Agora | Getting started with Edge Impulse

3 Dec 2021 | v1.0.0

www.embeddedplanet.com

1. Prerequisites

NOTE: This tutorial is based off of Edge Impulse's continuous motion recognition tutorial here

https://docs.edgeimpulse.com/docs/continuous-motion-recognition

In order to start uploading data from the EP_AGORA to Edge Impulse, you will need working installations of:

Mbed CLI Edge Impulse Data Forwarder Your preferred serial program (TeraTerm, PuTTY, Minicom, etc.)
https://docs.edgeimpulse.com/docs/cli-data-forwarder

Firstly, sensor data need to be collected and uploaded to the Edge Impulse cloud for training. The data are collected and sent like this:

Agora ----> Dev Machine running Edge Data Forwarder ----> Edge Impulse Cloud

For example here, datapoints can be collected by reading from the LSM9DS1 accelerometer sensor. Then, after reading from each axis, the data can be formatted and sent over the serial bus to the machine you are using for development.

2. Setting Up Edge Impulse Data Forwarder

The Edge Impulse forwarder is expecting individual sensor data to be separated with commas or tabs, with each datapoint being on a new line. The default baud rate of the forwarder is 115200. You can get the Agora to read and forward the data via an mbed program:

1. Use mbed-cli to start a project by running `mbed new acquire-data`. See mbed-cli's documentation for more information about starting and managing projects.
2. Next, you'll need to add the EP libraries to interface with the sensors. To do this, run `mbed add https://github.com/embeddedplanet/ep-oc-mcu/#17fedb4031ede8018d3916a51b3b6d2008dbb6af` in the directory of your project.
3. Then, you need to configure the project to use agora specific settings. Copy the following text into mbed_app.json, replacing the file's existing contents:

