

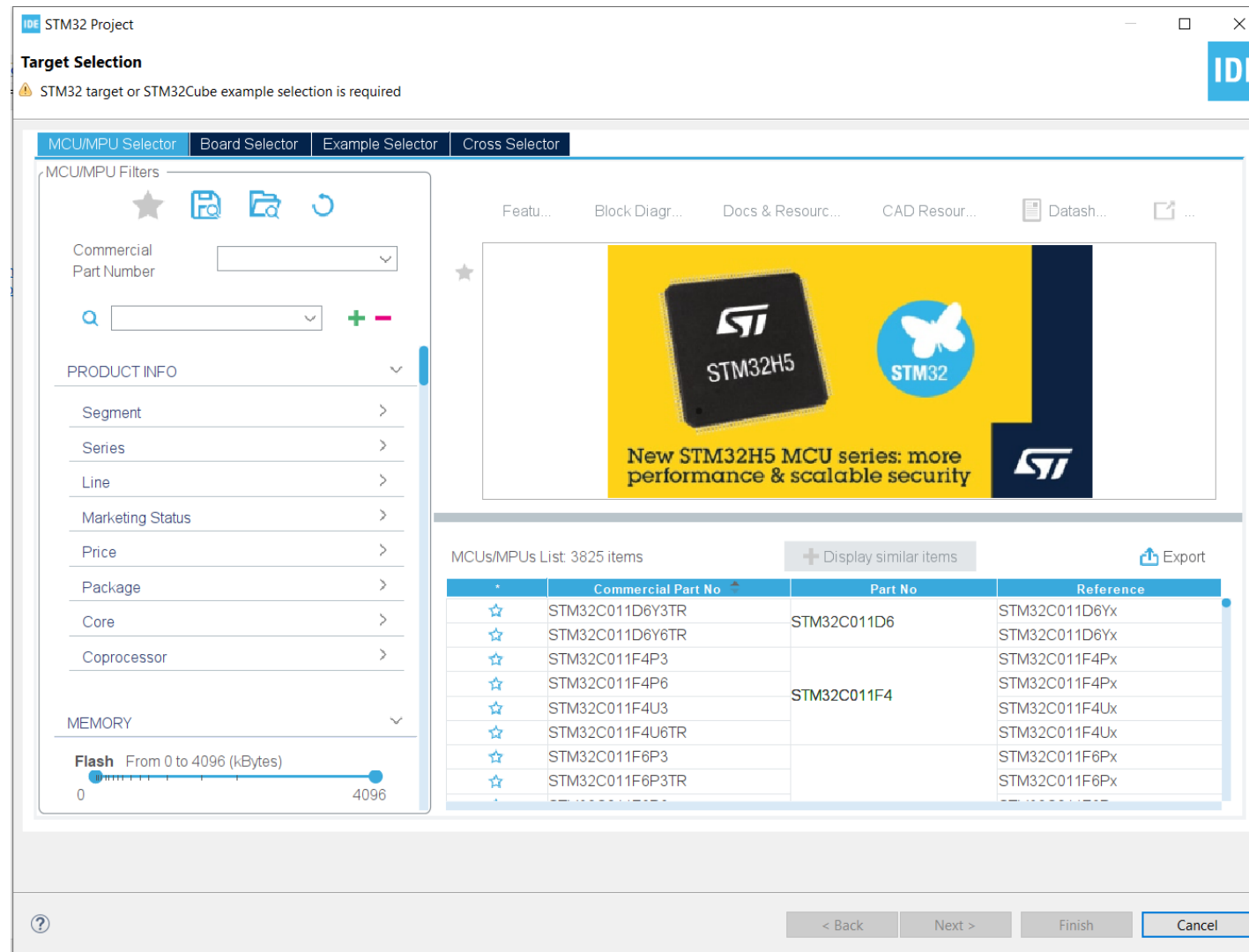
UCSD Embedded C Final Project

By

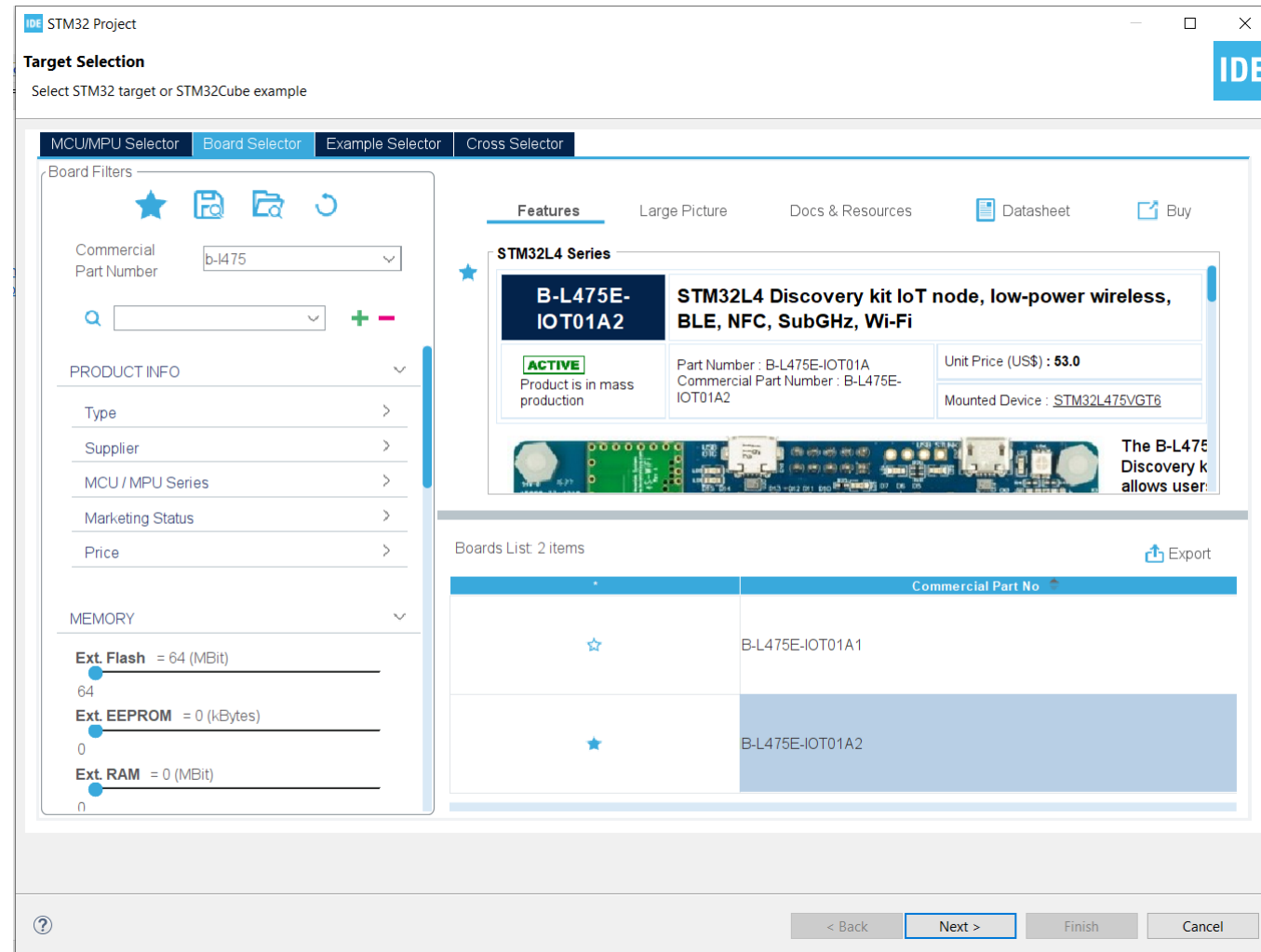
Hsuankai Chang

hsuankac@umich.edu

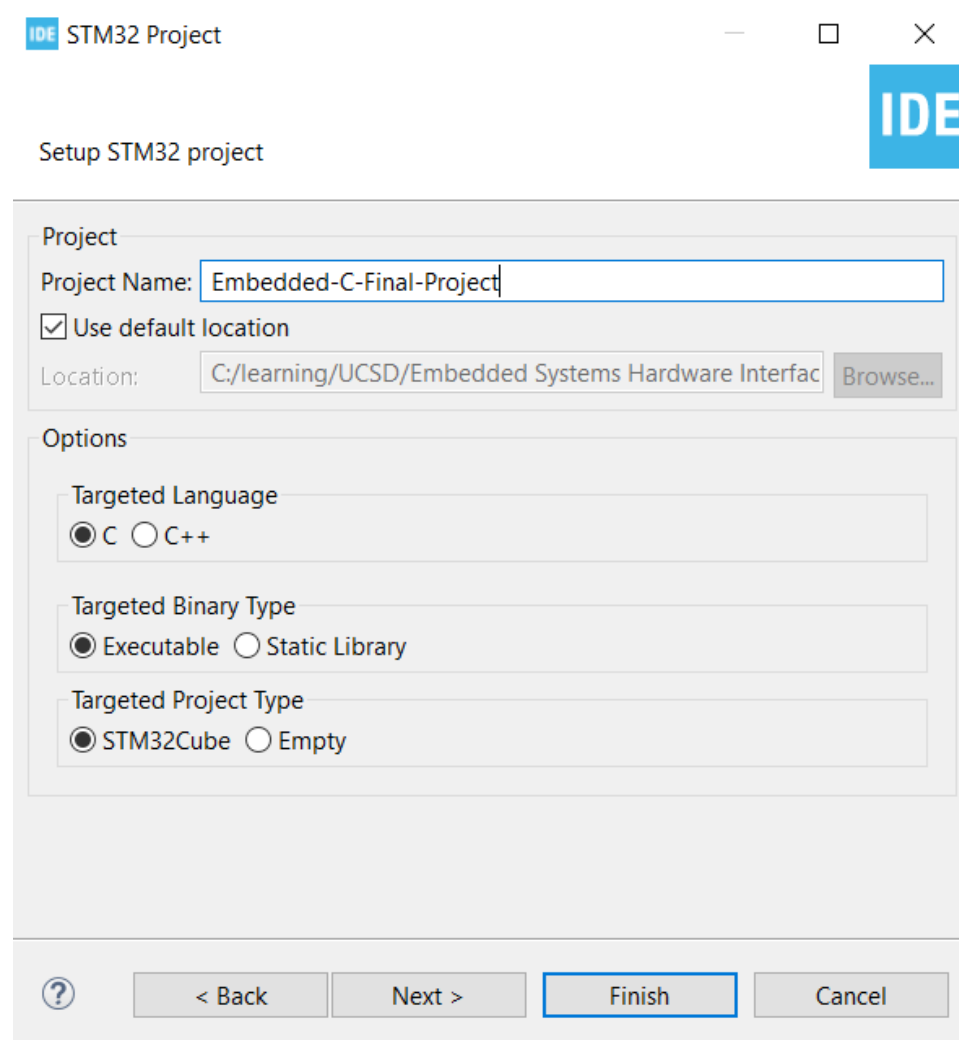
Step 1. Startup STM32CubeIDE and create new STM32 project



Step 2. Access board selector and type in the board you use, click Next



Step 3. Enter the project name then click Next



The image shows a 'Setup STM32 project' dialog box from an IDE. The window title is 'IDE STM32 Project'. The dialog is titled 'Setup STM32 project'. It has two main sections: 'Project' and 'Options'. In the 'Project' section, the 'Project Name' field contains 'Embedded-C-Final-Project'. Below it, the 'Use default location' checkbox is checked. The 'Location' field shows 'C:/learning/UCSD/Embedded Systems Hardware Interfac' with a 'Browse...' button to its right. The 'Options' section contains three groups of radio buttons: 'Targeted Language' with 'C' selected, 'Targeted Binary Type' with 'Executable' selected, and 'Targeted Project Type' with 'STM32Cube' selected. At the bottom, there is a row of buttons: a help icon (?), '< Back', 'Next >', 'Finish' (which is highlighted with a blue border), and 'Cancel'.

IDE STM32 Project

Setup STM32 project

Project

Project Name: Embedded-C-Final-Project

☒ Use default location

Location: C:/learning/UCSD/Embedded Systems Hardware Interfac Browse...

