ESERO-UK
UK CanSat Competition

The CanSat Competition

What is a CanSat?

A simulation of a real satellite, integrated within the volume and shape of a soft drink can.

The challenge

1. To fit all major subsystems found in a satellite, including power, sensors and communications, into the volume and shape of a soft drink can.
2. To provide a parachute to ensure the can survives the landing.
3. To carry out scientific experiments and transmit in-flight data to an Earth-based computer.

Who can take part?

The competition is open to school and college students over the age of 14

The CanSat Competition

Phase 1 - Call for proposals and team selection

Phase 2 - Teachers' introductory workshop

Phase 3 - CanSat construction and test activities

Phase 4 - Regional launch campaign

Phase 5 - National final launch campaign

Winners go on to the European final.

Benefits to schools

- a unique opportunity for students to have a first practical experience of a real space project
- learn by doing
- get acquainted with inquiry-based methodology typical of real-life scientific and technical professions
- acquire and/or reinforce fundamental Technology, Physics, and programming curricular concepts
- understand the importance of coordination and teamwork
- enhance communication skills
- increase knowledge of careers in the space industry




CanSat 2020/21

Increasing students' and teachers' knowledge of the UK space industry further:

- Use of spaceports for regional events – is this possible?
- Having STEM Ambassadors from Launch UK mentoring teams

Contact us

Tom Lyons
STEM Enrichment Coordinator
t.lyons@stem.org.uk

Rebecca Crawford-Richardson
ESERO-UK Project Officer
r.crawford-richardson@stem.org.uk

Website: www.stem.org.uk/esero/cansat

CanSat CPD Workshop 2021

Matthew Rowlings

Josh Duncan

James White

Tom Lyons

Schedule

0930	Introduction to the CanSat competition
1100	Lab 1: Soldering and GPIO control
1230	Lunch
1315	Lab 2: Sensing with I2C
1500	Coffee Break
1515	Lab 3: Radio comms using SPI
1615	Wrap up

What is CanSat?

CanSat 2019 - ESA events: https://flic.kr/s/aHsmFaLWQ4

What is CanSat?

- A simulation of a real space mission!
- ESA define two mission objectives:

  Primary mission: Measure atmospheric Temperature and Pressure during the descent of the CanSat

  Secondary Mission: A scientific or technological (or other!) mission of the student's choice

AWS CanSat – UK winners 2019
"Our secondary mission is to investigate and create a non-Newtonian fluid that absorbs the impact energy when the CanSat hits the ground"

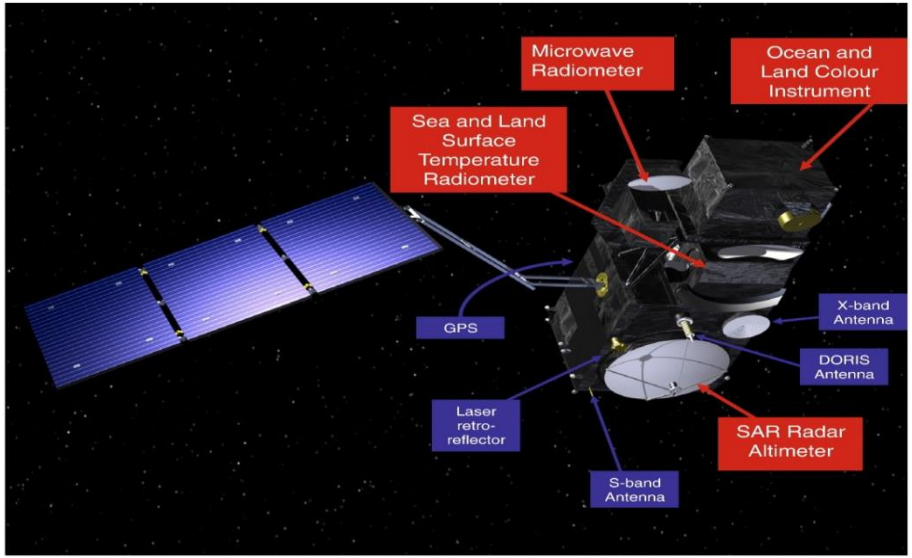
CanSat 2019 - ESA events: https://flic.kr/s/aHsmFaLWQ4

Project Beta – Netherlands winners 2019
Secondary mission was to release two smaller CanSats to allow a larger survey area for a platentary lander

CanSat 2019 - ESA events: https://flic.kr/s/aHsmFaLWQ4

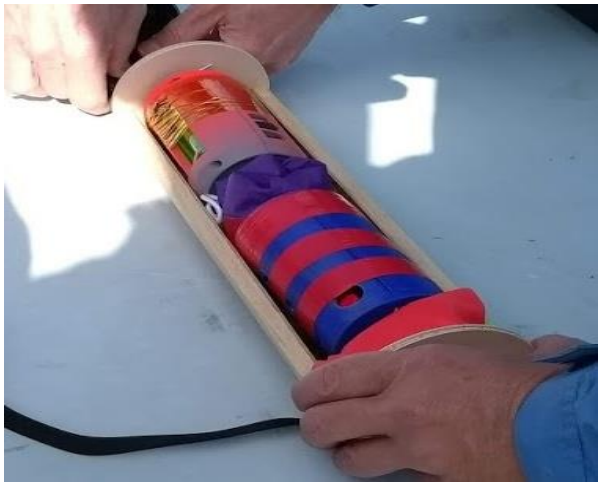
What is CanSat?

- Key subsystems to design:
  - Orbit
  - Scientific Instruments
  - Telemetry/Ground Stations
  - Data Handling
  - Power
  - Structure
  - Thermal
  - Propulsion
- Key Constraints:
  - Weight
  - Dimensions
  - Costs
  - Design/mission lifetime and environment



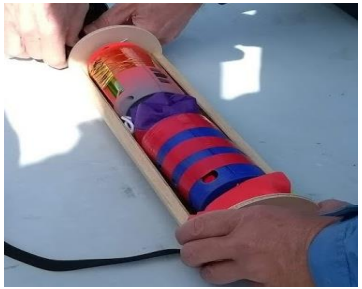
What is CanSat?

- A simulation of a real space mission!
- ESA define 16 mission constraints:
  - All the components of the CanSat must fit inside a standard soft drinks can (115 mm height and 66 mm diameter), with the exception of the parachute.
  - The mass of the CanSat must be between 300 grams and 350 grams.
  - The CanSat must have an easily accessible master power switch.
  - The CanSat's descent speed must not be lower than 5 m/s or higher than 12 m/s
  - The total budget of the final CanSat model should not exceed 500€ (£400)
  - The CanSat must be flight-ready upon arrival at the launch campaign
  - It must be possible for the systems to remain switched on for four continuous hours.



