Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/17/2021

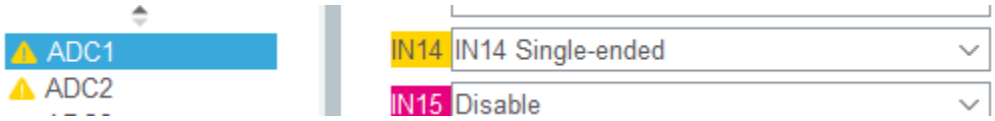
# Assignment 6: ADC

The following will document completion of the sixth assignment for ECE-40291, with the stated goals of:

1. Use STM32CubeMX to generate ADC1-IN14 code so that you can access the Arduino shield pin A0 on the STM32 Discovery IoT Board.
2. Use TrueStudio to include edit main.c to read the ARD_A0 value every 1 second.
3. Use TrueStudio to edit, build, run, and debug the code, using a breakpoint to display the value read from the ARD_A0 pin.
4. Try reading these three values from ARD_A0: (1) Floating (nothing connected to the pin); (2) GND (the ARD_A0 pin connected to ground); (3) 1.5 VDC battery connected to ARD_A0 (be sure to connect the GND of the battery to GND on the Arduino shield.

## 1. Use STM32CubeMX to generate ADC1-IN14 code so that you can access the Arduino shield pin A0 on the STM32 Discovery IoT Board.

After creating a new project for our disco board using the methods employed in previous assignments, we will need to open the project configurator to the ADC interface section and enable ADC1-IN14 to single ended mode from its default of disabled.




After closing the configurator, the IDE will rebuild the project files to include the appropriate ADC config function calls. We will want to jump in to main.c and verify that the ADC is being initialized to look at channel 14, or ARD_A0.

```
275  /** Configure Regular Channel
276  */
277  sConfig.Channel = ADC_CHANNEL_14;
278  sConfig.Rank = ADC_REGULAR_RANK_1;
279  sConfig.SamplingTime = ADC_SAMPLETIME_2CY
```

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/17/2021

**2. Use TrueStudio to include edit main.c to read the ARD_A0 value every 1 second.**

At this point, we can write the code needed to perform a read of the A0 input once a second, with an LED toggle in place to provide a visual feedback of the code executing. The first step will be to call the calibration function, *HAL_ADCEx_Calibration_Start()*, on ADC1 in single ended mode, as we configured it for earlier. Then in the *while(1)* loop, we will implement the LED toggle and a short delay with the HAL functions as appropriate, and then call the functions to enable the ADC, *HAL_ADC_Start()*, poll for a conversion completion, *HAL_ADC_PollForConversion()*, and then read the value into a local variable to be observed in the debug session. Note the breakpoint placed on the delay function, line 146 of main.c

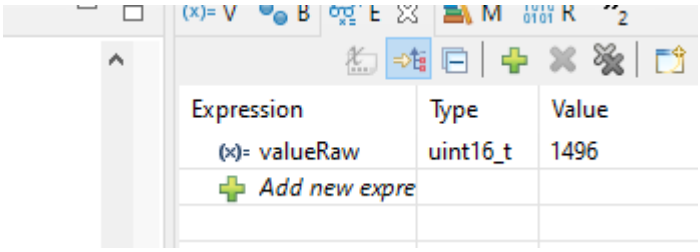
```
118   /* USER CODE BEGIN 2 */
119
120   // Calibrate ADC
121   HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
122
123   /* USER CODE END 2 */
124
125   /* Infinite loop */
126   /* USER CODE BEGIN WHILE */
127   while (1)
128   {
129     /* USER CODE END WHILE */
130
131     /* USER CODE BEGIN 3 */
132
133     // Blink to give indication of *something* happening
134     HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
135
136
137     // Enable ADC1
138     HAL_ADC_Start(&hadc1);
139
140     // Poll ADC1 for value
141     HAL_ADC_PollForConversion(&hadc1, 10);
142
143     uint16_t valueRaw = HAL_ADC_GetValue(&hadc1);
144
145     // Short delay
146     HAL_Delay(1000);
147
148   }
149   /* USER CODE END 3 */
```

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/17/2021

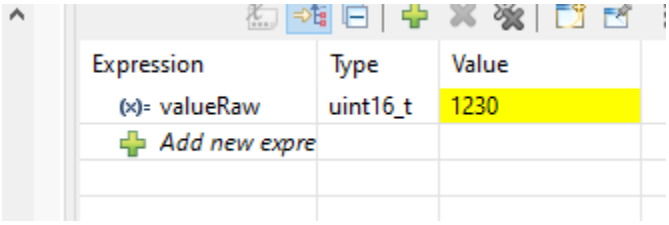
3. **Use TrueStudio to edit, build, run, and debug the code, using a breakpoint to display the value read from the ARD_A0 pin.**

At this point, we can compile and flash the disco board and enter into a debug session, stepping into the *while(1)* loop and watching the value of our holding variable, valueRaw.

