Wireless communications

IOT specialization | HBO-ICT Software Engineering

Ruben Smit (423723), Willem Dekker (436608)

Table of contents

1. Introduction

This document describes the answers and results of assignments for the Wireless Communication course of HBO-ICT as carried out by Willem Dekker and Ruben Smit. Firstly a higher level radio application using Bluetooth Low Energy enabling communication between three different devices is documented. Secondly all assingments about the low level NRF24L01+ radio are documented including the Led app. Finally our preformed research using the Packet Error Rate and Throughput test apps is documented.

2. High level Radio

This documentation describes our high level radio project in which we demonstrate Bluetooth Low Energy (BLE) communication between three different devices. First we will describe the goal of our system. Next the roles of the different devices are described followed by the services and characteristics used in the BLE communciation. We also note how to use the system and the inner workings of the code is explained. Finally some demonstrations of the working system are shown and how we tested the system.

2.1 System Goal

The goal of this system is to send the current angle of a potentiometer to an android application where it is displayed on a dashboard. From this dashboard it should be possible to control the angle of a servo motor, the angle of the servo should be shown on the dashboard. It should also be possible to link the angle of the servo motor to the angle of the potentiometer.

2.2 Roles of of the Devices

The system will consist of three devices: a NRF52, a Android phone and a FiPy. Each of these devices will have a different role and task in the system which will be described below.

2.2.1 NRF52

The NRF52 will act as a BLE Peripheral. A potentiometer will be attached to the NRF52 and the angle of this potentiometer can be read by a connected central.

2.2.2 FiPy

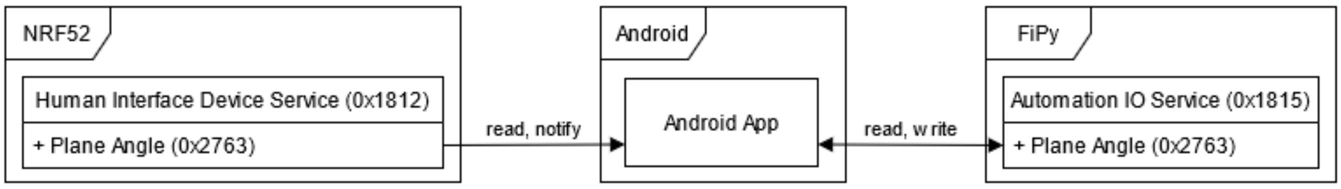
The FiPy will act as a BLE Peripheral. It will be attached to a servo and the current angle of this servo can be read by a connected central. It will also be possible to set a new angle for the servo.

2.2.3 Android phone

On the android phone a appplication will run that displays a dashboard. It will function as a BLE Central and connect to NRF52 and FiPy peripherals. It wil read the angle of the potentiometers of connected NRF52 peripherals and display those on the dashboard. When the angle of a potentiometer changes the dashboard is updated. The current angle of the servos of connected FiPy peripherals is also displayed. Using the dashboard the angle of the servos can be changed.

2.3 Services and Characteristics

Both peripheral types expose a BLE service with a characteristic, the Android phone connects to these peripherals and uses these services and characteristics. A description will be given of the services and characteristics available from each peripheral and whether you can read, write or be notified for each characteristic. A overview of the different services and characteristics and how they are used is shown in the figure below.



2.3.1 NRF52

- **Service: Human Interface Device (0x1812)** This service was chosen because the NRF52 acts like a special kind of HID (Human Interface Device). It takes human input and makes it available to connected devices.
- **Characteristic: Plane angle (0x2763)** This characteristic was chosen because we are looking for the plane angle to set the servo to. And we read the angle (resistance) of the potentiometer to get it. The angle is sent in degrees.
  - **Read** The current angle of the potentiometer can be read.
  - **Notify** When the angle of the potentiometer changes connected devices can be notified.

2.3.2 FiPy

- **Service: Automation IO (0x1815)** This service was chosen because we are altering the IO of the fipy.
- **Characteristic: Plane angle (0x2763)** This was chosen because we are setting the plane angle of the servo. The angle is send and set in degrees.
  - **Read** The current angle of the servo can be read.
  - **Write** The angle of the servo can be changed by writing a new value.

2.3.3 Android phone

The Android phone does not expose any services and characteristics. It does however read the angle of connected NRF52 peripherals and subscribes to notifications in changes to this angle. It also reads the angles of connected FiPy peripherals and writes a new angle if needed.

2.4 System Usage

The following chapter will describe how the system can be used. First we descibe how each component of the system, the NRF52, FiPy and Android phone, must be prepared. Next we show how all components can be used together and how the dashboard can be used.

2.4.1 Preparing the NRF52

Follow the following steps to prepare the NRF52 with the attached potentiometer to be used as a Human Interface Device.

Bill of materials

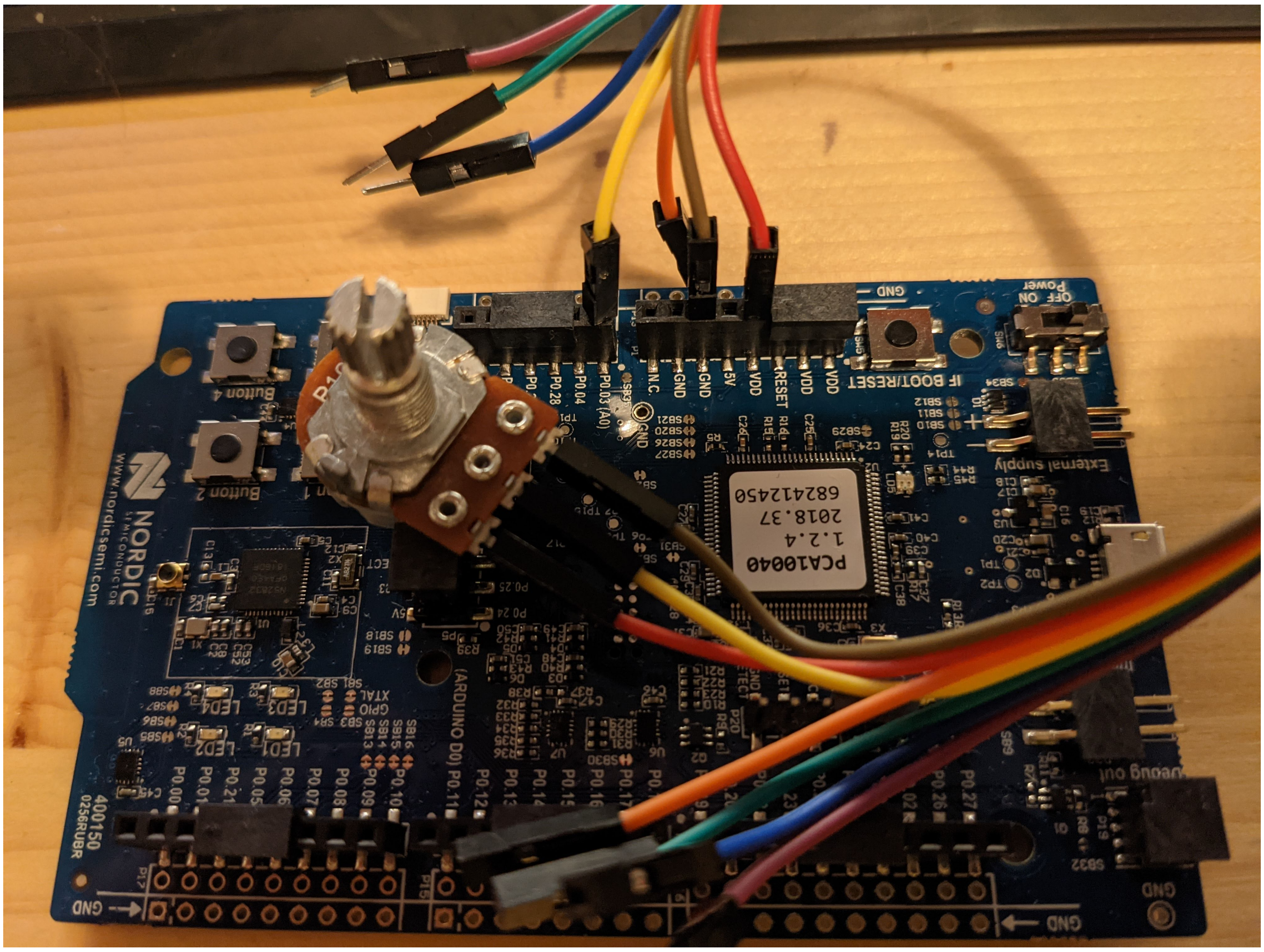
The following materials are needed:

- Nordic-nRF52-DK
- 10K Ohm potentiometer
- 3 Jumper wires male-female

Connecting the materials

Connect the potentiometer to the NRF50 according to the following schematic. In the image below a example is shown of how these are connected.

NRF52 pin	Potentiometer pin
VDD	Fixed end (P1)
A0	Wiper (P2)
GND	Fixed end (P3)



Uploading the code

Upload the code found in /nrf52 to the NRF52 using Visual Studio Code with the PlatformIO plugin.

2.4.2 Preparing the FiPy

Follwo the following steps to prepare the FiPy with the attached servo to be used as a Automation IO device.

Bill of materials

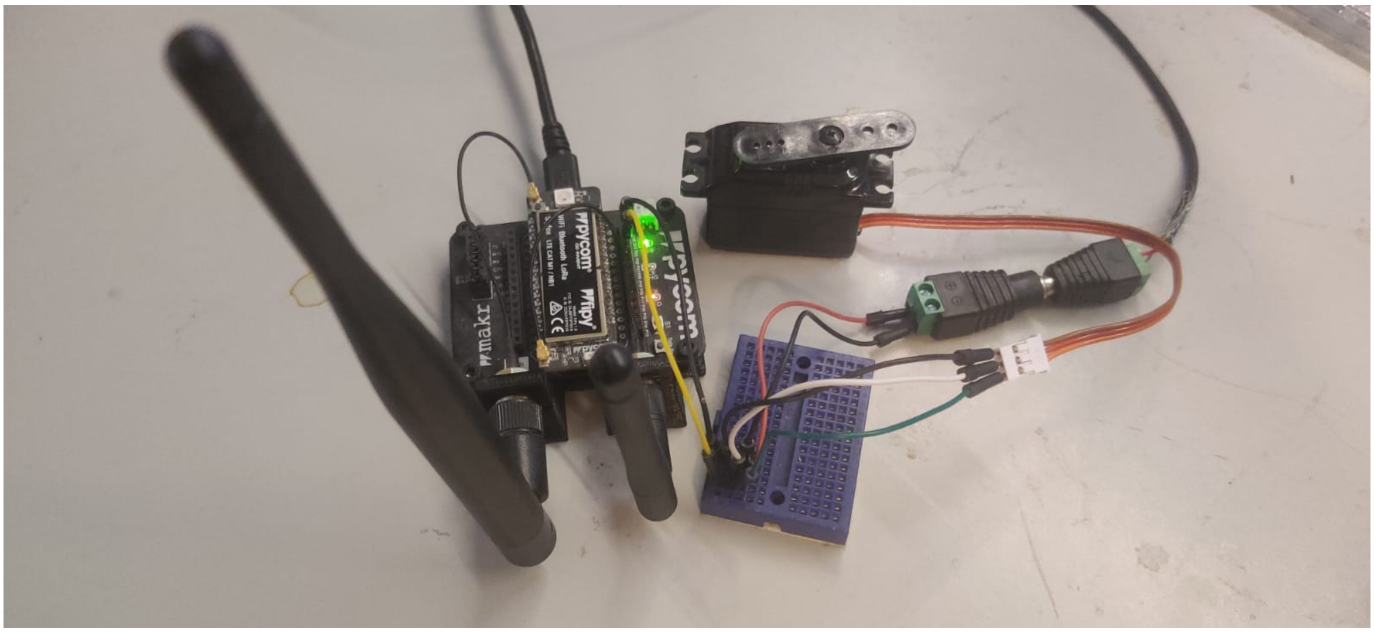
The following materials are needed:

- FiPy with Expansion board
- Servo motor
- 5V 500mA External power supply
- Breadboard
- 7 Jumper wires male-male

Connecting the materials

Connect the servo to the FiPy and the power supply on the breadboard according to the following schematic. In the image below a example is shown of how these are connected.

FiPy pin	Servo pin	External power supply (5V 500mA)
	5v	Positive
P23	Signal	
GND	GND	Negative



Uploading the code

Upload the code found in /fipy to the FiPy using Visual Studio Code with the Pycom plugin.