Options

Targeted Language

☒ C ☐ C++

Targeted Binary Type

☒ Executable ☐ Static Library

Targeted Project Type

☒ STM32Cube ☐ Empty

? < Back Next > Finish Cancel

Step 4. See the firmware package name, version and location



The screenshot shows a dialog box titled "STM32 Project" with a subtitle "Firmware Library Package Setup". The main text says "Setup STM32 target's firmware". The dialog is divided into three sections: "Target and Firmware Package", "Firmware and Software Package Repository", and "Code Generator Options".

Target and Firmware Package

Target Reference: B-L475E-IOT01A2

Firmware Package Name and Version: STM32Cube FW_L4 V1.17.2

Firmware and Software Package Repository

Location: C:\Users\hsuankai.chang\STM32Cube\Repository

See ['Firmware Updater'](#) for settings related to package installation

Code Generator Options

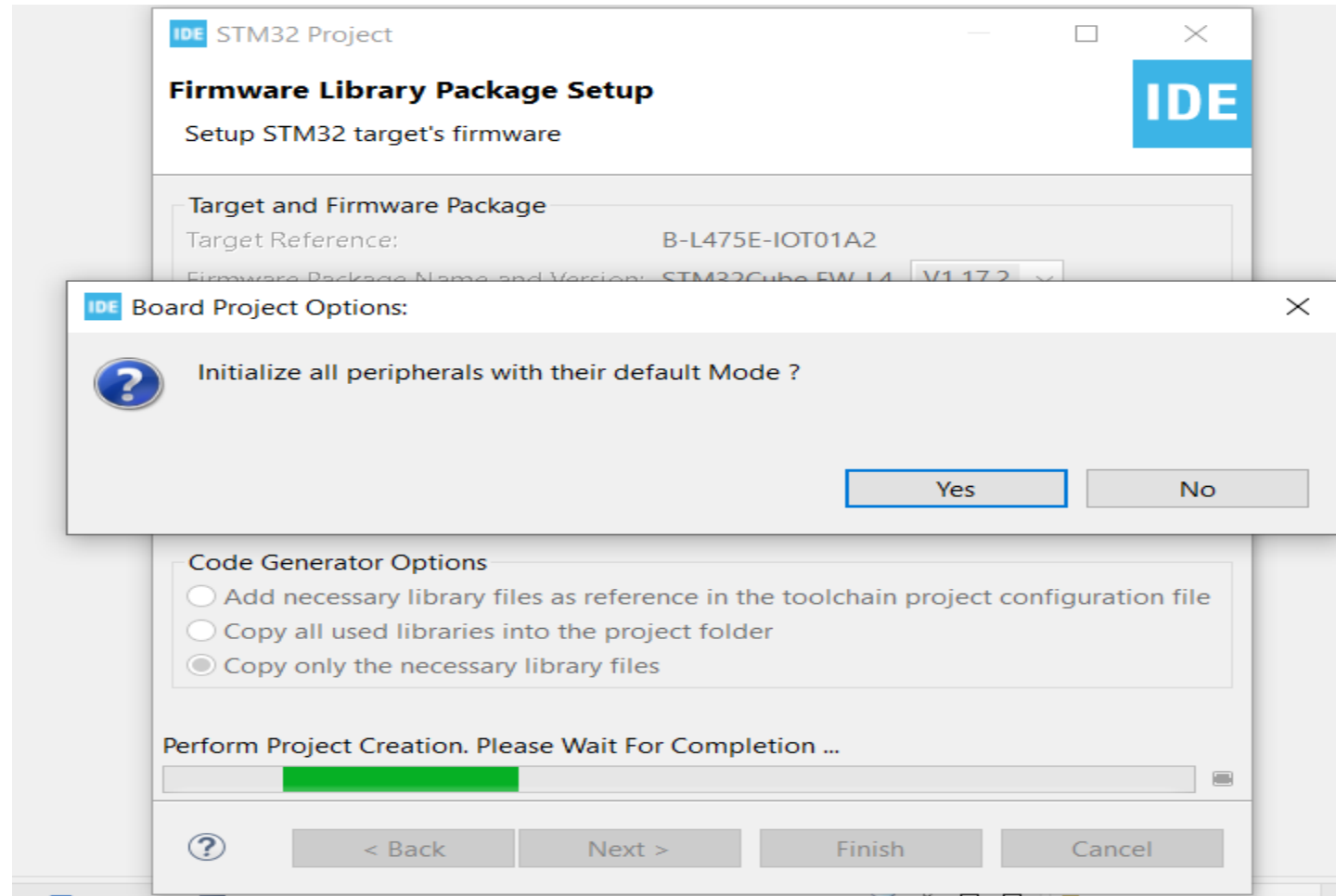
☐ Add necessary library files as reference in the toolchain project configuration file

☐ Copy all used libraries into the project folder

☒ Copy only the necessary library files

At the bottom, there are buttons for "?", "< Back", "Next >", "Finish", and "Cancel".

Step 5. Click yes to initialize all peripherals to default



Step 6. In the final project, I decide to demonstrate ADC reading in polling and interrupt mode, I2C temperature reading with HST221 using polling, interrupt and DMA mode, also using UART to transmit and receive message, and GPIO to toggle the LED pin. First start to enable ADC settings

The screenshot shows the STM32CubeIDE Pinout & Configuration window. The left sidebar lists various components under 'Categories', with 'ADC1' selected. The main panel displays the 'ADC1 Mode and Configuration' settings. The 'Mode' section shows channels IN11 through IN15, with IN11, IN12, and IN15 set to 'Disable', IN13 set to 'Disable', and IN14 set to 'IN14 Single-ended'. The 'Configuration' section includes a 'Reset Configuration' button and tabs for 'Parameter Settings', 'User Constants', 'NVIC Settings', 'DMA Settings', and 'GPIO Settings'. The 'NVIC Settings' tab is active, showing the 'NVIC Interrupt Table' with 'ADC1 and ADC2 interrupts' enabled.