What is CanSat?

- A simulation of a real space mission – yes!
- Its difficult!



How do I win CanSat?

CanSat 2019 - ESA events: https://flic.kr/s/aHsmFaLWQ4

Judging Criteria

- Four main areas of merit:
  1. Scientific Value (35%)

     Clear scientific purpose, good understanding of science, the quality of the technical reports
  2. Technical Achievement (35%)

     Technical complexity, performance of primary and secondary missions
  3. Professional Competences (20%)

     Teamwork and management, adaptability to changes in their plan and lessons learnt, quality of presentations
  4. Outreach (10%)

     Reach of project within schools and with the wider community , both online and offline

- For the UK competition the judges decide this on (TBC depending on COVID situation):

  Content of the final report

  The final presentation

  Observing and talking to the teams on the launch campaign

Common Pitfalls

1. Last minute changes or poorly accessible batteries leading to breaking the CanSat in the last minute
2. Lack of interpretation of results in the final presentation, needs to be linked back to the original mission objectives
3. Missions high in technical/scientific complexity but with poor justification or without a clear “story”
4. Physical dimensions just past our limits
5. Not enough testing of ground –> CanSat radio link or use of a backup (e.g. storing the CanSat data internally)
6. Launch site conditions not matching expectations (e.g. windiness, lack of facilities and environment conditions)

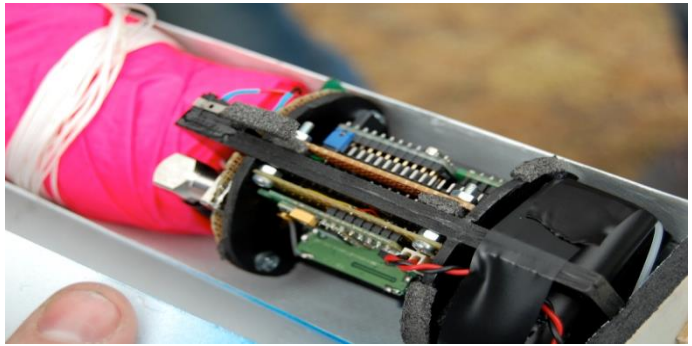
How do we build a CanSat?

CanSat 2019 - ESA events:
https://flic.kr/s/aHsmFaLWQ4

CanSat Design

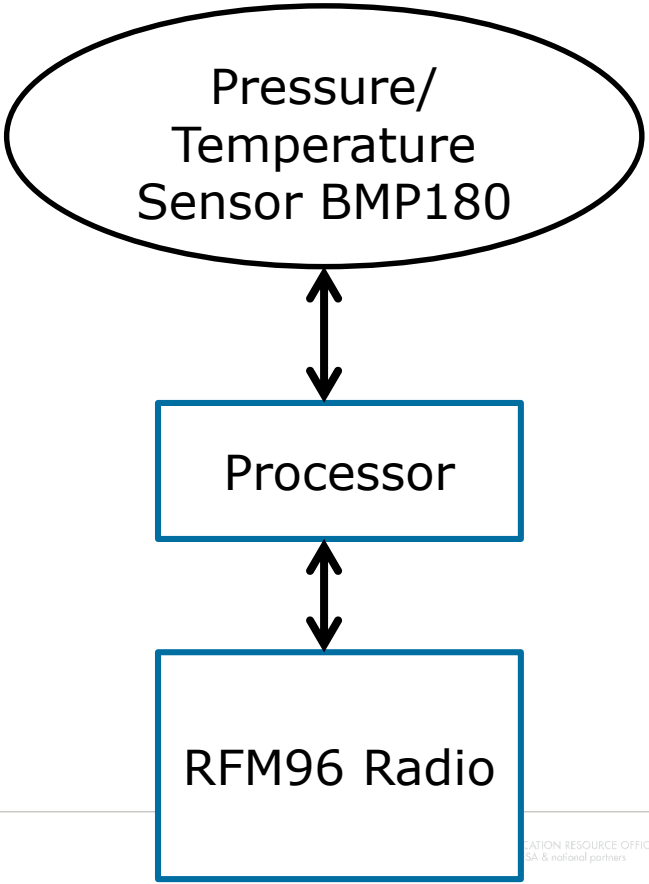
Once a scientific/technical objective has been chosen, the following key engineering elements are simultaneously addressed and refined to fulfil that objective:

- Mechanical Design
- Electrical Design
- Software Design
- Descent Mechanism Design
- Ground Support Equipment Design