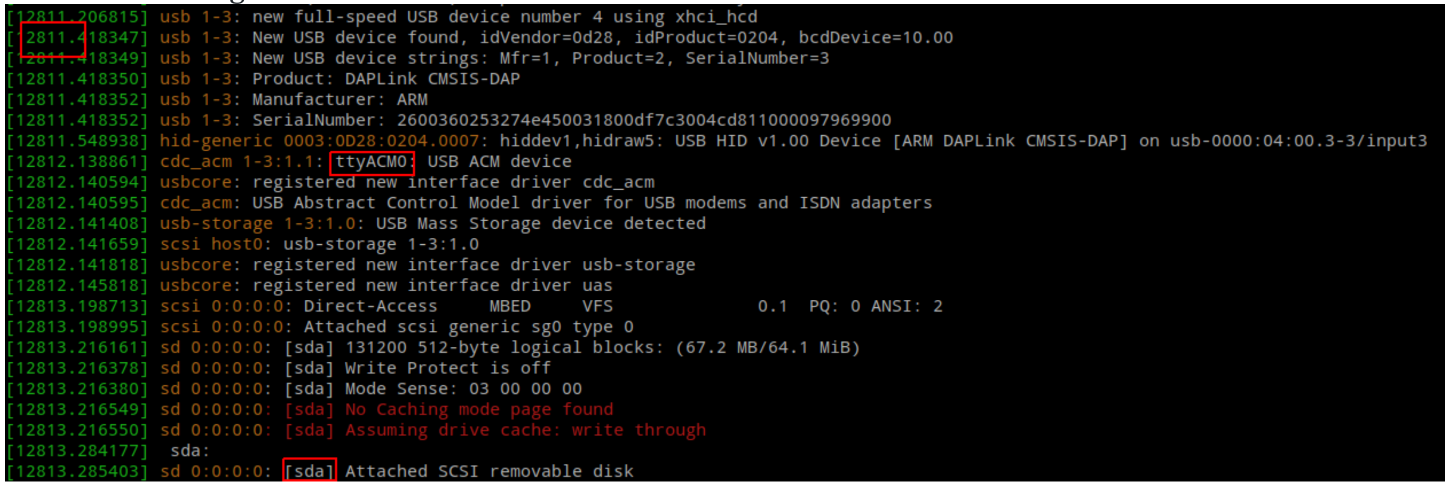
```
{
    "target_overrides": {
        "*": {
            "target.features_add"                       : ["BOOTLOADER", "STORAGE"],
            "platform.stdio-baud-rate"                  : 115200,
            "platform.stdio-convert-newlines"           : true,
            "platform.stdio-buffered-serial"            : true,
            "platform.stdio-flush-at-exit"              : true,
            "rtos.main-thread-stack-size"               : 5120,
            "mbed-trace.enable"                         : null,
            "events.shared-stacksize"                      : 2048,
            "nsapi.default-wifi-security"                 : "WPA_WPA2",
            "nsapi.default-wifi-ssid"                      : "\"SSID\"",
            "nsapi.default-wifi-password"                  : "\"PASSWORD\"",
            "nsapi.default-cellular-apn"                   : "\"APN\"",
            "nsapi.default-cellular-sim-pin"              : null,
            "nsapi.default-cellular-username"              : null,
            "nsapi.default-cellular-password"              : null
        },
        "EP_AGORA": {
            "target.features_remove"                        : ["BLE", "CRYPTOCELL310"],
            "target.header_offset"                         : "0x10000",
            "target.app_offset"                             : "0x10400",
            "target.components_remove"                        : ["QSPIF"],
            "target.components_add"                           : ["SPIF"],
            "target.macros_remove"                            : ["MBEDTLS_CONFIG_HW_SUPPORT"],
            "storage_filesystem.internal_base_address"  : "(MBED_ROM_START + MBED_BOOTLOADER_SIZE)",
            "storage_filesystem.rbp_internal_size"       : "(32*1024)",
            "storage.storage_type"                          : "FILESYSTEM",
            "storage_filesystem.filesystem"                : "LITTLE",
            "storage_filesystem.blockdevice"                : "SPIF",
            "storage_filesystem.external_base_address"  : "(0x0)",
            "storage_filesystem.external_size"            : "(1024*1024*1)",
            "drivers.uart-serial-rxbuf-size"              : 1024,
            "drivers.uart-serial-txbuf-size"              : 1024,
            "lwip.ipv4-enabled"                                : true,
            "lwip.ipv6-enabled"                                : true,
            "lwip.ppp-enabled"                                  : true,
            "lwip.tcp-enabled"                                 : true,
            "lwip.ethernet-enabled"                          : false,
            "lwip.mem-size"                                       : 22000,
            "nsapi.dns-response-wait-time"                 : 30000
        }
    },
    "config": {
        "bootloader-size": {
            "help"        : "Helper macro to enable calculation of rom regions. target.header_offset and target.app_offset still needs to be calculated manually, though.",
            "value"       : "(32*1024)",
            "macro_name": "MBED_BOOTLOADER_SIZE"
        }
    }
}
```

3. Here is the code that will be forwarding the data, which you can copy into `main.cpp`:

```
#include "mbed.h"
#include "LSM9DS1.h"

#define G_TO_MS2 9.8067

static const uint8_t LSM9DS1_ACCEL_GYRO_ADDRESS = 0x6A << 1;
static const uint8_t LSM9DS1_MAG_ADDRESS = 0x1C << 1;

DigitalOut sensor_power_enable(PIN_NAME_SENSOR_POWER_ENABLE);

I2C i2c(PIN_NAME_SDA, PIN_NAME_SCL);
```

```
LSM9DS1 lsm9ds1(i2c, LSM9DS1_ACCEL_GYRO_ADDRESS, LSM9DS1_MAG_ADDRESS);

int main()
{
    sensor_power_enable = 1;

    if (lsm9ds1.begin())
    {
        printf("LSM9DS1 online\n");
        lsm9ds1.calibrate();
    } else
    {
        printf("ERROR: LSM9DS1 offline!\n");
    }

    while(1)
    {
        lsm9ds1.readAccel();

        printf("%f,%f,%f\n",lsm9ds1.calcAccel(lsm9ds1.ax) * G_TO_MS2, lsm9ds1.calcAccel(lsm9ds1.ay) *
G_TO_MS2, lsm9ds1.calcAccel(lsm9ds1.az) * G_TO_MS2);
        ThisThread::sleep_for(1);
    }

    return 0;
}
```

