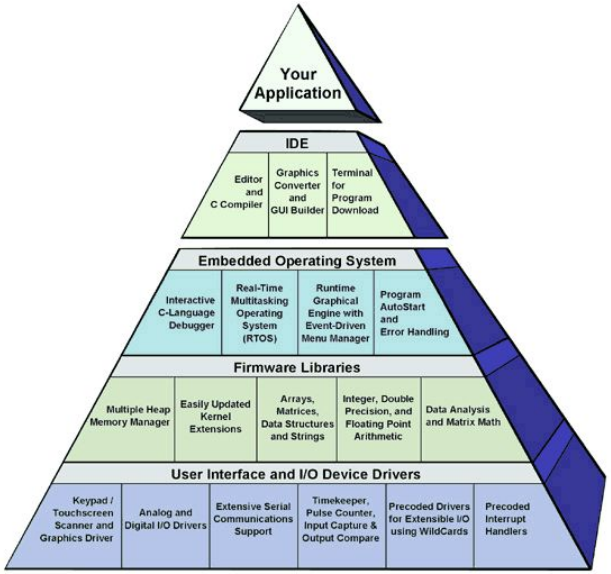
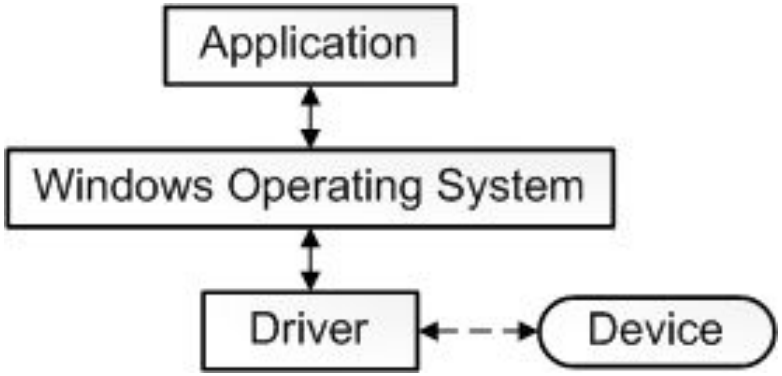
Device Drivers for Embedded Internet of Things (IoTs) Platforms





DCE
DEPT. OF COMPUTER ENGINEERING

Content

- Device Driver Overview
- Serial Communication Device Driver
  - Universal Asynchronous Receiver/Transmitter (UART)
  - Serial Peripheral Interface (SPI)
  - Inter-Integrated Circuit (I2C)
- Summary

What is Device Driver?

- **Device driver** is a **particular software application** that is designed to enable interaction with hardware devices.
- Device drivers are **operating system-specific** and **hardware-dependent**.

- For embedded platforms running OS, device drivers are considered as part of firmware.

- Firmware is the software which executes on the embedded system's CPU (written in assembler and C), was complied and the **binary burned onto an EPROM**.

Why Do We Need Device Driver?

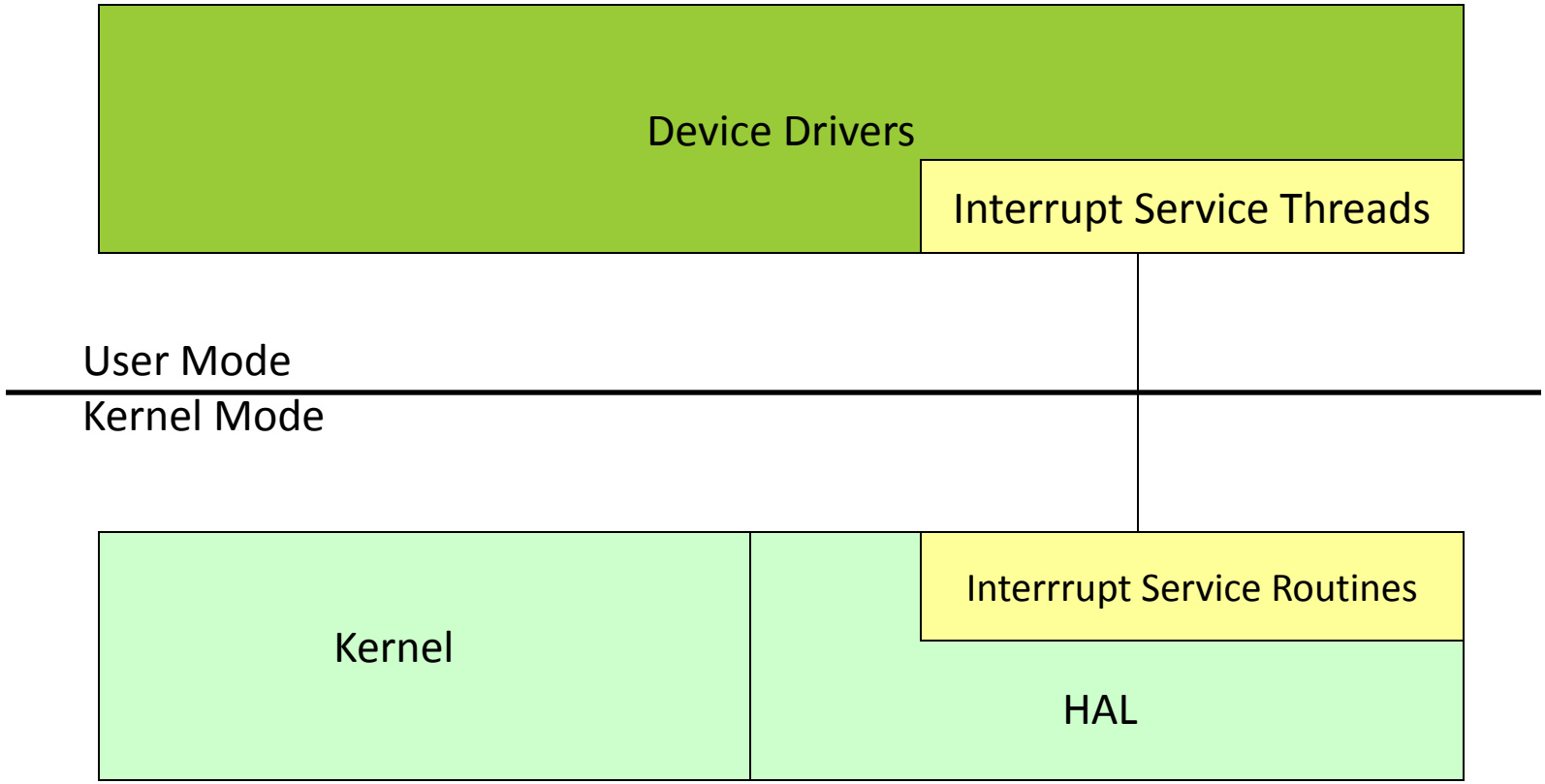
- Provide uniform APIs to access hardware.

