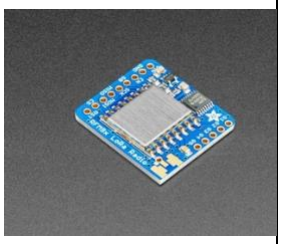
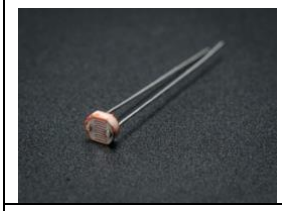
Kit List

Adafruit RFM96W LoRa Radio Transceiver Breakout - 433 MHz (RadioFruit) 	https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts Circuit python libraries: • **adafruit_rfm9x.mpy** • **adafruit_bus_device**
Adafruit RFM69HCW Transceiver Radio Breakout - 433 MHz (RadioFruit) 	https://learn.adafruit.com/adafruit-rfm69hcw-and-rfm96-rfm95-rfm98-lora-packet-padio-breakouts Circuit python libraries: • **adafruit_rfm69.mpy** • **adafruit_bus_device**
BME688 4-in-1 Air Quality Breakout (Gas, Temperature, Pressure, Humidity) 	https://docs.circuitpython.org/projects/bme680/en/latest/ https://learn.adafruit.com/adafruit-bme680-humidity-temperature-barometic-pressure-voc-gas/python-circuitpython Circuit python libraries: • **adafruit_bme680.mpy** • **adafruit_bus_device**

BMP280 Breakout - Temperature, Pressure, Altitude Sensor 	adafruit_bmp280.mpy https://docs.circuitpython.org/projects/bmp280/en/latest/api.html Circuit python libraries: • **adafruit_bmp280.mpy** • **adafruit_bus_device**
Adafruit DRV8833 DC/Stepper Motor Driver Breakout Board 	https://learn.adafruit.com/adafruit-drv8833-dc-stepper-motor-driver-breakout-board/downloads Circuit python libraries: • **adafruit_motor** • import pwmio
Photoresistor 	Circuit python libraries: import analogio
Camera **Arducam Mega 5MP SPI Camera Module - Autofocus** 	Possible coding: Circuit python using camara.py library Or using Using Arducam.py, OV2640_reg and OV5642_reg libraries

Adafruit RFM69HCW Transceiver Radio Breakout - 433 MHz

Pin

Radio RFM69/RFM9x

	Pico Pin
Clock (SCK)	**GP2**
MOSI	**GP3**
MISO	**GP4**
CS	**GP6**
RESET	**GP7**

```
import adafruit_rfm69

spi = busio.SPI(clock=board.GP2, MOSI=board.GP3, MISO=board.GP4)
cs = digitalio.DigitalInOut(board.GP6)
reset = digitalio.DigitalInOut(board.GP7)

rfm69 = adafruit_rfm69.RFM69(spi, cs, reset, 433.0)

Sending Data
rfm69.send('Hello world!')

Receiving Data
data = rfm9x.receive(timeout=5.0)
print("Receiving data",data)
datastr = ""

if data is not None:

    for num in data:
        datastr = datastr + chr(num)

    print(datastr)
```

If using **Adafruit RFM96W LoRa Radio Transceiver Breakout - 433 MHz (RadioFruit)**

```
import adafruit_rfm9x

spi = busio.SPI(clock=board.GP2, MOSI=board.GP3, MISO=board.GP4)
cs = digitalio.DigitalInOut(board.GP6)
reset = digitalio.DigitalInOut(board.GP7)
rfm9x = adafruit_rfm9x.RFM9x(spi,cs,reset,433.0,baudrate=921600)
```

BME688 4-in-1 Air Quality Breakout (Gas, Temperature, Pressure, Humidity)

BMP260

	Pico Pin
SDA	**GP0**
SCL	**GP1**

```
import adafruit_bme680

temp = 0
humidity = 0
altitude = 0
air = 0

 i2c = busio.I2C(scl=board.GP1,sda=board.GP0)
 bme = adafruit_bme680.Adafruit_BME680_I2C(i2c,address=0x76)
 # change this to match the location's pressure (hPa) at sea level
 bme.sea_level_pressure = 991
```

BMP280 Breakout - Temperature, Pressure, Altitude Sensor

```
import adafruit_bmp280

i2c = busio.I2C(scl=board.GP1,sda=board.GP0)
bme = adafruit_bmp280.Adafruit_BMP280_I2C(i2c,address=0x76)
# change this to match the location's pressure (hPa) at sea level
bme.sea_level_pressure = 991

temperature_offset = -5
temp = str(bme.temperature + temperature_offset) #Convert float to str
```