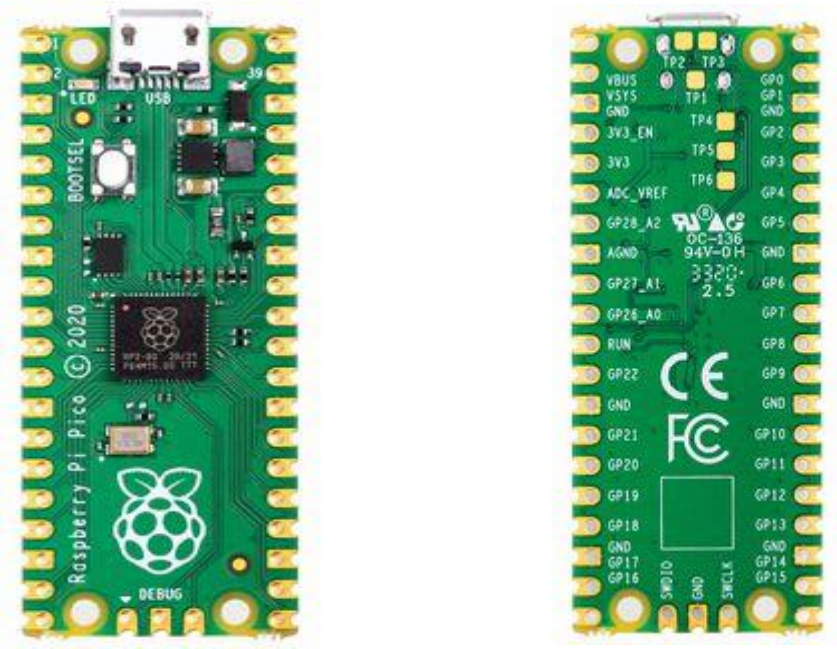


Today's CanSat

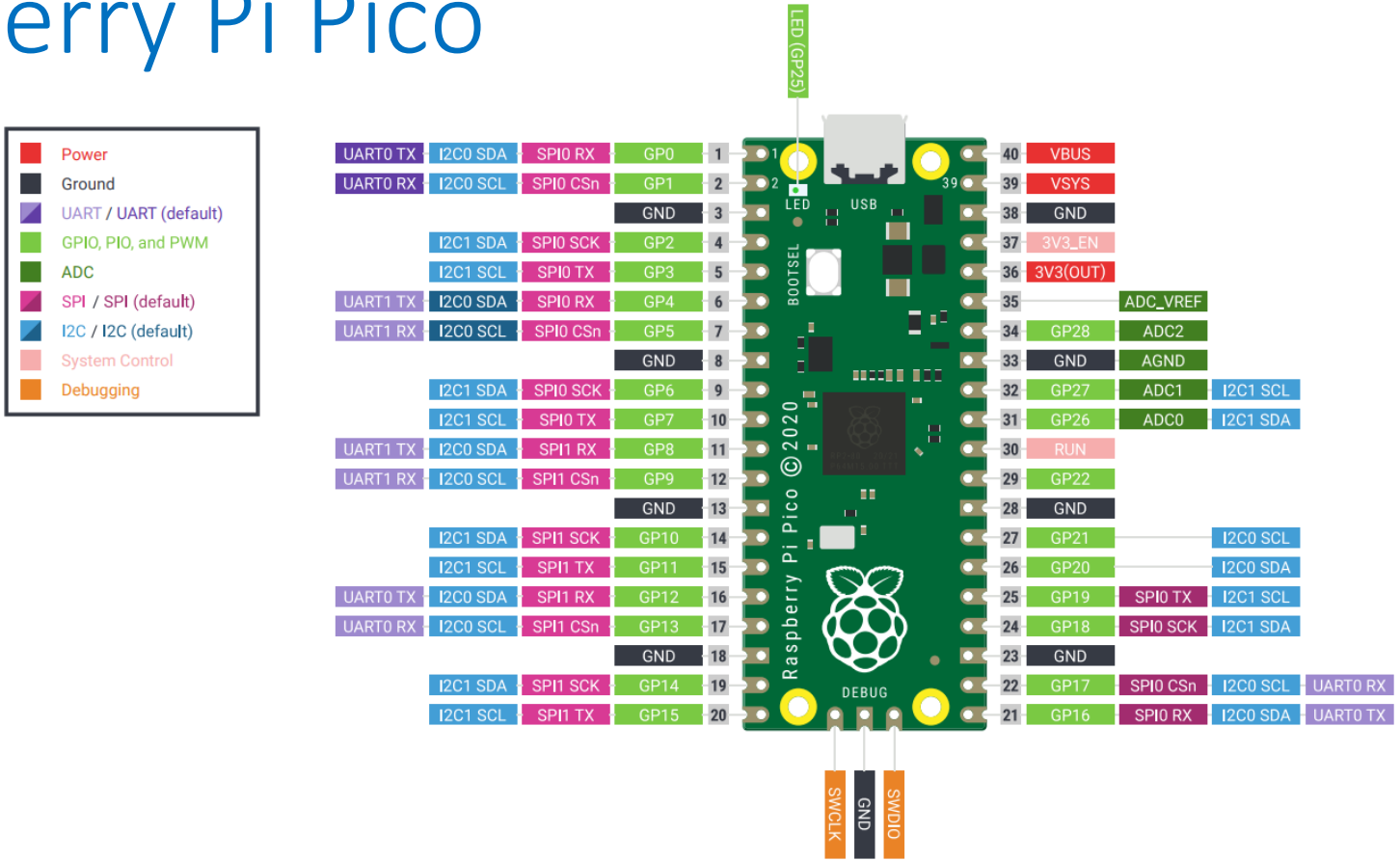
- Will fulfil the electrical, software and ground station aspects of the primary mission!
- Based around a Raspberry Pi Pico for the CanSat computer
- All components in the kit are available cheaply from online distributors



Raspberry Pi Pico



Raspberry Pi Pico



CircuitPython

- Allows us to program the Pi Pico in Python
- Turns the Pi into a USB drive for easy programming
- Loads our program into the Pi when power is applied
- Provides a Python *Read-Evaluate-Print-Loop (REPL)*

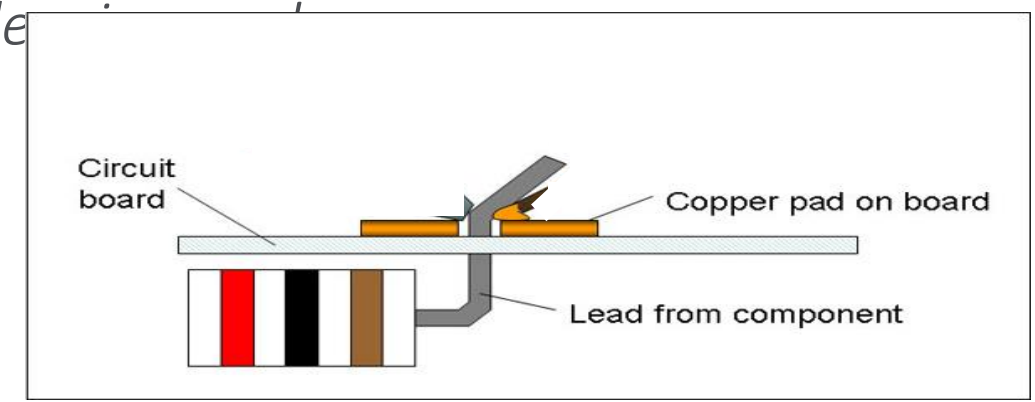
```
Adafruit CircuitPython 7.0.0 on 2021-09-20; Raspberry Pi Pico with rp2040
>>> |
```

- Our code is stored on the Pi in *code.py* and other files as required
- 3rd Party libraries are stored on the Pi in the */lib/* folder