- Custom platforms
  - Contain many peripheral devices and kernel supported CPU
    - Get kernel to boot on the board
    - Device drivers to allow applications to access peripheral devices

Device Driver Models

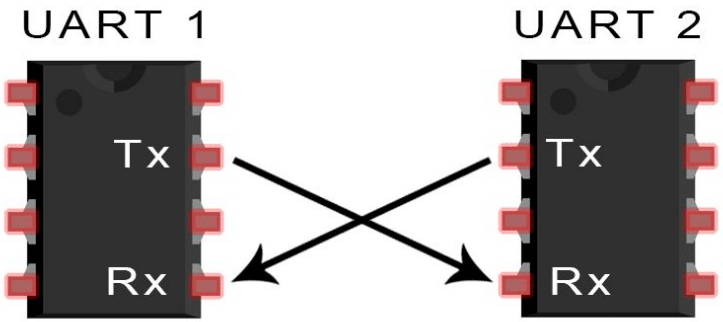
- Device drivers, over the years, have become very complex
  - Drivers are separated into classes
    - Serial, network, audio, video, touch panel, etc.
  - Layer approach is used
    - To support a new device, a layer is modified instead of rewriting the entire driver
    - Processing functions required for a given class often do not require modification

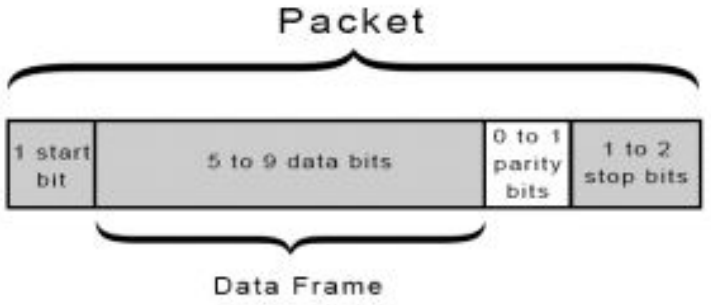
Device Driver Architecture

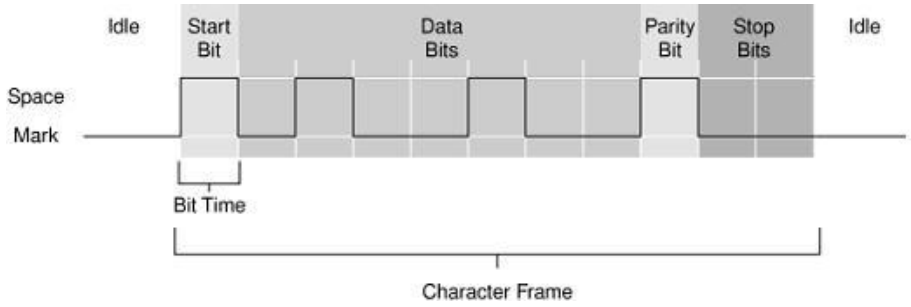


Serial Communication using UART

- The most popular connection in hardware devices



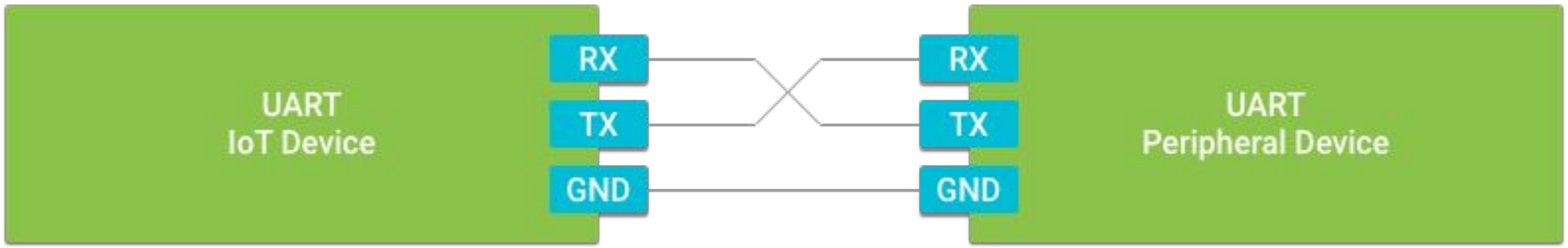




What is the difference between UART, RS232 and RS485?

