A8 Assignment – ESE516-SPRING 2022

DUE DATE: Thursday April 4 2022 before 11:59pm EST (By almost midnight). To be submitted on Google Drive of Team Folder on a folder called A8. GROUP PROJECT

Remember: Please submit your complete Atmel Project on Google Drive. The project must be complete (must have everything so we can compile it!).

1.) PRE-REQUISITE WORK

Before you start this week's project, there is some pre-requisite work to be done. Please complete these tasks!

A.)Build your system

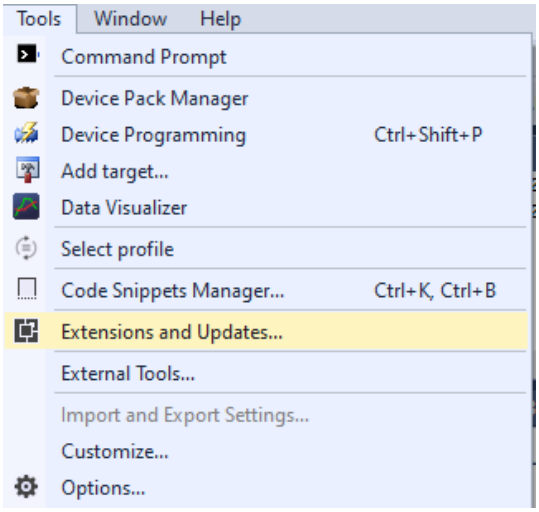
Please follow the instructions on the file "ESE516_ONLINE_BUILD_SPRING2021_A8.pptx" and build your system. Please double check before continuing!

B.)Install Percepio

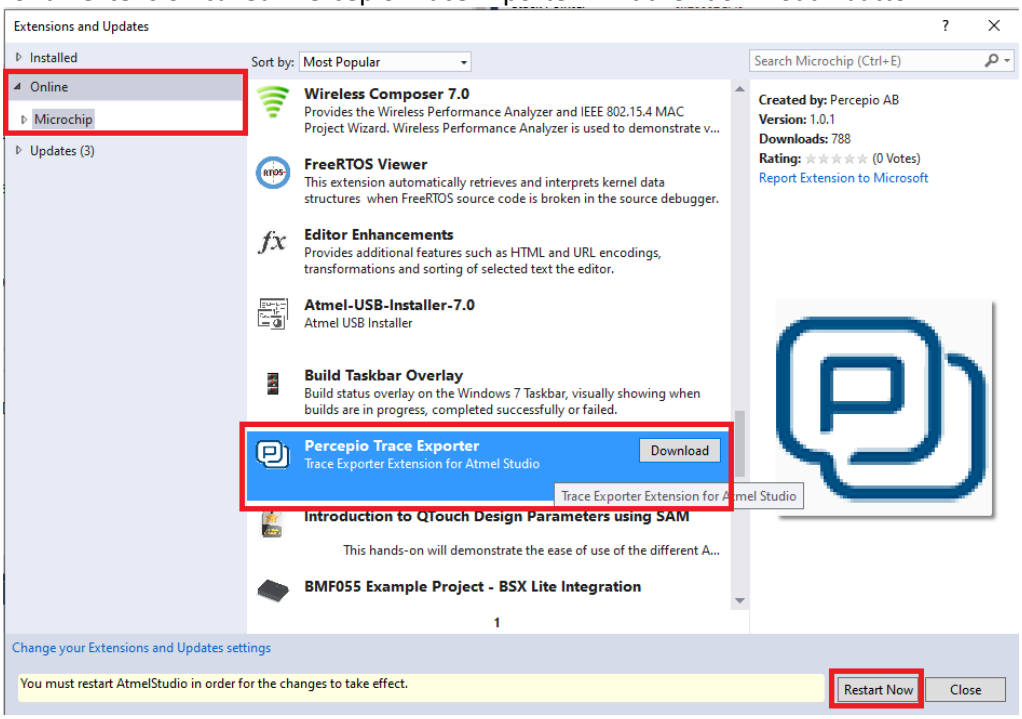
Please follow the following steps to install Percepio on your system.

Percepio Installation

1.) Open Atmel Studio
2.) Go to "Tools ->Extensions and Updates"



3.) The Extensions and Updates window will open. Choose “Online” on the bar to the left and then search for an extension called “Percepio Trace Exporter”. Hit the “download” button.



4.) Once the download is finished hit the “Restart Now” button.

