# ADS8681 ADC with Beaglebone Black in C

You need to enable the pins with uBoot overlays:

You need to edit /boot/uEnv.txt:

Change these two lines: (Change as per SPI that you are used)

#uboot\_overlay\_addr5=<file5>.dtbo

to

uboot\_overlay\_addr5=/lib/firmware/BB-SPIDEV1-00A0.dtbo

and then reboot.

In Below Example we use SPI1 (Tested Code in BBB) debian OS. Work with Linux.

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/ioctl.h>

#include <linux/spi/spidev.h>

#define SPI\_DEVICE "/dev/spidev1.0"

#define SPI\_MODE SPI\_MODE\_0

#define SPI\_BITS\_PER\_WORD 8

#define SPI\_SPEED\_HZ 1000000 // 1MHz

#define GPIO\_PATH "/sys/class/gpio"

#define SPI1\_CS2 "48" // Change this to the GPIO pin you want to control

int spi\_fd;

void spi\_init() {

spi\_fd = open(SPI\_DEVICE, O\_RDWR);

if (spi\_fd == -1) {

perror("Error opening SPI device");

exit(EXIT\_FAILURE);

}

// Set SPI mode

uint8\_t mode = SPI\_MODE;

if (ioctl(spi\_fd, SPI\_IOC\_WR\_MODE, &mode) == -1) {

perror("Error setting SPI mode");

close(spi\_fd);

exit(EXIT\_FAILURE);

}

// Set bits per word

uint8\_t bitsperword = SPI\_BITS\_PER\_WORD;

if (ioctl(spi\_fd, SPI\_IOC\_WR\_BITS\_PER\_WORD, &bitsperword) == -1) {

perror("Error setting SPI bits per word");

close(spi\_fd);

exit(EXIT\_FAILURE);

}

// Set SPI speed

uint32\_t speed = SPI\_SPEED\_HZ;

if (ioctl(spi\_fd, SPI\_IOC\_WR\_MAX\_SPEED\_HZ, &speed) == -1) {

perror("Error setting SPI speed");

close(spi\_fd);

exit(EXIT\_FAILURE);

}

}

void spi\_transmit(uint16\_t data) {

uint8\_t tx\_buf[2];

// Split 16-bit data into two 8-bit values

tx\_buf[0] = (data >> 8) & 0xFF;

tx\_buf[1] = data & 0xFF;

struct spi\_ioc\_transfer spi\_msg = {

.tx\_buf = (unsigned long)tx\_buf,

.len = sizeof(tx\_buf),

};

if (ioctl(spi\_fd, SPI\_IOC\_MESSAGE(1), &spi\_msg) == -1) {

perror("Error transmitting data over SPI");

close(spi\_fd);

exit(EXIT\_FAILURE);

}

}

uint16\_t spi\_receive() {

uint8\_t rx\_buf[2];

uint16\_t received\_data;

struct spi\_ioc\_transfer spi\_msg = {

.rx\_buf = (unsigned long)rx\_buf,

.len = sizeof(rx\_buf),

};

if (ioctl(spi\_fd, SPI\_IOC\_MESSAGE(1), &spi\_msg) == -1) {

perror("Error receiving data over SPI");

close(spi\_fd);

exit(EXIT\_FAILURE);

}

// Combine two 8-bit values into a single 16-bit value

received\_data = (rx\_buf[0] << 8) | rx\_buf[1];

return received\_data;

}

// Function to export a GPIO pin

void export\_gpio(const char \*gpio\_pin) {

FILE \*export\_file = fopen(GPIO\_PATH "/export", "w");

if (export\_file == NULL) {

perror("Error exporting GPIO");

exit(EXIT\_FAILURE);

}

fprintf(export\_file, "%s", gpio\_pin);

fclose(export\_file);

}

// Function to set the direction of a GPIO pin

void set\_gpio\_direction(const char \*gpio\_pin, const char \*direction) {

char path[255];

sprintf(path, GPIO\_PATH "/gpio%s/direction", gpio\_pin);

FILE \*direction\_file = fopen(path, "w");

if (direction\_file == NULL) {

perror("Error setting GPIO direction");

exit(EXIT\_FAILURE);

}

fprintf(direction\_file, "%s", direction);

fclose(direction\_file);

}

// Function to set a GPIO pin high

void set\_gpio\_high(const char \*gpio\_pin) {

char path[255];

sprintf(path, GPIO\_PATH "/gpio%s/value", gpio\_pin);

FILE \*value\_file = fopen(path, "w");

if (value\_file == NULL) {

perror("Error setting GPIO high");

exit(EXIT\_FAILURE);

}

fprintf(value\_file, "1");

fclose(value\_file);

}

// Function to set a GPIO pin low

void set\_gpio\_low(const char \*gpio\_pin) {

char path[255];

sprintf(path, GPIO\_PATH "/gpio%s/value", gpio\_pin);

FILE \*value\_file = fopen(path, "w");

if (value\_file == NULL) {

perror("Error setting GPIO low");

exit(EXIT\_FAILURE);

}

fprintf(value\_file, "0");

fclose(value\_file);

}

// Function to unexport a GPIO pin

void unexport\_gpio(const char \*gpio\_pin) {

FILE \*unexport\_file = fopen(GPIO\_PATH "/unexport", "w");

if (unexport\_file == NULL) {

perror("Error unexporting GPIO");

exit(EXIT\_FAILURE);

}

fprintf(unexport\_file, "%s", gpio\_pin);

fclose(unexport\_file);

}

int main() {

spi\_init();

float Voltage;

float divide\_Fact = 32768; // (2^15) half of 65536

// Export the GPIO pin

export\_gpio(SPI1\_CS2);

// Set the GPIO direction to out

set\_gpio\_direction(SPI1\_CS2, "out");

// uint16\_t data\_to\_send = 0xf0f0;

// Transmit data

// spi\_transmit(data\_to\_send);

// Example: Transmit and receive 16-bit data continuously

while (1) {

// Set GPIO low

set\_gpio\_low(SPI1\_CS2);

// Receive data

uint16\_t received\_data = spi\_receive();

set\_gpio\_high(SPI1\_CS2);

usleep(500000); // Sleep for 500ms (0.5 seconds)

Voltage = (received\_data - divide\_Fact) \* 0.000375;

printf("Calculated Voltage: %f \n", Voltage);

usleep(1000000); // Sleep for 100ms (adjust as needed)

}

// Unexport the GPIO pin

unexport\_gpio(SPI1\_CS2);

close(spi\_fd);

return 0;

}