UART Device Driver on Android Things

- UART stands for Universal Asynchronous Receiver Transmitter
- It is ***universal*** because both the data transfer speed and data byte format are configurable.
- It is ***asynchronous*** in that there are no clock signals present to synchronize the data transfer between the two devices
- UART data transfer is ***full-duplex***, meaning data can be sent and received at the same time



UART Implementation

- https://github.com/androidthings/sample-uartloopback
- Adding the required permission

```
<uses-permission
android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

- Managing the connection

```
PeripheralManager manager =
PeripheralManager.getInstance();
List<String> deviceList =
manager.getUartDeviceList();
if (deviceList.isEmpty()) {
    Log.i(TAG, "No UART port available on this
device.");
} else {
    Log.i(TAG, "List of available devices: " +
deviceList);
}
```

UART Implementation

- Open an UART Port

```
private UartDevice mDevice;
            PeripheralManager manager =
PeripheralManager.getInstance();
            mDevice =
manager.openUartDevice(UART_DEVICE_NAME);
        } catch (IOException e) {
            Log.w(TAG, "Unable to access UART
device", e);
        }
```

- Close an UART Port

```
@Override
    protected void onDestroy() {
        super.onDestroy();

        if (mDevice != null) {
            try {
                mDevice.close();
                mDevice = null;
            } catch (IOException e) {
                Log.w(TAG, "Unable to
close UART device", e);
            }
        }
    }
```

UART Implementation

- Configure the frame format

```
public void configureUartFrame(UartDevice uart)
throws IOException {
    // Configure the UART port
    uart.setBaudrate(115200);
    uart.setDataSize(8);
    uart.setParity(UartDevice.PARITY_NONE);
    uart.setStopBits(1);
}
```

- **Baudrate**: Communication speed in baud. In computer, it is equivalent to bits per second (bps)

UART Implementation

- Send a message

```
public void writeUartData(UartDevice uart)
throws IOException {
    byte[] buffer = {...};
    int count = uart.write(buffer,
buffer.length);
    Log.d(TAG, "Wrote " + count + " bytes to
peripheral");
}
```

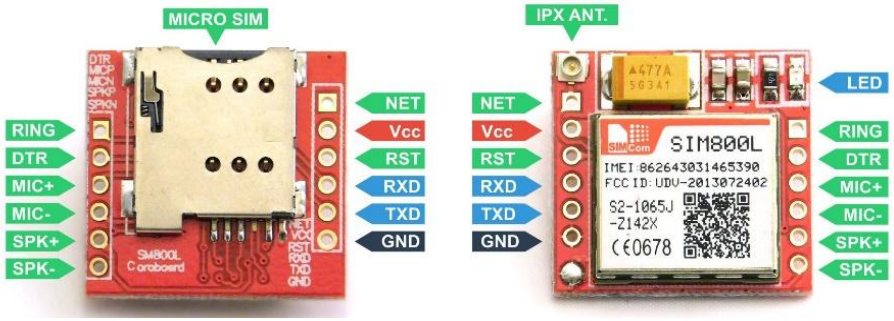
- Receive a message

```
private UartDeviceCallback mUartCallback = new
UartDeviceCallback() {
        @Override
        public boolean
onUartDeviceDataAvailable(UartDevice uart) {
            // Read available data from the UART
device
            try {
                readUartBuffer(uart);
            } catch (IOException e) {
                Log.w(TAG, "Unable to access UART
device", e);
            }
            return true;
        }

        @Override
        public void onUartDeviceError(UartDevice uart,
int error) {
            Log.w(TAG, uart + ": Error event " +
error);
```

```
public void readUartBuffer(UartDevice uart)
throws IOException {
    // Maximum amount of data to read at one
time
    final int maxCount = ...;
    byte[] buffer = new byte[maxCount];

