Andreas Vasberg, Patrik Viitala

Lab 1 - Group 5

Captains Log: 

We had already prepared the Arduino IDE before the lab on Andreas's computer. Unfortunately it didn't want to accept the trinket with the USB-drivers. When we then installed Arduino on Patrik's computer it didn't work either. After some searching and restarting Patrik tried with another USB-port and it worked. You could almost feel the magic.

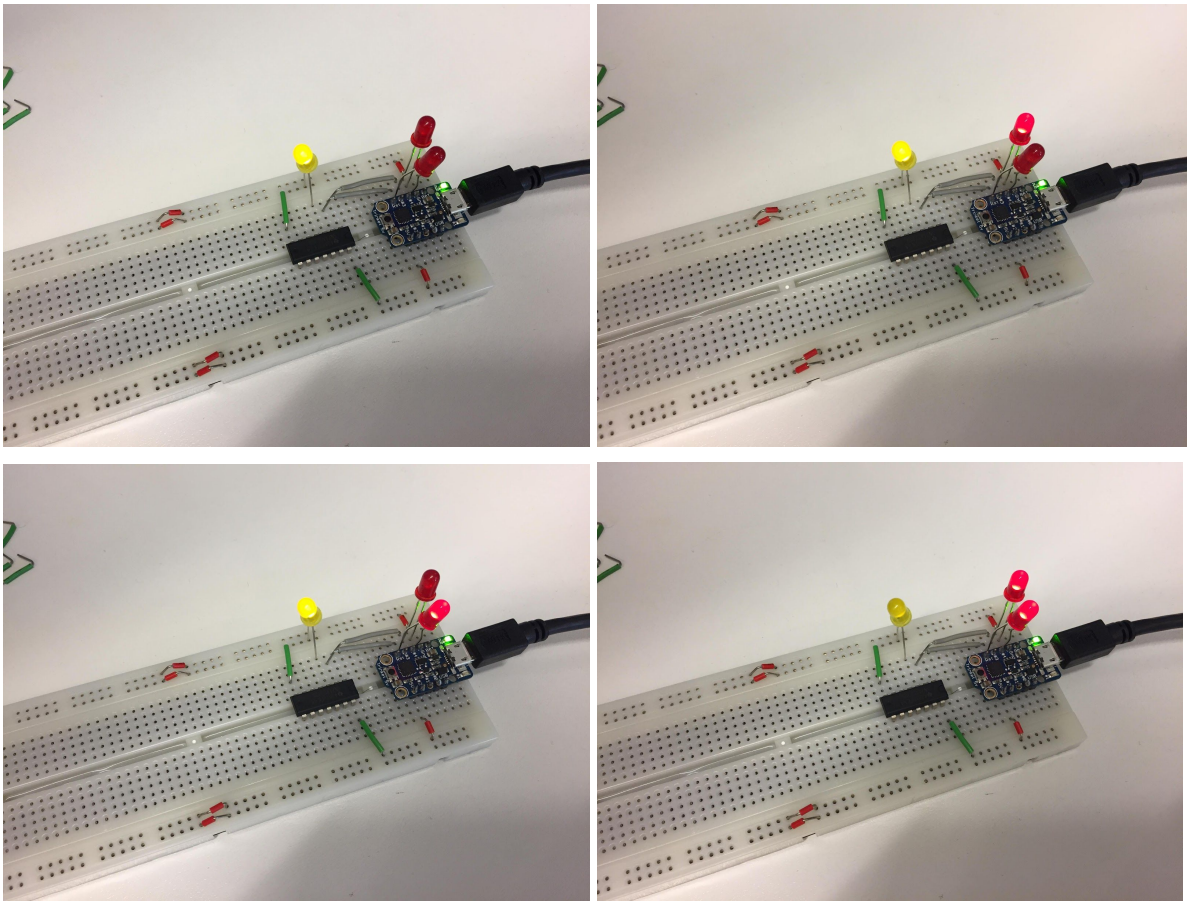
Now we could start working with the tasks. Starting with **task 1**.