Mode
IN11 Disable
IN12 Disable
IN13 Disable
IN14 IN14 Single-ended
IN15 Disable
<input type="checkbox"/> IN16 Single-ended
<input type="checkbox"/> Temperature Sensor Channel
<input type="checkbox"/> Vbat Channel
<input type="checkbox"/> Vrefint Channel
EXTI Conversion Trigger Disable

Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
ADC1 and ADC2 interrupts	<input checked="" type="checkbox"/>	0	0

Step 7. Next, start setting up the I2C, notice that I set up both the interrupt and DMA mode

Embedded-C-Final-Project.ioc × main.c stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c

Embedded-C-Final-Project.ioc - Pinout & Configuration

Pinout & Configuration | Clock Configuration | Project Manager

Software Packs | Pinout

I2C2 Mode and Configuration

Mode

I2C I2C

Configuration

Reset Configuration

Parameter Settings | User Constants | NVIC Settings | DMA Settings | GPIO Settings

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
DMA1 channel4 global interrupt	<input checked="" type="checkbox"/>	0	0
DMA1 channel5 global interrupt	<input checked="" type="checkbox"/>	0	0
I2C2 event interrupt	<input checked="" type="checkbox"/>	0	0
I2C2 error interrupt	<input type="checkbox"/>	0	0

n.c stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c

- Pinout & Configuration

Pinout & Configuration | Clock Configuration | Project Manager

Software Packs | Pinout

I2C2 Mode and Configuration

Mode

I2C I2C

Configuration

Reset Configuration

Parameter Settings | User Constants | NVIC Settings | DMA Settings | GPIO Settings

DMA Request	Channel	Direction	Priority
I2C2_RX	DMA1 Channel 5	Peripheral To Memory	Low
I2C2_TX	DMA1 Channel 4	Memory To Peripheral	Low

Add Delete

Step 8. Start coding the command line interface using UART

```
main.c × stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
476 MX_USB_OTG_FS_PCD_Init();
477 MX_ADC1_Init();
478 /* USER CODE BEGIN 2 */
479 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
480 /* USER CODE END 2 */
481
482 /* Infinite loop */
483 /* USER CODE BEGIN WHILE */
484 while (1)
485 {
486     /* USER CODE END WHILE */
487
488     /* USER CODE BEGIN 3 */
489     // Issue command prompt
490     char *prompt = "Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), "
491                  "6=ADC Reading(Interrupt)\n\r cmd> ";
492     HAL_UART_Transmit(&huart1, (uint8_t*)prompt, strlen(prompt), 1000);
493
494     // Wait for a single number entry
495     char ch;
496     HAL_UART_Receive(&huart1, (uint8_t*)&ch, 1, HAL_MAX_DELAY);
497
498     char *msg = "\r\n";
499     HAL_UART_Transmit(&huart1, (uint8_t*)msg, strlen(msg), 1000);
500
501     switch(ch)
502     {
503     case '1': do_who_am_i(); break;
504     case '2': do_temp_polled(); break;
505     case '3': do_temp_interrupt(); break;
506     case '4': do_temp_dma(); break;
507     case '5': adc_polling_average(); break;
508     case '6': adc_interrupt(); break;
509     // Fall through if none
510     }
511 }
512 /* USER CODE END 3 */
513 }
```

Step 9. Start to code up the I2C who am I register reading

```
main.c × stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
81
82 /* Private user code -----*/
83 /* USER CODE BEGIN 0 */
84 #define HST221_READ_ADDRESS      0xbf
85 #define HST221_WRITE_ADDRESS    0xbe
86
87 static uint8_t irq_complete = 0;
88 static uint8_t adc_irq_complete = 0;
89 static uint8_t status_flag = 0;
90 static uint16_t data = 0;
91 static uint8_t status_data = 0;
92
93 uint16_t find_max(uint16_t arr[], uint8_t size)
94 {
95     uint16_t max = arr[0];
96     for (int i = 1; i < size; i++)
97     {
98         if (arr[i] > max)
99             max = arr[i];
100     }
101     return max;
102 }
103
104 uint16_t find_min(uint16_t arr[], uint8_t size)
105 {
106     uint16_t min = arr[0];
107     for (int i = 1; i < size; i++)
108     {
109         if (arr[i] < min)
110             min = arr[i];
111     }
112     return min;
113 }
114
115 void do_who_am_i()
116 {
117     // Step 1. send sub address
118     // Write sub address
119     uint8_t who_am_i = 0xf; // WHO_AM_I register
120     HAL_StatusTypeDef status;
121     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &who_am_i, sizeof(who_am_i), 1000);
122
123     char buf[100];
124     snprintf(buf, sizeof(buf), "HAL_I2C_Master_Transmit: status: %u\r\n", status);
125     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
126
127     // Step 2. read from address to get WHO_AM_I
128     uint8_t data = 0x42;
129     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, &data, sizeof(data), 1000);
130
131     snprintf(buf, sizeof(buf), "HAL_I2C_Master_Receive: status: %u, data: 0x%x\r\n", status, data);
132     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
133
134 }
135
```

Step 10. Build and test the function, test is successful

COM4 - Tera Term VT

File Edit Setup Control Window Help

Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd>

HAL_I2C_Master_Transmit: status: 0

HAL_I2C_Master_Receive: status: 0, data: 0xbc

Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd> █

Step 11. Start coding the I2C reading in polling mode

```
main.c × stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
135
136 void do_temp_polled()
137 {
138     // Setup control register 1
139     uint8_t control_reg1 = 0x20;
140     uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
141     HAL_StatusTypeDef status;
142     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1), 1000);
143
144     // Start a conversion
145     uint8_t control_reg2 = 0x21;
146     uint8_t control_data2[] = {control_reg2, 0x01};
147     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2), 1000);
148
149     char buf[100];
150     snprintf(buf, sizeof(buf), "(One-shot enable) HAL_I2C_Master_Transmit: status: %u\r\n", status);
151     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
152
153     // Wait for conversion complete
154     uint8_t status_reg = 0x27;
155     uint8_t status_data = 0;
156     int count = 0;
157     while(count < 10)
158     {
159         // Send read status register sub command
160         status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg), 1000);
161         snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit: status: %u\r\n", count, status);
162         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
163
164         // Read conversion status
165         status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data), 1000);
166         snprintf(buf, sizeof(buf), "Status register: 0x%02x\r\n", status_data);
167         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
168
169         // Check for temperature conversion complete
170         if(status_data & 0x1)
171         {
172             // ...
173         }
174     }
175 }
```