    int count;
    while ((count = uart.read(buffer,
buffer.length)) > 0) {
        Log.d(TAG, "Read " + count + " bytes
from peripheral");
    }
}
```

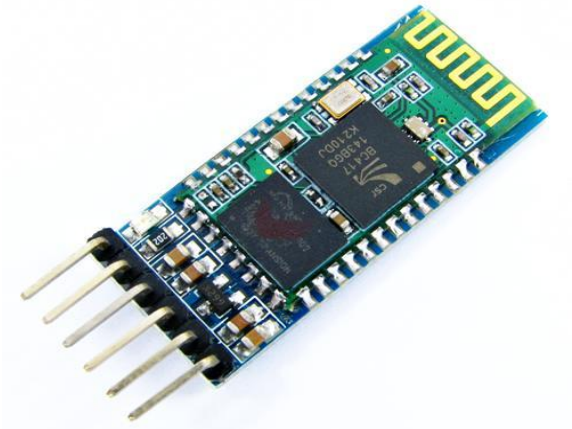
Wireless Modules using UART Interface

- **AT command** (AT stands for attention)



GSM/GPRS

Wifi
(ESP8266)



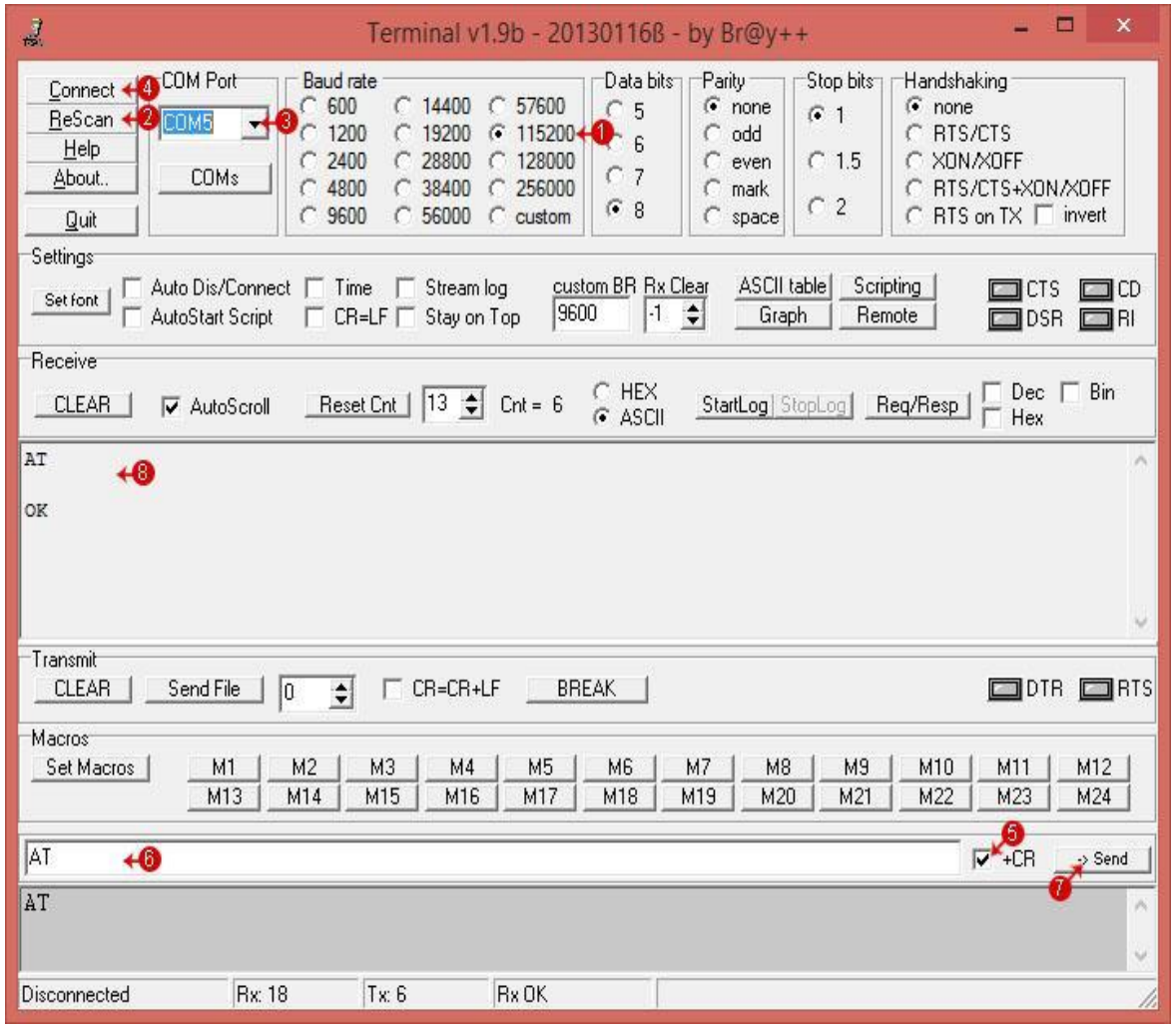
Bluetooth

XBee

LORA
(SX1728)

Hyper Terminal

https://www.dropbox.com/s/7xuwege5pjv6fis/Terminal.exe?dl=0



- 1. Set Baudrate
- 3. Select COM Port
- 4. Click Connect
- 5. Send a testing command (AT)

Example AT Commands for GSM/GPRS

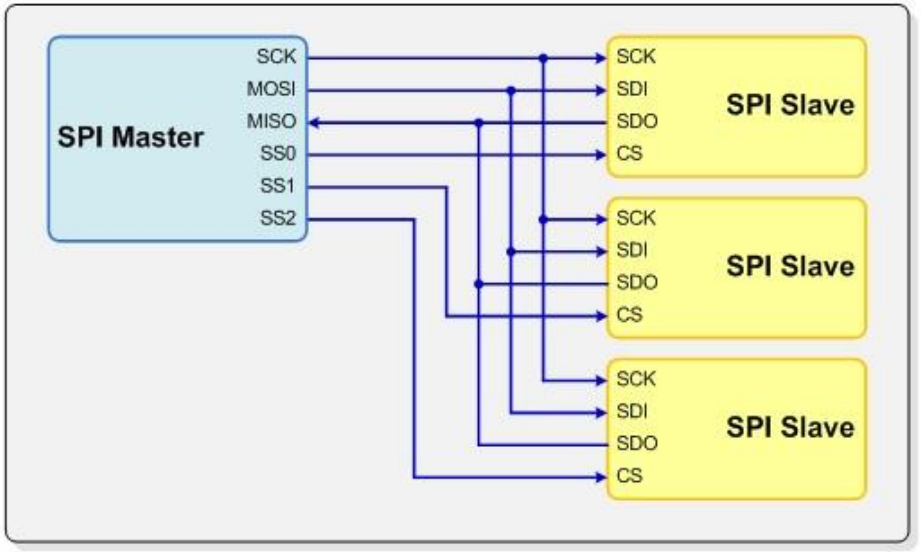
- Send a SMS message to a phone:
  - AT+CMGF=1
  - AT+CSCS="GSM"
  - AT+CMGS="84906362340"
  - Send text message
  - 0x1A
- Send a GET request:
  - AT+SAPBR=1,1
  - AT+HTTPINIT
  - AT+HTTPPARA="CID",1
  - AT+HTTPPARA="URL","http://www.iforce2d.net/test.php"
  - AT+HTTPACTION=0
  - AT+HTTPREAD
  - AT+HTTPTERM
- Note: Every AT command ends with carry return
  - \r\n
  - 0x0d 0x0a

Serial Communication using SPI