Task 1

After putting together the circuit by following the steps and looking at the figure in the instructions, we could write a simple script to test the NAND-gate:

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pins as an output.
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
      digitalWrite(3, LOW);
      digitalWrite(4, LOW);
      delay(1000);
      digitalWrite(3, HIGH);
      delay(1000);
      digitalWrite(3, LOW);
      digitalWrite(4, HIGH);
      delay(1000);
      digitalWrite(3, HIGH);
      delay(1000);
}
```

Andreas Vasberg, Patrik Viitala



We used red LEDs to monitor the input and a yellow LED to monitor the output. As can be seen in these pictures, the input and output matches the truth table for a NAND-gate.

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

We also figured out that when we connected the two red LEDs we lost connection to the computer when pressing the button or trying to start the adafruit. Therefore we unplugged them before uploading the code and then reconnected them.

Andreas Vasberg, Patrik Viitala

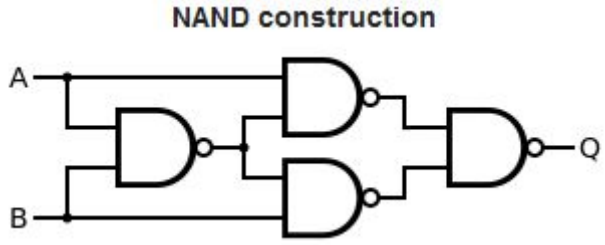
Task 2

Before we started writing code and building the circuit, we used pen and paper and a website called *nandgame.com* to visualize how it should be done. The simulations really helped a lot and quickly we had an idea on how to solve the problem.

From the diagram in the instructions, we saw that the circuit required a XOR-gate and an AND-gate.

A	B	A **XOR** B
0	0	0
0	1	1
1	0	1
1	1	0

An XOR-gate can be constructed using 4 NAND-gates like this:

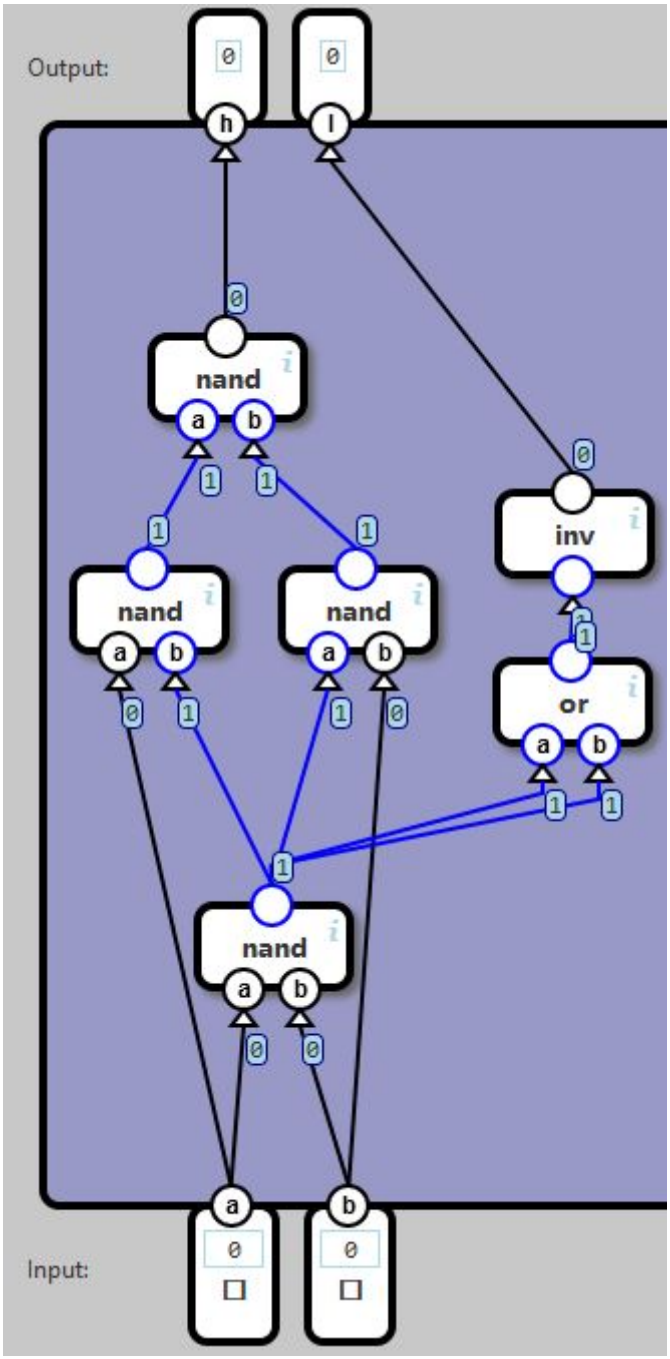


The same could be done for the AND-gate.