Adafruit DRV8833 DC/Stepper Motor Driver Breakout Board

DRV8833 motor controller

The DRV8833 motor controller can control the two motor coils inside the stepper motor

PICO

- Pico **pin 25 (GP19)** to driver **pin AIN2**
- Pico **pin 24 (GP18)** to driver **pin AIN1**
- Pico **GND** to driver **GND**
- Pico **3v3** to driver **VM**

If using a reset button

- Pico **pin 30 (RESET)** to reset button to **GND**

Power

- Power supply **5V+** to driver **SLP**
- Power supply **GND** to common ground

Motor (wiring color varies by motor)

- Orange **motor wire** to driver **AOUT1**
- Pink **motor wire** to driver **AOUT2**

```
def configureMotor():
    PWM_PIN_A = board.GP19 #AIN2
```

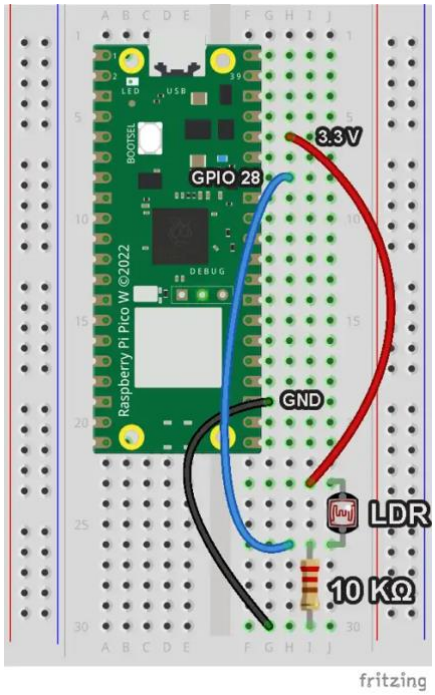
```
    PWM_PIN_B = board.GP18 #AIN1
    pwm_a = pwmio.PWMOut(PWM_PIN_A,frequency = 50)
    pwm_b = pwmio.PWMOut(PWM_PIN_B,frequency = 50)
    motor1 = motor.DCMotor(pwm_a,pwm_b)
    return motor1

def startMotor(motor1):
    motor1.throttle = 1
    print("throttle:",motor1.throttle)

def stopMotor(motor1):
    motor1.throttle = 0
    print("DC motor stop")
```

Photoresistor

Pico GP28




```
def configureLight():

    photoresistor = analogio.AnalogIn(board.GP26)
    return photoresistor

def checkLight(photoresistor):

    light = False

    lightLevel = photoresistor.value
    lightLevel = round(lightLevel/65535*100,2)


    if lightLevel < 90:
        print("Light!")
        light = True
    else:
        print("Dark!")
        light = False

    return light
```

Camera

Take picture on cansat and send data to ground station:

As of now, **CircuitPython** doesn't have native libraries for the **Arducam Mega** series, including support for its SPI or I2C interfaces.

However, you can use compatible CircuitPython SPI/I2C communication approach to interface with the Arducam Mega by translating commands from the Arducam's documentation.

Arducam Mega Communication

Using communication protocol:

- **SPI**

To use it in CircuitPython, you'll need to:

1. Manually send commands over SPI/I2C.
2. Interpret and process the returned data.

2. Required CircuitPython Libraries

Ensure your CircuitPython environment has the following libraries installed:

- adafruit_bus_device (for SPI/I2C communication).

3. Using SPI to Communicate with Arducam Mega

Here's an example template for SPI communication with the Arducam Mega:

1. Pin Configuration

To use the Arducam Mega with a Raspberry Pi Pico connected to the pins you specified, you can configure CircuitPython or MicroPython for SPI communication as follows:

	Pico Pin
VCC	**5V**
GND	**GND**
Clock (SCK)	**GP14**
MOSI	**GP15**
MISO	**GP16**
CS	**GP17**

VCC and GND: Power and ground connections.
SPI Pins (SCK, MISO, MOSI): Ensure the pins match your SPI peripheral configuration in software.
CS (Chip Select): Used to enable communication with the camera. Set this pin low to select the device.