Step 12. I2C temperature reading code continue, notice like just in assignment, I toggle between one-shot and auto increment mode

```
main.c x stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
169 // Check for temperature conversion complete
170 if(status_data & 0x1)
171 {
172     snprintf(buf, sizeof(buf), "New data available!\r\n");
173     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
174     break;
175 }
176 HAL_Delay(1000);
177 count++;
178 }
179
180 // Toggle between normal poll and address increment poll
181 static int toggle = 1;
182
183 if(toggle)
184 {
185     toggle = 0;
186
187     // Read temperature LSB
188     uint8_t temperature_lsb = 0x2a;
189     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
190     snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
191     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
192
193     uint8_t data_lsb = 0x42;
194     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_lsb, sizeof(data_lsb), 1000);
195     snprintf(buf, sizeof(buf), "(LSB) HAL_I2C_Master_Receive: status: %u, data_lsb: 0x%02x\r\n", status, data_lsb);
196     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
197
198     // Read temperature MSB
199     uint8_t temperature_msb = 0x2b;
200     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_msb, sizeof(temperature_msb), 1000);
201     snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
202     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
203
204     uint8_t data_msb = 0x42;
205     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_msb, sizeof(data_msb), 1000);
206     snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%02x\r\n", status, data_msb);
207     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
```

Step 13. I2C polling continue

```
main.c × stm32l4xx_it.c × stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
196 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
197 Embedded-C-Final-Project/Core/Src/stm32l4xx_it.c
198 // Read temperature MSB
199 uint8_t temperature_msb = 0x2b;
200 status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_msb, sizeof(temperature_msb), 1000);
201 snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Transmit: status: %u\r\n", status);
202 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
203
204 uint8_t data_msb = 0x42;
205 status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data_msb, sizeof(data_msb), 1000);
206 snprintf(buf, sizeof(buf), "(MSB) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%02x\r\n", status, data_msb);
207 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
208 }
209 else
210 {
211     toggle = 1;
212     // Read using auto increment
213     uint8_t temperature_lsb = 0x2a | 0x80;
214     status = HAL_I2C_Master_Transmit(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb), 1000);
215     snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit: status: %u\r\n", status);
216     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
217
218     uint16_t data = 0x4242;
219     status = HAL_I2C_Master_Receive(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data), 1000);
220     snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive: status: %u, data_msb: 0x%04x\r\n", status, data);
221     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
222 }
223
224 }
```

Step 14. Test the I2C polling, test is successful

VT COM4 - Tera Term VT

File Edit Setup Control Window Help

```
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), 6=ADC Reading(Interrupt)
cmd>
<One-shot enable> HAL_I2C_Master_Transmit: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit: status: 0
Status register: 0x03
New data available!
<LSB> HAL_I2C_Master_Transmit: status: 0
<LSB> HAL_I2C_Master_Receive: status: 0, data_lsb: 0xc7
<MSB> HAL_I2C_Master_Transmit: status: 0
<MSB> HAL_I2C_Master_Receive: status: 0, data_msb: 0x00
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), 6=ADC Reading(Interrupt)
cmd>
<One-shot enable> HAL_I2C_Master_Transmit: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit: status: 0
Status register: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit: status: 0
<Auto increment> HAL_I2C_Master_Receive: status: 0, data_msb: 0x00c6
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), 6=ADC Reading(Interrupt)
cmd> █
```

Step 15. I2C temperature reading interrupt mode coding

```
main.c x stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
225
226 void do_temp_interrupt()
227 {
228     irq_complete = 0;
229     char buf[100];
230     // Setup control register 1
231     uint8_t control_reg1 = 0x20;
232     uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
233     HAL_StatusTypeDef status;
234     status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1));
235     snprintf(buf, sizeof(buf), "(Control register 1) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
236     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
237     while(0 == irq_complete)
238     {
239         HAL_Delay(1000);
240     }
241
242     irq_complete = 0;
243     // Start a conversion but interrupt driven
244     int control_reg2 = 0x21;
245     uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
246     status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
247     snprintf(buf, sizeof(buf), "(One-shot Enable) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
248     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
249     while(0 == irq_complete)
250     {
251         HAL_Delay(1000);
252     }
253
254     irq_complete = 0;
255     // Wait for conversion complete
256     uint8_t status_reg = 0x27;
257     int count = 0;
258     while(count < 10)
259     {
260         // Send read status register sub command
261         status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
262         snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_IT: status: %u\r\n", count, status);
263         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
```


Step 16. I2C temperature reading interrupt mode coding continue

```
main.c × stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
261 status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
262 snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_IT: status: %u\r\n", count, status);
263 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
264 while(0 == irq_complete)
265 {
266     HAL_Delay(1000);
267 }
268
269 irq_complete = 0;
270 status_flag = 1;
271 // Read conversion status
272 status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
273 while(0 == irq_complete)
274 {
275     HAL_Delay(1000);
276 }
277
278 // Check for temperature conversion complete
279 if(status_data & 0x1)
280 {
281     snprintf(buf, sizeof(buf), "New data available!\r\n");
282     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
283     break;
284 }
285 HAL_Delay(1000);
286 count++;
287 }
288
289 irq_complete = 0;
290 // Read using auto increment
291 uint8_t temperature_lsb = 0x2a | 0x80;
292 status = HAL_I2C_Master_Transmit_IT(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb));
293 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit_IT: status: %u\r\n", status);
294 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
295 while(0 == irq_complete)
296 {
297     HAL_Delay(1000);
298 }
299
300 irq_complete = 0;
301 // Receive using interrupt
302 status = HAL_I2C_Master_Receive_IT(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data));
303 while(0 == irq_complete)
304 {
305     HAL_Delay(1000);
306 }
307 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_IT: status: %u\r\n", status);
308 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
309 }
310 }
```

