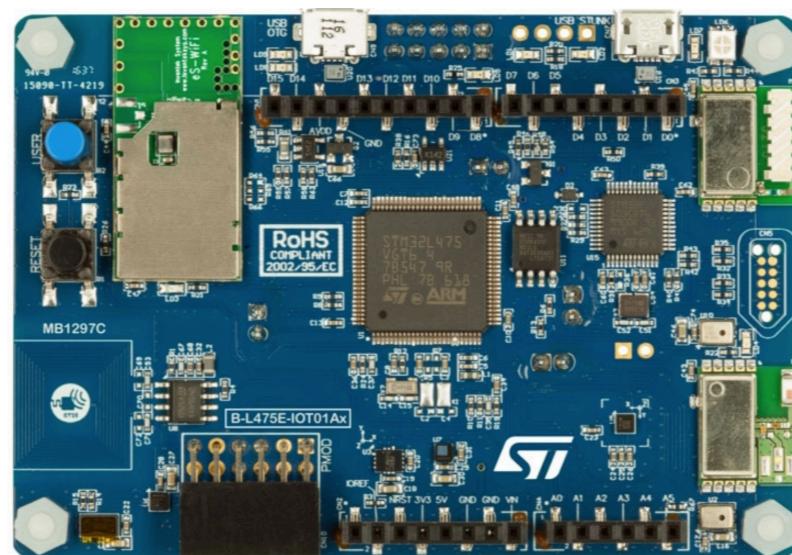


Embedded Systems Hardware Interfacing **ADC and DAC**

Norman McEntire



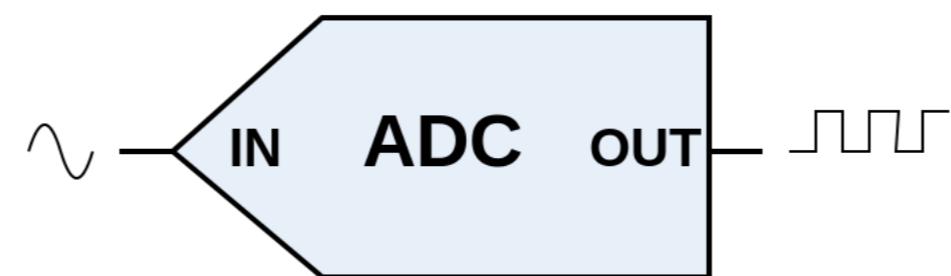
Contents

- Concepts
- Data Sheet - STM32L475
- User Manual - UM2153 - STM32L Discovery Kit for IoT
- Schematics - STM32L Discovery Kit for IoT
- API - STM32L HAL
 - Data Structures
 - Functions
- Hands-On Project

References

- https://en.wikipedia.org/wiki/Analog-to-digital_converter
- https://en.wikipedia.org/wiki/Digital-to-analog_converter
- https://www.st.com/resource/en/schematic_pack/b-l475e-iot01ax_sch.zip
- https://www.st.com/resource/en/user_manual/dm00347848-discovery-kit-for-iot-node-multichannel-communication-with-stm32l4-stmicroelectronics.pdf
- <https://www.st.com/resource/en/datasheet/stm32l475vg.pdf>

ADC



ADC Concepts

Analog to Digital Converter

- ADC Converts an analog signal into a digital signal
 - Example: Analog Sound Signal
 - Example: Analog Temperature Signal
- ADC converts continuous time and continuous amplitude to
 - Discrete time and discrete amplitude

ADC Concepts

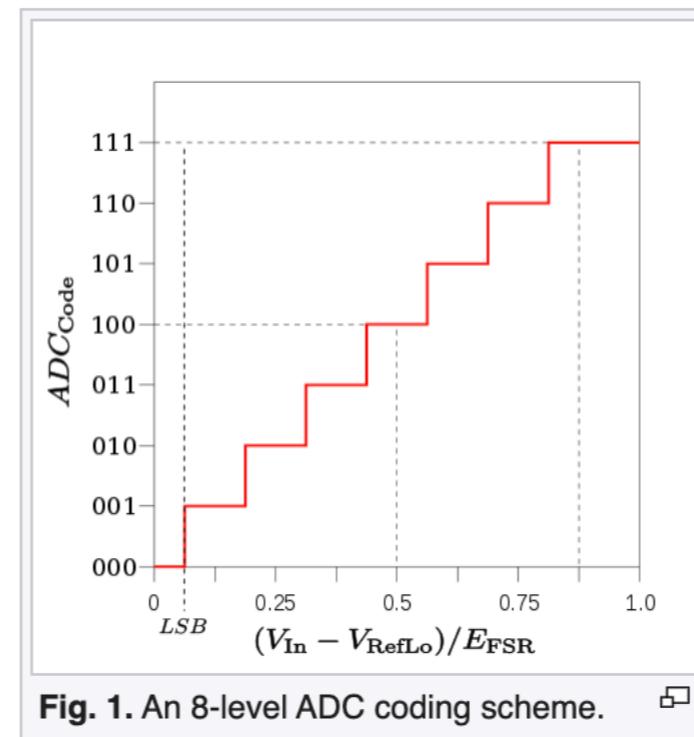
ADC Performance

- Performance ADC influenced by
 - Bandwidth
 - Sampling rate
 - SNR - Signal to noise ratio

ADC Concepts

Resolution - Bits

- The number of different discrete values produced over an analog range
 - Example: 8-bits give 256 different values



ADC Concepts

Resolution - Volts

- Resolution can also be expressed in volts
 - Change in voltage to change the output code
 - LSB (Least Significant Bit) Voltage
 - The resolution Q of ADC is equal to LSB voltage
 - $Q = \text{Full Scale Voltage} / 2^m$
 - Example: 3.3V / 12-bits

ADC Concepts

Oversampling

- