CS 2110 Homework 2
Digital Logic and the ALU

John, Corey, Kevin M, Haojun, Tam

Fall 2021

Contents

1 Overview

1.1 Purpose

You have learned about digital logic, including transistors, gates, and combinational logic. Gates (AND, OR, etc.) can be built using transistors, and Combinational Logic circuits (decoder, MUX, etc.) can be built using gates. Note how the concepts build up from transistors to gates to combinational logic. We have provided you with a tool called Circuitsim (available through Docker) that allows you to simulate building circuits, without actually using physical hardware.

The purpose of this assignment is for you to become proficient building gates out of transistors, and combinational logic circuits out of gates, using Circuitsim. You will put these components together to create an ALU (arithmetic logic unit) circuit, which will allow you to perform several specified math and logic operations using digital logic.

1.2 Task

You will complete three Circuitsim files, and build an ALU from the ground up, starting with transistors, and then gates, and then the remainder of your combinational logic. Please read this entire document for detailed instructions, including which digital logic components you are allowed to use or prohibited from using for each part of the assignment.

This document also contains tutorials on using Circuitsim and creating subcircuits, among other topics.

The steps to complete this assignment include:

1. Create the standard logic gates (NAND, NOR, NOT, AND, OR)
2. Create an 8-input multiplexer and an 8-output decoder
3. Use multiplexers to create a circuit that evaluates the sign of a number
4. Create a 1-bit full adder
5. Create an 8-bit full adder using the 1-bit full adder
6. Use your 8-bit full adder and other components to construct an 8-bit ALU

1.3 Criteria

You will submit three Circuitsim files that you produce to Gradescope: gates.sim, plexers.sim, and alu.sim. Your circuits must work properly and produce the desired results in order to receive credit. For the ALU portion, we will give partial credit for each operation that works properly.

Be sure to avoid using components that are disallowed for each phase of this assignment.

2 Optional Tutorial

Note: This tutorial is optional and to help you get acquainted with CircuitSim before you start this homework. If you're comfortable with this software, feel free to skip to section 2. CircuitSim is an interactive circuit simulation package. We will be using this program for the next couple of homework assignments. This is a tutorial to help you get acquainted with the software. CircuitSim is a powerful simulation tool designed for educational use. This gives it the advantage of being a little more forgiving than some of the more commercial simulators. However, it still requires some time and effort to be able to use the program efficiently. With this in mind, we present you with the following assignment:

2.1 Part 1 — Read Resources

Read through the following resources

- CircuitSim Wires Documentation `https://ra4king.github.io/CircuitSim/docs/wires/`
- Tutorial 1: My First Circuit `https://ra4king.github.io/CircuitSim/tutorial/tut-1-beginner`

2.2 Part 2 — Complete Tutorial 2

Complete Tutorial 2 `https://ra4king.github.io/CircuitSim/tutorial/tut-2-xor`

Instead of saving your file as **xor.sim**, save your file as **part1.sim**. As well, make sure you label your two inputs **a** and **b**, and your output as **c**, as well as rename your subcircuit to xor.

2.3 Part 3 — Complete Tutorial 3

Complete Tutorial 3 `https://ra4king.github.io/CircuitSim/tutorial/tut-3-tunnels-splitters`

Name the subcircuit **umbrella**, the input **in**, and the output **out**. Save your file as **part2.sim**.

3 Instructions

For this assignment, you will be using CircuitSim. The version is included in the 2110 Docker container. If you are having issues with Docker, it can also be found on canvas under Files -> Tools -> CircuitSim.jar.

3.1 Requirements

For the first part of this assignment (the logic gates), you may use **only those components found in the Wiring tab** (being the input/output pins, constants, probes, clocks, splitters, tunnels, and transistors), along with any of the sub-circuits that you create or that already exist.

For the second part of this assignment (the multiplexer and decoder), you may use **only those components found in the Wiring and Gates tabs** along with any of the sub-circuits that you create or that already exist, except in the sign evaluation circuit where you may use the **Plexer Tab** as well.

For the last part of this assignment, you may use only those components found in the **Wiring and Gates tabs (except for the XOR/XNOR gates)** as well as any of the sub-circuits that you create or that already exist. The term sub-circuit refers to any circuits that you have constructed in this part of the homework. If you're unsure on how to utilize a sub-circuit, check out *Section 5.* **For the ALU portion specifically, you may use the Plexer Tab as well.**

Use of anything not listed above will result in heavy deductions. Your need to have everything in their correctly named sub-circuit. More information on sub-circuits is given below.

Use tunnels where necessary to make your designs more readable, but do not overdo it! For gates, muxes, adders and decoders you can often get clean circuits just by placing your components well rather than using tunnels everywhere.

3.2 Part 1: 1-Bit Logic Gates

ALLOWED COMPONENTS: Wiring Tab and Circuits Tab

All of the circuits in this file are in the *gates.sim* file.

For this part of the assignment, you will create a transistor-level implementation of the NAND, NOT, NOR, AND, and OR logic gates.

Remember for this section that you are only allowed to use the components listed in the Wiring section, along with any of the logic gates you are implementing in CircuitSim. For example, once you implement the NOT gate you are free to use that subcircuit in implementing other logic gates. Implementing the gates in the order of the subcircuit tabs can be the easiest option.

As a brief summary of the behavior of each logic gate:

NAND (Inputs: A, B - Output: OUT)

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

NOR (Inputs: A, B - Output: OUT)

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

AND (Inputs: A, B - Output: OUT)

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

OR (Inputs: A, B - Output: OUT)

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

NOT (Input: IN - Output: OUT)

A	NOT A
0	1
1	0

Hint: Start by creating the NAND and NOT gates from transistors. Then use this gate as a subcircuit for implementing the others.

All of the logic gates must be within their named sub-circuits.

3.3 Part 2: Plexers

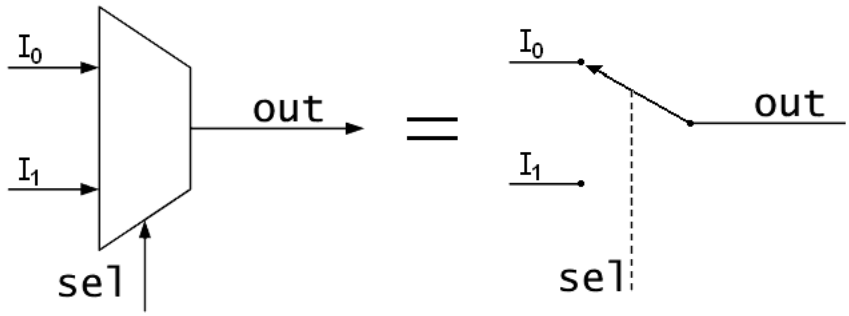
ALLOWED COMPONENTS: Wiring Tab, Circuits Tab, and Gates Tab

NOTE: For the sign evaluation circuit, you may use the Plexer tab as well.

All of the circuits in this file are in the *plexers.sim* file.

3.3.1 Multiplexer

The multiplexer you will be creating has 8 1-bit inputs (labeled appropriately as `A`, `B`, `C`, ..., `H`), a single 3-bit selection input (`SEL`), and one 1-bit output (`OUT`). The multiplexer uses the `SEL` input to choose a specific input line for forwarding to the output.

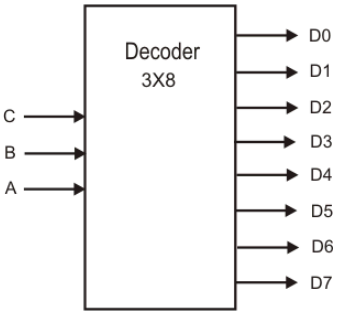


For example:

```
SEL = 000 ==> OUT = A
SEL = 001 ==> OUT = B
SEL = 010 ==> OUT = C
SEL = 011 ==> OUT = D
SEL = 100 ==> OUT = E
SEL = 101 ==> OUT = F
SEL = 110 ==> OUT = G
SEL = 111 ==> OUT = H
```

3.3.2 Decoder

The decoder you will be creating has a single 3-bit selection input (`SEL`), and eight 1-bit outputs (labeled `A`, `B`, `C`, ..., `H`). The decoder uses the `SEL` input to raise a specific output line, as seen below.