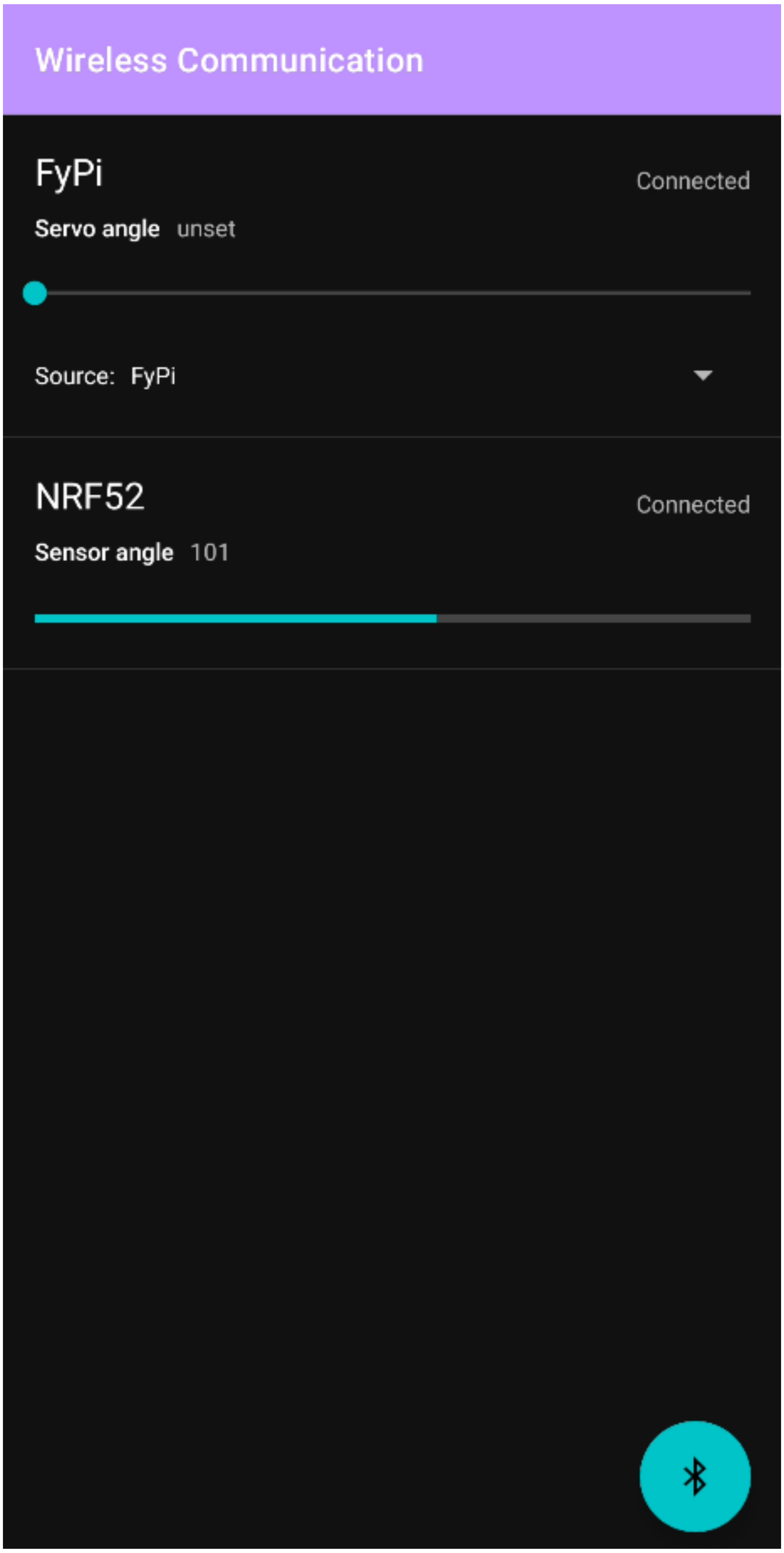
2.4.3 Preparing the Android phone

To use a Android phone as central with dashboard in the system, upload the code found in /android to a Android phone using Android studio.

2.4.4 Operation

To use the system follow the following steps:

1. Make sure all peripherals are turned on and prepared correctly.
2. Open the application on the Android phone
3. Press the big bluetooth button in the bottom right corner. The android device will now find and connect to all available peripherals. Please grant access to bluetooth and location services if requested.
4. A list of available peripherals and their status is shown once they are connected as can be seen in the image below.



Wireless Communication

FyPi

Connected

Servo angle unset

Source: FyPi

NRF52

Connected

Sensor angle 101

For each connected NRF52 a bar is shown displaying the current angle of the potentiometer. When the potentiometer is turned this bar is updated. Every connected FyPi is also shown in the list. It is possibe to set the angle of the FyPi using the slider. Using the source dropdown it is possible to connect the angle of the servo to the angle of a potentiometer. When the potentiometer is turned the angle of the servo will be matched.

2.5 Code

For every device a program has been written to enable the communication between the various components, connect the sensors and actuators and to display their status on the dashboard. Next we will explain how the code for each device works.

2.5.1 NRF52

The code for the NRF52 consists of a main function and a Human Interface Device service class and can be found in /nrf52/src/main.cpp.

Main function

On boot the main function is called. This function starts all services and is shown below.

```
int main() {
    BLE &ble_interface = BLE::Instance();
    events::EventQueue event_queue;
    HidService hid_service;
    BLEProcess ble_process(event_queue, ble_interface);

    ble_process.on_init(callback(&hid_service, &HidService::start));

    // bind the event queue to the ble interface, initialize the interface
    // and start advertising
    ble_process.start();

    // Process the event queue.
    event_queue.dispatch_forever();

    return 0;
}
```

First a instance of the ble interface and a event queue is created. Next a instance of the human interface device (HID) service is created. A bluetooth low energy process is created and the event queue and bluetooth interface are attached to it. When the bluetooth process is initiated the HID service is started. We start advertising and continue processing the event queue as long as the application is running.

Human Interface Device Service

The Human Interface Device Service manages the HID service and the angle characteristic for the GATT server.

```
void start(BLE &ble_interface, events::EventQueue &event_queue)
{
      if (_event_queue) {
        return;
    }

    _server = &ble_interface.gattServer();
    _event_queue = &event_queue;

    // register the service
    printf("Adding service\r\n");
    ble_error_t err = _server->addService(_hid_service);

    if (err) {
        printf("Error %u during service registration.\r\n", err);
        return;
    }

    // read write handler
    _server->onDataSent(as_cb(&Self::when_data_sent));
    _server->onDataRead(as_cb(&Self::when_data_read));

    // updates subscribtion handlers
    _server->onUpdatesEnabled(as_cb(&Self::when_update_enabled));
    _server->onUpdatesDisabled(as_cb(&Self::when_update_disabled));

    // print the handles
    printf("human interface device service registered\r\n");
    printf("service handle: %u\r\n", _hid_service.getHandle());
    printf("\angle characteristic value handle %u\r\n", _angle_char.getValueHandle());

    _event_queue->call_every(1000 /* ms */, callback(this, &Self::read_angle));
    _event_queue->call_every(1000 /* ms */, callback(this, &Self::blink_led));
}
```

When the service is started it registers the hid service to the GATT server. Next it registers the handlers for sending and reading data and enabling and disabling of updates. Two events are added to the event queue to be run every second, reading the angle of the potentiometer and blinking the heartbeat led.

```
void read_angle(void)
{
    uint8_t angle = (uint8_t) map(_angle_sensor.read(), 0, 1, 0, 180);
    printf("read angle as %i\r\n", angle);

    ble_error_t err = _angle_char.set(*_server, angle);
    if (err) {
        printf("write of the angle value returned error %u\r\n", err);
        return;
    }
}
```

After reading the angle it is mapped to a value of 0 to 180 degrees. Then the angle characteristic is updated with the new angle and any subscribers are notified.

2.5.2 FiPy

The code for the FiPy can be found in /fipy/main.py.

```
bluetooth = Bluetooth() # Get a bluetooth instance
bluetooth.set_advertisement(name='FyPi') # Set the name
bluetooth.callback(trigger=Bluetooth.CLIENT_CONNECTED | Bluetooth.CLIENT_DISCONNECTED, handler=conn_cb) # set up the callbacks for connect and disconnect events
bluetooth.advertise(True) # advertise the device
```

When the FiPy is booted a bluetooth instance is created and prepared for advertisement and started. Also connection and disconnection callbacks are added.

```
srv1 = bluetooth.service(uuid=0x1815 , isprimary=True) # set up the service to display the current angle of the servo
chr1 = srv1.characteristic(uuid=0x2763 , value=currentAngle) # set up the characteristic to get the server angle
char1_cb = chr1.callback(trigger=Bluetooth.CHAR_WRITE_EVENT, handler=char1_cb_handler) # set up the callback when writen to characteristic
```

Next the Automation IO service is created and a angle characteristic is added to the service. A callback is created to handle write events to the angle characteristic.

```
pwm = PWM(0, frequency=50) # make a pwm provider
servo = pwm.channel(0, pin='P23', duty_cycle=0.0) # Setup the pwm for the servo
setServoPwn(currentAngle) # Set the servo the the initial angle
```

Finally a Pulse With Modulation (PWM) provider is created with a channel to control the servo. The correct pwm value is determined for the current angle and the servo is moved to the initial angle.

```
def char1_cb_handler(chr, data):
    events, value = data # store the events and data
    if  events & Bluetooth.CHAR_WRITE_EVENT: # if the event was a write event
        currentAngle = int.from_bytes(value, "big") # get the value from the payload
        setServoPwn(currentAngle) # set the servo to the right position
        chr1.value(currentAngle) # update the value that is displayed over Bluetooth
        print("Set new angle: ", currentAngle)
```

When a write event occurs for the angle characteristic it is handled by the event handler. The new angle is read from the payload and stored. A new pwm value for the servo is calculated and the servo is moved to the right position. Finally the characteristic is updated with the new angle value.

