Serial Interfacing Between The DE0 Nano and The ESP8266 WiFi Module

This tutorial will go over how to use the ESP8266 as a WiFi hotspot (access point) and act as a server, and how to communicate with the DE0 Nano. For our project, the user would use a WiFi-enabled device such as their cellphone to communicate with the ESP8266 which would then forward the user's instructions to the DE0 Nano.

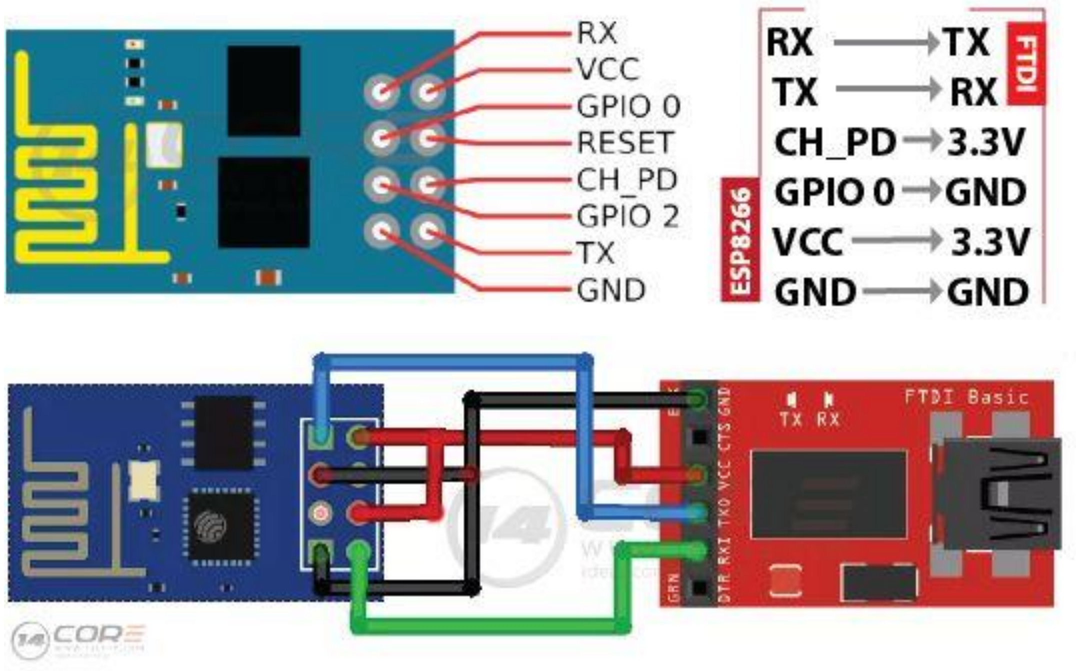
What You Will Need

- Altera DE0 Nano
- ESP8266 WiFi Module
- FTDI232 USB to Serial Module
- Wires
- Sockets
- A computer
- NIOS II IDE

