ChatGPT

Using Two Raspberry Pi Pico Boards to Wireless Transfer Images from a Dimension EasyCheck V2

Overview: We will use two Raspberry Pi Pico boards (preferably Pico **W** models for Wi-Fi) to intercept image files saved by the Dimension EasyCheck V2 digital scope and transfer them wirelessly to a computer. **Pico A** will act as a "smart SD card" interface on the scope, detect new JPEG images saved to the SD slot, and send them via Wi-Fi to **Pico B**. Pico B (connected to your PC via USB) will receive the image data, reconstruct the JPEG file, and save it. Finally, you will copy the image from Pico B to your Windows PC. The following guide provides exhaustive step-by-step instructions, including hardware wiring, software setup, MicroPython scripts for each Pico, and all necessary commands.

Background Research and Plan

Before building, it's important to understand the challenges and plan each part:

- **Interfacing with the SD Card Slot:** The EasyCheck V2 scope can save images to an SD card. We will **replace or tap into the SD card interface** with Pico A. Because the Raspberry Pi Pico (RP2040 microcontroller) does not natively act as an SD *device*, we can't simply "emulate" an SD card at full speed (that's very complex and generally requires FPGA or specialized hardware [1] [2]). Instead, we will **wire Pico A in parallel with the SD card lines** so it can access the card as a second master when the scope is not using it. MicroPython does not support the high-speed SDIO 4-bit mode on Pico, so we will use the simpler **SPI mode** for SD card communication [3]. (The SD card *does* support SPI fallback mode, which uses the MISO, MOSI, SCK, CS signals.)

- **Avoiding Bus Conflicts:** Having two masters (the scope's internal controller and Pico A) on one SD card bus requires care. One recommended approach is to connect the Pico through **series resistors** on the SPI lines to gently tap the signals without driving them at the same time as the scope [4]. In practice, this means Pico A will **only drive the SD lines when the scope is idle**, and stay in input mode otherwise. We will ensure Pico A accesses the card only after the scope finishes writing an image. (The EasyCheck V2 likely writes an image file each time you capture; we'll detect when a new file appears.) This method essentially uses the real SD card as a shared storage: the scope writes to it, and Pico A reads from it [4].

- **Wireless Communication (Pico to Pico):** To send data between the two Pico boards wirelessly, we will use Wi-Fi (since Pico W has built-in Wi-Fi). Both Picos will connect to your Wi-Fi network as stations (clients) [5]. One Pico will run a simple **server** socket and the other a **client**, implementing a basic TCP/IP transfer. (This avoids dealing with lower-level radio modules or protocols.) For reliability and simplicity, we will send the entire image file as a binary stream over a TCP socket. The concept is similar to having one Pico act as a small web server and the other as a client [5], but we'll use a custom protocol to transfer the file. Using MicroPython's `network` and `socket` modules, we can connect to Wi-Fi and send data over TCP sockets (the same mechanism used to transfer files on networks) [6] [7].

- **Data Format for Transfer:** Instead of “deconstructing” the image into text (which would bloat the size and complicate reconstruction), we will send the raw binary JPEG data. Pico A will read the JPG file in chunks and send those bytes directly. To let Pico B know when the file ends, Pico A will first send a **header** containing the file size (in bytes). Pico B will then know exactly how many bytes to expect for the image. This approach is simpler and faster than converting the image to text. (If needed, one could encode to Base64 text, but that adds ~33% overhead and isn’t necessary here.)

- **Reconstructing and Saving the Image on PC:** Pico B will receive the bytes and save them as a `.jpg` file on its internal flash filesystem. Because Pico B is connected to your PC via USB, you can then retrieve that file. We will demonstrate using Thonny IDE’s file interface to copy the file from Pico B to a folder on your Windows PC. (Alternatively, one could use a Python script with PySerial or `mpremote` to pull the file, but the Thonny method is straightforward.)

With this plan in mind, let’s proceed step by step.

Step 1: Gather Hardware and Tools

- **Raspberry Pi Pico boards:** 2 units. It’s strongly recommended to use **Raspberry Pi Pico W** for both, as we need wireless capability. (If you only have original Pico (no Wi-Fi), you’d have to attach external RF modules or use a wired connection, which is beyond this scope.)
- **MicroSD card and SD slot access:** You have two options:
- Use the scope’s existing microSD **card slot** by inserting a custom-built “interposer.” For example, you could take a microSD extension cable or breakout board and modify it. One idea is to use a microSD-to-SD adapter and solder wires from it to Pico, but an easier way is to get a **microSD breakout module** (with SPI interface) and actually insert a microSD card that both the scope and Pico share. However, having both the device and Pico on one card requires the wiring method described (with caution).

- **Alternatively**, use the scope’s USB port (it supports USB flash drives) with a different approach. *This guide assumes using the SD slot as requested.* (Note: If SD wiring proves too difficult, a future improvement might be to emulate a USB flash drive with a microcontroller, but MicroPython on Pico doesn’t support USB mass storage device mode by default, so we stick with the SD method.)

