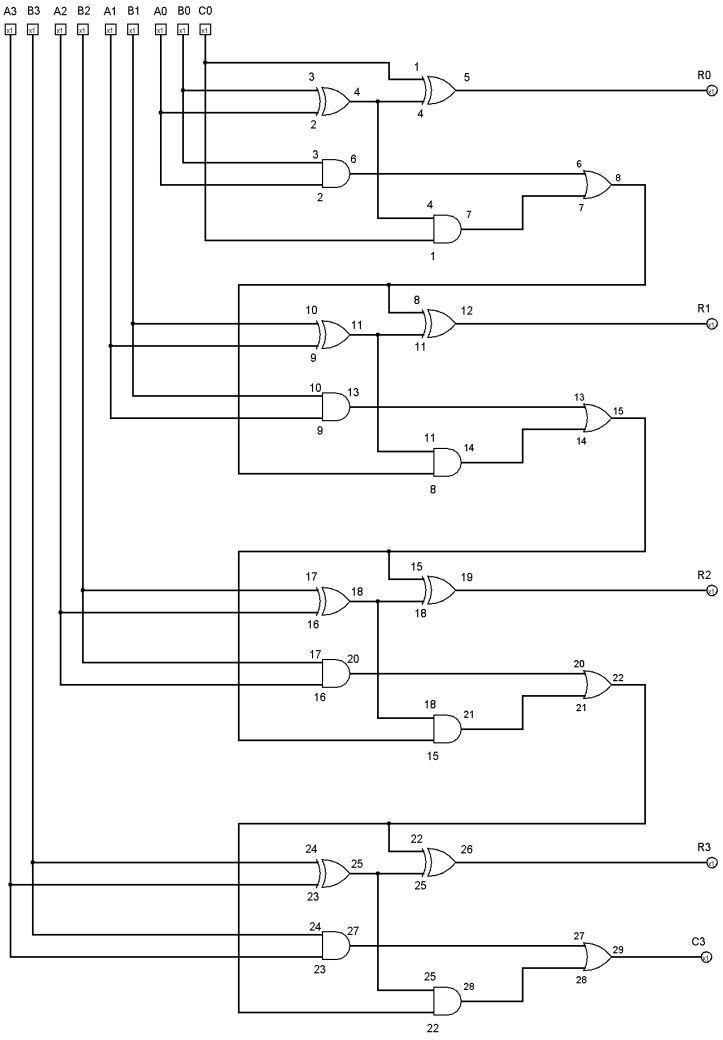

Figure 1: 4bit full adder

- `verifyResults` is used for checking the correctness of the MPC and the non-MPC evaluations. It accepts two strings as arguments that represent two inputs.
- `LocalTest` class is a modified version of [1]. Mainly, the `_print_ealuation` method has being modified in order to perform the MPC function evaluation as well as the non-MPC function evaluation. This method outputs Alice's inputs, Bob's inputs, the MPC circuit output, the non-MPC function output and a human readable `verifyResults` result.

3.2 n_bit_binary_adder.py file

- `add_n_bits` performs a binary sum using concatenated 1bit full adders. The number of bits of each input is passed as the third argument and it determines how

many 1bit full adders will be used.

- `make_n_bit_adder` is an high order function that dynamically generates the correct `add_n_bits` function according to the desired input size.

3.3 binary_adder.py file

- `full_adder` is entirely taken from [2] and simulates a 1bit full adder circuit.

4 Project dependencies

To run this project there are no additional dependencies needed other than those necessary for the github repository to work.

5 Steps to correctly run the script

Unzip roberto_tonino_cs_project.zip, then run the following command to go in the project root folder:

Shell

```
cd roberto_tonino_cs_project
```

In order to execute the script, run (if the command `python` does not work, try using `python3` instead):

Shell

```
python src/main.py local -c src/4bit-adder.json

# It is possible to run the following less verbose command as well
# The first one explains in a better way what operation it is doing
python src/main.py local
```

The output in the terminal will be of this shape:

Shell

```
Started running Yao Protocol...

Alice's wires [23, 16, 9, 2, 1]
Bob's wires [24, 17, 10, 3]

Total wires: 9
======== 4-bit full adder ========
Alice[23, 16, 9, 2, 1] = 00000 - Bob[24, 17, 10, 3] = 0000  ->
↪  Outputs[26, 19, 12, 5, 29] = 00000  -  Correct result = 00000 - Are
↪  they equal? Yes
Alice[23, 16, 9, 2, 1] = 00000 - Bob[24, 17, 10, 3] = 0001  ->
↪  Outputs[26, 19, 12, 5, 29] = 00010  -  Correct result = 00010 - Are
↪  they equal? Yes
Alice[23, 16, 9, 2, 1] = 00000 - Bob[24, 17, 10, 3] = 0010  ->
↪  Outputs[26, 19, 12, 5, 29] = 00100  -  Correct result = 00100 - Are
↪  they equal? Yes
...
...
Alice[23, 16, 9, 2, 1] = 11111 - Bob[24, 17, 10, 3] = 1111  ->
↪  Outputs[26, 19, 12, 5, 29] = 11111  -  Correct result = 11111 - Are
↪  they equal? Yes
```

`Alice[23, 16, 9, 2, 1] = xxxxx` represents the wire numbers in the `src/4bit-adder.json` file and the correspondent bits and are mapped to the inputs in the following way:

$$23, 16, 9, 2, 1 = A_3, A_2, A_1, A_0, C_0$$

`Bob[24, 17, 10, 3] = xxxxx` represents the wire numbers in the `src/4bit-adder.json` file and the correspondent bits and are mapped to the inputs in the following way:

$$24, 17, 10, 3 = B_3, B_2, B_1, B_0$$

`Outputs[26, 19, 12, 5, 29] = xxxxx` represents the wire numbers in the `src/4bit-adder.json` file and the correspondent bits and are mapped to the inputs in the following way:

$$26, 19, 12, 5, 29 = R_3, R_2, R_1, R_0, C_3$$

`Correct result = xxxxx` represents the result of the non-MPC evaluation of the function. It will be equal to the previous output.

References

[1] Olivier Roques. *ojroques/garbled-circuit*. GitHub, Dec. 2020. URL: https://github.com/ojroques/garbled-circuit.

[2] wray. *Binary Adder*. Gist Github, 2018. URL: https://gist.github.com/wray/2de11900a9cb51bed34fb8d77151bb24.