Circuit Python Code

Sending an image using the camera.py library

```
import board
import busio
import digitalio
import time
from Camera import Camera  # Replace with the correct library for your Arducam
import adafruit_rfm69

'''
#################### PINOUT ####################
Camera pin - Pico Pin
VCC  - 3V3
GND  - GND
SCK  - GP18
MISO - GP16
MOSI - GP19
CS   - GP17
RFM69 PINOUT:
SCK   - GP2
MOSI  - GP3
MISO  - GP4
CS    - GP6
RESET - GP7
VCC   - 3V3
GND   - GND
'''

# Initialize SPI for Camera
spi_cam = busio.SPI(clock=board.GP18, MISO=board.GP16, MOSI=board.GP19)

# Chip Select Pin for Camera
cs_cam = digitalio.DigitalInOut(board.GP17)
cs_cam.direction = digitalio.Direction.OUTPUT
cs_cam.value = True  # Deselect by default

# Onboard LED Pin
onboard_LED = digitalio.DigitalInOut(board.LED)  # Onboard LED
onboard_LED.direction = digitalio.Direction.OUTPUT

# Initialize the Camera
cam = Camera(spi_cam, cs_cam)

# Set camera settings
```

```
cam.resolution = '640x480'
cam.set_brightness_level(4)  # Example: Brightness +4
cam.set_contrast(-3)         # Example: Contrast -3

# Function to initialize RFM69
def radio_rfm69():
    spi_radio = busio.SPI(clock=board.GP2, MOSI=board.GP3,
MISO=board.GP4)
    cs_radio = digitalio.DigitalInOut(board.GP6)
    reset_radio = digitalio.DigitalInOut(board.GP7)
    rfm69 = adafruit_rfm69.RFM69(spi_radio, cs_radio, reset_radio, 433.0)
    return rfm69

# Initialize RFM69 Radio
rfm69 = radio_rfm69()

# Capture and send image information
try:
    onboard_LED.value = True  # Turn on onboard LED to indicate activity
    cam.capture_jpg()
    time.sleep(0.05)  # Short delay
    image_path = 'image.jpg'
    cam.saveJPG(image_path)  # Save the captured image
    onboard_LED.value = False  # Turn off onboard LED
    print("Image capture complete.")

    # Read the image file into a buffer
    with open(image_path, 'rb') as img_file:
        buffer = img_file.read()  # Read entire image into buffer
        print(f"Buffer size: {len(buffer)} bytes.")

        # Define chunk size (adjust as necessary)
        chunk_size = 100  # For example, 100 bytes per chunk
        num_chunks = len(buffer) // chunk_size + (1 if len(buffer) % chunk_size > 0
else 0)

        # Send the buffer in chunks
        for i in range(num_chunks):
            start = i * chunk_size
            end = min(start + chunk_size, len(buffer))
            chunk = buffer[start:end]

            # Send the chunk over RFM69
            rfm69.send(chunk)
            print(f"Sent chunk {i+1}/{num_chunks} ({len(chunk)} bytes).")
            time.sleep(0.1)  # Small delay between chunks to avoid overloading
the radio
```

```
    print("Image transmission complete.")

except Exception as e:
    print(f"An error occurred: {e}")
```

Alternative using Arducam.py, OV2640_reg and OV5642_reg

```
from Arducam import *  # Ensure this is in the same directory as the script
import time

def send(rfm9x, data):
    rfm9x.send(data)
    print(f"Sending {len(data)} bytes of data")

def camera_capture_and_send():
    once_number = 128  # Number of bytes to read per burst
    buffer = bytearray(once_number)  # Buffer to hold image chunks

    # Initialize the camera
    mycam = ArducamClass(OV5642)
    if not mycam.Camera_Detection():
        print("Camera not detected!")
        return

    print("Camera detected.")
    mycam.Spi_Test()
    mycam.Camera_Init()
    mycam.Spi_write(ARDUCHIP_TIM, VSYNC_LEVEL_MASK)
    time.sleep(1)

    # Configure the camera for JPEG capture
    mycam.clear_fifo_flag()
    mycam.set_format(JPEG)
    mycam.OV5642_set_JPEG_size(OV5642_320x240)

    # Start capturing the image
    mycam.flush_fifo()
    mycam.clear_fifo_flag()
    mycam.start_capture()

    # Get the length of the data in the FIFO
    length = mycam.read_fifo_length()
    print(f"Image length: {length} bytes")

    if length == 0 or length > 0xFFFF:
        print("Invalid image length. Aborting.")
        return
```

```
    # Read and send image data in chunks
    count = 0
    mycam.SPI_CS_LOW()  # Begin SPI transfer
    mycam.set_fifo_burst()

    try:
        while count < length:
            # Determine the number of bytes to read in this iteration
            bytes_to_read = min(once_number, length - count)

            # Read from FIFO and send
            mycam.spi.readinto(memoryview(buffer)[:bytes_to_read])
            send(buffer[:bytes_to_read])  # Ensure only valid data is sent
            count += bytes_to_read
            print(f"Sent {bytes_to_read} bytes. Total sent: {count}/{length} bytes")

            time.sleep(0.05)  # Optional: delay between bursts

    except Exception as e:
        print(f"Error during image transfer: {e}")

    finally:
        mycam.SPI_CS_HIGH()  # End SPI transfer
        mycam.clear_fifo_flag()

    print("Image transmission complete.")

# Run the capture and send function
camera_capture_and_send()
```