- **Level shifting / resistors:** The Pico and SD card both use 3.3V logic, so no level shifter is needed (do **not** apply 5V to SD lines!). However, to avoid contention on SPI lines, prepare some resistors (e.g. ~1 kΩ) to put in series with at least the clock and MOSI lines from the scope to Pico. These will protect against both devices driving the line simultaneously. You might also use a tri-state buffer or MOSFET-based SD card sharing circuit if you have one, but resistors will do for a simple solution [4].

- **Breadboard and jumper wires:** For prototyping the connections between Pico A and the SD card (or breakout). If soldering directly to the scope’s SD slot, ensure you have fine wire and good soldering tools. An SD card pinout will be needed (provided in the wiring step below).

- **Computer with Windows:** to run Thonny IDE and connect to Pico B over USB.

- **Software Tools:**

- **Thonny IDE** (recommended) – an easy IDE for MicroPython on Pico. If you don't have it, download and install Thonny from thonny.org on your Windows PC.
- **MicroPython firmware for Pico W** – download the latest MicroPython `.uf2` file for Pico W from the Raspberry Pi site. (For example, from the RaspberryPi documentation downloads page.)
- We will provide all MicroPython scripts; no additional libraries except one: the `sdcard.py` driver (which we will install on Pico A).

Step 2: Flash MicroPython on Both Pico Boards

1. **Put the Pico into bootloader mode:** Push and hold the **BOOTSEL** button on the Pico, then connect it to your PC via USB. Release BOOTSEL after plugging in. The Pico will mount as a USB mass storage device (RPI-RP2).
2. **Flash the MicroPython firmware:** Download the `micropython-uterminal-pico-w-...uf2` (or similarly named) UF2 file for Pico W. In Windows File Explorer, simply drag and drop this `.uf2` file onto the RPI-RP2 drive. The Pico will reboot automatically into MicroPython.
3. *Alternative:* In Thonny, you can go to **Tools > Options > Interpreter**, select "MicroPython (Raspberry Pi Pico)" and click "Install or update firmware…", then follow prompts to flash MicroPython.
4. **Repeat** for the second Pico. Ensure both Pico A and Pico B have MicroPython running.
5. To verify, open Thonny. In the bottom-right corner, select the Pico's serial port and MicroPython interpreter. You should see a `>>>` REPL prompt. Try typing `print("Hello")` and press Enter – it should respond in the Thonny shell. This confirms MicroPython is working.

Step 3: Hardware – Wiring Pico A to the Dimension Scope's SD Slot

> **Important:** Disconnect power from the Dimension scope while wiring to avoid shorts. Work carefully to not short the SD pins.

Identify SD card pins: The microSD card slot typically has 8 pins. For SPI mode we care about: **CLK**, **CMD**, **DAT0**, **DAT3**, plus power and ground. In SPI mode:
- **CLK** = clock (generated by master),
- **CMD** = command line, used as MOSI (master-out, slave-in),
- **DAT0** = data line 0, used as MISO (master-in, slave-out),
- **DAT3** = data line 3, repurposed as CS (chip select) in SPI mode.

On a microSD connector, pins are usually numbered as:

```
Pin 1: DAT2
Pin 2: DAT3 (CS in SPI)
Pin 3: CMD (MOSI)
Pin 4: VDD (3.3V)
Pin 5: CLK
Pin 6: VSS (Ground)
Pin 7: DAT0 (MISO)
Pin 8: DAT1
```

(If your slot has a card-detect switch, there may be additional pins for that, but not needed here.)

Wire connections: Use the diagram above to locate the pins. Connect them to Pico A as follows:

- **Ground:** SD VSS pin to Pico GND (common ground). This is crucial for a common reference.
- **3.3V Power:** The Dimension scope's SD slot 3.3V pin (VDD) should normally power the SD card. You have two options: (a) **Leave it powered by the scope** and connect Pico's 3.3V pin *to the same node* (so Pico and SD share power and reference – the Pico will also be powered via USB but sharing 3.3V helps level reference). Or (b) power the SD from Pico's 3.3V. In practice, you can tie the Pico's 3V3 output to the SD VDD so both scope and Pico see the same high level. The scope will then either supply 3.3V or the Pico will – ensure that the scope's 3.3V regulator and Pico's 3.3V are at the same potential to avoid back-powering issues. Easiest is to connect the 3.3V lines together. This way, when the scope is on, it powers the card (and indirectly Pico's 3.3 bus); when scope is off but Pico on, Pico could power the card if needed. (Monitor heating if any; if in doubt, do not drive 3.3V from Pico into the scope when the scope is off.)
- **SD CLK to Pico SCK:** Connect SD CLK (pin 5) to Pico GPIO **GP10** (which is SCK on SPI1) or GP2 (SCK on SPI0) – we will use SPI0 by default in code. In MicroPython SPI, you can use different SPI buses; we'll use **SPI0** with default SCK on GP18 by default for Pico (or we can assign custom pins). To avoid confusion, we'll explicitly define pins in code. For now, physically connect SD CLK to a Pico GPIO that we'll configure as SCK (we will use GP2 for SCK in this example).
- **SD CMD to Pico MOSI:** Connect SD CMD (pin 3) to Pico GPIO (we will use **GP3** for MOSI).
- **SD DAT0 to Pico MISO:** Connect SD DAT0 (pin 7) to Pico GPIO (we will use **GP4** for MISO).
- **SD DAT3 to Pico CS:** Connect SD DAT3 (pin 2) to a Pico GPIO for chip-select (we will use **GP5** as CS).
- **Series resistors:** If possible, insert ~1 kΩ resistor in series on the lines SD CLK, CMD, and DAT0 going into Pico (this can be on the Pico's input side). For example, between the SD CLK pin and Pico GP2, use a resistor. Do similarly for SD CMD to GP3. These resistors let the scope's controller dominate the line when it's actively driving, and protect the Pico from sourcing/sinking large currents if both try to drive. Pico's MISO (GP4) connecting to SD DAT0 can also have a resistor if Pico might ever drive that line (in SPI, MISO is output from card, so Pico won't drive it – so that one is less critical). Also put a resistor on the CS line (SD DAT3 to GP5) – the scope likely drives CS to select the card; Pico will also toggle it when it wants to use the card. Having a resistor there avoids conflict.
- **Card detect (optional):** If the SD slot has a card-detect switch, you might wire that to a Pico pin to sense insert/remove. Not strictly needed; we know a card is "present" because Pico itself is the card or it's always inserted.