Input		Output
A	B	F = A.B
0	0	0
0	1	0
1	0	0
1	1	1

With one NOR-gate left we knew that the only way to get the output 1 was to have both the inputs be 0. Coincidentally, the only way to get the output 0 from a NAND-gate is when both the inputs are 1. By having the inputs A and B go through the NAND-gate and the output of that gate go through the NOR-gate, we had constructed an AND-gate.

Andreas Vasberg, Patrik Viitala



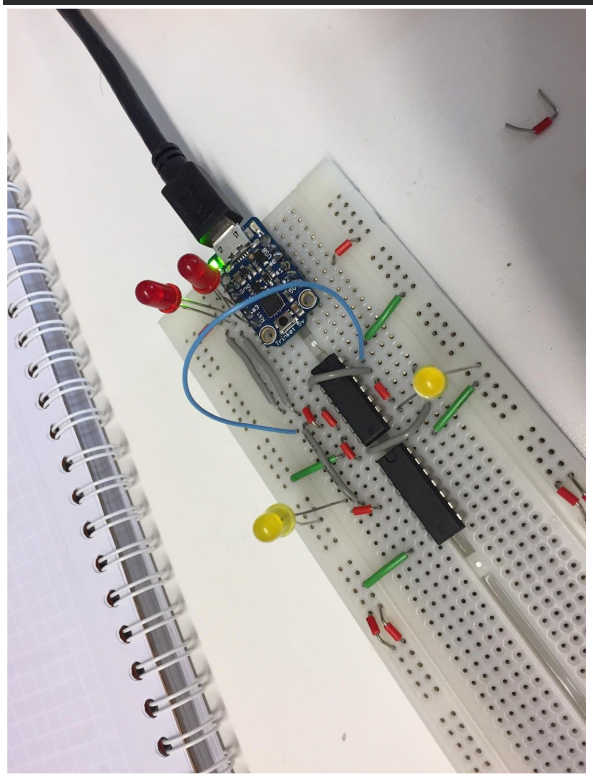
Using *nandgame.com* we could simulate and verify that the circuit worked.

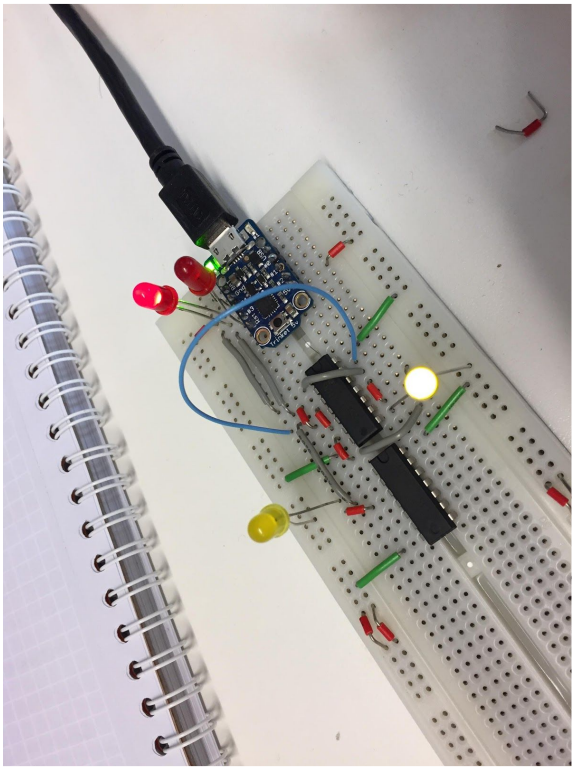
We then made a schematic for the circuit on paper and built it. The two red LEDs were for showing which pin was high or low while the left yellow LED was carry and right yellow LED was SUM.