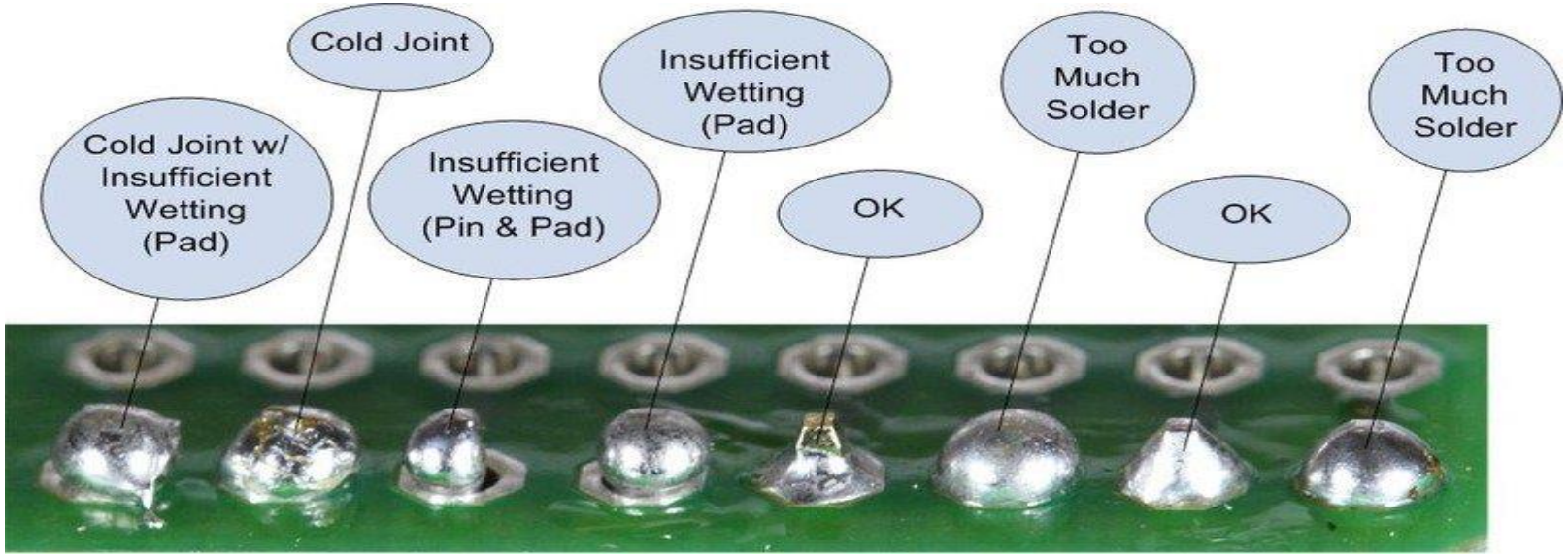
Lab 1: Soldering + GPIO

Soldering

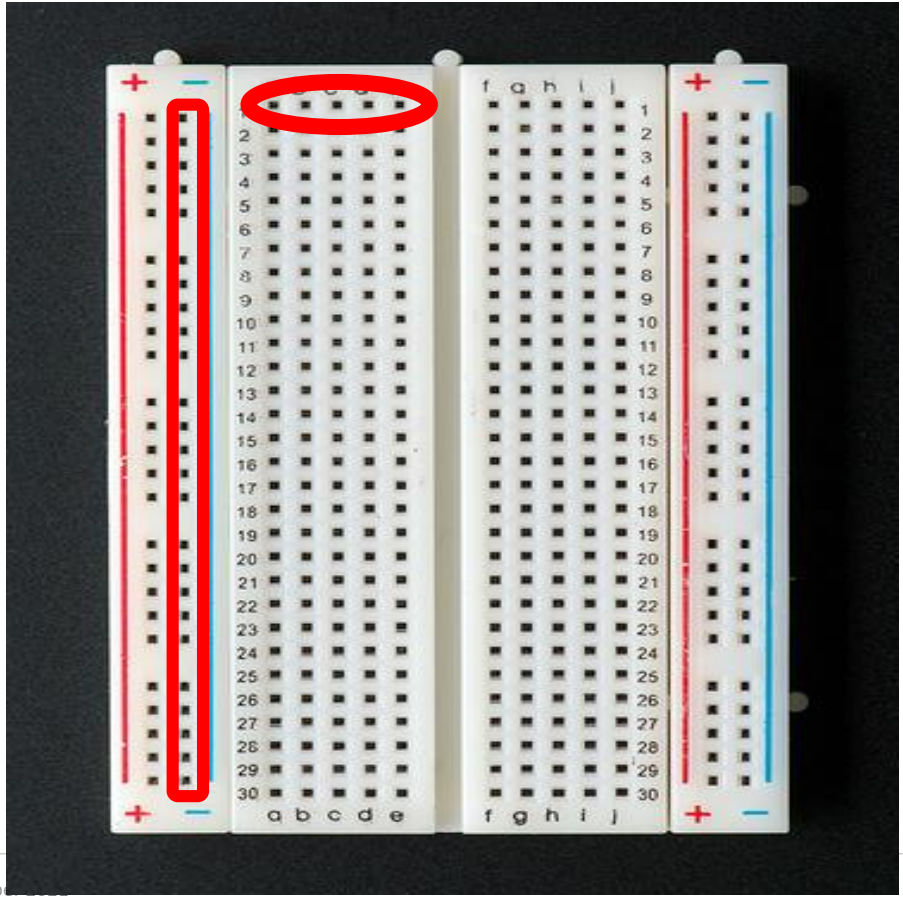
1. Place component through hole
2. Heat copper pad **and** component leg with soldering iron
3. Introduce solder slowly -> *le*
4. Withdraw solder
5. Withdraw soldering iron
6. Inspect joint



Soldering

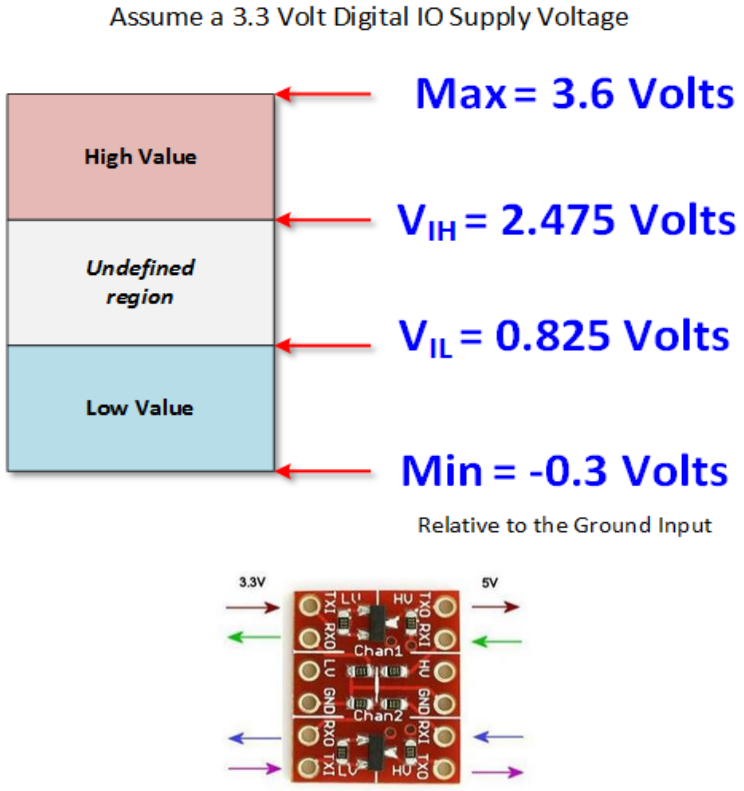

Cold Joint w/ Insufficient Wetting (Pad)
Cold Joint
Insufficient Wetting (Pin & Pad)
Insufficient Wetting (Pad)
OK
Too Much Solder
OK
Too Much Solder