5.) This will install the Percepio Plugin in your system. Now, please download Tracealyzer from the A8 google drive folder

6.) When asked for a license please use ONLINE ACTIVATION and use this code:
YCUY0-XF4VO-PJ4GQ-RB3M6-6QSVX

C.)Download Starter Project

Download the starter project from the A8 Folder (link) and unzip to your computer. Double click on “ESE516_STARTER_CODE.atsln” to open the project.

1.) Developing FreeRTOS Driver – I2C [100 points]

The given project has two I2C sensors – The Seesaw 44 RBG Screen and the I2C IMU. Before we get to use them, we have to finish the I2C driver that is FreeRTOS compatible.

For this, the project has the files "I2CDriver.c" (and .h). In these files we have the following initialized.

```
I2cInitializeDriver: Initializes the I2C driver. It sets the SERCOM 0 to use pins PA08 (SDA) and
PA09 (SCL). It also registers callbacks for  RX Complete, TX Complete, and Error. It also
initializes a Mutex and a Binary semaphore used by the system.
```

To do: Students will fill out the following functions to finalize the inner working on the I2C Driver. The driver's architecture attempts to do the following:

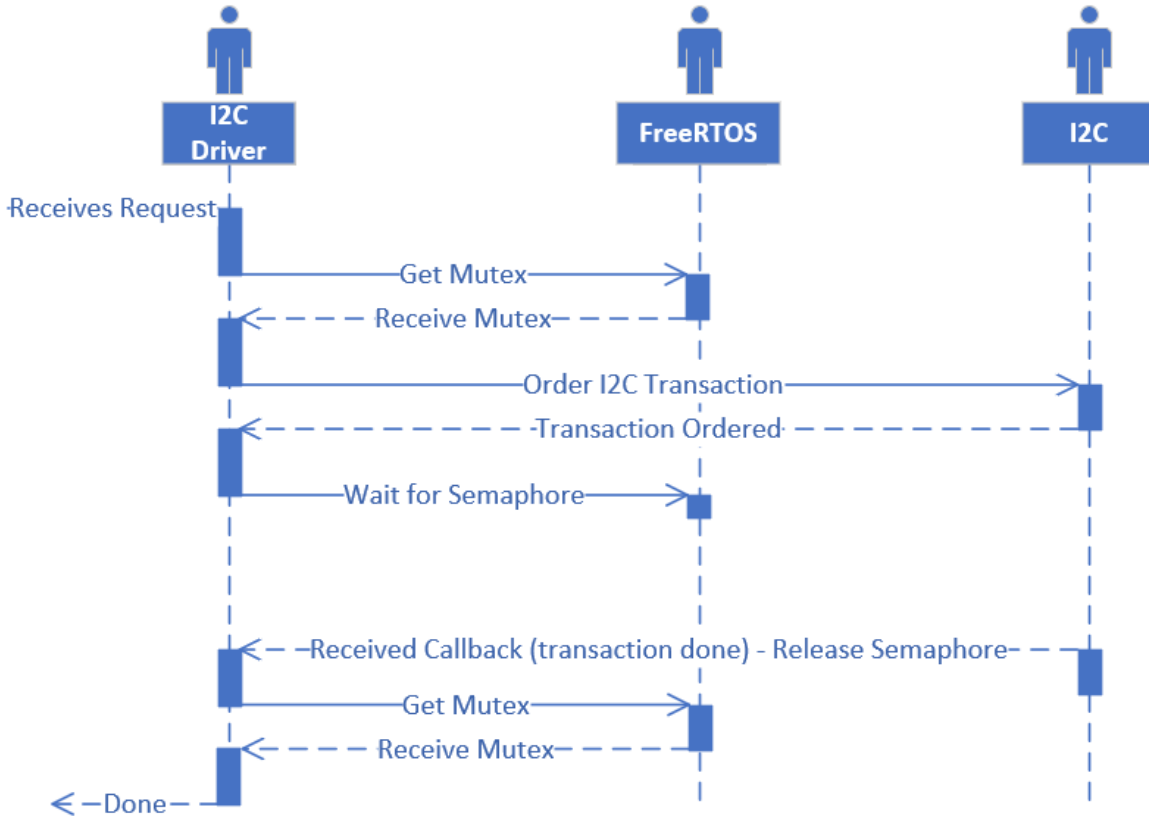


Figure 2, Desired combination of I2C!

Students to Fill Out (please read comments to determine what to do):

```
int32_t I2cGetMutex(TickType_t waitTime){

int32_t I2cFreeMutex(void){
```

```
I2cReadDataWait: TIP: This function writes an to an address and register and attempts to read.
Check the function "I2cWriteDataWait" to see how a simple write was implemented
```

How to test: Once you are done, you should be able to run the system. You will see that the IMU will fail (that is Q2) but the Neotrellis initialization will be ok! If it is not working, both initializations will fail.