Step 20. Test the code, test is successful

COM4 - Tera Term VT

File Edit Setup Control Window Help

```
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), 6=ADC Reading(Interrupt)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_IT: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_IT: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit_IT: status: 0
<Status register receive> HAL_I2C_Master_Receive_Callback: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_IT: status: 0
<Auto increment> HAL_I2C_Master_Receive_Callback: data: 0x00cf
<Auto increment> HAL_I2C_Master_Receive_IT: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), 6=ADC Reading(Interrupt)
cmd> █
```

Step 18. I2C temperature reading DMA mode coding

```
main.c x stm3214xx_it.c stm3214xx_hal_adc.c startup_stm321475vgtx.s stm3214xx_hal_uart.c
311 void do_temp_dma()
312 {
313     irq_complete = 0;
314     char buf[100];
315     // Setup control register 1
316     uint8_t control_reg1 = 0x20;
317     uint8_t control_data1[] = {control_reg1, 0x85}; // output registers not updated until MSB and LSB reading, 1 Hz
318     HAL_StatusTypeDef status;
319     status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, control_data1, sizeof(control_data1));
320     snprintf(buf, sizeof(buf), "(Control register 1) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
321     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
322     while(0 == irq_complete)
323     {
324         HAL_Delay(1000);
325     }
326
327     irq_complete = 0;
328     // Start a conversion but interrupt driven
329     int control_reg2 = 0x21;
330     uint8_t control_data2[] = {control_reg2, 0x01}; // One-shot enable
331     status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, control_data2, sizeof(control_data2));
332     snprintf(buf, sizeof(buf), "(One-shot Enable) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
333     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
334     while(0 == irq_complete)
335     {
336         HAL_Delay(1000);
337     }
338     irq_complete = 0;
339     // Wait for conversion complete
340     uint8_t status_reg = 0x27;
341     int count = 0;
342     while(count < 10)
343     {
344         // Send read status register sub command
345         status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, &status_reg, sizeof(status_reg));
346         snprintf(buf, sizeof(buf), "[%d] (status_reg) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", count, status);
347         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
348         while(0 == irq_complete)
```

Step 19. I2C temperature reading DMA mode coding continue

```
main.c × stm32l4xx_it.c stm32l4xx_hal_adc.c startup_stm32l475vgtx.s stm32l4xx_hal_uart.c
347 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
348 while(0 == irq_complete)
349 {
350     HAL_Delay(1000);
351 }
352
353 irq_complete = 0;
354 status_flag = 1;
355 // Read conversion status
356 status = HAL_I2C_Master_Receive_DMA(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&status_data, sizeof(status_data));
357 while(0 == irq_complete)
358 {
359     HAL_Delay(1000);
360 }
361
362 // Check for temperature conversion complete
363 if(status_data & 0x1)
364 {
365     snprintf(buf, sizeof(buf), "New data available!\r\n");
366     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
367     break;
368 }
369 HAL_Delay(1000);
370 count++;
371 }
372 irq_complete = 0;
373 // Read using auto increment
374 uint8_t temperature_lsb = 0x2a | 0x80;
375 status = HAL_I2C_Master_Transmit_DMA(&hi2c2, HST221_WRITE_ADDRESS, &temperature_lsb, sizeof(temperature_lsb));
376 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Transmit_DMA: status: %u\r\n", status);
377 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
378 while(0 == irq_complete)
379 {
380     HAL_Delay(1000);
381 }
382 irq_complete = 0;
383 // Receive using interrupt
384 status = HAL_I2C_Master_Receive_DMA(&hi2c2, HST221_READ_ADDRESS, (uint8_t*)&data, sizeof(data));
385 while(0 == irq_complete)
386 {
387     HAL_Delay(1000);
388 }
389 snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_DMA: status: %u\r\n", status);
390 HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
391 }
```

Step 20. Test the code, test is successful

```
VT COM4 - Tera Term VT
File Edit Setup Control Window Help
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), 6=ADC Reading(Interrupt)
cmd>
<Control register 1> HAL_I2C_Master_Transmit_DMA: status: 0
<One-shot Enable> HAL_I2C_Master_Transmit_DMA: status: 0
[0] <status_reg> HAL_I2C_Master_Transmit_DMA: status: 0
<Status register receive> HAL_I2C_Master_Receive_Callback: data: 0x03
New data available!
<Auto increment> HAL_I2C_Master_Transmit_DMA: status: 0
<Auto increment> HAL_I2C_Master_Receive_Callback: data: 0x00d3
<Auto increment> HAL_I2C_Master_Receive_DMA: status: 0
Options: 1=WHO_AM_I, 2=Temp(Polling), 3=Temp(Interrupt), 4=Temp(DMA), 5=ADC Reading(Polling), 6=ADC Reading(Interrupt)
cmd> █
```

Step 21. ADC polling reading coding

```
393 void adc_polling_average()
394 {
395     HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
396     HAL_Delay(1000);
397
398     uint16_t value[12];
399     uint16_t value_sum = 0;
400
401     // Poll 12 times
402     for(int i = 0; i < 12; i++)
403     {
404         // ADC start conversion
405         HAL_ADC_Start(&hadc1);
406         // Poll for results, timeout is 10 us
407         HAL_ADC_PollForConversion(&hadc1, 10);
408         value[i] = HAL_ADC_GetValue(&hadc1);
409         value_sum += value[i];
410     }
411
412     // Sum of value array minus the max and min value
413     float value_avg = (float)(value_sum - find_max(value, 12) - find_min(value, 12)) / 10;
414
415     float voltage = value_avg * (3.3 / 4096);
416
417     // Send value to console
418     char buf[100];
419     snprintf(buf, sizeof(buf), "ARD-A0: raw: %f, volts: %f\r\n", value_avg, voltage);
420
421     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
422 }
423
```

Step 21. ADC polling reading coding continue, find max and min function

```
93 uint16_t find_max(uint16_t arr[], uint8_t size)
94 {
95     int max = arr[0];
96     for(int i = 0; i < size; i++){
97         if(arr[i] > max){
98             max = arr[i];
99         }
100     }
101     return max;
102 }
103
104 uint16_t find_min(uint16_t arr[], uint8_t size)
105 {
106     int min = arr[0];
107     for(int i = 0; i < size; i++){
108         if(arr[i] < min){
109             min = arr[i];
110         }
111     }
112     return min;
113 }
114
```