For example:

```
SEL = 000 ==> A = 1, BCDEFGH = 0
SEL = 001 ==> B = 1, ACDEFGH = 0
```

```
SEL = 010 ==> C = 1, ABDEFGH = 0
SEL = 011 ==> D = 1, ABCEFGH = 0
SEL = 100 ==> E = 1, ABCDFGH = 0
SEL = 101 ==> F = 1, ABCDEGH = 0
SEL = 110 ==> G = 1, ABCDEFH = 0
SEL = 111 ==> H = 1, ABCDEFG = 0
```

3.3.3 Sign Evaluation

ALLOWED COMPONENTS: Wiring Tab, Circuits Tab, Gates Tab, and Plexer Tab

In this step, you will construct a circuit that takes an 8-bit two's complement input and evaluates whether the input is negative, zero, or positive. Based on this, this circuit will output a 3-bit bit-vector called NZP where the most significant bit is 1 *iff* the input is negative, the middle bit is 1 *iff* the input is zero, and the least significant bit is 1 *iff* the input is positive. Only 1 bit of the output should be set at any given time. Zero is not considered a positive number. **For example, if the input to this circuit is positive, then it will output 001**

With that in mind, set the correct bit and **implement this circuit in the Sign Evaluation subcircuit**

Hint: Recall that in two's complement, the most significant bit can be used to determine the sign of a number!

3.4 Part 3: Adders & ALUs

ALLOWED COMPONENTS: Wiring Tab, Circuits Tab, and Gates Tab

BANNED COMPONENTS: XOR Gate, XNOR Gate

NOTE: For the ALU specifically, you may use the Plexer tab as well.

All of the circuits in this file are in the *alu.sim* file.

3.4.1 1-Bit Adder

The full adder has three 1-bit inputs (`A`, `B`, and `CIN`), and two 1-bit outputs (`SUM` and `COUT`). The full adder adds `A + B + CIN` and places the sum in `SUM` and the carry-out in `COUT`.

For example:

```
A = 0, B = 1, CIN = 0 ==> SUM = 1, COUT = 0
A = 1, B = 0, CIN = 1 ==> SUM = 0, COUT = 1
A = 1, B = 1, CIN = 1 ==> SUM = 1, COUT = 1
```

Hint: Making a truth table of the inputs will help you.

3.4.2 8-Bit Adder

For this part of the assignment, you will daisy-chain 8 of your 1-bit full adders together in order to make an 8-bit full adder.

This circuit should have two 8-bit inputs (`A` and `B`) for the numbers you're adding, and one 1-bit input for `CIN`. The reason for the `CIN` has to do with using the adder for purposes other than adding the two inputs.

There should be one 8-bit output for `SUM` and one 1-bit output for `COUT`.

3.4.3 Basic 8-Bit ALU

ALLOWED COMPONENTS: Wiring Tab, Circuits Tab, Gates Tab, and Plexer Tab

BANNED COMPONENTS: XOR Gate, XNOR Gate

Note that this rule recursively extends to any component that your ALU uses, so you will need to ensure that no other part of the ALU uses the XOR or XNOR gates

You will first create a simple 8-bit ALU, using the 8-bit full adder you created previously.

For this ALU, we will be using a multiplexer. This ALU has two **8-bit** inputs for `A` and `B` and one **2-bit** input for `OP`, the op-code for the operation in the list below. It has one **8-bit** output named `OUT`. The following 4 operations will be selected from:

00. Addition	`[A + B]`
01. AND	`[A & B]`
10. NOT	`[NOT A]`
11. Pass	`[PASS A]`

NOTE: the **Pass** operation simply refers to outputting the value of A unchanged.

Notice that **NOT** and **Pass** only operate on the `A` input. **They should NOT rely on `B` being a particular value.**

The provided autograder will check the op-codes according to the order listed above (Addition (`00`), AND (`01`), etc.) and thus it is important that the operations are in this exact order.

3.4.4 Intermediate 8-Bit ALU

ALLOWED COMPONENTS: Wiring Tab, Circuits Tab, Gates Tab, and Plexer Tab

BANNED COMPONENTS: XOR Gate, XNOR Gate

Note that this rule recursively extends to any component that your ALU uses, so you will need to ensure that no other part of the ALU uses the XOR or XNOR gates

You will next create a different 8-bit ALU, with identical structure as the previous, but more complex operations as outlined below:

00. isEqual	`[A == B]`
01. isMultipleOf8	`[A % 8 == 0]`
10. MultiplyBy7	`[7 * A]`
11. Maximum	`[max(A, B)]`

Notice that **MultiplyBy7** and **isMultipleOf8** only operate on the `A` input. **They should NOT rely on `B` being a particular value.**

For the **MultiplyBy7** operation, although there is potential for overflow, we don't need to worry about it in this homework.