4. Lastly run a build with `mbed compile -t GCC_ARM -m EP_AGORA` and then copy the build output to the agora. On Linux, you will need to check how the device enumerated, which can be done by running `dmesg` right after you connect the agora. Near the bottom of the output, you should see something like this:



```
[12811.206815] usb 1-3: new full-speed USB device number 4 using xhci_hcd
[12811.418347] usb 1-3: New USB device found, idVendor=0d28, idProduct=0204, bcdDevice=10.00
[12811.418349] usb 1-3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[12811.418350] usb 1-3: Product: DAPLink CMSIS-DAP
[12811.418352] usb 1-3: Manufacturer: ARM
[12811.418352] usb 1-3: SerialNumber: 2600360253274e450031800df7c3004cd811000097969900
[12811.548938] hid-generic 0003:0D28:0204.0007: hiddev1,hidraw5: USB HID v1.00 Device [ARM DAPLink CMSIS-DAP] on usb-0000:04:00.3-3/input3
[12812.138861] cdc_acm 1-3:1.1: ttyACM0: USB ACM device
[12812.140594] usbcore: registered new interface driver cdc_acm
[12812.140595] cdc_acm: USB Abstract Control Model driver for USB modems and ISDN adapters
[12812.141408] usb-storage 1-3:1.0: USB Mass Storage device detected
[12812.141659] scsi host0: usb-storage 1-3:1.0
[12812.141818] usbcore: registered new interface driver usb-storage
[12812.145818] usbcore: registered new interface driver uas
[12813.198713] scsi 0:0:0:0: Direct-Access     MBED     VFS              0.1  PQ: 0 ANSI: 2
[12813.198995] scsi 0:0:0:0: Attached scsi generic sg0 type 0
[12813.216161] sd 0:0:0:0: [sda] 131200 512-byte logical blocks: (67.2 MB/64.1 MiB)
[12813.216378] sd 0:0:0:0: [sda] Write Protect is off
[12813.216380] sd 0:0:0:0: [sda] Mode Sense: 03 00 00 00
[12813.216549] sd 0:0:0:0: [sda] No Caching mode page found
[12813.216550] sd 0:0:0:0: [sda] Assuming drive cache: write through
[12813.284177]  sda:
[12813.285403] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

"sda" corresponds to the mass storage enumeration, and ttyACM0 corresponds to the serial port. Replace the following instances of "sda" and "ttyACM0" with the values from your dmesg output. To upload the build to the agora, mount the mass storage device: `sudo mount /dev/sda /mnt` and then copy the program over: `sudo cp BUILD/EP_AGORA/GCC_ARM/send-data.hex /mnt && sync`. Then, set up your serial program (such as TeraTerm) to communicate with the device "/dev/ttyACM0" with a baud rate of 115200 and no flow control.

After uploading the sampling code to the Agora, you can run the edge-impulse-data-forwarder and follow its prompts to provide your login credentials. The data forwarder is a nodeJS application that opens a websocket with the Edge Impulse cloud. This lets you control and label sampling from their web application instead of having to deal with files of sampled data on your local machine.

3. Collecting Data

Now, you can more closely follow Edge Impulse's tutorial at Step 2: "Collecting your first data."
https://docs.edgeimpulse.com/docs/continuous-motion-recognition

At step 6, select "C++ library" from the deployment menu and then click "Build." Then, in the parent directory of your project, you'll need to import Edge Impulse's example standalone project, which contains the boilerplate code needed to use the classifier you made. After that, you will need to follow a similar procedure to the data-forwarder to use agora-specific sensors and settings. All of that can be encapsulated in these steps:

1. Run `mbed import https://github.com/edgeimpulse/example-standalone-inferencing-mbed`
2. Extract the .zip file you downloaded from step 6 of Edge Impulse's tutorial in the new project directory.
3. Run `mbed add https://github.com/embeddedplanet/ep-oc-mcu/#17fedb4031ede8018d3916a51b3b6d2008dbb6af`
4. Copy the mbed_app.json of the acquire-data project to this one. You will need to add a line to the "*" block of "target_overrides," which is: "target.printf_lib": "std"
5. Now, copy this code to source/main.cpp:

```
#include "mbed.h"
#include "LSM9DS1.h"
#include "ei_run_classifier.h"

#define G_TO_MS2 9.8067

static const uint8_t LSM9DS1_ACCEL_GYRO_ADDRESS = 0x6A << 1;
static const uint8_t LSM9DS1_MAG_ADDRESS = 0x1C << 1;

DigitalOut sensor_power_enable(PIN_NAME_SENSOR_POWER_ENABLE);

I2C i2c(PIN_NAME_SDA, PIN_NAME_SCL);
LSM9DS1 lsm9ds1(i2c, LSM9DS1_ACCEL_GYRO_ADDRESS, LSM9DS1_MAG_ADDRESS);

static int64_t sampling_freq = EI_CLASSIFIER_FREQUENCY; // in Hz.
static int64_t time_between_samples_us = (1000000 / (sampling_freq - 1));

// to classify 1 frame of data you need EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE values
static float features[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE];

// set baud rate of serial port to 115200
static BufferedSerial serial_port(USBTX, USBRX, 115200);
FileHandle *mbed::mbed_override_console(int fd) {
    return &serial_port;
```

```
}

int main()
{
    Timer t;
    t.start();

    sensor_power_enable = 1;

    if (lsm9ds1.begin())
    {
        printf("LSM9DS1 online\n");
        lsm9ds1.calibrate();
    } else
    {
        printf("ERROR: LSM9DS1 offline!\n");
    }

    while (1) {
        // fill the features array
        for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix +=
EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME) {
            int64_t next_tick = t.read_us() + time_between_samples_us;

            lsm9ds1.readAccel();

            // copy accelerometer data into the features array
            features[ix + 0] = lsm9ds1.calcAccel(lsm9ds1.ax) * G_TO_MS2;
            features[ix + 1] = lsm9ds1.calcAccel(lsm9ds1.ay) * G_TO_MS2;
            features[ix + 2] = lsm9ds1.calcAccel(lsm9ds1.az) * G_TO_MS2;

            while (t.read_us() < next_tick) {
                /* busy loop */
            }
        }

        // frame full? then classify
        ei_impulse_result_t result = { 0 };

        // create signal from features frame
        signal_t signal;
        numpy::signal_from_buffer(features, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);

        // run classifier
        EI_IMPULSE_ERROR res = run_classifier(&signal, &result, false);
        ei_printf("run_classifier returned: %d\n", res);
        if (res != 0) return 1;

        // print predictions
        ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): \n",
            result.timing.dsp, result.timing.classification, result.timing.anomaly);

        // print the predictions
        for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
            ei_printf("%s:\t%.5f\n", result.classification[ix].label,
result.classification[ix].value);
```

```
        }
    #if EI_CLASSIFIER_HAS_ANOMALY == 1
        ei_printf("anomaly:\t%.3f\n", result.anomaly);
    #endif
    }
}
```

6. Lastly, compile and run on your device!

```
$ mbed compile -t GCC_ARM -m EP_AGORA
$ sudo cp BUILD/EP_AGORA/GCC_ARM/example-standalone-inferencing-mbed.hex /mnt && sync
```

Use your favorite serial port reading program to see the classifier output!