2.5.3 Android

The android application consists of several components. First a global overview will be given of these components and their relation. Next each component will be described in detail. The components of the application are:

- **Main Activity** Is started when the application boots. It initializes all other components of the application and manages the bluetooth connections and scanning.
- **Device Model** Stores all information about and handles the communication with a single peripheral.
- **Bluetooth Devices Provider** Stores and manages a list of all connected peripherals.
- **Bluetooth Devices List Adapter** Handles the displaying of the list of connected peripherals.

When the Main Activity is started and the bluetooth connection button is pressed the Android phone will start scanning for available peripherals. Once a peripheral is found a Device Model is created for the peripheral and stored in the Bluetooth Devices Provider.

The Main Activity will supply the Bluetooth Devices List Adapter with the list of devices to be displayed. When a new device is found the Main Activity will notify the List Adapter of the change. When a property of a device changes it wil notify the Main Activity which in turn will notify the List Adapter. In this way all changes to the devices will be displayed in the list.

Main Activity

When the Main Activity is started the `onCreate` method is called. It contains the following code.

```
// Initializes Bluetooth adapter.
final BluetoothManager bluetoothManager =
        (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
bluetoothAdapter = bluetoothManager.getAdapter();

// Ensures Bluetooth is available on the device and it is enabled. If not,
// displays a dialog requesting user permission to enable Bluetooth.
if (bluetoothAdapter == null || !bluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}

bluetoothDevicesListAdapter = new BluetoothDevicesListAdapter(this);

FloatingActionButton fab = findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        scanLeDevice();
    }
});
```

In it the Bluetooth manager and adapter are initialized. We ensure bluetooth is enabled and permission is given to use the bluetooth services. Next the List Adapter is initialized for displaying the list of connected peripherals. A listner is created for the floating action button that, when the button is pressed, starts scanning for bluetooth devices.

```
private void scanLeDevice() {
    Log.i(TAG, "Started scanning for devices");
    BluetoothDevicesProvider.clear();
    if (!mScanning) {
        // Stops scanning after a pre-defined scan period.
        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                mScanning = false;
                bluetoothLeScanner.stopScan(leScanCallback);
                Log.i(TAG, "Stopped scanning for devices");
            }
        }, SCAN_PERIOD);

        // Scan for devices matching the filter and use the callback for found devices
        mScanning = true;
        List<ScanFilter> filters = deviceFilters();
        ScanSettings settings = new ScanSettings.Builder().setScanMode(ScanSettings.SCAN_MODE_LOW_LATENCY).build();
        bluetoothLeScanner.startScan(filters, settings, leScanCallback);
    } else {
        // Stop scanning if the button is pressed again
        mScanning = false;
        bluetoothLeScanner.stopScan(leScanCallback);
        Log.i(TAG, "Stopped scanning for devices");
    }
}
```

When the scanning is started a delayed handler is created that will stop the scanning after a predefined period. A filter is applied to the scanner to only return NRF52 and FiPy devices. Also a callback is defined to handle any found devices.

```
private ScanCallback leScanCallback =
        new ScanCallback() {
            @Override
            public void onScanResult(int callbackType, ScanResult result) {
                // Get the bluetooth device
                Log.i(TAG, "Found a bluetooth device!");
                super.onScanResult(callbackType, result);
                BluetoothDevice bluetoothDevice = result.getDevice();

                if (!BluetoothDevicesProvider.contains(bluetoothDevice.getAddress())) {
                    // Add the device to the list and observe it
                    Log.i(TAG, "Adding new device to list");
                    Device device = new Device(bluetoothDevice, getContext());
                    BluetoothDevicesProvider.addDevice(device);
                    device.addObserver(getObserver());
                }
            }
        };
```

The scan callback creates a Device model for each found device and adds it to the Bluetooth Devices Provider. Also the Main Activity is adde as a observer to the Device model.

```
public void update(Observable observable, Object o) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            int index = BluetoothDevicesProvider.deviceList.indexOf((Device) observable);
            bluetoothDevicesListAdapter.notifyItemChanged(index);
        }
    });
}
```

When there are changes to a Device model the Main Activity, as observer, is notified. Consequently it will start a runner on the UI thread to notify the List Adapter about changes to the item.

Device Model

```
public Device(BluetoothDevice bluetoothDevice, Context context) {
    this.device = bluetoothDevice;
    this.context = context;

    Log.i(TAG, "Attempt connecting to gatt service.");
    bluetoothGatt = device.connectGatt(context, false, gattCallback);
}
```

When created the Device model stores the bluetooth device and attempts to connect to the gatt service of the device. The gatt callback handles the events that emanate from the gatt service. The following events are handled:

- **onConnectionStateChange** When the connection state is changed is is stored and all observers are notified and a attempt is made to discover the services.
- **onServicesDiscovered** When services are discovered a attempt is made to determine the service type.
- **onCharacteristicChanged** When a notification is recieved that a characteristich has changed a attempt is made to read the characteristic.
- **onCharacteristicRead** When the angle characteristic is read the format of the data is determined and the angle is stored.
- **onCharacteristicWrite** When a characteristic is successfully written this is logged.

```
private void setServiceType(BluetoothGatt gatt) {
    for (BluetoothGattService service: gatt.getServices()) {
        UUID uuid = service.getUuid();
        Log.i(TAG, "Found service: " + uuid + " for device: " + device.getName());
        if (uuid.equals(UUID_HUMAN_INTERFACE_DEVICE_SERVICE)) {
            // If the device is a sensor, subscribe to new sensor data
            Log.i(TAG, "This is a sensor!");
            deviceType = TYPE_SENSOR;
            subscribeToSensorData(service);
        } else if (uuid.equals(UUID_GENERIC_ATTRIBUTE_SERVICE)) {
            Log.i(TAG, "This is a servo!");
            deviceType = TYPE_SERVO;
        }
    }
}
```

When a service is detected it is matched against the sensor and servo service UUIDs. When a match is found the device type is stored. For sensors a attempt is made to subscribe to the sensor data.

```
private void subscribeToSensorData(BluetoothGattService service) {
    for (BluetoothGattCharacteristic characteristic: service.getCharacteristics()) {
        Log.i(TAG, "Found characteristic: " + characteristic.getUuid() + " for device: " + device.getName());
    }

    BluetoothGattCharacteristic characteristic = service.getCharacteristic(UUID_PLANE_ANGLE_CHARACTERISTIC);
    bluetoothGatt.setCharacteristicNotification(characteristic, true);
    BluetoothGattDescriptor descriptor = characteristic.getDescriptor(CLIENT_CHARACTERISTIC_CONFIG);
    descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
    bluetoothGatt.writeDescriptor(descriptor);

    Log.i(TAG, "Subscribed to sensor characteristic: " + characteristic);

    bluetoothGatt.readCharacteristic(characteristic);
}
```

When subscribing to the sensor data the angle characteristic is retrieved and the bluetooth gatt server of the android phone is instructed to subscribe to notifications for the characteristic. Next a attempt is made to read the angle characteristic.

```
private void setAngle(int angle) {
    if (angle != this.angle) {
        this.angle = angle;
        setChanged();
        notifyObservers();
    }
}
```

When a new angle is stored and it differs from the previous angle, the observers of the Device model are notified.

```
public void writeAngle(int angle, boolean notify) {
    if (angle != this.angle) {
        this.angle = angle;
        if (notify) {
            setChanged();
            notifyObservers();
        }
        Log.i(TAG, "Set angle to: " + angle);

        BluetoothGattService service = bluetoothGatt.getService(UUID_GENERIC_ATTRIBUTE_SERVICE);
        BluetoothGattCharacteristic characteristic = service.getCharacteristic(UUID_PLANE_ANGLE_CHARACTERISTIC);
        characteristic.setValue(angle, BluetoothGattCharacteristic.FORMAT_UINT8, 0);
        bluetoothGatt.writeCharacteristic(characteristic);
    }
}
```

When a new angle is written to a Device model a attempt is made to write it to the characteristic. If specified the observers of the model are notified of the change.

