Integrating Flipper Zero as a Serial RFID Peripheral

For BeagleY-AI and Embedded Linux

Date: December 3, 2025
Guide has been tested on:

- **Target:** BeagleY-AI (Debian 12 Bookworm)
- **Peripheral:** Flipper Zero (Firmware 1.4.2 or later)
- **Host OS:** Linux (for cross-compilation)

1. Introduction

Standard RFID breakout boards (like the RC522) are common in embedded systems, but they often require complex SPI/I2C wiring and are limited to specific frequencies. The **Flipper Zero** is a multi-tool capable of reading Low-Frequency (125kHz), High-Frequency (NFC), and UHF tags. This guide explains how to offload RF processing to the Flipper Zero and transmit captured Unique IDs (UIDs) to a host Linux computer (BeagleY-AI) via a **UART** connection. This effectively turns the Flipper Zero into a "smart" peripheral for your embedded system.

This document guides the user through:

1. Wiring the Flipper Zero to the BeagleY-AI GPIO header.
2. Configuring the Linux Device Tree to enable the specific UART port.
3. Writing a custom Flipper application (.fap) to read tags and send data.
4. Writing a C program on the BeagleY-AI to receive and parse the data.

2. Hardware Setup

2.1 Wiring Diagram

UART (Universal Asynchronous Receiver-Transmitter) requires a **cross-over** connection. The Transmit (TX) pin of one device connects to the Receive (RX) pin of the other.

WARNING: Both devices operate on **3.3V Logic**. Do **NOT** connect the 5V pin from the Flipper to the BeagleY-AI GPIO, or you risk damaging the processor.

Connections:

Flipper Zero Pin	BeagleY-AI Header Pin	Function
Pin 8 or 18 (GND)	Pin 6 (GND)	Common Ground (Essential)
Pin 13 (TX)	Pin 10 (GPIO 15 / RX)	Flipper sends data to Beagle
Pin 14 (RX)	Pin 8 (GPIO 14 / TX)	Beagle sends data to Flipper

3. Configuring the BeagleY-AI (Target)

By default, the specific header pins (8 and 10) on the BeagleY-AI might not be configured as a serial port in the OS. We must load a **Device Tree Overlay** to enable them.

3.1 Enabling the UART Overlay

1. Open a terminal on your BeagleY-AI or SSH to it.
2. Edit the boot configuration file:

```
(byai)$ sudo nano /boot/firmware/extlinux/extlinux.conf
```

3. Locate the fdtoverlays line. You need to append the specific overlay for UART on the main header. Add the following path to the list:

```
/overlays/k3-am67a-beagley-ai-uart-ttyama0.dtbo
```

4. Save and exit (Ctrl+O, Enter, Ctrl+X), then reboot:

```
(byai)$ sudo reboot
```

3.2 Verification

After the board reboots, verify that the serial device exists. On the BeagleY-AI, this port is mapped to /dev/ttyAMA0.

```
(byai)$ ls -l /dev/ttyAMA0
crw-rw---- 1 root dialout 204, 64 Apr 11 10:00 /dev/ttyAMA0
```

If you see the file details, the hardware is ready.

4. Flipper Zero Application (Sender)

We need a custom app on the Flipper to handle the RFID hardware and send the data. Standard Flipper apps do not stream data over UART.
Prerequisite: Install the **uFlipper Build Tool (ufbt)** on your host PC to compile the code.

```
(host)$ pipx install ufbt
```

4.1 The Code Logic

The Flipper firmware is multi-threaded. **Crucial:** You cannot perform heavy operations (like UART transmission) inside the RFID callback, or the device will crash (furi_check failed). We use a **Message Queue** pattern:

1. **Callback:** Scans tag -> puts data in Queue.
2. **Main Loop:** Reads Queue -> Sends data over UART.

