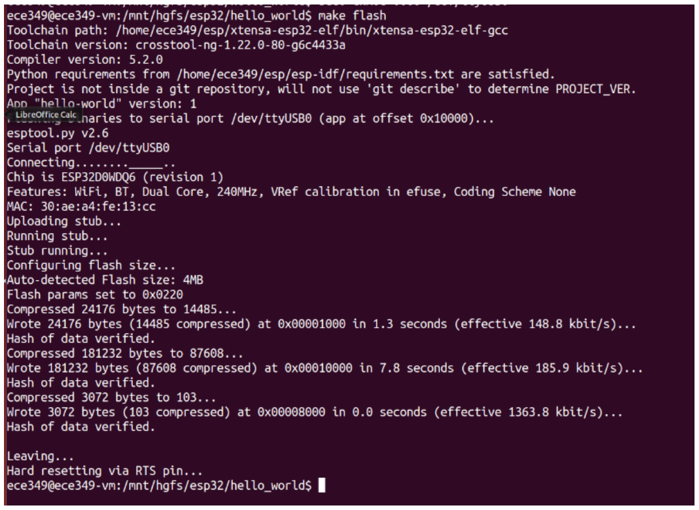
Progress Report 2/10/19 - 2/16/19
Ronit Banerjee, Team: A2

The beginning of this project required quite a bit of setup, thus I had 3 small tasks to do over this week and uptil next Wednesday. However, once the infrastructure has been built there will probably be longer tasks that occupy almost the entire week.

My deliverables for this week were to

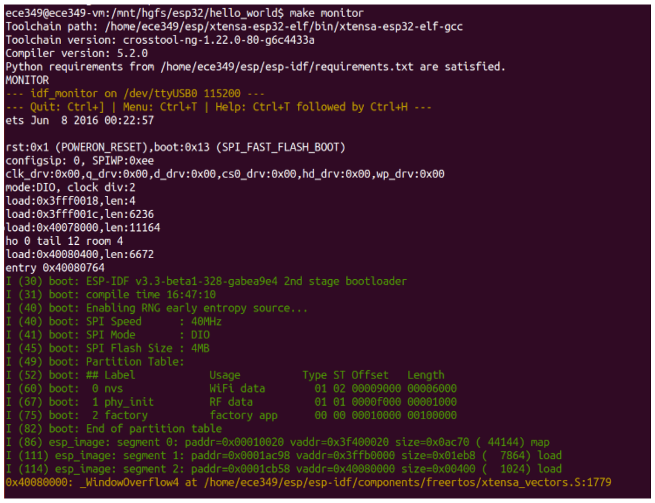
1. Familiarize myself with the esp32, setup the toolchain
2. Connect it to CMU's wifi, create a simple telnet server on my laptop to send data to and from it
3. Get the ESP32 sampling sound from the I2S mems microphone breakout board that we purchased.

To ensure portability and so that other teammates could get access to the toolchain easily, I decided to use a virtual machine to setup the build and debug environment, I used the 18349 virtual machine as a staring point and installed the necessary tools. We are now able to program the board over micro usb. While programming over microusb is convenient, we do not plan on having a microusb port on the esp32 PCB, as such I also had to verify that the board could be programmed directly from the gpio lines. With a little tweaking to the UART controller I was able to achieve this. This will also enable us to breadboard parts of our PCB before sending out to print. Next, the debug infrastructure had to be set up. I was able to get serial debugging working over serial and using RTOS , we get access printf, breakpoints, register dumps etc.



```
ece349@ece349-vm:/mnt/hgfs/esp32/hello_world$ make flash
Toolchain path: /home/ece349/esp/xtensa-esp32-elf/bin/xtensa-esp32-elf-gcc
Toolchain version: crosstool-ng-1.22.0-80-g6c4433a
Compiler version: 5.2.0
Python requirements from /home/ece349/esp/esp-idf/requirements.txt are satisfied.
Project is not inside a git repository, will not use 'git describe' to determine PROJECT_VER.
App "hello-world" version: 1
[illegible]naries to serial port /dev/ttyUSB0 (app at offset 0x10000)...
esptool.py v2.6
Serial port /dev/ttyUSB0
Connecting........_____..
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: 30:ae:a4:fe:13:cc
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 24176 bytes to 14485...
Wrote 24176 bytes (14485 compressed) at 0x00001000 in 1.3 seconds (effective 148.8 kbit/s)...
Hash of data verified.
Compressed 181232 bytes to 87608...
Wrote 181232 bytes (87608 compressed) at 0x00010000 in 7.8 seconds (effective 185.9 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds (effective 1363.8 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
ece349@ece349-vm:/mnt/hgfs/esp32/hello_world$
```



```
ece349@ece349-vm:/mnt/hgfs/esp32/hello_world$ make monitor
Toolchain path: /home/ece349/esp/xtensa-esp32-elf/bin/xtensa-esp32-elf-gcc
Toolchain version: crosstool-ng-1.22.0-80-g6c4433a
Compiler version: 5.2.0
Python requirements from /home/ece349/esp/esp-idf/requirements.txt are satisfied.
MONITOR
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:6236
load:0x40078000,len:11164
ho 0 tail 12 room 4
load:0x40080400,len:6672
entry 0x40080764
I (30) boot: ESP-IDF v3.3-beta1-328-gabea9e4 2nd stage bootloader
I (31) boot: compile time 16:47:10
I (40) boot: Enabling RNG early entropy source...
I (40) boot: SPI Speed      : 40MHz
I (41) boot: SPI Mode       : DIO
I (45) boot: SPI Flash Size : 4MB
I (49) boot: Partition Table:
I (52) boot: ## Label            Usage          Type ST Offset   Length
I (60) boot:  0 nvs              WiFi data        01 02 00009000 00006000
I (67) boot:  1 phy_init         RF data          01 01 0000f000 00001000
I (75) boot:  2 factory          factory app      00 00 00010000 00100000
I (82) boot: End of partition table
I (86) esp_image: segment 0: paddr=0x00010020 vaddr=0x3f400020 size=0x0ac70 ( 44144) map
I (111) esp_image: segment 1: paddr=0x0001ac98 vaddr=0x3ffb0000 size=0x01eb8 (  7864) load
I (114) esp_image: segment 2: paddr=0x0001cb58 vaddr=0x40080000 size=0x00400 (  1024) load
0x40080000: _WindowOverflow4 at /home/ece349/esp/esp-idf/components/freertos/xtensa_vectors.S:1779
```