Soldering

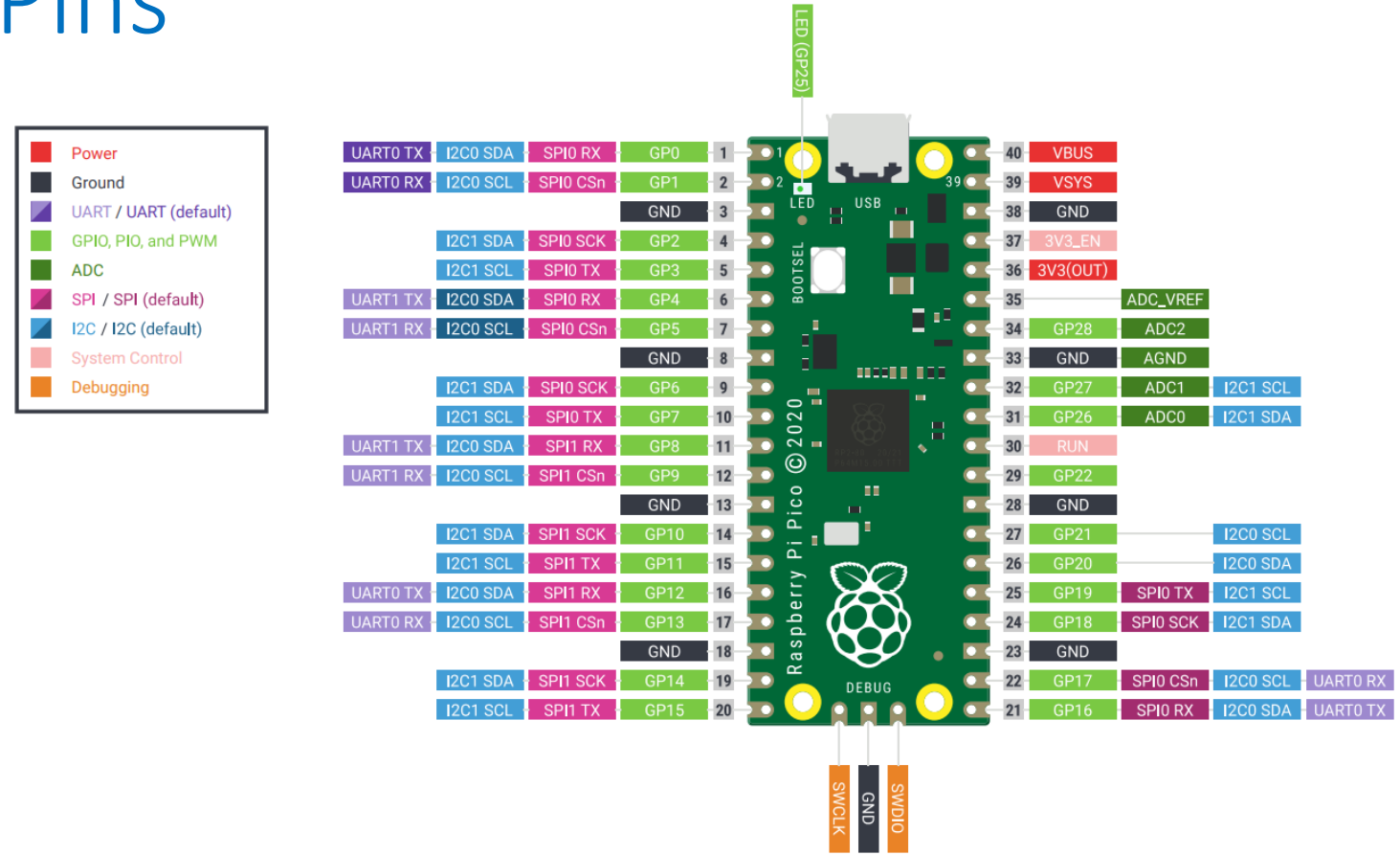

f g h i j
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
a b c d e
f g h i j

GPIO Pins

- General Purpose Input/Output Pins
- Can be used to either output voltages or read in voltages
- The voltage read is converted to a binary “True”/1 or “False”/0
- The voltage output can be either 0V or the chip’s voltage
- Interpretation is dependent on chip’s voltage – do not mix 3.3 and 5V systems without converters!



GPIO Pins


Power
Ground
UART / UART (default)
GPIO, PIO, and PWM
ADC
SPI / SPI (default)
I2C / I2C (default)
System Control
Debugging
LED (GP25)
LED
USB
BOOTSEL
Raspberry Pi Pico © 2020
DEBUG
SWCLK
GND
SWDIO
1 UART0 TX I2C0 SDA SPI0 RX GP0
2 UART0 RX I2C0 SCL SPI0 CSn GP1
3 GND
4 I2C1 SDA SPI0 SCK GP2
5 I2C1 SCL SPI0 TX GP3
6 UART1 TX I2C0 SDA SPI0 RX GP4
7 UART1 RX I2C0 SCL SPI0 CSn GP5
8 GND
9 I2C1 SDA SPI0 SCK GP6
10 I2C1 SCL SPI0 TX GP7
11 UART1 TX I2C0 SDA SPI1 RX GP8
12 UART1 RX I2C0 SCL SPI1 CSn GP9
13 GND
14 I2C1 SDA SPI1 SCK GP10
15 I2C1 SCL SPI1 TX GP11
16 UART0 TX I2C0 SDA SPI1 RX GP12
17 UART0 RX I2C0 SCL SPI1 CSn GP13
18 GND
19 I2C1 SDA SPI1 SCK GP14
20 I2C1 SCL SPI1 TX GP15
40 VBUS
39 VSYS
38 GND
37 3V3_EN
36 3V3(OUT)
35 ADC_VREF
34 GP28 ADC2
33 GND AGND
32 GP27 ADC1 I2C1 SCL
31 GP26 ADC0 I2C1 SDA
30 RUN
29 GP22
28 GND
27 GP21 I2C0 SCL
26 GP20 I2C0 SDA
25 GP19 SPI0 TX I2C1 SCL
24 GP18 SPI0 SCK I2C1 SDA
23 GND
22 GP17 SPI0 CSn I2C0 SCL UART0 RX
21 GP16 SPI0 RX I2C0 SDA UART0 TX