4.2 Code Listing: rfid_uart.c

Create a folder named rfid_uart and add this code.

```
#include <furi.h>
#include <furi_hal.h>
#include <lfrfid/lfrfid_worker.h>
#include <lfrfid/protocols/lfrfid_protocols.h>

#define UART_CH FuriHalSerialIdUsart
#define BAUDRATE 9600

// Event Queue logic to prevent crashes
typedef struct { char rfid_data[32]; } AppEvent;

void send_uart(FuriHalSerialHandle* handle, const char* str) {
    furi_hal_serial_tx(handle, (uint8_t*)str, strlen(str));
    furi_hal_serial_tx(handle, (uint8_t*)"\r\n", 2); // Newline is
important!
}

static void read_callback(LFRFIDWorkerReadResult result, ProtocolId
protocol, void* ctx) {
    // 1. Get Data
    // 2. Format to Hex String
    // 3. furi_message_queue_put(queue, &event, 0);
```

```
    // (Full implementation in provided sample zip)
}

int32_t rfid_uart_app(void* p) {
    // Init UART
    FuriHalSerialHandle* serial = furi_hal_serial_control_acquire(UART_CH);
    furi_hal_serial_init(serial, BAUDRATE);

    // Init RFID Worker
    ProtocolDict* dict = protocol_dict_alloc(lfrfid_protocols,
LFRFIDProtocolMax);
    LFRFIDWorker* worker = lfrfid_worker_alloc(dict);
    lfrfid_worker_start_thread(worker);
    lfrfid_worker_read_start(worker, LFRFIDWorkerReadTypeAuto,
read_callback, app);

    // Main Loop
    AppEvent event;
    while(furi_message_queue_get(queue, &event, FuriWaitForever) ==
FuriStatusOk) {
        send_uart(serial, event.rfid_data);
    }

    // Cleanup (Order is critical to avoid crashes!)
    lfrfid_worker_stop(worker);
    lfrfid_worker_stop_thread(worker); // STOP THREAD BEFORE FREEING
    lfrfid_worker_free(worker);
    // ... cleanup serial ...
    return 0;
}
```

4.3 Building and Installing

Connect your Flipper Zero via USB and run:

```
(host)$ ufbt launch
```

This compiles the app, uploads it, and starts it automatically.

5. BeagleY-AI C Client (Receiver)

Now we write the C program on the BeagleY-AI to listen for the data.

5.1 Configuring Serial (termios)

To read raw data, we must disable Linux's default terminal processing (Canonical mode, Echo).

```
int uart_fd = open("/dev/ttyAMA0", O_RDWR | O_NOCTTY | O_NDELAY);

struct termios options;
tcgetattr(uart_fd, &options);

// Set Baud Rate
cfsetispeed(&options, B9600);
cfsetospeed(&options, B9600);

// Raw Input Mode (Disable Echo and Canonical)
options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);

// Apply Settings
tcsetattr(uart_fd, TCSANOW, &options);
```

5.2 Reading Data

```
We use a non-blocking read() loop.
char buffer[64];
int len = read(uart_fd, buffer, sizeof(buffer)-1);

if (len > 0) {
    buffer[len] = '\0'; // Null terminate
    // Strip newlines sent by Flipper
    buffer[strcspn(buffer, "\r\n")] = 0;
    printf("RFID Tag Detected: %s\n", buffer);
}
```

6. Troubleshooting

6.1 "Input/output error" on BeagleY-AI

Problem: When running the C code, perror prints Input/output error.
Cause: The device file /dev/ttyAMA0 exists in the filesystem, but the pins are not active in the kernel Device Tree.
Solution: Re-check **Section 3.1**. Ensure the .dtbo overlay is added to extlinux.conf and the board has been rebooted.

6.2 Flipper Zero Crashes on Exit

Problem: The app works, but when you press "Back" to exit, the Flipper crashes with furi_check failed.
Cause: You freed the LFRFIDWorker memory while the background thread was still running.
Solution: You must call lfrfid_worker_stop_thread(worker) **before** calling lfrfid_worker_free(worker).

6.3 Garbage Data Received

Problem: The Beagle receives ???? or random symbols instead of the Hex ID.
Cause: Baud rate mismatch.
Solution: Ensure both rfid_uart.c (Flipper) and your Beagle C code are set to exactly **9600 baud**. Also, verify the Ground connection is tight.

7. Conclusion

You have now successfully integrated a complex RF analysis tool into your embedded Linux system using standard serial protocols. This architecture allows the BeagleY-AI to offload high-complexity tasks (RF decoding) to specialized peripherals while maintaining centralized control.