Double-check all connections for shorts or miswiring. The Pico's pins we chose (GP2, GP3, GP4, GP5) correspond to SPI0 by MicroPython's default mapping (we'll confirm in code). If you use different pins, we can adjust the code accordingly.

Hardware verification (optional): After wiring, you might test that the scope still recognizes the SD card (if you left a card in or if Pico is acting as one). Turn on the scope – if it complains about SD card, you may need to adjust wiring or ensure the Pico isn't interfering. Ideally, the scope should operate as normal, saving images to what it thinks is an SD card. (If Pico fully replaced the card without an actual SD present, then Pico needs to act as the card – which our MicroPython approach will do by initializing the card interface. So keep reading.)

Step 4: Prepare MicroPython on Pico A (Scope Interface)

Now we will set up Pico A to read the SD card and send files. This involves copying an SD card driver library to Pico A and then writing our main script.

1. **Connect Pico A to Thonny:** Plug Pico A into your PC (if not already) and select it in Thonny (bottom-right, select the COM port and "MicroPython (Raspberry Pi Pico)" as interpreter). You should see a prompt.
2. **Install `sdcard.py` library on Pico A:** MicroPython doesn't have built-in SD card support on Pico, but we have a driver module to handle it. We will create a file called `sdcard.py` on Pico A:
3. Download the MicroPython SD card driver code (for SPI) from the official source or use the one from Random Nerd Tutorials. For convenience, here is the library code:

```python
# sdcard.py - MicroPython SD card SPI driver
# Source: MicroPython drivers (modified by Brenton Schulz, 2022)
from micropython import const
import time

_CMD_TIMEOUT = const(1000)
_R1_IDLE_STATE = const(1 << 0)
_R1_ILLEGAL_COMMAND = const(1 << 2)
_TOKEN_CMD25 = const(0xFC)
_TOKEN_STOP_TRAN = const(0xFD)
_TOKEN_DATA = const(0xFE)

class SDCard:
    def __init__(self, spi, cs, baudrate=1320000):
        self.spi = spi
        self.cs = cs
        self.cmdbuf = bytearray(6)
        self.dummybuf = bytearray(512)
        self.tokenbuf = bytearray(1)
        for i in range(512):
            self.dummybuf[i] = 0xFF
        self.dummybuf_mv = memoryview(self.dummybuf)
        # Initialize card
        self.init_card(baudrate)

    def init_spi(self, baudrate):
        try:
            # On RP2, machine.SPI doesn't have MASTER attr; just call init
            self.spi.init(baudrate=baudrate, phase=0, polarity=0)
        except AttributeError:
            # On other ports (ESP8266/ESP32)
            self.spi.init(baudrate=baudrate, phase=0, polarity=0)

    def init_card(self, baudrate):
```

```
        # Init CS pin
        self.cs.init(self.cs.OUT, value=1)
        # Start with slow clock
        self.init_spi(100_000)  # 100 kHz for init
        # Clock card at least 80 cycles with CS high
        for _ in range(16):
            self.spi.write(b"\xff")
        # CMD0: go to idle state
        for _ in range(5):
            if self.cmd(0, 0, 0x95) == _R1_IDLE_STATE:
                break
        else:
            raise OSError("no SD card")
        # CMD8: determine card version
        r = self.cmd(8, 0x01AA, 0x87, 4)
        if r == _R1_IDLE_STATE:
            self.init_card_v2()
        elif r == (_R1_IDLE_STATE | _R1_ILLEGAL_COMMAND):
            self.init_card_v1()
        else:
            raise OSError("couldn't determine SD card version")
        # Get number of sectors
        if self.cmd(9, 0, 0, 0, False) != 0:
            raise OSError("no response from SD card")
        csd = bytearray(16)
        self.readinto(csd)
        if csd[0] & 0xC0 == 0x40:  # CSD v2.0
            self.sectors = ((csd[8] << 8 | csd[9]) + 1) * 1024
        elif csd[0] & 0xC0 == 0x00:  # CSD v1.0
            c_size = ((csd[6] & 0x3) << 10) | (csd[7] << 2) | (csd[8] >> 6)
            c_size_mult = ((csd[9] & 0x3) << 1) | (csd[10] >> 7)
            self.sectors = (c_size + 1) * (2 ** (c_size_mult + 2))
        else:
            raise OSError("SD card CSD format not supported")
        # CMD16: set block length to 512 bytes
        if self.cmd(16, 512, 0) != 0:
            raise OSError("can't set 512 block size")
        # Switch to high speed
        self.init_spi(baudrate)

    def init_card_v1(self):
        for i in range(_CMD_TIMEOUT):
            self.cmd(55, 0, 0)
            if self.cmd(41, 0, 0) == 0:
                self.cdv = 512
                return
        raise OSError("timeout waiting for v1 card")
```

```