- **S**erial **P**eripheral **I**nterface (SPI) is an interface bus commonly used to send data between **microcontrollers** and small peripherals such as shift **registers, sensors, and SD cards**

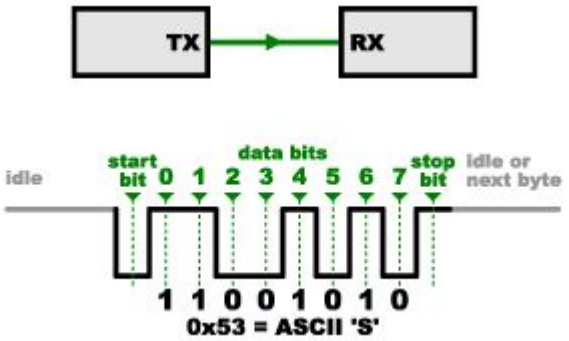
- Separate clock (SCK), data lines (MISO, MOSI) and chip select (CS) are used.

- Synchronous protocol

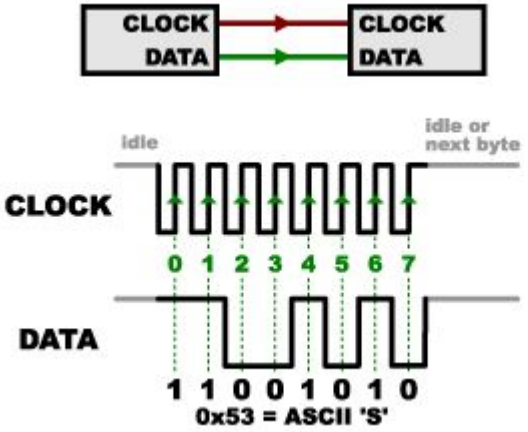


Asynchronous vs Synchronous Protocol

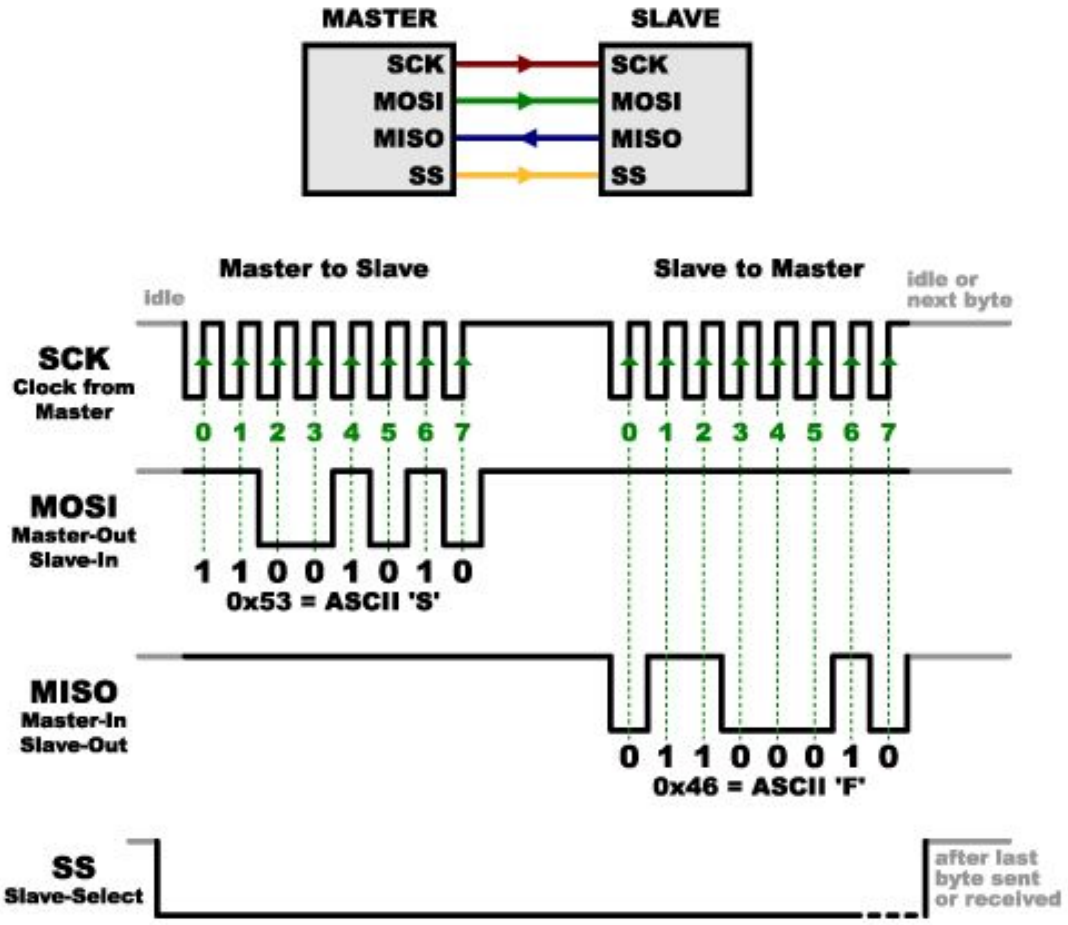
- Asynchronous (UART):
  - There is no CLOCK
  - Clock drift issue ☐ low speed



- Synchronous (SPI):
  - There is a CLOCK line
  - High speed



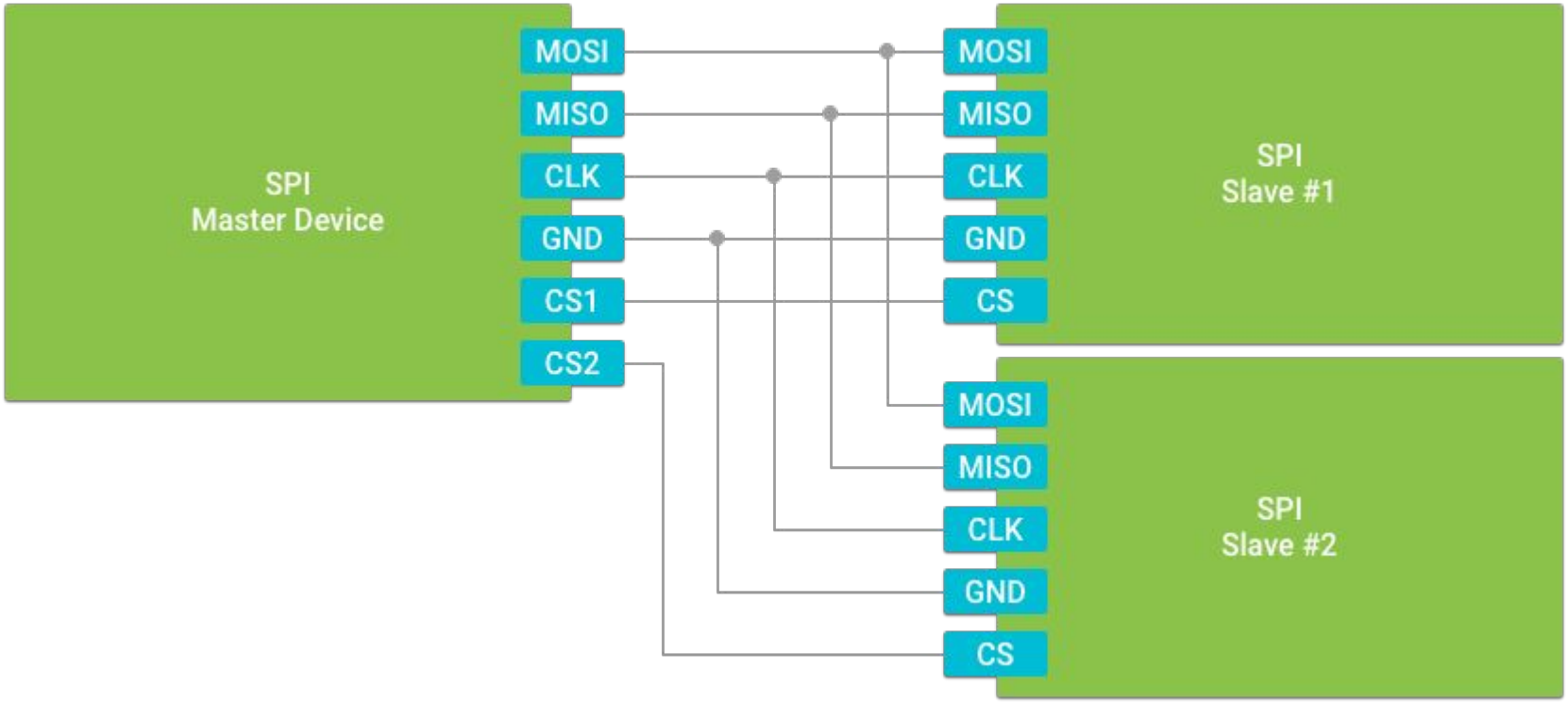
SPI Protocol



- **Clock is generated by the master**

SPI Device Driver on Android Things

- Serial Peripheral Interface (SPI) devices are typically found where fast data transfer rates are required (e.g. external non-volatile memory and graphical displays)
- Clock speeds are typically in the 16MHz to 25MHz range.



Implement SPI

- Adding the required permission