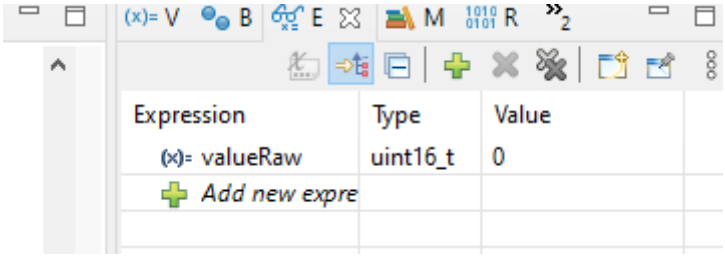



4. **Try reading these three values from ARD_A0: (1) Floating (nothing connected to the pin); (2) GND (the ARD_A0 pin connected to ground); (3) 1.5 VDC battery connected to ARD_A0 (be sure to connect the GND of the battery to GND on the Arduino shield.**

With the pin floating, we can see garbage value in our holding variable. After stepping through the loop several times, the ADC floated around and eventually settled to somewhere around this value:



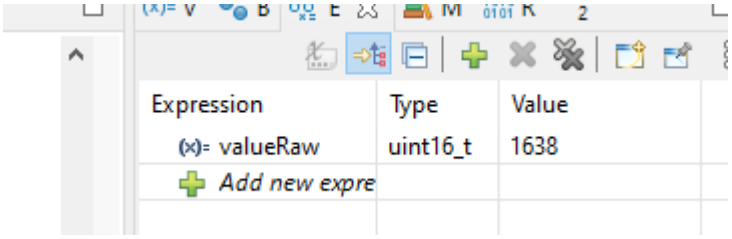

With the pin grounded, we see zero, as would be expected:



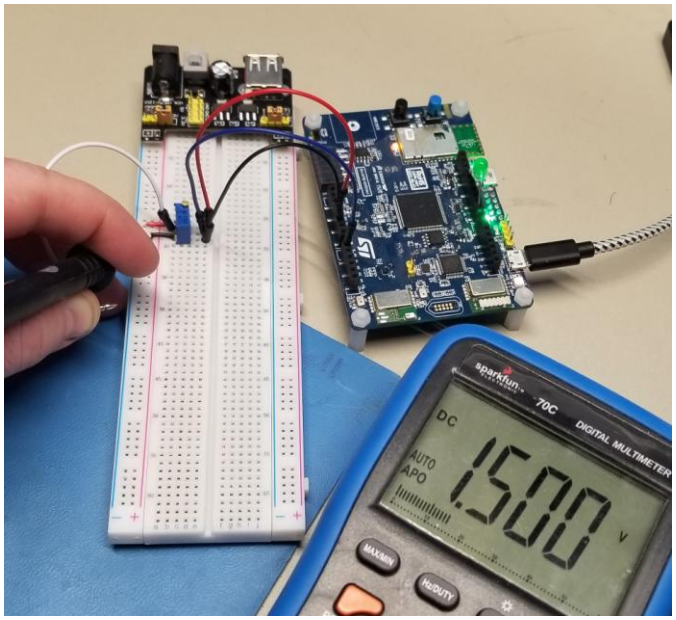

And finally with a potentiometer tuned to 1.5 volts and confirmed with a DMM, we get a reading that was hovering +/-5 or so counts from this reading:

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/17/2021




Interestingly, when this is converted (valueRaw * (3.3v / 4096)), this is about 1.3 volts, indicating that something is either off with my meter, or there was some type of connection or config issue on the board side. After drafting this report, I came back and checked a second reading at 2.5v and was getting a value of 2807, or ~2.26v. In the other ADC assignment, I had noted that the mid range readings seemed to hold a fair margin of error and again, am not sure if I should attribute it to my meter being out of calibration or some issue on my breadboard setup.




**Closing Thoughts**

This exercise was easy to accomplish, mainly from having already completed the ADC assignment in ECE-40293 and having worked out the various gotcha's and potential missteps while working through those User Stories. As I reflected in that report, I interact with many ADCs in my day to day role at work and this experience was a great way to again get closer to that hardware as well as to reinforce the skills learned in the other course.