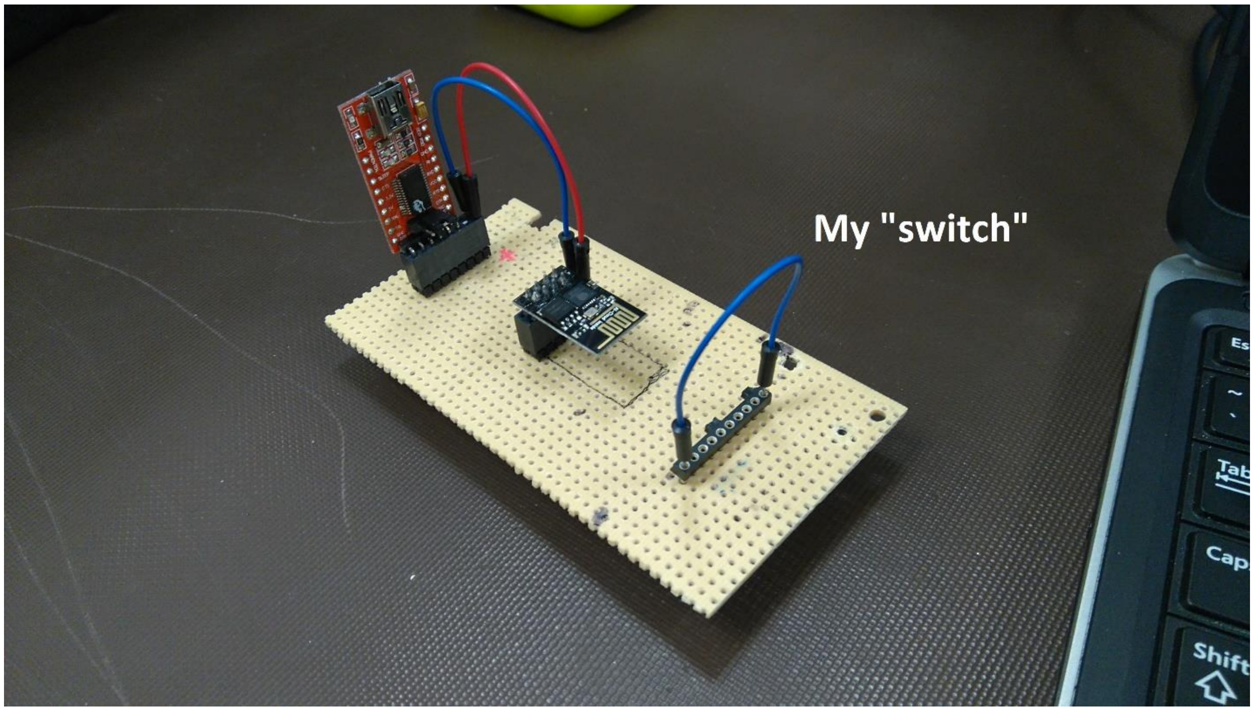
There is a 1MB limit to the appnotes submission, so refer to this repository for complete sample project: https://github.com/heiga/ESP8266_AppNotes

Setup Instructions

1) Connect the FTDI232 to the ESP8266.
   a. Follow the schematic shown below but **DO NOT connect GPIO_0 to GND.** Put a switch in between. This is for programming purposes. When programming the ESP8266, GPIO_0 must be grounded, then to keep what is programmed in flash memory, GPIO_0 must be disconnected from ground before disconnecting power. CH_PD **MUST** be connected to VCC (3.3V) at all times.
   b. The generic ESP8266 has a standardized pinout, but make sure you are using the proper pins on the FTDI232. The one we used has an inverted pinout compared to the one pictured below.



https://s-media-cache-ak0.pinimg.com/736x/68/85/e8/6885e849eafc4e52b064327ca906e1d4.jpg



I used a simple socket and wire instead as a "switch"

2) Install and configure the Arduino IDE on your computer. Here is a good guide: http://www.arduinesp.com/getting-started
3) Write a sketch for your ESP8266. There is one we used for the IO demo provided in the zip (esperio.ino). Uploading is the exact same as if you were uploading to an Arduino.
4) Open Quartus and then Qsys and add a UART (RS-232 Serial Port) to your project. This will connect to the ESP8266 via 2 GPIO connections. Refer to the provided project for how to connect it.
5) Connect a socket for the ESP8266 to your DE0 Nano. Make sure you are connecting the Nano's RX to the ESP8266's TX, and the Nano's TX to the ESP8266's RX. Using a socket allows you to pull the ESP8266 out of your DE0 system for reprogramming.

Notes:

- **ENSURE** that the ESP8266 is connected to 3.3V and not 5V. The FTDI232 can output either 5V or 3.3V, so if you plug the ESP8266 into 5V, it could fry.
- **HAVE** external power ready for the ESP8266. You **PROBABLY SHOULDN'T** power it with the 3.3V rail from the DE0 as it can only provide up to 200mA and the ESP8266 can draw up to 300mA. For our robot, we had to get a separate DCDC regulator for the ESP8266.
    - When programming, you **can** power the ESP8266 with only the FTDI232 outputting 3.3V
    - When testing however (ie performing WiFi tasks), you **must** power it externally as the FTDI232 only outputs 50mA in 3.3V mode.
- **To access the ESP8266's network**, access it like any other network. If you have mobile data enabled, disable it as your device will try to access any URL you enter via that first because the ESP8266's network is not connected to the internet.
- **You can set up the ESP8266 to connect to the real Internet** but we disqualified this option for 2 reasons. First, because it would require us to buy a domain name so our project would not be demonstrable for long after we graduate. Second, it would rely on the university's WiFi network never changing its requirements.