```
<uses-permission
android:name="com.google.android.things.permission.USE_PERIPHERAL_IO" />
```

- Managing the device connection

```
PeripheralManager manager =
PeripheralManager.getInstance();
List<String> deviceList =
manager.getSpiBusList();
if (deviceList.isEmpty()) {
    Log.i(TAG, "No SPI bus available on this
device.");
} else {
    Log.i(TAG, "List of available devices: " +
deviceList);
}
```

Implementation SPI

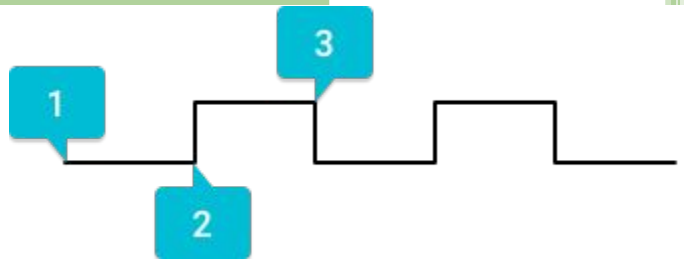
- Open SPI Port

```
try {
            PeripheralManager manager =
PeripheralManager.getInstance();
            mDevice =
manager.openSpiDevice(SPI_DEVICE_NAME);
        } catch (IOException e) {
            Log.w(TAG, "Unable to access SPI
device", e);
        }
```

- Close SPI Port

```
@Override
    protected void onDestroy() {
        super.onDestroy();

        if (mDevice != null) {
            try {
                mDevice.close();
                mDevice = null;
            } catch (IOException e) {
                Log.w(TAG, "Unable to
close SPI device", e);
            }
        }
    }
```

Implement SPI



- Configure SPI connection
  - MODE0: Clock signal idles low, data is transferred on the leading clock edge
  - MODE1: Clock signal idles low, data is transferred on the trailing clock edge
  - MODE2: Clock signal idles high, data is transferred on the leading clock edge
  - MODE3: Clock signal idles high, data is transferred on the trailing clock edge

```
public void configureSpiDevice(SpiDevice device) throws
IOException {
    // Low clock, leading edge transfer
    device.setMode(SpiDevice.MODE0);

    // 16MHz, 8BPW, MSB first
    device.setFrequency(16000000);
    device.setBitsPerWord(8);

device.setBitJustification(SpiDevice.BIT_JUSTIFICATION_M
SB_FIRST);
}
```

Implement SPI

- Transferring and Receiving Data

```
public void sendCommand(SpiDevice device, byte[] buffer)
throws IOException {
    // Shift data out to slave
    device.write(buffer, buffer.length);