    def init_card_v2(self):
        for i in range(_CMD_TIMEOUT):
            time.sleep_ms(50)
            self.cmd(58, 0, 0, 4)
            self.cmd(55, 0, 0)
            if self.cmd(41, 0x40000000, 0) == 0:
                self.cmd(58, 0, 0, 4)
                self.cdv = 1
                return
        raise OSError("timeout waiting for v2 card")

    def cmd(self, cmd, arg, crc, final=0, release=True, skip1=False):
        self.cs(0)
        # Create and send command packet
        buf = self.cmdbuf
        buf[0] = 0x40 | cmd
        buf[1] = arg >> 24
        buf[2] = arg >> 16
        buf[3] = arg >> 8
        buf[4] = arg
        buf[5] = crc
        self.spi.write(buf)
        if skip1:
            self.spi.readinto(self.tokenbuf, 0xFF)
        # Wait for response (MSB of response byte clear means success)
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            if not (self.tokenbuf[0] & 0x80):
                # Read any remaining bytes (e.g., R7 response)
                for _ in range(final):
                    self.spi.write(b"\xff")
                if release:
                    self.cs(1)
                    self.spi.write(b"\xff")
                return self.tokenbuf[0]
        # Timeout
        self.cs(1)
        self.spi.write(b"\xff")
        return -1

    def readinto(self, buf):
        # Read block into buf
        self.cs(0)
        # Wait for start token 0xFE
        for i in range(_CMD_TIMEOUT):
            self.spi.readinto(self.tokenbuf, 0xFF)
            if self.tokenbuf[0] == _TOKEN_DATA:
                break
```

```
        else:
            self.cs(1)
            raise OSError("timeout waiting for response")
        # Read data
        mv = self.dummybuf_mv if len(buf) == 512 else
memoryview(self.dummybuf)[:len(buf)]
        self.spi.write_readinto(mv, buf)
        # Read 2-byte CRC
        self.spi.write(b"\xff\xff")
        self.cs(1)
        self.spi.write(b"\xff")

    def write(self, token, buf):
        # Write a block of data (token is start byte)
        self.cs(0)
        self.spi.read(1, token)
        self.spi.write(buf)
        self.spi.write(b"\xff\xff")
        # Check response
        if (self.spi.read(1, 0xFF)[0] & 0x1F) != 0x05:
            self.cs(1)
            self.spi.write(b"\xff")
            return
        # Wait for write to finish
        while self.spi.read(1, 0xFF)[0] == 0x00:
            pass
        self.cs(1)
        self.spi.write(b"\xff")

    def write_token(self, token):
        # Send a special token (like STOP_TRAN token)
        self.cs(0)
        self.spi.read(1, token)
        # Wait for write complete
        while self.spi.read(1, 0xFF)[0] == 0x00:
            pass
        self.cs(1)
        self.spi.write(b"\xff")
```

4. In Thonny, **File > New** to open a new editor tab. Paste the entire `sdcard.py` code above into it.
5. Go to **File > Save as...**, choose **Raspberry Pi Pico** as the destination (this saves to Pico's flash). Name the file `sdcard.py` and click OK. The file will be saved on Pico A.

6. *(This is exactly what Random Nerd Tutorials suggests: creating and saving the* `sdcard.py` *library on the Pico* [8] *.)*

7. **Verify SD card access (optional test):** We can run a quick test to ensure Pico A can talk to the SD card. In Thonny's editor, open another new file (or use the REPL directly) and enter:

```
from machine import SPI, Pin
import sdcard, os

# SPI0 on Pico: we use GP2=SCK, GP3=MOSI, GP4=MISO, GP5=CS
spi = SPI(0, sck=Pin(2), mosi=Pin(3), miso=Pin(4))
cs = Pin(5, Pin.OUT)
sd = sdcard.SDCard(spi, cs)        # Initialize SD card
os.mount(sd, "/sd")                # Mount at /sd
print("Files on card:", os.listdir("/sd"))
```

Save this as, say, `sd_test.py` on Pico (or just run it). Stop any running script and use the **Run** button. It should print a list of files on the SD (the scope's image files) [9]. If you see an error or empty list:

8. Check wiring and that a card is present. If the scope is writing to an actual SD card, that card should be in place. If Pico is acting as the card without an actual SD, the above code *should* still work (it initializes the card – effectively making Pico enumerate it). If "no SD card" error, the scope might not be releasing the bus. Try powering off the scope and just have Pico connected to the card (if you use a separate card).
9. Once it prints directory contents, unmount the card: `os.umount("/sd")` (or just reset the Pico). We will incorporate mounting in our main script, so it's okay to leave it mounted for now, but in code we'll mount fresh.

