Using ADC on SPI (Serial Peripheral Interface) Guide for BeagleY-AI

by Matthew Stewart
Sept. 28, 2025

Guide has been tested on

BeagleY-AI (Target):	**Debian 13**
PC OS (host):	**Debian 13**

This document guides the user through

1. Setting up the hardware SPI

Table of Contents

1. SPI..2

Formatting

1. Commands for the host Linux's console are show as:
   `(host)$ echo "Hello PC world!"`
2. Commands for the target (BeagleY-AI) Linux's console are shown as:
   `(byai)$ echo "Hello embedded world!"`

Revision History

- Sept 28, 2025: Using ADC on SPI with Beagle-Y AI. Inspired by SPI guide by Dr. Brian Fraser

1. SPI

The ADC connected via the Serial Peripheral Interface (SPI). It is a four wire protocol:

- Serial Clock (`SCLK`): This is the clock that controls when data is transmitted.
- Master Out, Slave In (`MOSI`): Transmits data from the CPU to the ADC
- Master In, Slave out (`MISO`): Transmits data from the device back to the CPU.
- Chip Enable (`CE`): Also called chip select, or device enable. This allows multiple devices to share the same SPI bus.

1. Connect the wires from the BYAI header to the breadboard. Pay attention to the data flow direction and check the wiring before powering up. BYAI documentation will show you the header pinout and the ADC datasheet will show you the ADC pintout.
2. Check to make sure the SPI can be seen on the BYAI.
   - `(byai)$ ls -l /dev/spidev*`
   - This will show you the SPI busses.
3. Loopback test (optional but recommended)
   - On the BYAI disconnect the MOSI and MISO from the ADC connect the MISO to the MOSI. This will have any data going out coming right back.
   - `(byai)$ sudo spidev_test -D /dev/spidev0.0 -s 250000 -v -S 5 -p "Talkin on the SPI."`
   - You should get an output that looks something like this:

     ```
     spi mode: 0x0
     bits per word: 8
     max speed: 250000 Hz (250 kHz)
     TX | 54 61 6C 6B 69 6E 20 6F 6E 20 74 68 65 20 53 50 49 2E __ __ __ __ __ __ __ __ __ __ __ __ __ __  |Talkin on the SPI.|
     RX | 54 61 6C 6B 69 6E 20 6F 6E 20 74 68 65 20 53 50 49 2E __ __ __ __ __ __ __ __ __ __ __ __ __ __  |Talkin on the SPI.|
     ```
4. Now we’re going to start building up a .c file to access the SPI. Starting with includes:
   - ```
     #include <stdio.h>
     #include <stdint.h>
     #include <unistd.h>
     #include <fcntl.h>
     #include <sys/ioctl.h>
     #include <linux/spi/spidev.h>
     ```
5. The we create a read_ch function to read one channel from the ADC
   - `static int read_ch(int fd, int ch, uint32_t speed_hz) {`
     ```

- `fd` is the file descriptor for the SPI device
- `ch` is the channel number on the ADC
- `speed_hz` SPI clock speed
- `tx` this is our request message to the ADC
- `rx` this is our receive buffer

6. We then have a structure that has all of the data that we want to send in one SPI transfer. Look at the ADC datasheet and see if ou can understand this line
   - ```
     uint8_t tx[3] = { (uint8_t)(0x06 | ((ch & 0x04) >> 2)),
                          (uint8_t)((ch & 0x03) << 6),
                          0x00 };
     ```
7. A buffer for the received data
   - ```
     uint8_t rx[3] = { 0 };
     ```
8. Now we put all of this together in a structure to get ready to send
   - ```
     struct spi_ioc_transfer tr = {
         .tx_buf = (unsigned long)tx,
         .rx_buf = (unsigned long)rx,
         .len = 3,
         .speed_hz = speed_hz,
         .bits_per_word = 8,
         .cs_change = 0
     };
     ```
9. This is the line where we actually ask the kernel to send the SPI message and get the reply
   - ```
     if (ioctl(fd, SPI_IOC_MESSAGE(1), &tr) < 1) return -1;
     ```
10. And now we actually send back the 12 bit data from this function.
    - ```
      return ((rx[1] & 0x0F) << 8) | rx[2];  // 12-bit result
      ```

    ```
    }
    ```
11. The following is a main function to set things up and read from each of the channels.
12. Point to the SPI 0 device and set some global variables
    - ```
      int main(void) {
          const char* dev = "/dev/spidev0.0";
          uint8_t mode = 0;       // SPI mode 0
          uint8_t bits = 8;
          uint32_t speed = 250000;
      ```
13. Open the device (treated like a file)
    - ```
      int fd = open(dev, O_RDWR);
      ```
14. Send settings to the kernel
    - ```
      if (fd < 0) { perror("open"); return 1; }
       if (ioctl(fd, SPI_IOC_WR_MODE, &mode) == -1) { perror("mode"); return
      1; }
      ```

```
    if (ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits) == -1) { perror("bpw");
return 1; }
    if (ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed) == -1)
{ perror("speed"); return 1; }
```

15. Get ADC data from channels 0 and 1
16. ```
    int ch0 = read_ch(fd, 0, speed);
    int ch1 = read_ch(fd, 1, speed);
    ```
17. Print them to standard out.
    - ```
      printf("CH0=%d  CH1=%d\n", ch0, ch1);
      ```
18. Close the file.
    - ```
      close(fd);
       return 0;
      }
      ```
19. Put that together into a .c file and you should be able to compile this program to get data from the ADC. (remember to cross compile)
20. To run this program you will need to use the sudo command. This program is accessing the kernel directly and the OS doesn't want just anyone doing that.
      ```