The next step was to connect the device to CMU's wifi, I had to get the device's mac address and then register it on net reg, the device was able to connect to cmu's wifi, but for some reason it was not able to connect to the telnet server on my laptop that was on the same network. I think this is because the ports on my latop were not open and at the time I could not figure out how to do it but we still needed to verify that the wifi controller was working. Most of the ports on the cmu unix server are open. So, I ran a small echo server on the unix machines and got the esp32 to echo whatever I typed on my laptop that was sshed to cmu unix (the code for the server has been included). Thus, I was able to verify it actually connected. However, our goal is to be able to perform all our computation on a laptop.

I tried disabling the firewall but even then we could not connect. While sending data over CMU's wifi is the ultimate goal, we needed a workaround at least for now so that we can start work in parallel on other aspects of the project. My laptop can create wifi hotspots and I made the esp32 connect directly to the laptop and transfer data. I was able to get the simple echo server working, for now the task of sending data over cmu's wifi will be pushed to later on the schedule so as to focus on other aspects of the project.

The next deliverable was to connect the microphone to the esp32 to generate some data. A precursor to this task was to decide between Electret condenser microphones and mems microphone. I checked out one of each from IDEATE lending, the EC microphone could detect sound over a larger range of frequencies, and we thought this would enable us to pick up more features. However, when I connected the EC mic to the ESP32 ADC, I found that the results were very noisy, and the CPU was able to send very few samples of data per second. The danger of having low throughput is that the ESP32 sram will fill up and we'll start dropping data. We are sampling at 44KHz, which means we get 44000 integers every second, which comes to 171 kB/s. The esp32 has 512kB of SRAM, and since we are recording over intervals of 10 mins, our throughput has to be around 150 kB/s otherwise we run the risk of dropping data.

The mems device was better in terms of noise, but we still faced the issue of throughput. To get around this, we purchased a MEMS mic with very high signal to noise ratio (65 dB), this was much higher than the previous EC mic (32 B) and the previous mems mic (28 dB). This mic also works with i2s, which means we can use the hardware i2s controller with dma to get signal samples from the mic (I have attached the code). In theory this should increase throughput. Unfortunately, I was not able to get the i2s to work. The processor is driving the clock, but we are just getting noise from the device. The problem is either with the code or the device (most likely the former). However, just as a sanity check, I will use an Arduino and a built in i2s library just to ensure that the device works.

I am on schedule for all tasks, the i2s task has time till Wednesday and there is slack till the weekend. My focus for next week will be to extract sound from esp32 and send it to the server. If I finish the task before the weekend, I will work on making the processor sleep and wakeup on a gpio interrupt. We did not purchase the breakout board for our wakeup mic, because it was almost $100 even though the actual mic is under $2 instead, we will use an analog mic and an analog comparator to develop the functionality on the esp32 until we get our pcbs and we can test it on the actual wakeup mic.