If all good, Pico A can read the SD. Now let's proceed to the main script for Pico A.

Step 5: Pico A Main Script – Detect and Send New Images

We'll create a MicroPython script that runs on Pico A to continually watch for new `.jpg` files and send them to Pico B over Wi-Fi.

Plan for Pico A script: - Initialize Wi-Fi and connect to your WLAN (as a station client). - Define the IP address or hostname of Pico B's server and a port number for communication. - Initialize SPI and mount the SD card at `/sd`. - Keep track of already-seen files in the `/sd` directory. - Loop forever: periodically check for new files. When a new JPEG file is detected, pause briefly (to ensure the scope finished writing the file), then open the file and transmit it to Pico B: - Establish a socket connection to Pico B's server. - Send a 4-byte length header (file size). - Stream the file data in chunks over the socket. - Close the socket and mark the file as "sent" (so we don't send it again). - Small enhancements: we'll only consider files with a ".jpg" extension for sending. We will also handle Wi-Fi reconnection if needed (if Wi-Fi drops, the script could try to reconnect).

Find Pico B's IP: To configure Pico A, we need to know where to send the data. We will set up Pico B's server *next*, but planning ahead: either assign Pico B a static IP or get it via DHCP and then update Pico A's script. The simplest is: run Pico B's script to print its IP (we'll do that in Step 6), then note that IP and put it into Pico

A's code below. (Alternatively, use mDNS or hostnames if supported, but to keep it simple we'll use a hardcoded IP address.)

For now, let's write Pico A's code (we will fill in `SERVER_IP` after we have it):

1. **Open a new file in Thonny**, and start writing Pico A's script, e.g. call it `picoA_send.py`:

```
import network, socket
from machine import SPI, Pin
import sdcard, os, time

# --- User Configurations ---
SSID = "YOUR_WIFI_SSID"
PASSWORD = "YOUR_WIFI_PASSWORD"
SERVER_IP = "x.x.x.x"      # <-- To be set: IP address of Pico B
PORT = 8000                # Port on which Pico B server listens
CHECK_INTERVAL = 2         # seconds between SD card checks
# ---------------------------

# 1. Connect to Wi-Fi
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
print("Connecting to WiFi...")
wlan.connect(SSID, PASSWORD)
# Wait for connection
max_wait = 15
while max_wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    max_wait -= 1
    print("Waiting for Wi-Fi connection...")
    time.sleep(1)

if wlan.status() != 3:
    raise RuntimeError("Wi-Fi connection failed")
print("Connected to Wi-Fi, IP address:", wlan.ifconfig()[0])

# 2. Initialize SD card interface (SPI)
spi = SPI(0, sck=Pin(2), mosi=Pin(3), miso=Pin(4))
cs = Pin(5, Pin.OUT)
sd = sdcard.SDCard(spi, cs)
os.mount(sd, "/sd")
print("SD card mounted. Initial files:", os.listdir("/sd"))

# Track files already seen
seen_files = set(os.listdir("/sd"))
```

```
print("Pico A is now monitoring for new images...")

try:
    while True:
        # Refresh file list
        current_files = set(os.listdir("/sd"))
        # Identify new files
        new_files = [f for f in current_files - seen_files if 
f.lower().endswith(".jpg")]
        if new_files:
            for fname in new_files:
                fullpath = "/sd/" + fname
                # Wait a moment to ensure file is done writing
                print(f"New image detected: {fname}, waiting for write to 
finish...")
                time.sleep(1)  # adjust delay if needed
                try:
                    # Get file size
                    fsize = os.stat(fullpath)[6]
                except Exception as e:
                    print("Error getting file size:", e)
                    fsize = 0
                if fsize == 0:
                    print("File size is 0, skipping send.")
                    seen_files.add(fname)
                    continue

                # Open socket to Pico B
                addr = socket.getaddrinfo(SERVER_IP, PORT)[0][-1]
                sock = socket.socket()
                try:
                    sock.connect(addr)
                except Exception as e:
                    print("Failed to connect to Pico B:", e)
                    sock.close()
                    break  # break out of for-loop (you could also retry)
                print(f"Connected to Pico B at {SERVER_IP}:{PORT}, sending 
{fname} ({fsize} bytes)...")
                try:
                    # Send 4-byte size (big-endian)
                    sock.send(fsize.to_bytes(4, 'big'))
                    # Send file data in chunks
                    with open(fullpath, "rb") as f:
                        while True:
                            buf = f.read(1024)
                            if not buf:
                                break
                            sock.send(buf)
```

```
                    print("File sent successfully.")
                except Exception as e:
                    print("Error during send:", e)
                sock.close()

# Optionally, mark file as sent (move or rename on SD, or delete if not needed)
                # For now, we just add to seen_files so it won't resend
                seen_files.add(fname)
        # Update seen_files for any other changes
        seen_files |= current_files
        time.sleep(CHECK_INTERVAL)
finally:
    # Cleanup on exit
    os.umount("/sd")
    spi.deinit()  # (machine.SPI might not have deinit; alternatively, just
pass)
    wlan.disconnect()
```

1. **Edit Wi-Fi credentials:** In the script above, replace `"YOUR_WIFI_SSID"` and `"YOUR_WIFI_PASSWORD"` with your Wi-Fi network's name and password. This is required for the Pico to connect to your WLAN.