ESP8266 to DE0 Nano Communication

If you read the ioesper.ino file provided, you will see how to set up the ESP8266 as a hotspot and server, and that sending information to the DE0 Nano is simply a 1 line task.

Just use **Serial.print("n")** where **n** is the character you wish to send. These are ASCII characters which are received as their integer value on the DE0 side.

On the DE0 Side:

In the main C file you will need to initialize a queue and task (if the wifi stuff is not going in another task).

All this code is in the included files

// Instantiation of the queue (in the main C file)

```
#define WIFI_PACKAGE_BUFFER_LENGTH     1
OS_EVENT* wifiPackageQueue;
void* wifiPackageBuffer[WIFI_PACKAGE_BUFFER_LENGTH];
```

// Initializations in the main C file

```
    if(OSTaskCreateExt(wifi_test,
                        NULL,
                        (void *)&wifi_test_stk[TASK_STACKSIZE-1],
                        WIFI_TEST_PRIORITY,
                        WIFI_TEST_PRIORITY,
                        wifi_test_stk,
                        TASK_STACKSIZE,
                        NULL,
                        0))
      {
            printf("Wifi task creation failure\n");
      }

    if(alt_ic_isr_register(WIFI_UART_IRQ_INTERRUPT_CONTROLLER_ID,
                                       WIFI_UART_IRQ,
                                       &wifi_uart_interrupt,
                                       NULL,
                                       NULL))
      {
        printf("wifi interrupt failed\n");
      }

    wifiPackageQueue = OSQCreate(wifiPackageBuffer,
WIFI_PACKAGE_BUFFER_LENGTH);
```

// Stuff in the wifi task

```
void wifi_test(void* pdata){
        uint8_t err;

      uint8_t wifiReceive;

        while (1){
                //do stuff
              // OSSemPend(BUTTON_SEM, 0, &err);
              printf("Hello from wifi\n");

              IOWR_ALTERA_AVALON_PIO_DATA(SPEAKER_BASE, 0x1);
              printf("speak \n");
          wifiReceive = (uint8_t) OSQPend(wifiPackageQueue, 0, &err);
          printf("Received form Wifi: %d \n", wifiReceive);
          motorCommand(wifiReceive);

        }
        //OSTimeDlyHMSM(0, 0, 0, CAM_WAIT_MS);
}

void wifi_uart_interrupt(void * context){
    uint8_t read = 0;
    printf("Hello from wifi interrupt\n");
    while(!(IORD_ALTERA_AVALON_UART_STATUS(WIFI_UART_BASE) &
ALTERA_AVALON_UART_STATUS_RRDY_MSK));
    read = IORD_ALTERA_AVALON_UART_RXDATA(WIFI_UART_BASE);

    OSQPost(wifiPackageQueue, (void*) read);
}

void motorCommand(uint8_t input) {
    if (input == WIFI_MOTOR_CW) {
            IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_L_BASE, MOTOR_FORWARD);
            IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_R_BASE, MOTOR_FORWARD);
    }
    if (input == WIFI_MOTOR_CCW) {
        IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_L_BASE, MOTOR_REVERSE);
            IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_R_BASE, MOTOR_REVERSE);
    }
    if (input == WIFI_MOTOR_STOP) {
        IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_L_BASE, MOTOR_SHORTSTOP);
            IOWR_ALTERA_AVALON_PIO_DATA(MOTOR_R_BASE, MOTOR_SHORTSTOP);
    }
}
```

The flow is quite simple, the hard part is dealing with microC intricacies.

When the UART (serial) component picks up a character from the ESP8266, an interrupt fires and puts the character on a queue which the wifi task is pending on. The wifi task then takes this character and does with it whatever you program it to. In our case, we had it control a motor.