Bluetooth Devices Provider

```
public static List<Device> deviceList;
public static Map<String, Device> deviceMap;
public static BluetoothDevicesListAdapter adapter;
```

The BluetoothDevicesProvider maintains a list of all connected devices. It also stores the instance of the devices list adapter that displays the list of devices so it can notify the adapter of any changes to the list.

```
public static void addDevice(Device device) {
    if (!contains(device.getDeviceAddress())) {
        deviceList.add(device);
        deviceMap.put(device.getDeviceAddress(), device);
        adapter.notifyDataSetChanged();
    }
}
```

New devices can be added to the list and the adapter will be notified of the change.

```
public static void clear() {
    for(Device device: deviceList) {
        device.disconnect();
    }

    deviceList.clear();
    deviceMap.clear();
    adapter.notifyDataSetChanged();
}
```

When the list with devices must be cleared each device is disconnected and the lists are cleared. The adapter is notified of the changes to the list.

Bluetooth Devices List Adapter

The list adapter displays a list containing three kinds of devices:

- Unknown devices where the service type is not yet determined.
- Sensor devices with a Human Interface Device service.
- Servo devices with a Automation IO service.

Each device type has its own view and view binder. In the `onCreateViewHolder` and `onBindViewHolder` functions the correct view and binder are determined.

```
public int getItemViewType(int position) {
    Device device = BluetoothDevicesProvider.deviceList.get(position);

    if(device.getDeviceType() == SENSOR){
        return SENSOR;
    } else if (device.getDeviceType() == SERVO) {
        return SERVO;
    } else {
        return UNKNOWN;
    }
}
```

The view type is determined for the current list position by getting the device from the devices provider and matching the type.

SERVO VIEW

The servo view contains a seekbar to change the angle of the servo and a dropdown to select the source of the angle.

```
holder.sbAngle.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
        if (!device.usesSource()) {
            device.writeAngle(seekBar.getProgress(), false);
            holder.tvAngle.setText(String.valueOf(seekBar.getProgress()));
        }
    }
```

The `OnSeekBarChangeListener` writes a new angle to the device when the position of the seekbar is changed.

```
List<Device> sources = new ArrayList<>();
sources.add(device);
for (Device sourceDevice: BluetoothDevicesProvider.deviceList) {
    if (!sourceDevice.equals(device) && sourceDevice.getDeviceType() == SENSOR) {
        sources.add(sourceDevice);
    }
}
ArrayAdapter<Device> spinnerAdapter = new ArrayAdapter(context, R.layout.source_list_item, R.id.tvSourceName, sources.toArray());
holder.spSource.setAdapter(spinnerAdapter);
if (device.usesSource()) {
    holder.spSource.setSelection(sources.indexOf(device.getSource()));
}

holder.spSource.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
        Device source = sources.get(i);
        if (source.equals(device)) {
            device.unsetSource();
        } else {
            device.setSource(source);
        }
    }
```

A list of available sources is created by getting all devices from the device provider and filtering them on device type. For each available source a dropdown item is created. The currently used source is pre-selected in the dropdown. When a item in the dropdown is selected the `OnItemSelectedListener` handles the event and sets the new source for the servo device.

2.6 Tests

Three tests where preformed to validate the system.

2.6.1 Potentiometer rotation test

For this test a Sensor peripheral and a Android phone running the application where used. The goal of the test was to determine if the angle of the potentiomenter on the peripheral was correctly displayed on the dashboard. Also tested was if the angle of the potentiometer was changed the dashboard was updated with the new angle in a timely manner.

The test proved to be successfull. The angle displayed on the dashboard matched the angle of the potentiometer and changes in the angle of the potentiometer where displayed correctly on the dashboard within a second.

2.6.2 Servo rotation test

This test was preformed using a Servo peripheral and a Android phone running the application. The goal of the test was to determine if the angle of the servo matched the setting on the dashboard. Also tested was if the angle of the servo changed according to the input on the dashboard.

The test was successfull. The angle displayed matched the angle of the servo and when the angle was changed on the dashboard the servo quickly rotated to the new angle.

2.6.3 Combined rotation test

During this test both a Sensor and Servo peripheral where used, as well as a Android phone running the application. The goal was to test if the angle of the servo would match the angle of the potentiometer when the sensor was selected as source for the servo on the dashboard.

It was possible to link the angle of the servo to the angle of the potentiometer during the tests. The tests where therefore successfull.

2.7 Demonstrations

The following video's demonstrate the working system in several ways:

- **Full app demonstration** app-demo.mp4 displaying connecting sensors and servo's, displaying changes in angle of the sensor, changing the angle of the servo and linking the angle of the sensor to the servo.
- **Fipy with phone demonstration** fipy-with-phone-demo.mp4 displaying controlling the angle of the servo using the app.
- **NRF52 with phone demonstration** NRF52-with-phone-demo.mp4 displaying changing the angle of the potentiometer and displaying the changes in the app.

3. Low level radio

3.1 Theory

The theory assignments for the Wireless Communication course consist of general questions about the NRF24L01+ radio. These general questions are answered in this chapter. Furthermore a table describing the name and function of each NRF24L01+ pin has been added, with their connection to the Arduino uno compatible pins.

3.1.1 General questions

1. **What is the maximum output power from the NRF24 module?**
   - 0 dBm 2
   - **How does that compare to the output power of a 3G/4G mobile phone?**
     - Between 0 and 30 dBm 1
2. **What is a frequency band?**
   - The operating frequencys that a device works in.
3. **Which frequency bands can be used by this radio? (check the datasheet)**
   - From 2.400GHz to 2.525GHz 2
   - **Which bands are used by WiFi and Bluetooth?**
     - Wifi: 2401 MHz to 2495 MHz 3
     - Bluethooth: 2400 to 2483.5 MHz 4
   - **Where does the NRF24 overlap with these bands?**
     - From stat to finnish only the 5G wifi is not overlapped
4. **What is the wavelength in meters for the used NRF24 frequency band?**
   - 300.000.000 / 2.400.000.000 = 0.125m
   - 300.000.000 / 2.483.500.000 = 0.120m
5. **Which data rates does the radio support?**
6. 250kbps, 1Mbps and 2Mbps 2
7. **How many channels does the radio support?**
   - 126 channels so 8 bit 2
   - **What is the relationship between channels and frequencies?**
   - F0= 2400 + RF_CH MHz you add the channel to 2400 and you get the frequency 2
8. **Which modulation technique does the radio use? Explain how this modulation technique works in your own words.**
   - GFSK: Gaussian frequency shift keying is a modulation method for digital communication. it uses a Gaussian filter before the signal is send. FSK is the input in this system this is modulation where the frycuentcy of a signal is changed depending if the bit is a 1 or 0
9. **Power consumption in the idle mode states (power-down, standby-I and standby-II). Explain in your own words what the different states do.**
   - power-down: 900nA, this mode has the radio turned off so only the standby of the chip is used
   - standby-I: 26µA, this mode reduces power by running the radio in a half turned off way
   - standby-II: 320µA, the same mode as standby-i but with extra buffers so the startup time is lower