    // Read the response
    byte[] response = new byte[32];
    device.read(response, response.length);
    ...
}
```

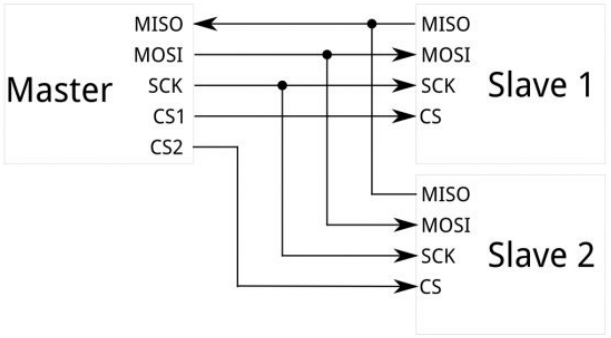
- If 2 bytes are sent and 2 bytes will be received, how many bytes for the buffer (the second parameter in sendCommand function)???

Serial Communication using I2C

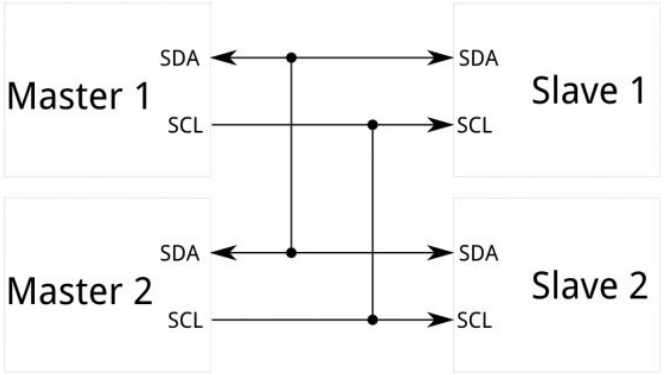
- Inter-integrated Circuit (I^2C) Protocol is a protocol intended to allow multiple "slave" digital integrated circuits ("chips") to communicate with one or more "master" chips.
- Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device.
- Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information.

I2C vs SPI

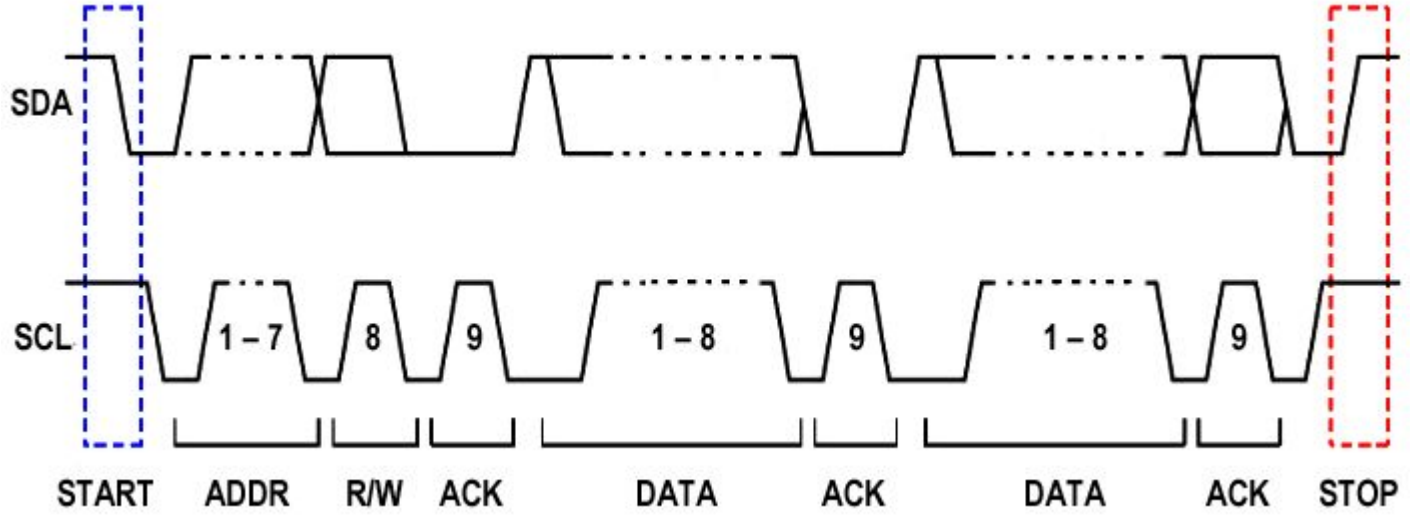
- Number of pins ☐ Difficult in tight PCB layout
- SPI only allows one master on the bus
- SPI is good for high data rate **full-duplex**



- Only two pins, like asynchronous serial, but can support up to 128 slave devices
- Support a multi-master system
- Data rates is at 100kHz or 400kHz



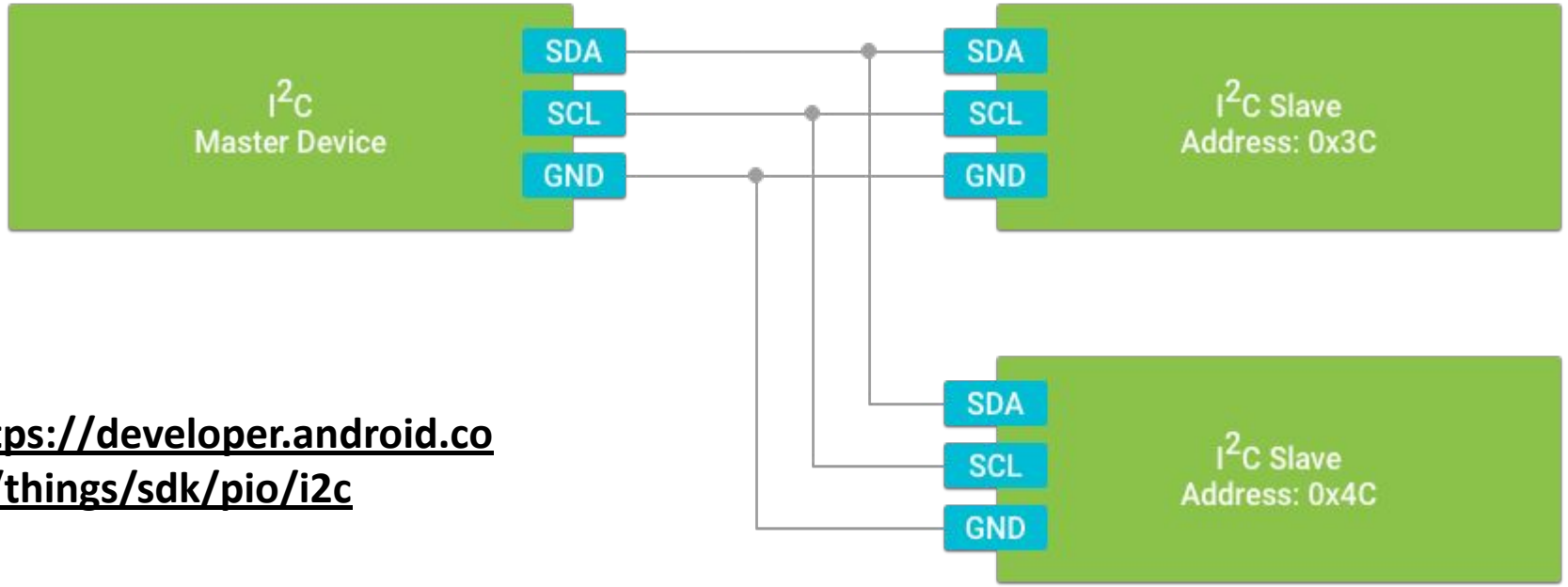
I2C Protocol



- Master sends START condition and controls the clock (SCL)
- Master sends a unique 7-bit slave address
- Master sends Read/Write bit: 0 write to slave, 1: read from slave
- Slave which address is matched send ACK bit