2.) COMPLETING THE IMU INTERFACE FUNCTIONS[50 points]

For the IMU we downloaded a driver direct from the manufacturer. The only thing we need to do is fill out functions "`platform_write`" and "`platform_read`" on file "IMU/lsm6dso_reg.c".

To do: Please fill the files mentioned above. Read the comments of the functions on what they have to do. Use the I2C driver functions that you helped complete in the previous questions (`I2cWriteDataWait and I2cReadDataWait).`

How to test: There is a CLI command called "imu". If the device is working correctly, the IMU should successfully return an X,Y,Z acceleration value. If it is not working correctly, it will return 0 for all the axes.

3.) COMPLETING NEOTRELLIS CLI COMMANDS[50 points]

The following exercises are meant to show a common method of developing HW module drivers before diving into doing the main application. The first tasks will be to complete some CLI commands that are meant to show you how the FreeRTOS CLI works.

The commands for the student team to complete can be found on the File “CliThread.c” (on the CliThread folder). For this point the commands to implement are:

Command on the CLI: *led [keynum] [Red] [Green] [Blue]*
CLI Parameters: 4 parameters

- **Keynum:** Number from 0 to 15. Represents the LED number we want to turn on the device
- **Red:** Number from 0 to 255. Determines the intensity of the Red LED (0 being off, 255 being full on)
- **Green:** Number from 0 to 255. Determines the intensity of the Green LED (0 being off, 255 being full on)
- **Blue:** Number from 0 to 255. Determines the intensity of the Blue LED (0 being off, 255 being full on)

Function for student’s to complete:

```
BaseType_t CLI_NeotrellisSetLed() ---Line 336
```

About the Neotrellis Device:

The skeleton functions have good tips on how to get started. The I2C driver and the initialization of the device has been already done. There is also functions that turn LEDs on and off that will make the first task very easy. Please check "SeesawDriver.c" to see what these functions are.

The following will be important for the following question:
The button presses are somewhat more complicated. How the Neotrellis system works is that it will keep a running buffer of button presses and releases, in chronological order. Our MCU can:

- Query the system into how many events are there in the buffer (Command `SeesawGetKeypadCount()` does this for you).
- Read a certain number of events from the buffer (Command `SeesawReadKeypad()` does this for you).

What the second CLI command must do is read every event and then decode the events into human-readable strings. The Neotrellis device returns a byte per each event. Each event is coded in the following format:
BYTE (8 bits)

Most Significant 6 bytes : Key number Will return a key number based on the following diagram. You can use the macro `NEO_TRELLIS_SEESAW_KEY(number)` To turn this number into a 0-15 button number	**Last two significant bytes: Action** 0x02: Button was released 0x03: Button was pressed