2. **Leave** `SERVER_IP` **for now** – we will fill this in after setting up Pico B to get its IP address.

3. **Save the file:** Go to **File > Save as**, choose **Raspberry Pi Pico**, and name it (for example) `main.py` (so it will run automatically on boot) or `picoA_send.py` if you prefer manual running. If you name it `main.py`, the script will execute whenever Pico A boots or resets. That is useful if you want this to run headless on the device.

We will **not run this yet** – we need to set up Pico B first and get its IP.

However, a few notes on the Pico A code: - It uses `network.WLAN` to connect Wi-Fi [6] and waits until connected (status `3` means success) [10] [11], then prints the IP. - It mounts the SD card and prints the file list [9]. - It then continuously checks for new ".jpg" files. We use `os.listdir("/sd")` to get files and a `seen_files` set to track which we've already processed. - When a new file is found, it waits 1 second, then gets its size via `os.stat`. If the file size is zero (possibly if caught too early), it skips sending. - It then connects to Pico B's IP on the specified port. We send the file size as 4 bytes, then open the file and send its contents in 1024-byte chunks. - After sending, it closes the socket and marks the file as sent by adding to `seen_files` (so even if it remains on SD, we won't resend). *(In a real scenario, you might remove or rename the file on SD to prevent infinite growth, but since the scope might need the file intact, we'll leave it and just not resend.)* - The `try/finally` ensures that if the loop exits, we unmount the SD and disconnect Wi-Fi gracefully.

We will revisit Pico A to plug in the correct `SERVER_IP` once we have it from Pico B. So keep this script handy.

Step 6: Pico B Main Script – Receive Image and Save to File

Next, configure Pico B, which will run a server to accept connections from Pico A and receive the image data. Pico B is connected to your laptop via USB, and will ultimately hold the received file for transfer to PC.

Plan for Pico B script: - Connect to Wi-Fi (same SSID/PW as Pico A). - Start a TCP server socket listening on a chosen port (e.g. 8000). - When a client (Pico A) connects, accept the connection. - Receive the first 4 bytes to get the incoming file size. - Receive the file bytes in a loop until we've got the entire image. - Save the data to a JPEG file on Pico B's flash storage. - We'll name the files sequentially (to avoid overwriting if multiple images come in). For example, first image as `image_001.jpg`, next as `image_002.jpg`, etc. - Print some status to REPL (so we know on PC that something happened, and also to get Pico B's IP for configuring Pico A).

Wi-Fi Note: Ensure Pico B is also within Wi-Fi coverage. Both Picos should connect to the same network (e.g. your home router) so they can communicate. Alternatively, one Pico could be an AP and the other connect to it, but that's more complex; using your existing Wi-Fi network is easier.

Let's write Pico B's server script:

1. **Open a new file in Thonny** (make sure you select Pico **B** this time under Interpreter). Name it e.g. `picoB_receive.py` (or make it `main.py` on Pico B if you want auto-start).
2. **Write the following code:**

```
import network, socket
import time, os

# --- User Config ---
SSID = "YOUR_WIFI_SSID"
PASSWORD = "YOUR_WIFI_PASSWORD"
PORT = 8000
# -------------------

# Connect to Wi-Fi
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
print("Connecting Pico B to WiFi...")
wlan.connect(SSID, PASSWORD)
for i in range(15):  # wait up to 15 seconds
    if wlan.isconnected():
        break
    print(".", end="")
    time.sleep(1)
print()
if not wlan.isconnected():
    raise RuntimeError("Failed to connect to Wi-Fi")
ip = wlan.ifconfig()[0]
```

```
print("Pico B connected to Wi-Fi, IP:", ip)

# Set up TCP server
addr_info = socket.getaddrinfo("0.0.0.0", PORT)[0][-1]
server_socket = socket.socket()
server_socket.bind(addr_info)
server_socket.listen(1)
print("Server listening on {}:{}".format(ip, PORT))

# Counters for saved images
img_count = 0

try:
    while True:
        print("Waiting for incoming connection...")
        conn, addr = server_socket.accept()
        print("Accepted connection from", addr)
        # Receive 4-byte length
        length_data = b''
        try:
            # Make sure we get 4 bytes for size
            while len(length_data) < 4:
                chunk = conn.recv(4 - len(length_data))
                if not chunk:
                    break
                length_data += chunk
        except Exception as e:
            print("Error receiving length:", e)
            conn.close()
            continue
        if len(length_data) != 4:
            print("Invalid length header. Connection closed.")
            conn.close()
            continue
        file_size = int.from_bytes(length_data, 'big')
        print(f"Receiving image of {file_size} bytes...")

        # Generate filename
        img_count += 1
        filename = "image_{:03d}.jpg".format(img_count)
        try:
            f = open(filename, "wb")
        except Exception as e:
            print("Failed to open file for writing:", e)
            conn.close()
            continue

        bytes_received = 0
```

```
        try:
            # Receive the file data
            while bytes_received < file_size:
                data = conn.recv(1024)
                if not data:
                    break  # connection closed unexpectedly
                f.write(data)
                bytes_received += len(data)
            f.close()
        except Exception as e:
            print("Error receiving file data:", e)
            f.close()
        conn.close()
        print(f"File received ({bytes_received}/{file_size} bytes), saved as
{filename}")
        # Optionally, if bytes_received != file_size, you could mark error.
```

1. **Edit Wi-Fi credentials:** Put the same SSID and PASSWORD as used in Pico A's script.