Step 22. Test the code, test is successful

```
VT COM4 - Tera Term VT
File Edit Setup Control Window Help
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd>
ARD-A0: raw: 1997.500000, volts: 1.609314
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd>
ARD-A0: raw: 2002.000000, volts: 1.612939
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd> █
```


Step 22. Test the code, test is successful

```
VT COM4 - Tera Term VT
File Edit Setup Control Window Help
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd>
ARD-A0: raw: 1997.500000, volts: 1.609314
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd>
ARD-A0: raw: 2002.000000, volts: 1.612939
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd> █
```

Step 23. ADC interrupt coding

```
424 void adc_interrupt()  
425 {  
426     adc_irq_complete = 0;  
427     HAL_ADC_Start_IT(&hadc1);  
428     while(adc_irq_complete == 0){  
429         HAL_Delay(1000);  
430     }  
431     char buf[100];  
432     snprintf(buf, sizeof(buf), "ADC conversion interrupt mode done\r\n");  
433     HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);  
434 }  
435
```

Step 24. Test the code, test is successful

```
VT COM4 - Tera Term VT
File Edit Setup Control Window Help
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd>
ARD-A0: raw: 1983, volts: 1.597632
ADC conversion interrupt mode done
Options: 1=WHO_AM_I, 2=Temp<Polling>, 3=Temp<Interrupt>, 4=Temp<DMA>, 5=ADC Reading<Polling>, 6=ADC Reading<Interrupt>
cmd> █
```

Step 25. Interrupt call back function implementation for I2C

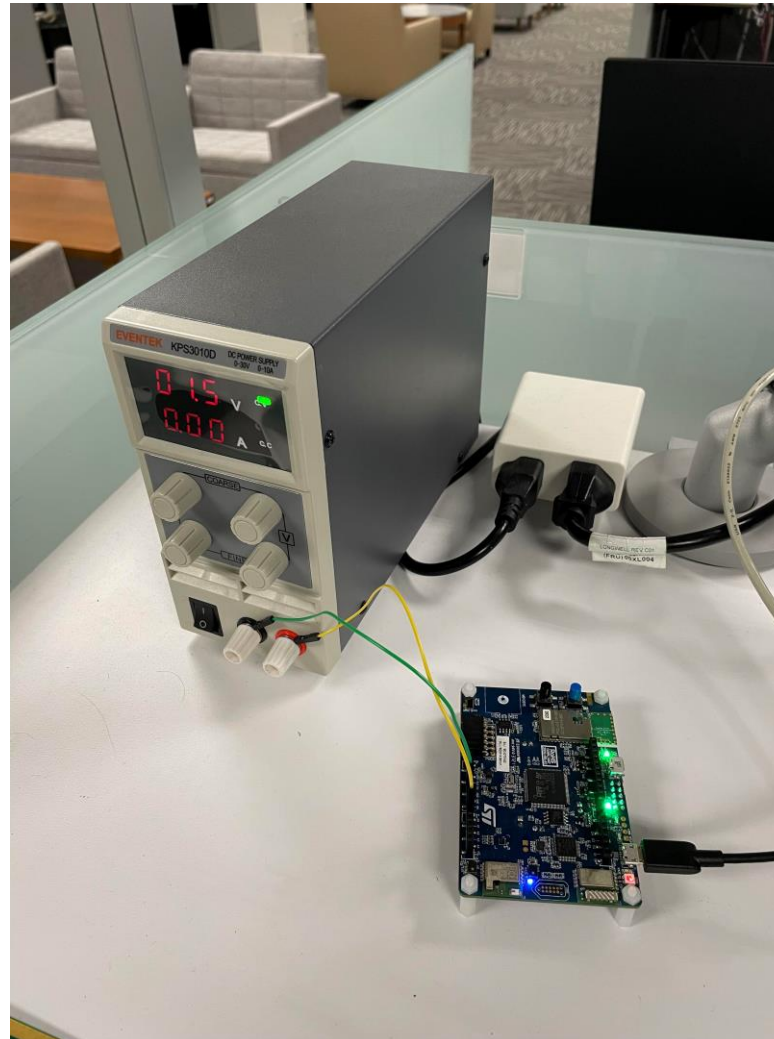
```
1122 /* USER CODE BEGIN 4 */
1123 void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)
1124 {
1125     HAL_GPIO_TogglePin(LED2_GPIO_Port, LED2_Pin);
1126     irq_complete = 1;
1127 }
1128
1129 void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c)
1130 {
1131     char buf[100];
1132     if(status_flag == 1)
1133     {
1134         status_flag = 0;
1135         snprintf(buf, sizeof(buf), "(Status register receive) HAL_I2C_Master_Receive_Callback: data: 0x%02x\r\n", status_data);
1136         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
1137     }
1138     else
1139     {
1140         snprintf(buf, sizeof(buf), "(Auto increment) HAL_I2C_Master_Receive_Callback: data: 0x%04x\r\n", data);
1141         HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
1142     }
1143     irq_complete = 1;
1144 }
```