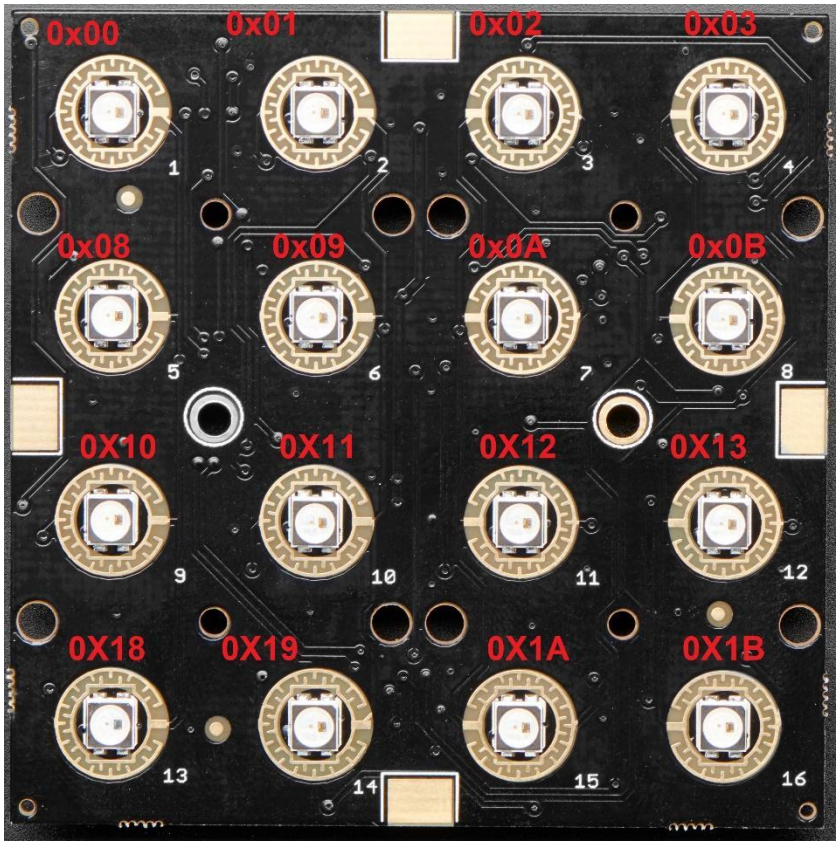


Figure 2: Red text shows the number the Neotrellis sends (6 bits value). The white number show the LED number + 1 (LED 16 in PCB will be LED15 in our code. LED 1 in PCB will be LED0 in our code)

Example of values received by the system

0X43

In binary:
0xb0100 0011

We get the first 6 bits to determine the LED number and the last two to get the action

Number: 0b0100 00 -? Adding two zeros on the left -> 0b0001 0000= 0x10 -> Button # 9 of PCB -> Button #8 of our code. If you use macro NEO_TRELLIS_SEESAW_KEY() on the value 0x10 you would have gotten 0x08.

Last two bytes : 0b11 -> Button was pressed.

4.) UI Thread – Handle Buttons [100 points]

The last step of this assignment is to implement code on the UI thread -currently empty- to do the following:

- Read the button buffer (Similar to how you did on the CLI part).
- Turn the LED of buttons that are pressed, and turn off the LEDs of the buttons that are not pressed (similar to the CLI command you also did before)

For this, please make your code inside the state machine state `UI_STATE_HANDLE_BUTTONS` on the file "UiHandlerThread.c"

If you want a more advanced challenge (THIS IS NOT REQUIRED FOR GRADING!) try to make the LEDs fade out for a small period of time after the user has just release the button press. A simple way to do this can be to perform a decay function as discussed in class.

Measuring time passing from two events is easy on FreeRTOS! You can use the function "TickType_t xTaskGetTickCount(void)" to get the current RTOS Tick time. Here is an example on how to use it:

```
TickType_t startTime =  xTaskGetTickCount(); //Get current time

while(1)
{
//Wait until some time has passed to do something - When time is
bigger than 200 ticks
if(xTaskGetTickCount() - startTime > 200)
     {
          startTime =  xTaskGetTickCount(); //Save ticktime for
next round
          //Perform the function you want to happen only every
delta T - Like updating the LED values to decay them!
     }
}
```