Half-adder

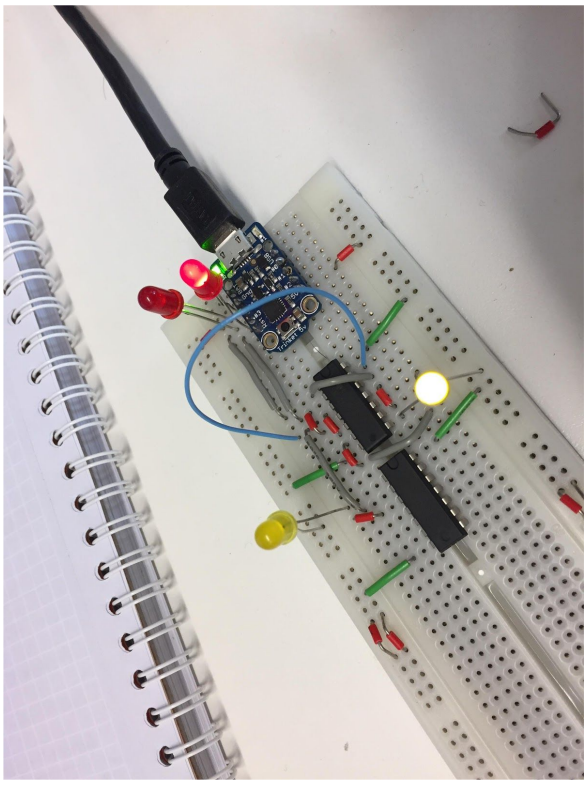
```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pins as an output.
  pinMode(3, OUTPUT);
```

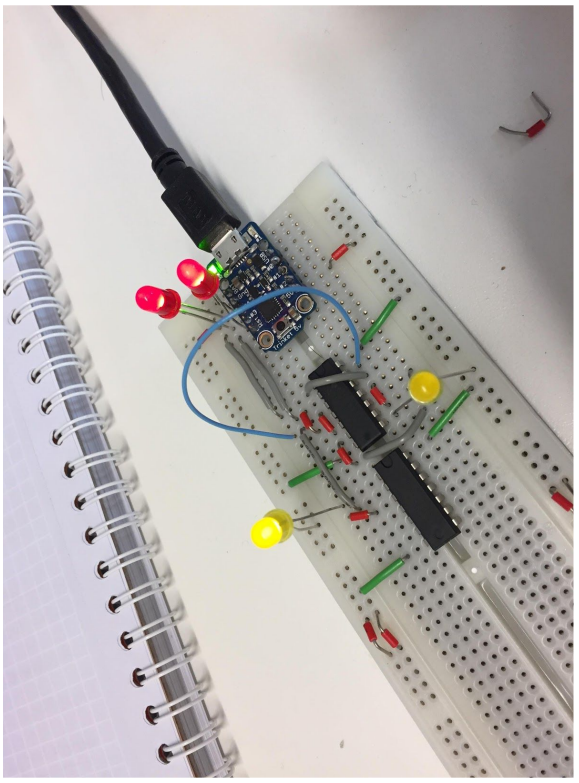
```
  pinMode(4, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
      digitalWrite(3, LOW);
      digitalWrite(4, LOW);
      delay(1000);
      digitalWrite(3, HIGH);
      delay(1000);
      digitalWrite(3, LOW);
      digitalWrite(4, HIGH);
      delay(1000);
      digitalWrite(3, HIGH);
      delay(1000);
}
```





Andreas Vasberg, Patrik Viitala



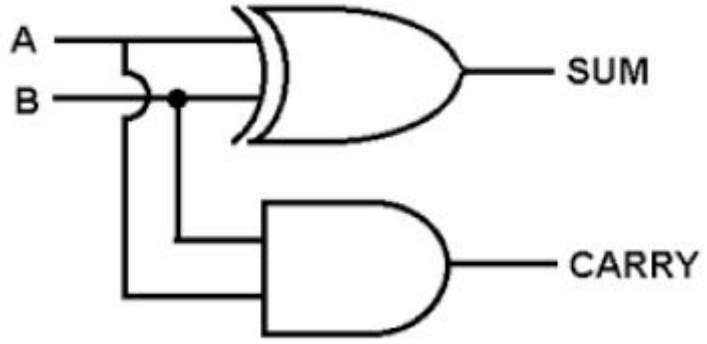


Truth Table			
Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

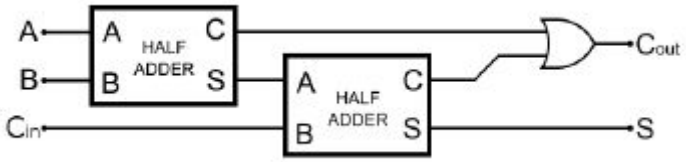
No problem at all. **Task 3** is the only thing left.