Receiving the image

When you receive the image data from the camera, the main steps to process it involve:

1. **Reassembling the Data**: Combining chunks into a complete image.
2. **Saving the Image**: Writing the reassembled data to a file.
3. **Decoding (if necessary)**: If the data is encoded (e.g., Base64 or JPEG), decode it into a usable image format.
4. **Displaying or Further Processing**: Render the image on a display or perform further image processing.

Below is a step-by-step explanation with sample code for each stage:

1. Reassemble the Data

When receiving data chunks, you need to ensure they are in the correct order and form the complete image.

Example:

```
def receive_and_reassemble(receive_function, total_length, chunk_size=128):
    """
    Reassemble the image data received in chunks.
    :param receive_function: Function to receive data chunks.
    :param total_length: Total length of the data to be received.
    :param chunk_size: Size of each data chunk.
    :return: Reassembled bytearray containing the complete image.
    """
    received_data = bytearray()
    total_received = 0

    while total_received < total_length:
        # Receive a chunk
        chunk = receive_function(chunk_size)
        received_data.extend(chunk)
        total_received += len(chunk)
        print(f"Received {total_received}/{total_length} bytes")

    return received_data
```

- receive_function: Function to retrieve chunks of data (e.g., via UART, SPI, or radio).
- chunk_size: Number of bytes to request per iteration.

2. Save the Image

Once you have the complete data in a byte array, save it to a file for verification or later use.

Example:

```
def save_image(data, file_name="received_image.jpg"):
    """
    Save the reassembled image data to a file.
    :param data: Byte array containing image data.
    :param file_name: Name of the file to save the image to.
    """
    with open(file_name, "wb") as f:
        f.write(data)
    print(f"Image saved as {file_name}")
```

3. Decode the Image (if necessary)

If the data is in a compressed or encoded format (e.g., Base64 or JPEG), decode it before displaying or processing.

Base64 Decoding:

If the image data was Base64-encoded during transmission:

```
import base64

def decode_base64(data):
    """
    Decode Base64-encoded data.
    :param data: Base64-encoded byte array.
    :return: Decoded byte array.
    """
    return base64.b64decode(data)
```

Verify or Decode the Image Format:

If the data is in JPEG format, ensure you have the necessary libraries to open or render it (e.g., **Pillow** in Python).

4. Display the Image

After saving or decoding, render the image on a display or view it on a PC.

Viewing on a PC (Using Pillow):

If the received data is saved as a JPEG file:

```
from PIL import Image

def display_image(file_name="received_image.jpg"):
    """
    Display an image using Pillow on a PC.
    :param file_name: Name of the image file to display.
    """
    img = Image.open(file_name)
    img.show()
```

5. Further Processing

Once you have reassembled the data, saved, and verified it as a proper image, you can:

- **Analyze**:
- Use OpenCV for computer vision tasks (e.g., edge detection, object recognition).
- **Compress**:
- Reduce the file size for storage or retransmission.
- **Transmit Further**:
- Forward the image to another system, cloud storage, or web interface.

Complete Example

Here's a full workflow combining reception, reassembly, saving, and displaying:

```
# Mock function for receiving data (replace with your implementation)
def mock_receive(chunk_size):
    return bytearray([0xFF] * chunk_size)  # Replace with actual data reception

# Main workflow
total_length = 1024  # Replace with the actual image size
chunk_size = 128

# Step 1: Reassemble
image_data = receive_and_reassemble(mock_receive, total_length,
chunk_size)

# Step 2: Save
save_image(image_data, "received_image.jpg")

# Step 3: Display
display_image("received_image.jpg")
```

Key Notes

Error Checking:
Implement checks for corrupted data using sequence numbers.
Chunk Ordering:
Ensure chunks are received and processed in the correct order.
Storage Considerations:
Ensure the receiving system has enough memory to buffer large images.