See other FreeRTOS useful functions here: https://www.freertos.org/a00021.html

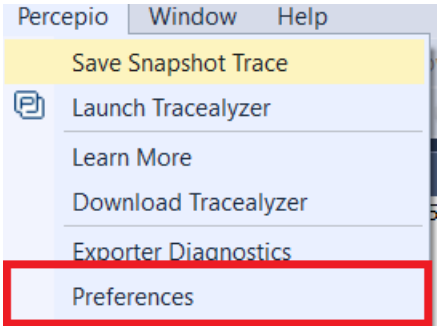
A8 RUBRIC

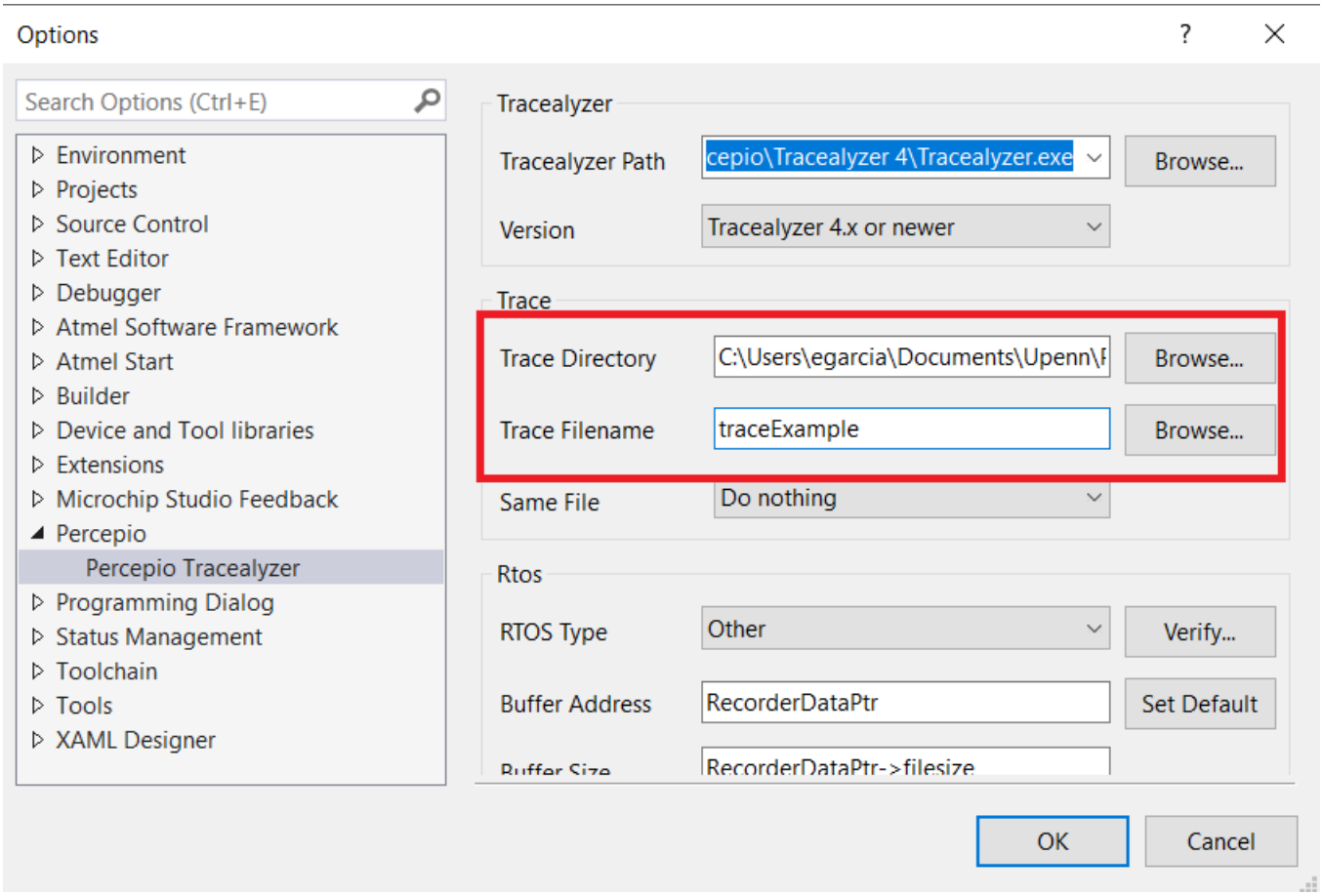
Points	Rubric
-300 points	If the project is incomplete or has missing files, it will incur this penalty.
100 points	I2C Driver works for read and write
50 points	CLI command imu works
50 points	CLI command to light up LEDs works
100 points	Task UiHanlder had the “Button Handler” code added to it. Buttons light up when pressed and turn off when not pressed anymore.
+20 points	Provided picture asked in appendix

APPENDIX: Do the following for extra 20 points – Using Percepio

Once you are done, we can use Percepio to test everything is ok, and to get a better understanding of the system! Now, place a breakpoint on line 118 of “main.c”.

When the breakpoint is hit, we can use Percepio to download a snapshot of what is the latest going on inside the MCU. To do this, once stopped, go to the Percepio toolbar on Atmel Studio and hit “Preferences” Give the file a path and name:





Then, choose “Save Snapshot Trace” on the Percepio menu. This should open Percepio!

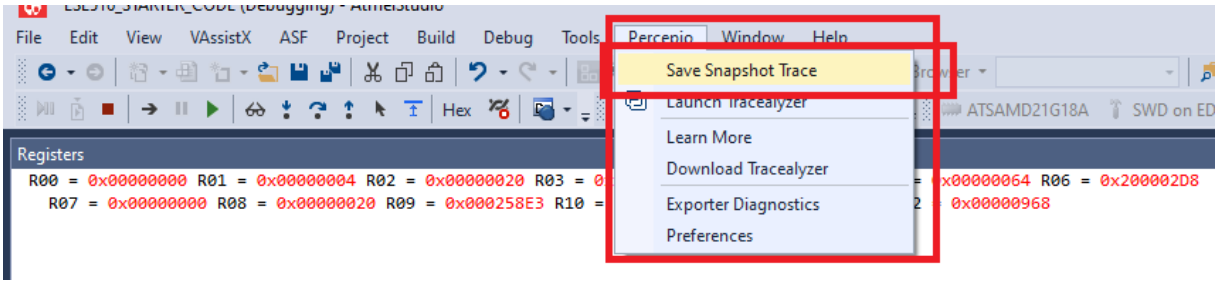


Figure 3. How to open Percepio once a breakpoint is hit and we are paused

Once Percepio is opened, you can click around and see events of the system. Your system should look similar to the one below. I added some comments to show what went down.

Please submit a picture of your percepio trace with your project.