Task 3

Here we messed up the outputs in the beginning since the schematic from the instructions was flipped.



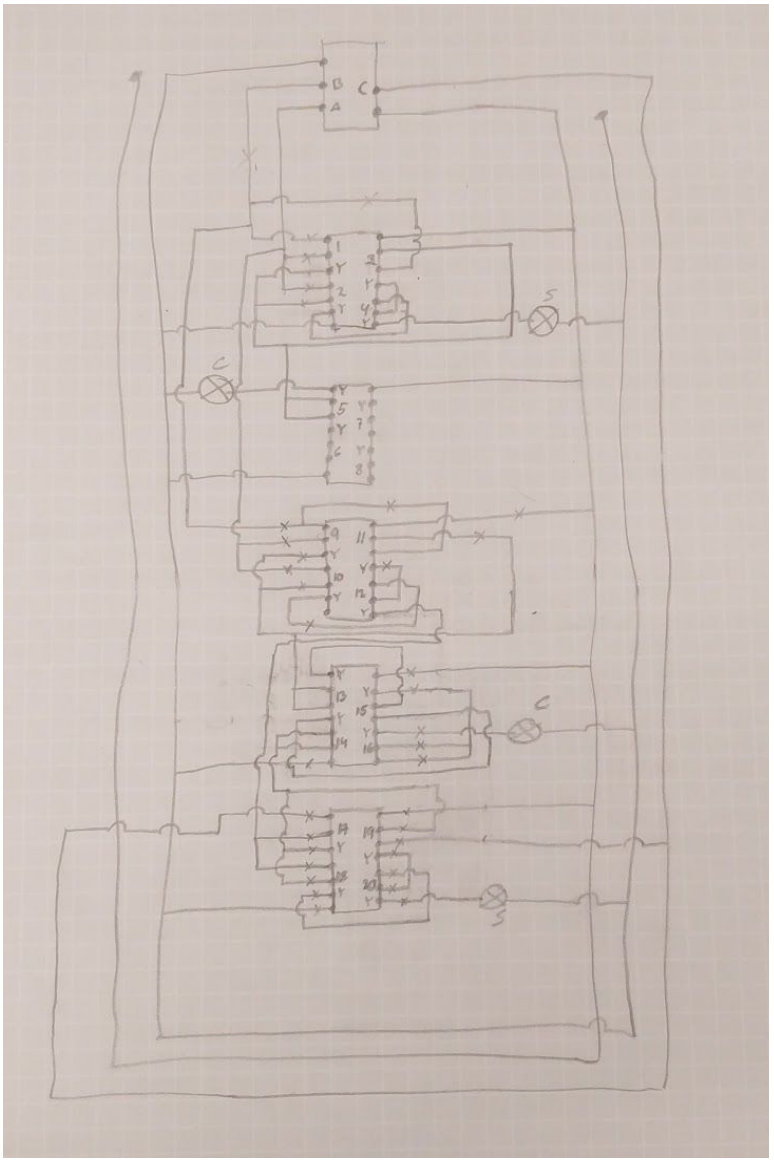
In the schematic for the half adder, the sum was at the top, while the carry was at the bottom. Due to technical errors with the adafruit in the beginning of the lab, we were short on time and decided to rush things. We therefore assumed that the order of the outputs was the same on the full-adder.