2. **Save this file** to Pico B (if you want it to run on boot, name it `main.py`; otherwise save as `picoB_receive.py` and you'll run it manually).

3. **Run Pico B's server script:** Click the Run button in Thonny. The script will output status in the Thonny Shell. You should see something like:

   ```
   Connecting Pico B to WiFi...
   ... (dots) ...
   Pico B connected to Wi-Fi, IP: 192.168.1.100
   Server listening on 192.168.1.100:8000
   Waiting for incoming connection...
   ```

   This means Pico B is ready. **Copy the IP address** shown (e.g. `192.168.1.100`). This is what we will put into Pico A's `SERVER_IP`. *(If the IP printed as `192.168.x.y`, that's correct. If it prints `0.0.0.0` or similar, it didn't connect – check the SSID/PW or signal.)*

Leave Pico B's script running, waiting for a connection.

Step 7: Finalize Pico A Script with Server IP and Test the System

Now go back to Pico A's script in Thonny. Update the `SERVER_IP` variable to the IP address you got from Pico B.

Example: if Pico B's IP was `192.168.1.100`, then:

```
SERVER_IP = "192.168.1.100"
```

Make sure the port number in Pico A script matches the Pico B script's port (we used 8000 for both).

Save the changes to Pico A (if it was `main.py`, it will auto-run on reset; if not, run it manually after saving).

Run Pico A's script: With Pico B's server still running and waiting, execute Pico A's script. If you saved it as `main.py`, you can simply press the reset button on Pico A (it will start automatically). If not, click Run in Thonny for Pico A's file.

Watch Pico A's output in Thonny: It should connect to Wi-Fi and print its IP, mount the SD, list initial files, then start monitoring:

```
Connecting to WiFi...
Connected to Wi-Fi, IP address: 192.168.1.101
SD card mounted. Initial files: ['image001.jpg', 'image002.jpg']
Pico A is now monitoring for new images...
```

(The file list will vary; that's just an example.)

Now, **test the image capture on the scope**: Use the Dimension EasyCheck V2 to capture a new fiber endface image. When it saves, Pico A should detect the new file on the SD card.

For instance, if a new file `image003.jpg` is created by the scope, Pico A's log will show:

```
New image detected: image003.jpg, waiting for write to finish...
Connected to Pico B at 192.168.1.100:8000, sending image003.jpg (345678
bytes)...
File sent successfully.
```

Over on Pico B's Thonny output, you should see:

```
Accepted connection from ('192.168.1.101', 12345)
Receiving image of 345678 bytes...
File received (345678/345678 bytes), saved as image_001.jpg
Waiting for incoming connection...
```

This indicates the transfer was successful. Pico B saved the file as `image_001.jpg` on its flash.

If you capture another image on the scope, the process repeats: Pico A sends it, Pico B saves as `image_002.jpg`, and so on. Both Picos should handle multiple transfers in sequence.

Troubleshooting tips: - If Pico A cannot connect to Pico B, check that `SERVER_IP` is correct and that both are on the same network. Ensure your Wi-Fi signal is okay. You can also ping the Pico B's IP from your PC to see if it's reachable. - If file transfer hangs or the size doesn't match, ensure the Wi-Fi is stable. Large images might take a couple of seconds; our code doesn't print progress for every chunk (to keep it simple), but you can add prints if needed. - If the scope writes a file but Pico A doesn't detect it: maybe the scope hasn't closed the file yet or uses a different extension. Verify the scope's saved filenames and extension (likely `.jpg` or `.jpeg`). Adjust `endswith(".jpg")` if needed (e.g., `.jpeg`). Also, ensure `CHECK_INTERVAL` isn't too long or too short; 2 seconds is a balance. You can make it 1 second for faster response. - If you encounter crashes due to memory (unlikely for small files, but if images are large and Python can't handle listing a huge directory), consider processing one file at a time or freeing memory by removing sent files.

Step 8: Transfer the Image from Pico B to Your Computer

Once Pico B has the image file(s), you need to copy them to your Windows PC. We'll do this using Thonny's file manager:

1. In Thonny, stop Pico B's script (if it's still running, press the **Stop** button). This will close the server loop (and also free the file so we can access it).
2. Go to **View > Files** in Thonny. This opens a file browser pane. You should see two sections: "This computer" (your PC) and "Micropython device" (the Pico B).
3. Click the **Micropython device** section. It will list the files stored on Pico B's flash. You should see `image_001.jpg`, `image_002.jpg`, etc. (along with any .py files you saved).
4. **Download the file:** Right-click on `image_001.jpg` and choose **Save to PC** (or a similarly phrased option). Thonny will prompt where to save – navigate to your desired folder on your PC (for example, create a folder `PicoImages` in Documents) and save the file. Repeat for any additional images (image_002.jpg, etc.).
5. *Alternative:* You can also drag and drop from the device pane to a folder in Windows Explorer (in some Thonny versions). Or use the **"≡" menu (three dots)** in the file pane – there is an option to **Download** the selected file [12].
6. Verify on your PC that the JPEG file opens (use an image viewer). It should be the captured endface image from the scope.