Step 26. Interrupt call back function implementation for ADC

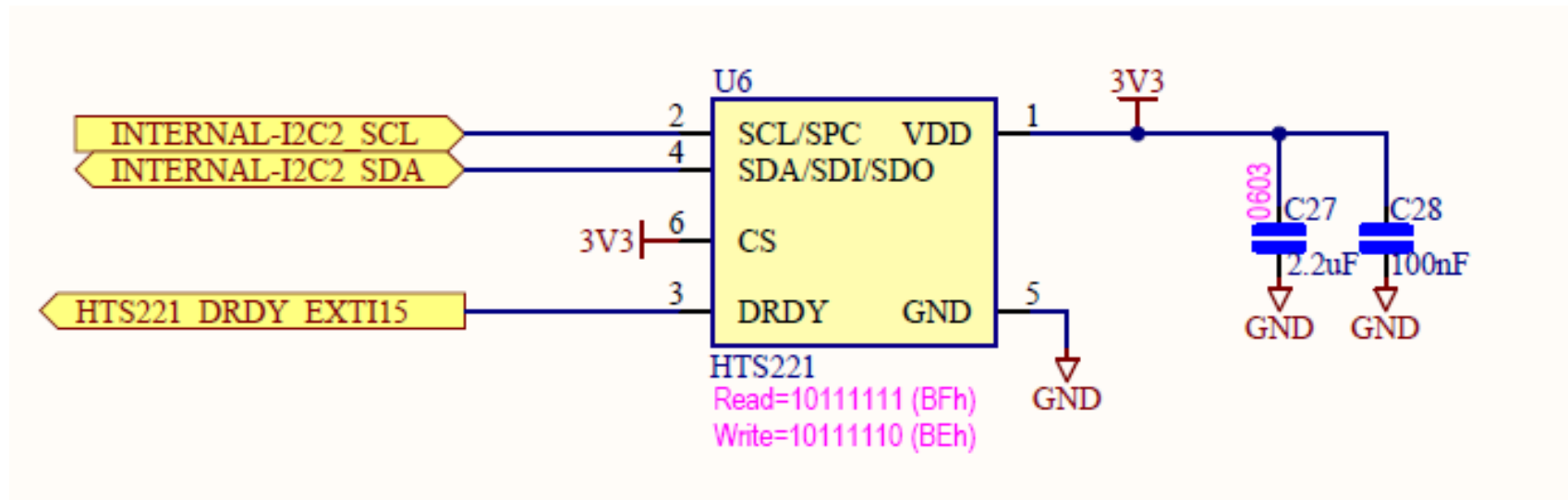
```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
    uint16_t value = HAL_ADC_GetValue(&hadc1);
    float voltage = value * (3.3 / 4096);

    // Send value to console
    char buf[100];
    snprintf(buf, sizeof(buf), "ARD-A0: raw: %u, volts: %f\r\n", value, voltage);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), 1000);
    adc_irq_complete = 1;
}
```

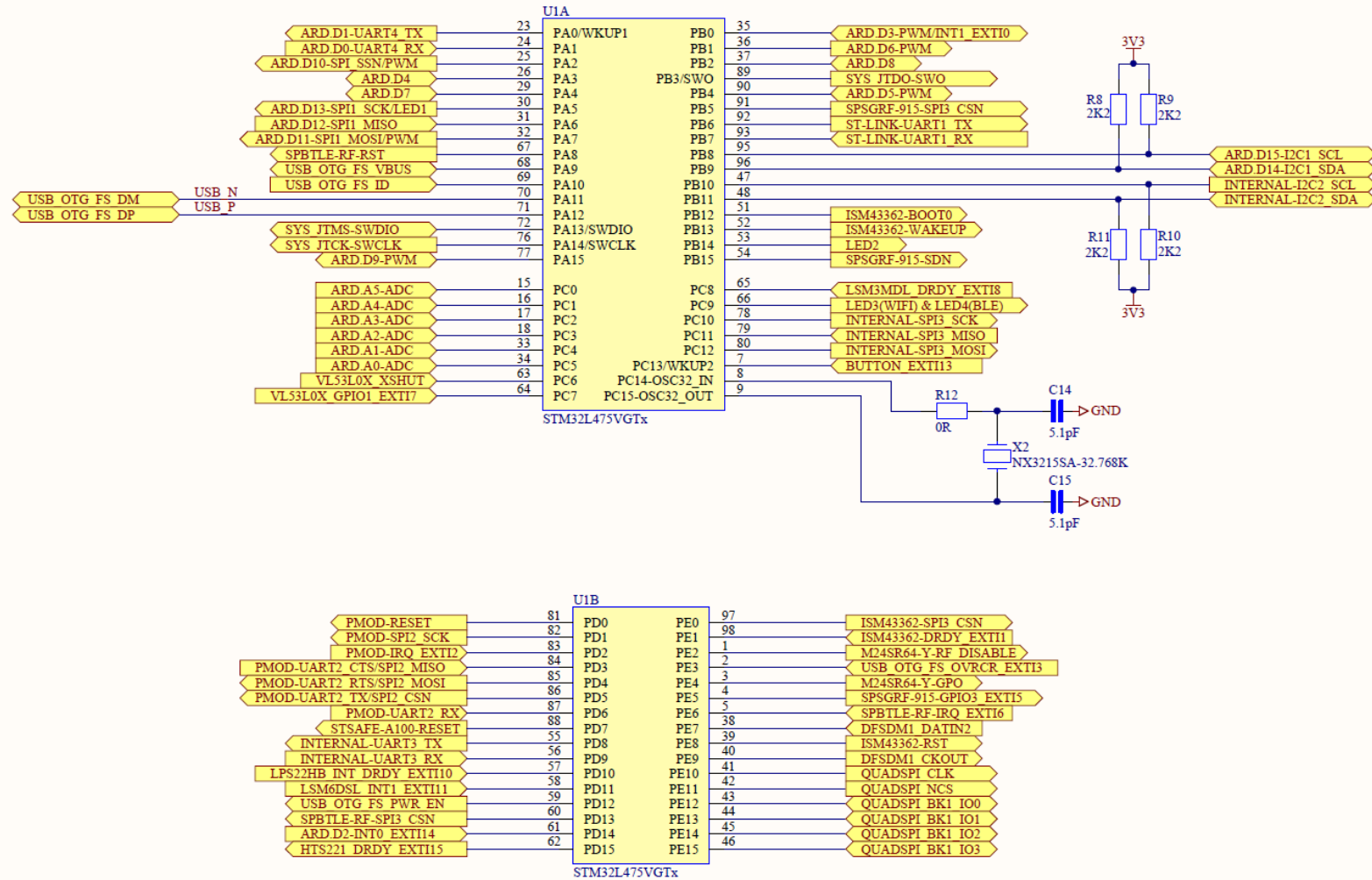
Appendix, wire connection to ARD-A0 pin with 1.5V, note that we actually read 1.6v, since the power supply is not very accurate



Appendix, schematic for HTS221



Appendix, schematic for processor side



Appendix, schematic for ST-LINK UART1