By looking closer, it becomes apparent that the outputs are the opposite way in the schematic for the full-adder. After getting more time we sat down and looked at our schematic piece by piece and eventually got it to work.

We had 5 NOR-gates available before we made the OR-gate in the full-adder. After finishing the full-adder, we were left with 3 NOR-gates. We tested the one-bit full adder circuit and it worked perfectly. We then connected the half-adder and the full-adder to each other.

Andreas Vasberg, Patrik Viitala



Our schematics before we linked half-adder carry out output to the full-adder's carry in input.

Full-adder

```
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pins as an output.
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
      digitalWrite(2, LOW);
      digitalWrite(3, LOW);
      digitalWrite(4, LOW);
```

```
      delay(1000);
      digitalWrite(2, HIGH);
      delay(1000);
      digitalWrite(2, LOW);
      digitalWrite(3, HIGH);
      delay(1000);
      digitalWrite(3, LOW);
      digitalWrite(4, HIGH);
      delay(1000);
      digitalWrite(4, LOW);
      digitalWrite(2, HIGH);
      digitalWrite(3, HIGH);
      delay(1000);
      digitalWrite(2, LOW);
      digitalWrite(4, HIGH);
      delay(1000);
      digitalWrite(3, LOW);
      digitalWrite(2, HIGH);
      delay(1000);
      digitalWrite(3, HIGH);
      delay(1000);
}
```

In this script, pin 2 was the carry-in. We verified that this circuit worked without taking any pictures.

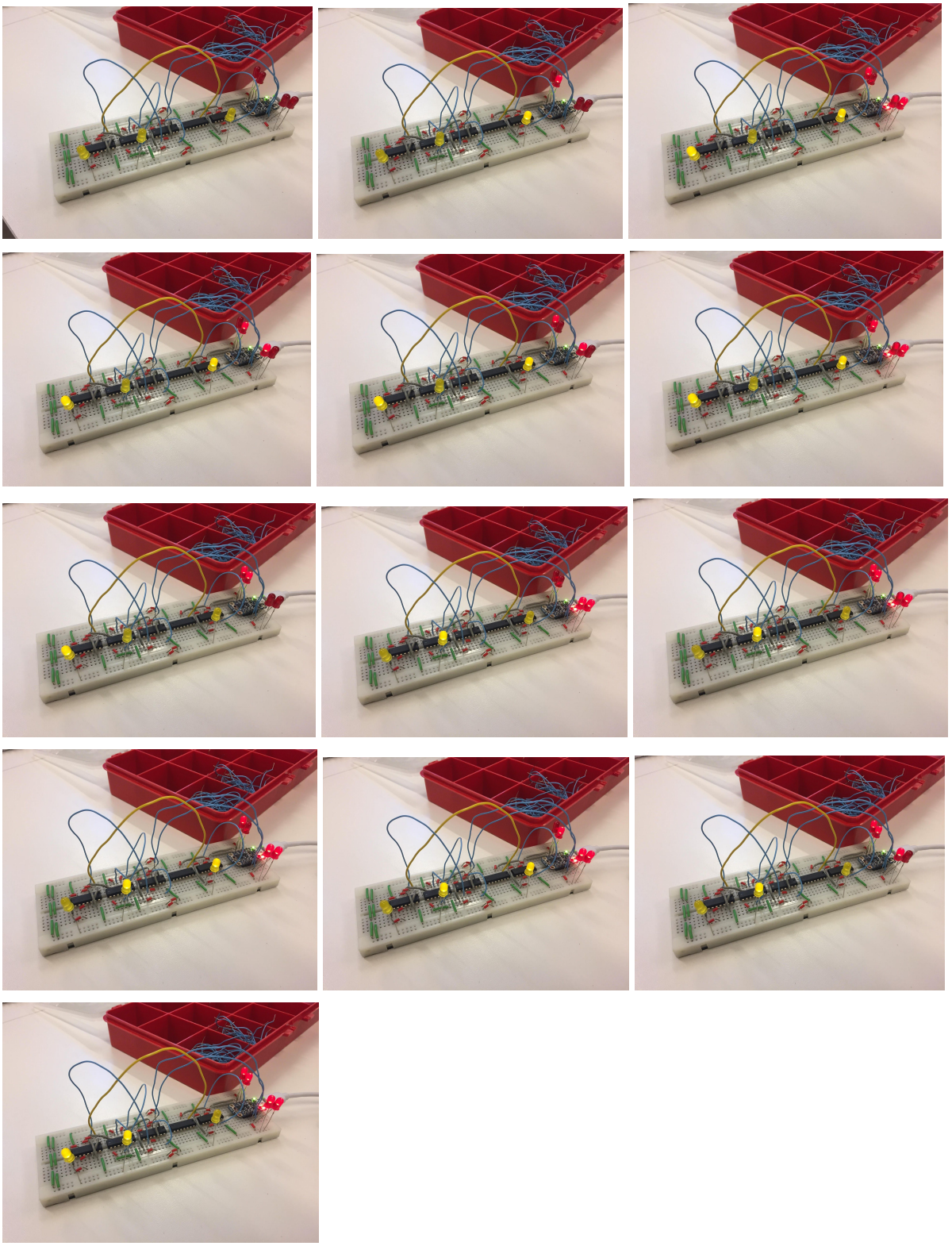
2-bit adder

```
int a0 = 3;
int b0 = 4;
int a1 = 1;
int b1 = 2;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pins as an output.
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
      digitalWrite(1, LOW);
      digitalWrite(2, LOW); // 1
      digitalWrite(3, LOW);
```

```