(If you prefer command-line: you could use `mpremote fs cp :image_001.jpg .` *to copy via MicroPython's command-line tool, or use PuTTY and* `ampy`*. But the Thonny GUI is simplest.)*

1. Once copied, you can delete the image from Pico B's flash to free space: right-click the file in Thonny's device pane, choose **Remove**. (The Pico has limited flash, ~1.3MB for file storage by default, so it's wise to offload images promptly especially if they are large.)

Congratulations! You have successfully captured an image on the EasyCheck V2, had Pico A intercept it, sent it wirelessly to Pico B, and saved it on your computer.

(Optional) Tips and Enhancements

- **Autostart on Power-Up:** If you name the scripts `main.py` on each Pico, they will run automatically whenever the Pico is powered. This way, the whole chain can work without manual intervention (as long as the Wi-Fi is available and the scope and Picos are powered). You could power Pico A from the

scope's 12V (with a regulator) or another source so it's always on when the scope is on. Pico B can be powered by the PC's USB or any USB power.
- **Indicator LEDs:** You might use Pico's onboard LED to indicate activity (e.g., blink when sending or receiving). MicroPython on Pico W lets you control the LED via `Pin("LED", Pin.OUT)`.
- **File Name Handling:** We kept Pico B's file names simple (`image_001.jpg` etc.). If you want the original filename, you could modify Pico A to send the name as well (e.g., send a short header with name length and name). This would complicate the protocol slightly. The current approach reliably avoids collisions by numbering.
- **Deleting files on SD:** If the SD card on the scope fills up over time, you might want Pico A to remove images after sending. Be cautious: if the scope expects the file to remain (for example, it might not care, but if the user wants to review on the device, deleting might be undesirable). If safe, you can call `os.remove("/sd/"+fname)` after sending. Ensure the scope isn't accessing it. Given the scope likely doesn't track old images beyond saving them, deleting after transfer could be okay.
- **Error Recovery:** Our scripts have basic error handling (they won't crash on common issues), but if Wi-Fi drops or the scope writes many files quickly, you may need to add more robust logic (e.g., if Pico A can't connect to B, retry after a delay; if file transfer fails mid-way, maybe retry). You can tailor this to your needs.
- **Security:** This transfer is unencrypted on the local network. If that's a concern, consider having them on an isolated network or implement a simple encryption in code. Given it's just images on presumably a lab network, this is probably fine.

Throughout this guide, we built on research and known solutions for MicroPython. We confirmed that **using SPI mode for SD card access is the viable approach on Pico** [3], and we used a community-provided MicroPython SD card driver to mount the filesystem [9]. We also leveraged the Pico W's Wi-Fi capabilities to link two devices on the same network, as demonstrated in similar projects (one Pico can run a server while another acts as client) [5]. By following the meticulous steps above, you now have a working system to wireless transfer images from the EasyCheck V2's SD slot to your PC, using two Pico microcontrollers and MicroPython.

Sources:

- MicroPython SD card SPI usage and driver setup [3] [9]
- StackExchange discussion on SD card interception with microcontroller (wiring via resistors) [4]
- Raspberry Pi Pico W wireless communication (client/server on same Wi-Fi) [5]
- MicroPython Wi-Fi connection example (Pico W connecting to network) [6] [7]

---

[1] [2] arduino - Emulating an SD-Card with a microcontroller (or Raspberry Pi) - Electrical Engineering Stack Exchange
https://electronics.stackexchange.com/questions/390548/emulating-an-sd-card-with-a-microcontroller-or-raspberry-pi

[3] [8] [9] Raspberry Pi Pico: MicroSD Card Guide with Datalogging (MicroPython) | Random Nerd Tutorials
https://randomnerdtutorials.com/raspberry-pi-pico-microsd-card-micropython/

[4] microcontroller - Is it possible to emulate an SD as a slave? - Electrical Engineering Stack Exchange
https://electronics.stackexchange.com/questions/218245/is-it-possible-to-emulate-an-sd-as-a-slave

[5] Wireless Communication Between Two Raspberry Pi Pico W Boards
https://thepihut.com/blogs/raspberry-pi-tutorials/wireless-communication-between-two-raspberry-pi-pico-w-boards?srsltid=AfmBOooOqC7wWInN5iuNkdiTTIGZv1crY2lRdPj7nT0CYqZd0n7Jy2HD

[6] [7] [10] [11] Raspberry Pi Pico W: Getting Started with Wi-Fi (MicroPython) | Random Nerd Tutorials
https://randomnerdtutorials.com/raspberry-pi-pico-w-wi-fi-micropython/

[12] Save files off Pico W in Windows - Raspberry Pi Forums
https://forums.raspberrypi.com/viewtopic.php?t=342876