Software

- *Libraries* allow code other people have written to be easily included in your code:

```
import digitalio
```

- *Functions* allow library code to be used:

```
rfm9x.send(message)
```

- *Loops* allow code to be repeated a number of times (or whilst a logical condition is true):

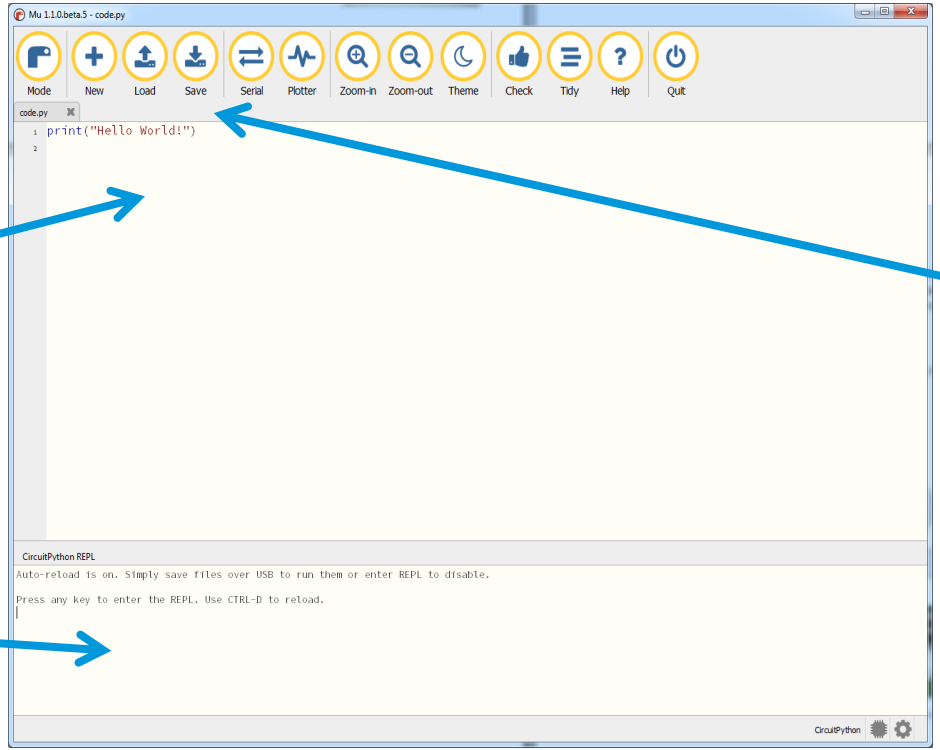
```
while True:
    #toggle the LED
    led.value = not led.value
    time.delay(1)
```

- *Comments* are ignored when compiling your code. Please encourage their use! :-)

```
#send the temperature and pressure readings over the radio
```

Software (CircuitPython using Mu)

- Mu Editor for simple CircuitPython development



Python Code

Program Output/REPL

Save to upload code to Pi Pico and start running

Software

```
#add external libraries created by third parties
import board
import digitalio
import time

#GPIO settings for LED
led = digitalio.DigitalInOut(board.GP25)
led.direction = digitalio.Direction.OUTPUT

while True:
    #toggle the LED
    led.value = not led.value
    time.delay(1)
```

Import Libraries

Assign GPIO pin

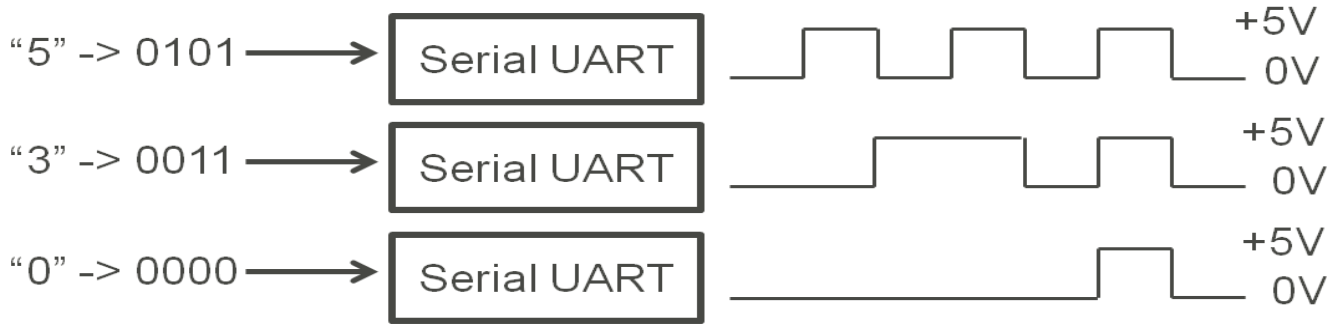
Setup GPIO pin as output

Write to GPIO

Call Library function

Serial Communication

- ***U**niversal **A**synchronous **R**eceiver-**T**ransmitter*
- Converts values into binary streams of data



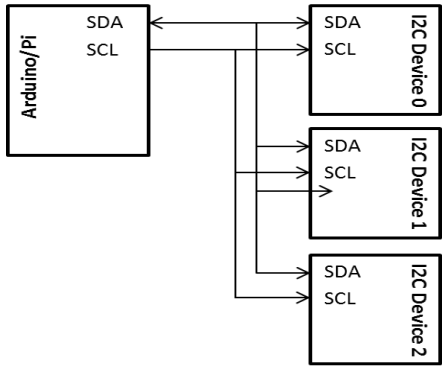
- Useful for sending text-based data over a wire
- Simple, only two wires required
- Has also been used in CanSat to communicate with GPS sensors and 3G modems

Lab 2: Sensors + I2C

I2C Communication

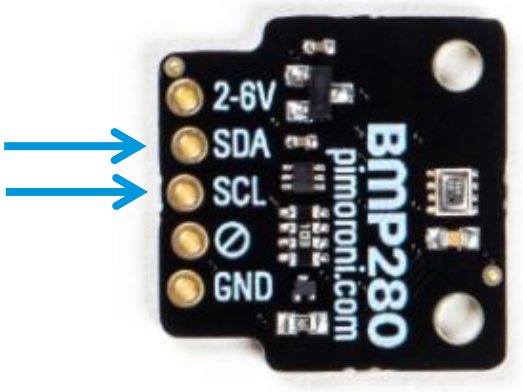
- ***I**nter-**I**ntergrated **C**ircuit*
- Another serial protocol, like UART
- However allows more than one device to be connected (1008!):