- Data (8bit) is transfered

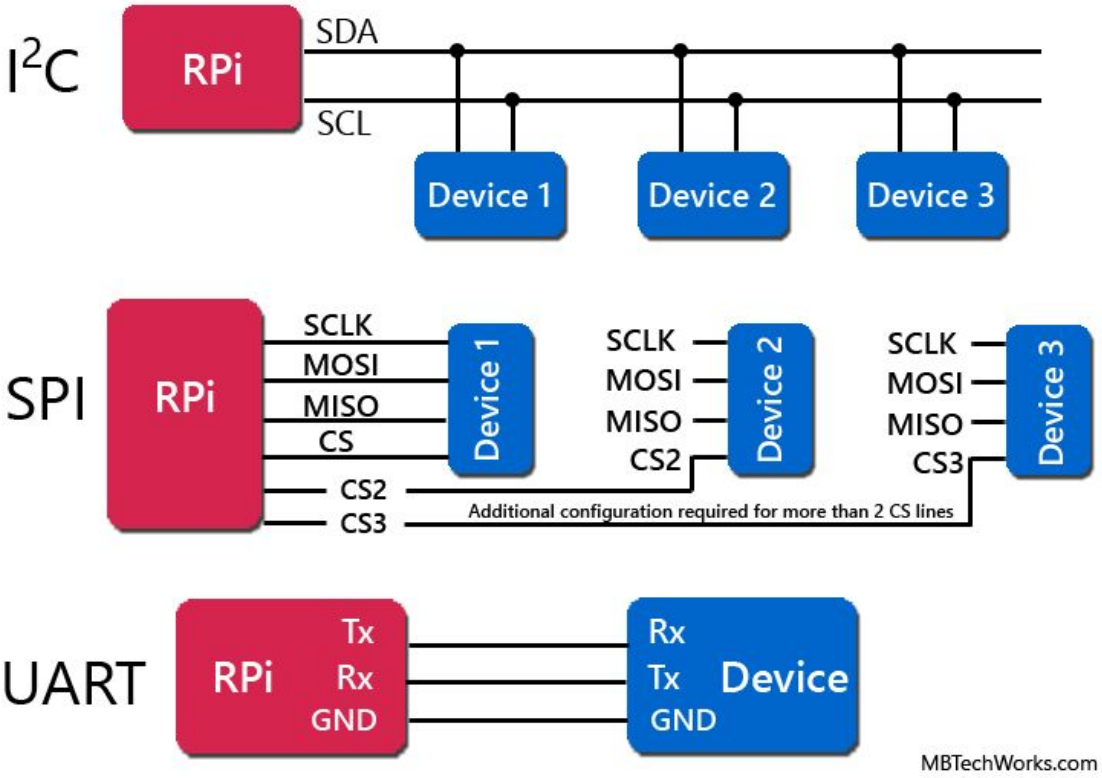
Implement I2C

- I2C is a ***synchronous*** serial interface. The device in control of triggering the clock signal is known as the ***master***. All other connected peripherals are known as ***slaves***. Each device is connected to the same set of data signals to form a ***bus***.



https://developer.android.com/things/sdk/pio/i2c

Summary



- UART = Universal Asynchronous Receiver / Transmitter
- SPI = Serial Peripheral Interface
- I^2C = Inter-Integrated Circuit

Serial Communications Methods

Name	Description	Function
I^2C	Inter-Integrated Circuit	Half duplex, serial data transmission used for short-distance between boards, modules and peripherals. Uses 2 pins.
SPI	Serial Peripheral Interface bus	Full-duplex, serial data transmission used for short-distance between devices. Uses 4 pins.
UART	Universal Asynchronous Receiver-Transmitt	Full-duplex, Asynchronous, serial data transmission

Conclusion

- **UART** - simple; not high speed; no clock needed; limited to one device connected to the Pi.
- **I2C** - faster than UART, but not as fast as SPI; easier to chain many devices; Pi drives the clock so no sync issues.
- **SPI** - fastest of the three; Pi drives the clock so no sync issues; practical limit to number of devices on the Pi.