      digitalWrite(4, LOW);
      delay(1000);
      digitalWrite(b0, HIGH); // 2
      delay(1000);
      digitalWrite(b0, LOW); // 3
      digitalWrite(b1, HIGH);
      delay(1000);
      digitalWrite(b0, HIGH); // 4
      delay(1000);
      digitalWrite(b0, LOW); // 5
      digitalWrite(b1, LOW);
      digitalWrite(a0, HIGH);
      delay(1000);
      digitalWrite(b0, HIGH); // 6
      delay(1000);
      digitalWrite(b0, LOW); // 7
      digitalWrite(b1, HIGH);
      delay(1000);
      digitalWrite(b0, HIGH); // 8
      delay(1000);
      digitalWrite(b0, LOW); // 9
      digitalWrite(b1, LOW);
      digitalWrite(a0, LOW);
      digitalWrite(a1, HIGH);
      delay(1000);
      digitalWrite(b0, HIGH); // 10
      delay(1000);
      digitalWrite(b0, LOW); // 11
      digitalWrite(b1, HIGH);
      delay(1000);
      digitalWrite(b0, HIGH); // 12
      delay(1000);
      digitalWrite(b0, LOW); // 13
      digitalWrite(b1, LOW);
      digitalWrite(a0, HIGH);
      delay(1000);
      digitalWrite(b0, HIGH); // 14
      delay(1000);
      digitalWrite(b0, LOW); // 15
      digitalWrite(b1, HIGH);
      delay(1000);
      digitalWrite(b0, HIGH); // 16
      delay(1000);
}
```

Here we defined the outputs to make it more clear what we tested in each row. We matched

the sequence of a truth table to easily verify that the circuit was correct.



The right yellow LED is s0, The middle yellow LED is carry and the left yellow LED is s1. The red LEDs connected to pin 3 and 4 are inputs A0 and B0 respectively. The red LEDs connected to pin 1 and 2 are inputs A1 and B1 respectively.

Inputs				Outputs		
a1	a0	b1	b0	c	s1	s0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0

Inputs				Outputs		
a1	a0	b1	b0	c	s1	s0
1	0	0	0	0	0	1
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Again, the results on the breadboard matched the truth table.