For the **isMultipleOf8** and **isEqual** operations, return `00000001` if the condition is true, and `00000000` otherwise.

The provided autograder will check the op-codes according to the order listed above (isEqual (`00`), isMultipleOf8 (`01`), etc.) and thus it is important that the operations are in this exact order.

3.5 Running the Autograder

To run the autograder, type the following command into your terminal while in the homework 2 directory:

```
java -jar hw02-tester.jar
```

Make sure all the tests have been passed. Keep in mind that even if you get full credit from the autograder, we reserve the right to test for more cases. Note: To guarantee that the autograder runs without error, run it from your Docker container. To do this, either run `./cs2110docker.sh` and open the terminal inside the graphical client, or run `./cs2110docker.sh -it` to open a shell directly in your terminal.

4 Deliverables

Please upload the following files onto the assignment on Gradescope:

1. `gates.sim` — `NOT, NAND, NOR, AND, OR`
2. `plexers.sim` — `Decoder, MUX, Sign Evaluation`
3. `alu.sim` — `1-Bit Adder, 8-Bit Adder, Basic 8-Bit ALU, Intermediate 8-Bit ALU`

Be sure to check your score to see if you submitted the right files, as well as your email frequently until the due date of the assignment for any potential updates.

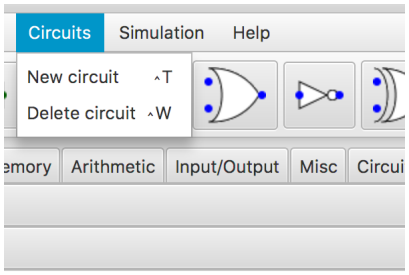
No partial credit will be given for incorrect outputs for Part 1, Part 2, and the adder-portion of Part 3. For the ALUs, partial credit will be awarded on a per-operation basis, wherein each operation must perform successfully to be awarded credit. Because of this, we urge you to check your score before the due date.

5 Sub-Circuit Tutorial

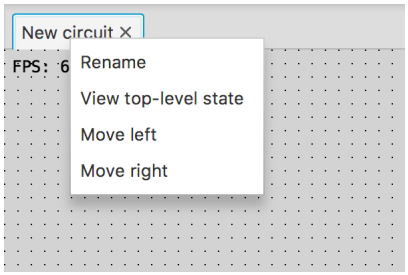
As you build circuits that are more and more sophisticated, you will want to build smaller circuits that you can use multiple times within larger circuits. Sub-circuits behave like classes in Object-Oriented languages. Any changes made in the design of a sub-circuit are automatically reflected wherever it is used. The direction of the IO pins in the sub-circuit correspond to their locations on the representation of the sub-circuit.

To create a sub-circuit:

1. Go to the "Circuits" menu and choose "New circuit"

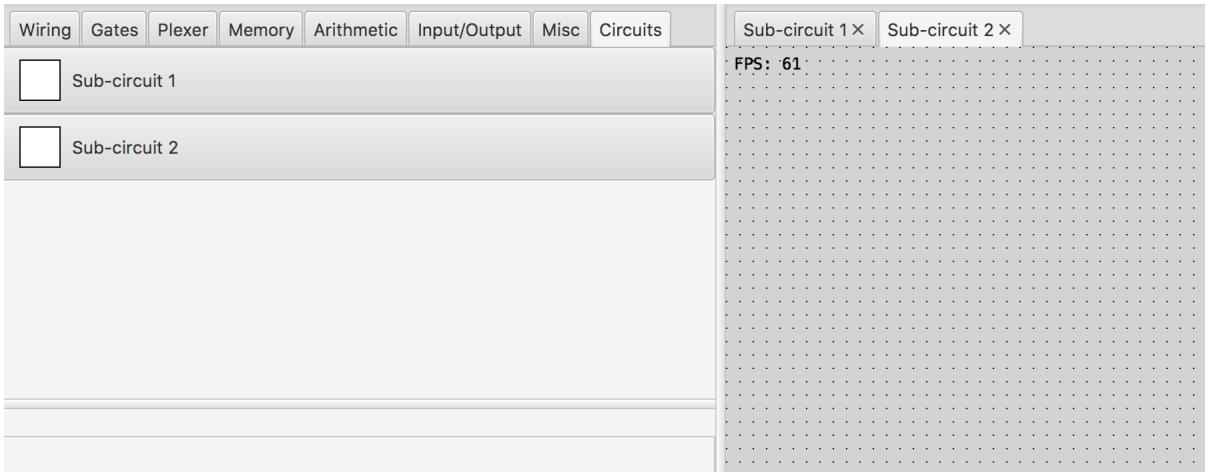


2. Name your circuit by right-clicking on the “New circuit” item and selecting “Rename”

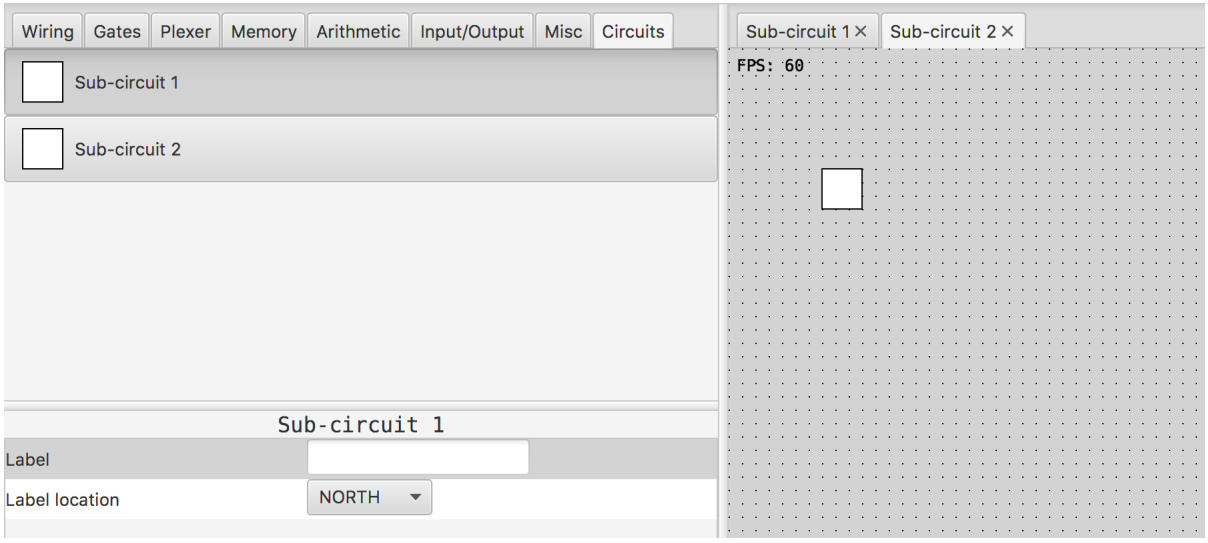


To use a sub-circuit:

1. Click the “Circuits” tab next to the “Misc” tab



2. Select the circuit you wish to use and place it in your design



6 Rules and Regulations

6.1 General Rules

1. Starting with the assembly homeworks, any code you write must be meaningfully commented. You should comment your code in terms of the algorithm you are implementing; we all know what each line of code does.

2. Although you may ask TAs for clarification, you are ultimately responsible for what you submit. This means that (in the case of demos) you should come prepared to explain to the TA how any piece of code you submitted works, even if you copied it from the book or read about it on the internet.

3. Please read the assignment in its entirety before asking questions.

4. Please start assignments early, and ask for help early. Do not email us the night the assignment is due with questions.

5. If you find any problems with the assignment it would be greatly appreciated if you reported them to the author (which can be found at the top of the assignment). Announcements will be posted if the assignment changes.

6.2 Submission Conventions

1. All files you submit for assignments in this course should have your name at the top of the file as a comment for any source code file, and somewhere in the file, near the top, for other files unless otherwise noted.

2. When preparing your submission you may either submit the files individually to Canvas/Gradescope or you may submit an archive (zip or tar.gz only please) of the files. You can create an archive by right clicking on files and selecting the appropriate compress option on your system. Both ways (uploading raw files or an archive) are exactly equivalent, so choose whichever is most convenient for you.

3. Do not submit compiled files, that is .class files for Java code and .o files for C code. Only submit the files we ask for in the assignment.

4. Do not submit links to files. The autograder does not understand it, and we will not manually grade assignments submitted this way as it is easy to change the files after the submission period ends.