10. **Power consumption in the transmit state at the four different output levels.**
    - 0dBm output power: 11.3 mA
    - -6dBm output power: 9.0 mA
    - -12dBm output power: 7.5 mA
    - -18dBm output power: 7.0 mA
11. **Power consumption in the receive mode state for the three different air data rates.**
    - Supply current 2Mbps: 13.5 mA
    - Supply current 1Mbps: 13.1 mA
    - Supply current 250kbps: 12.6mA
12. **What is Enhanced ShockBurst and what are the most important functions? Describe this in your own words.**
    - Enhanced ShockBurst (ESB) is a protocol supporting two way data transfer it takes care of importend features such as retransmission, buffering, packet acknowledgment.
13. **On what layer in the ISO/OSI model does this operate?**
    - layer 2
14. **Look at the format of the radio packet (section 7.3 on page 28 of the datasheet) and describe the function of the following fields:**
    - Address: addres of the reciever this is for what radio the packet is ment
    - PID (Packet Identification): this part contains all the meta data of the packet like the legth of the payload, packet information (is it a retransmit or not)
    - No acknowledgement flag should it send a ack back to the sender this so it knows you got the message
    - CRC: this is a checksum bytes these are set by means of a calculation
15. **Enhanced ShockBurst is able to automatically validate a packet. Describe how this process works in your own words. Give an example of a packet that is broken and describe how the radio is able to detect this.**
    - before transmission the CRC bytes are calcculated by the radio this is done over the hole packet exept the crc bytes this is than recalculated at the receving end and checked, if the bytes dont match you can detect that the transmission was not succesfull.
16. **What conditions define a good address (see section 7.3.2 of thedatasheet)? Give an example of a good 5-byte address that adheres to these conditions. Write the address in hexadecimal and binary format.**
    - Adresses with more than one level shift and no continuation of the preamble can be considered good addresses.
    - 0x313D371F2F, 0011 0001 0011 1101 0011 0111 0001 1111 0010 1111
17. **Give an example of a bad 5-byte address and explain why this is not a good address. Write the address in hexadecimal and binary.**
    - 0x000FFFFFFF, 0000 0000 0000 1111 1111 1111 1111 1111 1111 1111
    - Only one level shift
18. **Find on the internet what the cost / price (high and low) is of this module.**
    - Zeer goedkoop, rond de €0,56 ex verzenden uit china 5 en €2,50 ex verzenden uit nederland 6.

19. **Findthree alternative radiomodules or shields and for each module**
    a. **RFM69HW** 7
        - What technique is used for communication: FSK, GFSK, MSK, GMSK, OOK modulations, make your own protocol
        - What is the cost of the shield / module: €4,75
        - What software is available for this module / shield: Arduino library

    b. **HC-05** 8
        - What technique is used for communication: Bluetooth
        - What is the cost of the shield / module: €6,00
        - What software is available for this module / shield: Arduino library

    c. **433MKIT** 9
        - What technique is used for communication: AM, make your own protocol
        - What is the cost of the shield / module: €2,50
        - What software is available for this module / shield: Arduino library
20. **Add a table to you report that describes the name and the function of every NRF24 pin. Include to which pin of the Arduino Uno header the NRF24 pin is connected.**
    - See nRF24L01+ pinout table
21. **The Hello World! example has the TX and RX address configured to a default value. What do you think would happen If a classroom full of students would start using the same address?**
    - Huilen. Due to interference most messages will not be transmitted properly.
22. **Have a look at the method 'setRegister'. This method is used quite often in the library. What is the function of this method and what is happening in the radio when this function is called?**
    - The radio is configured and controlled using registers. Every register has a address and controls a diferent setting. This method sets the value of a register and therefore configures the radio.
23. **At the top of the .cppfile, have a look at the list of '_NRF24L01P_REG...' definitions. What are these values and where can you find them in the NRF24 datasheet?**
    - These are the addresses of the different registers. These can be found under 9.1 in the datasheet 2.
24. **Find in the datasheet what 'ART' stands for and explain this in your own words.**
    - Auto Re-Transmission. It automagically retransmits a package if the reciever did not confirm that it revieved id (did not send a ACK).

3.1.2 nRF24L01+ pinout table

#	Name	Function	Uno pin
1	GND	Ground	Ground
2	VCC	Voltage source	3.3 volt
3	CE	Chip Enable	9
4	CSN	Chip Select Not	8
5	SCK	Serial Clock (SPI)	13
6	MOSI	Master Out, Slave In (SPI)	11
7	MISO	Master In, Slave Out (SPI)	12
8	IRQ	Interrupt	7

3.2 Led App

In this chapter we walk about the design and implementation of the led app.

3.2.1 Pinout and functions of NRF24L01

PIN number	Arduino connection	function
1	GND	GND: this provides the ground to the nrf this is needed so the reference voltage for the control signals is known
2	Ioref	VCC: This is providing the power to the chip for it to turn on this needs to be 3.3V
3	D9	CE: Chip enable is the pin that manages whether the chip should listen to the SPI commands send
4	D10	CSN: is the chip select not pin this pin is used to enable the spi bus
5	D13	SCK: Serial clock provides the clock pulse for the spi communication to work
6	D11	MOSI: Master out slave in pin this is the input pin for data from the microcontroller to the NRF
7	D12	MISO: Master in slave out pin this is the output pin for data drom the nrf to the microcontroller
8	D7	IRQ: This is the inrerrupt pin an wil be active low when triggerd

3.2.2 Packet definition

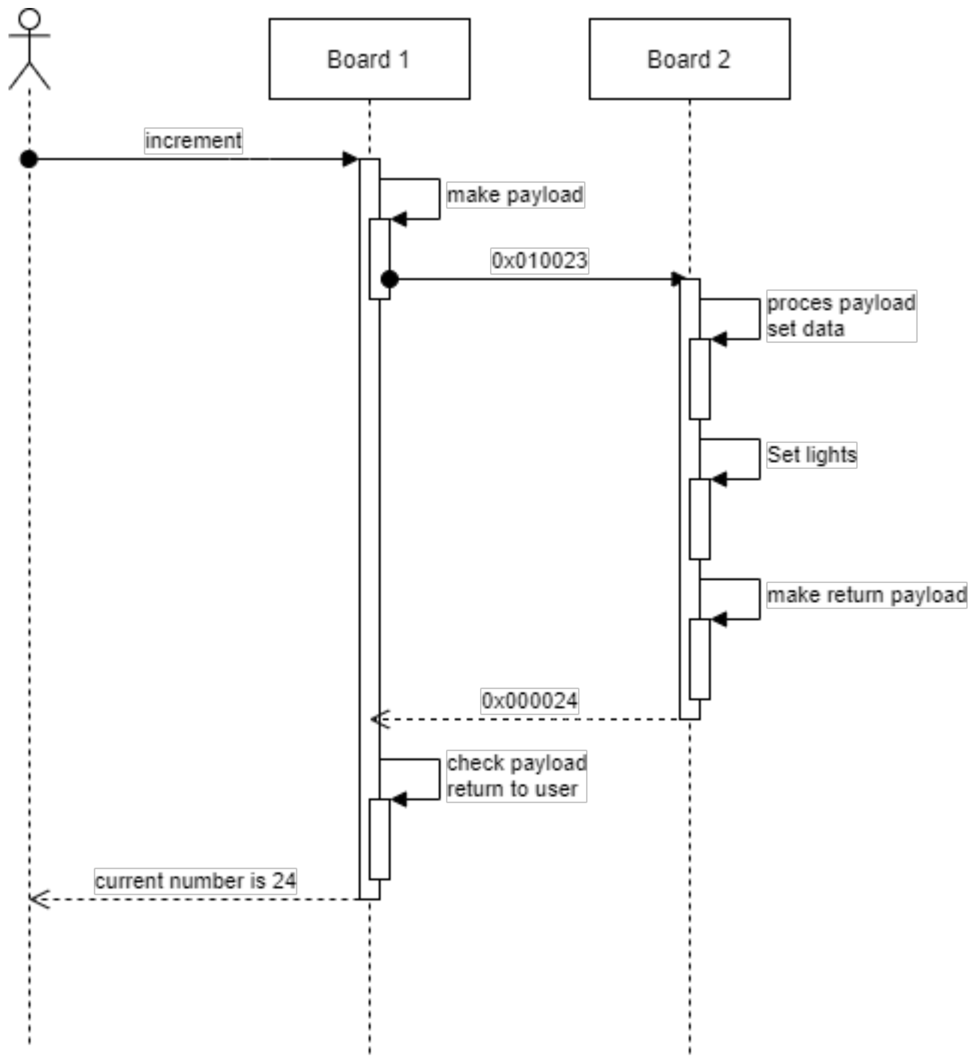
The data that needs to be transported between the two radio's is the following: should i go to the next pin, should i go the previous pin, and what pin am I on? This data can be fit into 3 variables so 3 bytes. The packet would look something like this:

	Next	previus	current pin
Packet data	0x00	0x01	0x01
Values	0	1	1

So, this packet tells you need to go back one pin and the current pin is 1. So, a packet containing 0x010003 would be go to next pin current pin is 3. This way the minimal amount of data is send over the air.

3.2.3 Communication between the boards

The boards would send a package described above to the other board and increment or decretent the pointer value given in the packet this packet would sync the boards if there was a discrepancy between them. This is shown in the picture below.



3.2.4 Implementation

During the implementation of this project, we stumbled on a few roadblocks. The code that was provided in the assignment was out of date and would not compile a few days where spend on trying to get this code to work and we did not get it running. We have tried platformIO to compile it the browser-based IDE and their own IDE but the sample code did not work. Then we tried working with the MCP23508 and while we could get LEDs to light up. After a few nights work we did not get the interrupts working not on the Nucleo or on a Arduino UNO which was used for sanity checking our way of thinking. We even after consulting the documentation of the MCP we did not get it to work. We beleave some adress asighment did not go right or wat not interperted right. The configuration was:

Register	Address	Value	binary
IODIR	0x00	0xc0	1100 0000
IPOL	0x01	0x00	0000 0000
GPINTEN	0x02	0xc0	1100 0000
DEFVAL	0x03	0xc0	1100 0000
INTCON	0x04	0xc0	1100 0000
IOCON	0x05	0x0A	0000 1010
GPPU	0x06	0xC0	1100 0000
INTF	0x07	0x00	0000 0000
INTCAP	0x08	0x00	0000 0000
GPIO	0x09	0x00	0000 0000
OLAT	0x0A	0x00	0000 0000

We also ran into the issue with the interrupts during the microcontrollers course. Where we did not get the interrupts to work.

During the implementation for the radio communication, we did not get any communication with the radios via the STM. We tried example code, tried different radios we had some new ones laying around. All with no luck we know the radios were in working condition since we tested them with an Arduino before and the communication worked there. We decided to keep working on this while we started work on the other assignment. This so if we get it working on the other assignment, we could use this knowledge to fix the problems we were facing here.

In the end we did not succseed in getting te communication working between the two devices.

3.3 Packet Error Rate App

This chapter describes the research preformed for the packet error rate app assignment.

3.3.1 Introduction

The goal of the Packet Error Rate app is to measure the packet error rate when sending a number of messages while varying the output power for sending, the frequency channels, data rates and Auto Retransmit settings of the NRF24L01+. Firstly the problem and research question is defined. Secondly the methodology used to answer the research question is described. Thirdly the results of the research is presented. Finally a conclusion and answer to the research question is given.

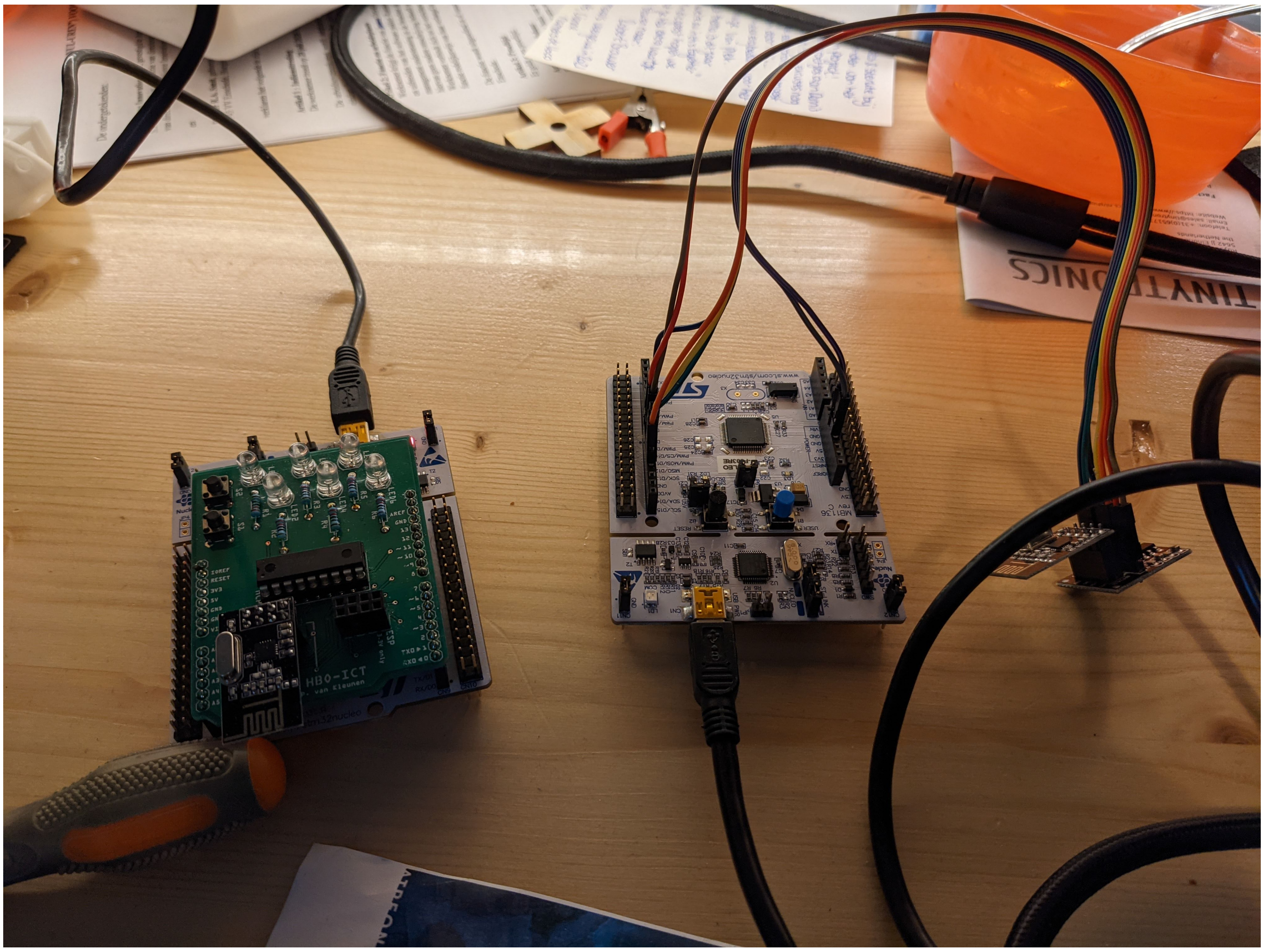
3.3.2 Problem definition

The parameters for the output power, the frequency channels, data rates and Auto Retransmit settings influence the packet error rate of the app. To prevent package loss the optimal setting of the parameters should be determined. Therefore our research question is:

1. What combination of output power, the frequency channels, data rates and Auto Retransmit settings of the NRF24L01+ gives the lowest package loss.