SDA: slave data
SCL: slave clock

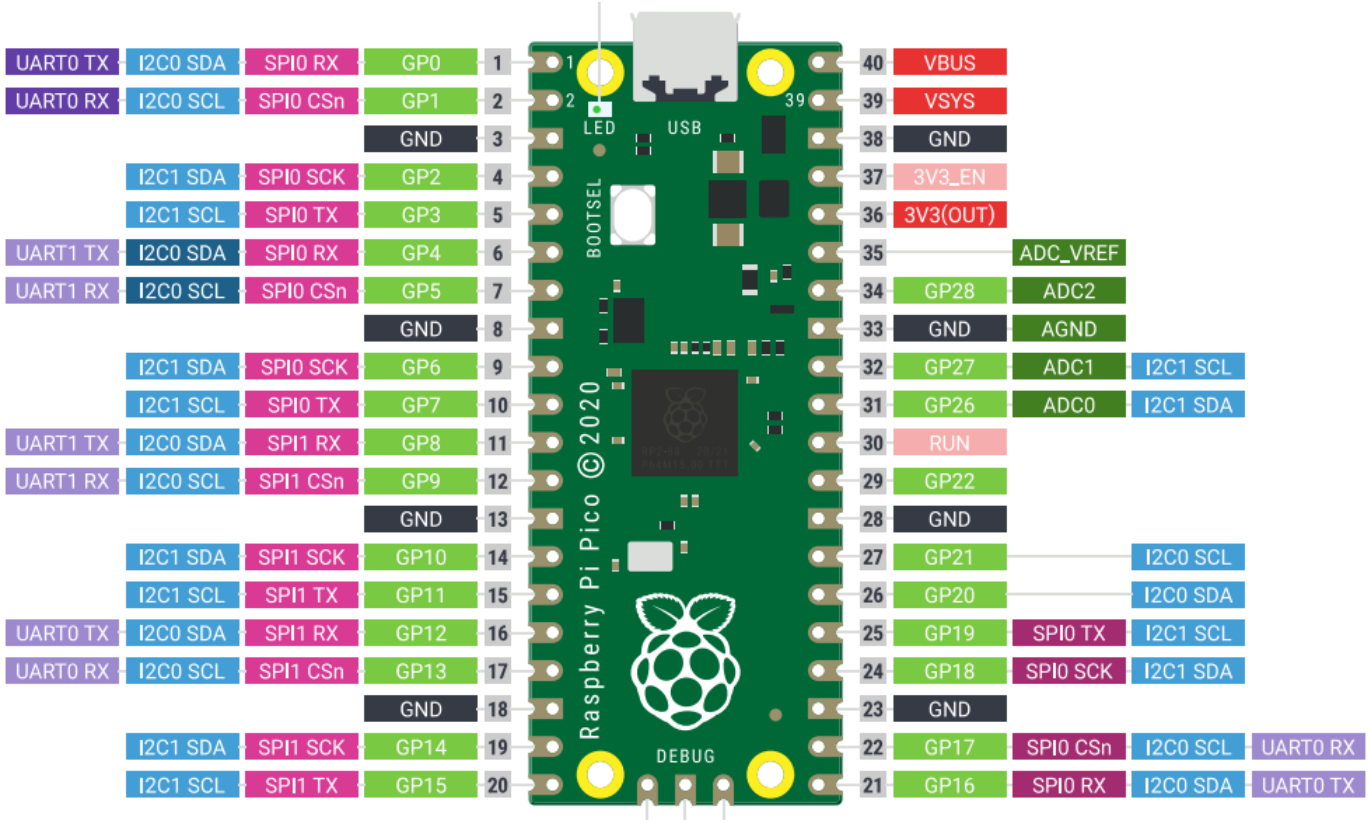


- Usually works with a vendor provided library
- Simple, only two wires required
- Has been used in CanSat to communicate with many sensors – gas, accelerometers, Analog-to-Digital

I2C Communication



I2C Communication



Using the I2C with BMP280

- First Import BMP280 Library:

```
import board
import busio
import adafruit_bmp280
```

- Define the I2C pins:

```
i2c = busio.I2C(scl = board.GP15, sda = board.GP14)
```

- Create a BMP280 object and connect to I2C:

```
bmp280_sensor = adafruit_bmp280.Adafruit_BMP280_I2C(i2c, address=0x76)
```

- Read the temperature and pressure:

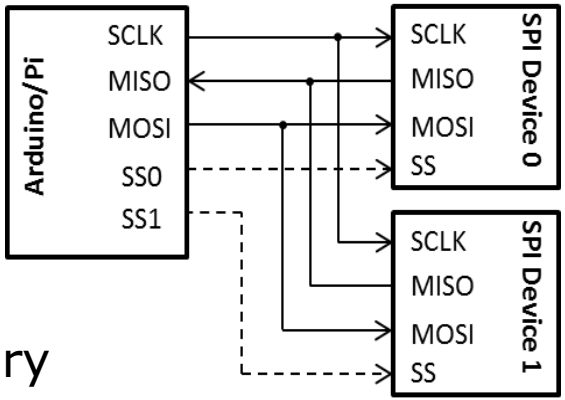
```
def read_temperature():
    return bmp280_sensor.temperature

def read_pressure():
    return bmp280_sensor.pressure
```

Lab 3: Radio Communication + SPI

SPI Communication

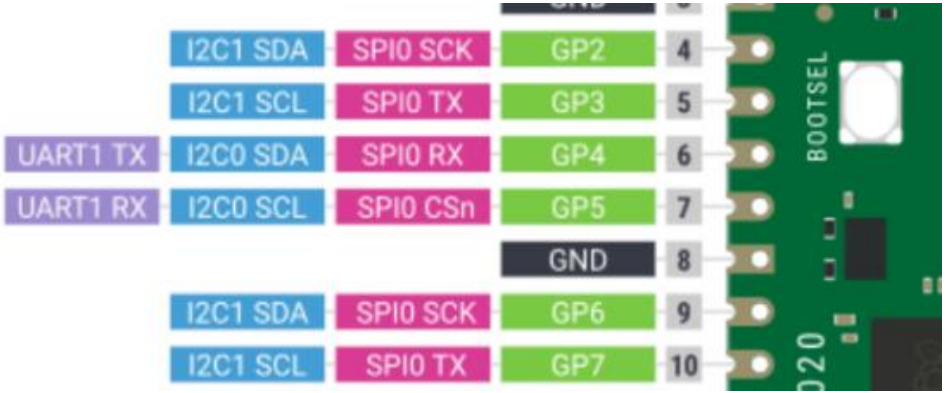
- ***S**erial **P**eripheral **I**nterface*
- Yet another serial protocol!
- Higher performance than UART and I2C.
- But requires more wiring:

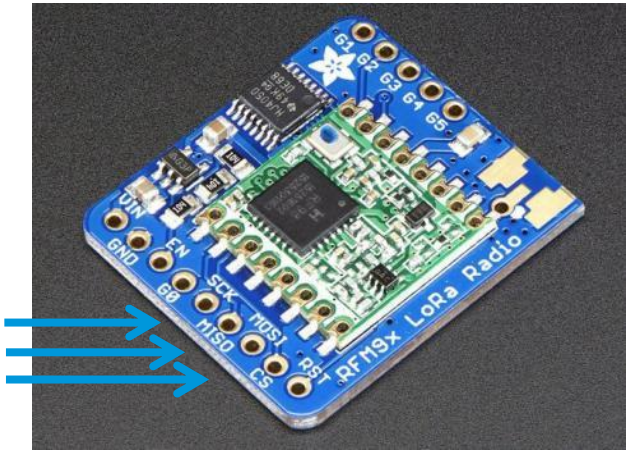
SCLK: Serial Clock.
MISO: Master Input / Slave Output.
MOSI: Master Output / Slave Input.
SS0/CE0: Slave Select / Chip Enable.



- Again, works with a vendor provided library
- Three wires + 1 for each device required
- Used to communicate with more complex devices: cameras, memory cards, WiFi modems

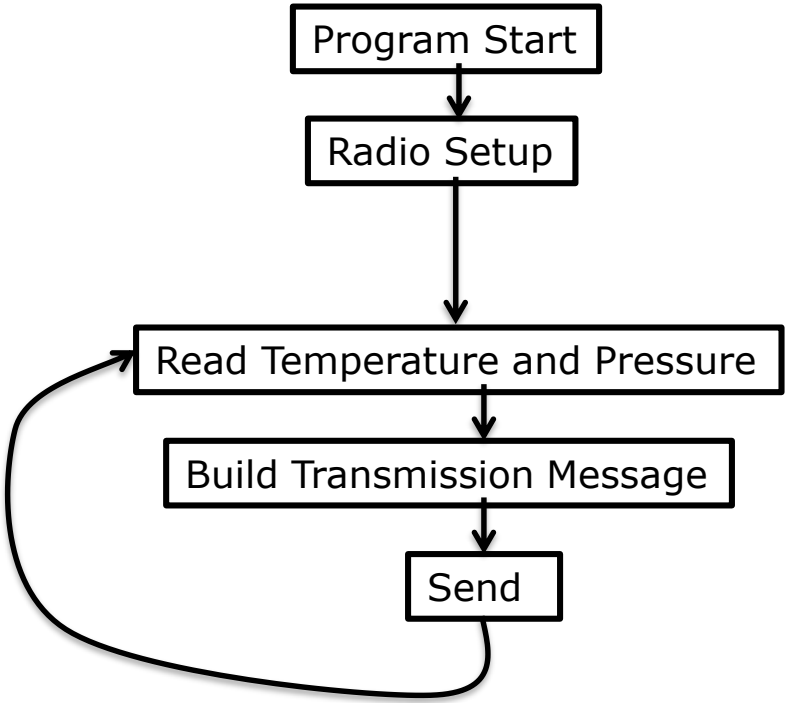
SPI Communication

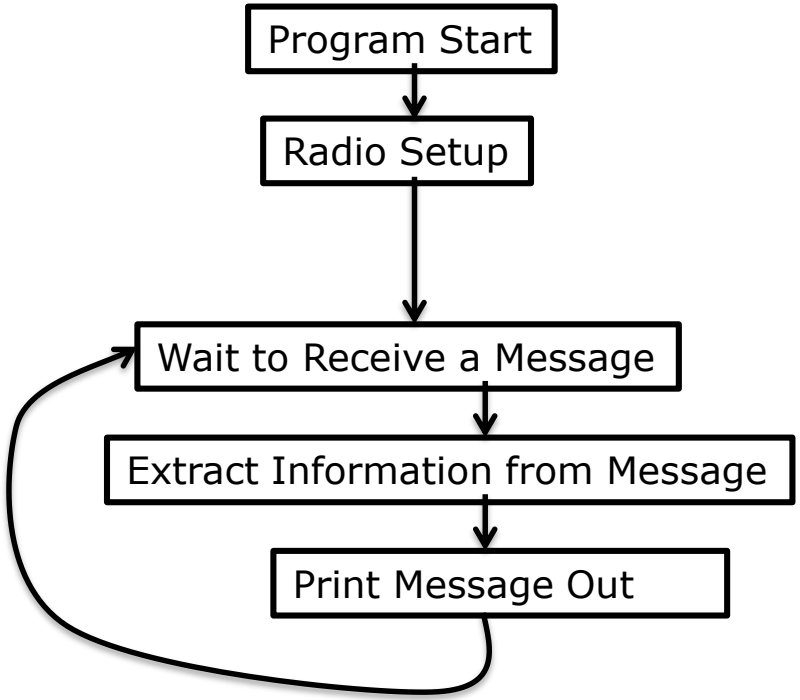




- Pi Pico SPI TX is MOSI *(Master Out, Slave In)*
- Pi Pico SPI RX is MISO *(Master In, Slave Out)*

Radio Software





Radio Setup

- First assign the SPI bus pins:

```
spi = busio.SPI(clock=board.GP2, MOSI=board.GP3, MISO=board.GP4)
```

- Also assign GPIOs for CS and RST pins:

```
cs = digitalio.DigitalInOut(board.GP6)
reset = digitalio.DigitalInOut(board.GP7)
```

- Create an RFM9x radio object from these pins:

```
rfm9x = adafruit_rfm9x.RFM9x(spi, cs, reset, 433.0)
```

The 433.0 argument is the frequency that your radio chip is designed for, 433MHz in our case

Radio Transmission

- Sending data:

```
def send(message):
    rfm9x.send(message)
```

- Message needs to be a string with any values etc converted to string form, i.e. use the .format() method:

```
radio.send("[CANSAT NAME] Temperature: {:.2f} Pressure {:.0f}".format(room_temp, room_pressure))
```

Receiving Data

- Receiving data:

```
def try_read():
    return rfm9x.receive(timeout=1.0)
```

- receive() will wait for the number of settings set in *timeout*
- Received data will be a byte array. You will need to convert it to a text string for printing:

```
str(radio_message, 'ascii')
```

- You can also extract the data from it, take a look at Python arrays for more information