6.3 Submission Guidelines

1. You are responsible for turning in assignments on time. This includes allowing for unforeseen circumstances. If you have an emergency let us know **IN ADVANCE** of the due time supplying documentation (i.e. note from the dean, doctor's note, etc). Extensions will only be granted to those who contact us in advance of the deadline and no extensions will be made after the due date.

2. You are also responsible for ensuring that what you turned in is what you meant to turn in. After submitting you should be sure to download your submission into a brand new folder and test if it works. No excuses if you submit the wrong files, what you turn in is what we grade. In addition, your assignment must be turned in via Canvas/Gradescope. Under no circumstances whatsoever we will accept any email submission of an assignment. Note: if you were granted an extension you will still turn in the assignment over Canvas/Gradescope.

6.4 Syllabus Excerpt on Academic Misconduct

Academic misconduct is taken very seriously in this class. Quizzes, timed labs and the final examination are individual work.

Homework assignments are collaborative, In addition many if not all homework assignments will be evaluated via demo or code review. During this evaluation, you will be expected to be able to explain every aspect of your submission. Homework assignments will also be examined using computer programs to find evidence of unauthorized collaboration.

What is unauthorized collaboration? Each individual programming assignment should be coded by you. You may work with others, but each student should be turning in their own version of the assignment. Submissions that are essentially identical will receive a zero and will be sent to the Dean of Students' Office of Academic Integrity. Submissions that are copies that have been superficially modified to conceal that they are copies are also considered unauthorized collaboration.

You are expressly forbidden to supply a copy of your homework to another student via electronic means. This includes simply e-mailing it to them so they can look at it. If you supply an electronic copy of your homework to another student and they are charged with copying, you will also be charged. This includes storing your code on any site which would allow other parties to obtain your code such as but not limited to public repositories (Github), pastebin, etc. If you would like to use version control, use github.gatech.edu

6.5 Is collaboration allowed?

Collaboration is allowed on a high level, meaning that you may discuss design points and concepts relevant to the homework with your peers, share algorithms and pseudo-code, as well as help each other debug code. What you shouldn't be doing, however, is pair programming where you collaborate with each other on a single instance of the code. Furthermore, sending an electronic copy of your homework to another student for them to look at and figure out what is wrong with their code is not an acceptable way to help them, because it is frequently the case that the recipient will simply modify the code and submit it as their own.

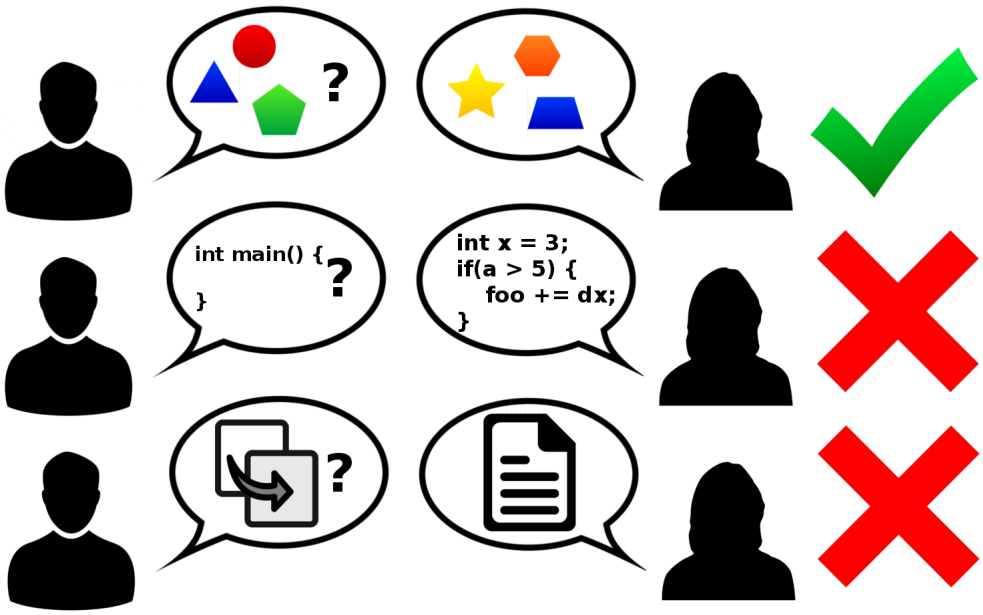


Figure 1: Collaboration rules, explained colorfully