# Communication - I2C

The aim of this project is to create a multi-master I2C bus, with which transmit data from the masters to the display.. The system must contain 2 master devices and 1 slave, being the sensors and display

## Concepts

### Hardware based solution

While this does not adhere to the I2C specification, it is a simple solution to the problem. The hardware based solution is based on using GPIO pins to force control of the bus, by directly controlling another master. Due to the requirements of the project, this solution is not applicable.

### Token based solution (AKA Token Passing Media Access Control)

An access token is passed in a regular pattern between devices. This token is used to determine which device has control of the bus. The token is passed from device to device, until it reaches the device that wants to take control of the bus. The device that wants to take control of the bus, will then take control of the bus, and pass the token to the next device. The device that has control of the bus, will pass the token to the next device, and relinquish control of the bus. This goes on until the system shuts down. In pseudocode, this would look like the following:

```
Loop forever
{
    Wait for token
    Take control of the bus
    Verify I have control of the bus
    If I have control of the bus
    {
        Do something
    }
    Relinquish control of the bus
    Pass token to next device
}
```

### Register based control (AKA Carrier Sense Media Access Control)

With this concept, the goal of either master is to take control of the bus by writing to the control register. The control register is a 1 byte register, which is located at address 0x00. The control register is used to set the state of the bus. The following states are possible:

| Value | Description |
|---|---|
| 0x00 | Bus free |
| 0x01 | Master 1 |
| 0x02 | Master 2 |

The control register is read by all masters. If a master wants to take control of the bus, it writes to the control register. If the bus is free, the master takes control of the bus. If the bus is taken by another master, the master waits until the bus is free. If the bus is taken by itself, the master does nothing. In pseudocode, this would look like the following:

### Collision Avoidance

An interesting aspect of the I2C bus is the fact that *low* bits have a higher priority than *high* bits. This means that if two masters write to the same register, the master with the lowest address will win. This could be leveraged to our advantage.

In theory, we could read & write the data in turn. If we do this on a bit by bit basis, we can verify that the data being sent is correct. For this purpose, we could attach a IO pin to the bus, and continously read what is being sent over the line. This would allow us to verify that the data is correct.

### Feedback

After discussing the options with Felix, we decided to go with Carrier Sense Media Access Control.

## Implementation

### Device Addresses

| Device | Address |
|---|---|
| Temperature | 0x01 |
| Humidity | 0x02 |
| Display | 0x03 |

### Basic Collision Avoidance

The first step is to test my theory on a basic collision avoidance system. By connecting two arduino's to the bus, I can verify that the system works.

### Hardware

### Software

### Results

## Display

### OLED

To make my life easier, I have made a function for printing lines to the display. This function will automatically increment the cursor, so that the next line will be printed below the previous line. This function also automatically displays the data, so that the user does not have to call the display function. The function is as follows:

```
int cursorY = 0;

void PrintLine(String title, int data)
{
  oled.print(title);
  oled.print(data);
  oled.display();
  cursorY += 16;
  oled.setCursor(0,cursorY);
}

void ResetCursor()
{
  oled.setCursor(0,0);
```

```
  cursorY = 0;
}
```

This in turn, allows me to simply loop through the data, and print it to the display. The code for this is as follows:

```
void loop()
{
    PrintLine("Temp: ", temp);
    PrintLine("Hum: ", hum);

    oled.clear(PAGE);
    delay(DELAY); // Delay to prevent flooding the bus, as well as have some modicum of stal
}
```

### Recieving I2C

## References