3.3.3 Methodology

Two Nucleos will be exquipped with a NRF24L01+, one using the IoT shield and one using a breakout board. Both Nucleos will be connected to a computer using USB cables. A example of this setup is shown in the image below.



One of the Nucleos will act as the transmitter, the other as the reciever. They will listen to serial commands recieved via the USB cable to synchonosly cycle through different combinations of settings. The transmitter will send a fixed number of packets. The reciever will measure the total amount of packets recieved. The transmitter will record the time it took to send all packets. A csv file will be generated containing the value of each setting, the amount of packets sent and recieved and the total transmit time.

3.3.4 Results

Unfortunately we did not succeed in getting the nrf24l01+ running with a Nucleo. Please see the implementation of the LED app chapter for a more detailed description of the problems we faced and our attempts to overcome these.

Ultimately we where unable to collect any results. In the folder /nrf24-packet-error-rate a preliminary setup for the app can be found. This is a work-in-progress application and not near the final solution.

3.3.5 Conclusion

Due to the lack of results a conclusion can not be given.

3.4 Throughput test app

This chapter describes the research preformed for the Throughput test app assignment.

3.4.1 Introduction

For the NRF24L01+ a data rate can be configured. This is the amount of data that will be transmitted in a period of time. In practice this data rate is not reached. This research tries to determine the actual data rate of the NRF24L01+. Firstly the problem and research question is defined. Secondly the methodology used to answer the research question is described. Thirdly the results of the research is presented. Finally a conclusion and answer to the research question is given.

3.4.2 Problem definition

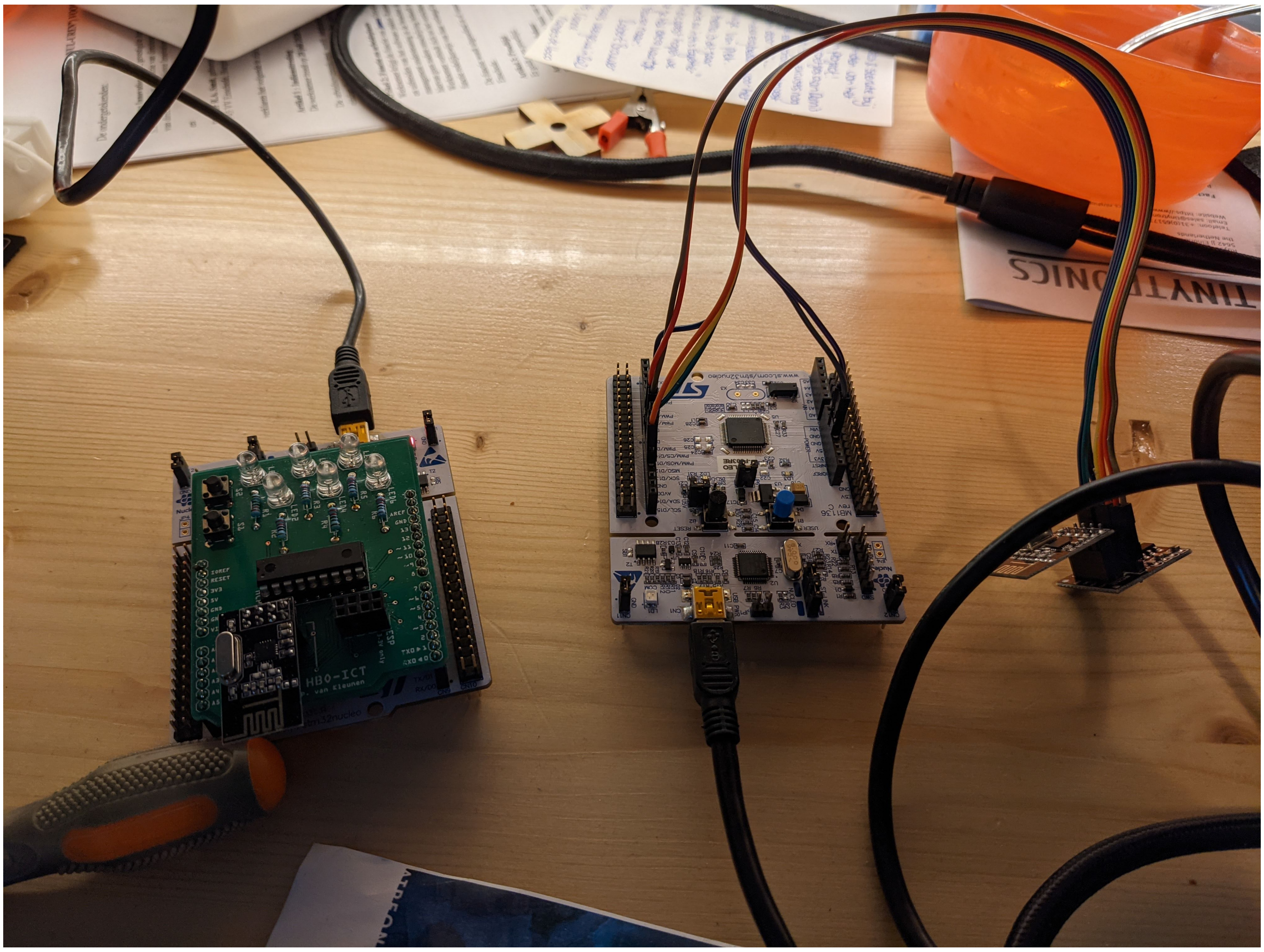
The actual data rate of the NRF24L01+ does not precicely match the data rate set in the settings. It is unknow what the actual datarate is and therfore unknown how long it takes to transmit a large amount of data. Therefore the following research questions are determined:

1. How long does it take to transmit 10.000 32-bit numbers (320.000 bits in total) from a transmitter to a reciever for a set data rate.
2. What is the actual data rate when transmitting 320.000 bits and how much does it differ from the set data rate.
3. What causes the difference between the actual and set data rate.

During the tests all numbers must be recieved by the reciever.

3.4.3 Methodology

Two Nucleos will be exquipped with a NRF24L01+, one using the IoT shield and one using a breakout board. Both Nucleos will be connected to a computer using USB cables. A example of this setup is shown in the image below.



One of the Nucleos will act as the transmitter, the other as the reciever. The transmitter will transmit the 10.000 32-bit numbers to the reciever. The reciever will acknowledge if the packet has been recieved using auto acknowledge. The transmitter will record the total time it takes to transmit all numbers as well as the total amount of retries. This will be sent to a serial monitor via the USB cable.

3.4.4 Results

Due to problems with using the NRF24L01+ on a Nucleo we where unable to complete this assignment and get any results. Please see the implementation of the LED app chapter for a more detailed description of the problems we faced and our attempts to overcome these.

3.4.5 Conclusion

Due to a lack of results we can not give a conclusion.

How long does it take to transmit 10.000 32-bit numbers (320.000 bits in total) from a transmitter to a reciever for a set data rate

We where unable to answer this question.

What is the actual data rate when transmitting 320.000 bits and how much does it differ from the set data rate

We where unable to answer this question.

What causes the difference between the actual and set data rate

Although we do not have results one of the reasons that the set data rate is not reached is the auto acknowledge communication. This adds a overhead to the communication and limits the actual data rate.

https://github.com/RubenSmit/wireless-communication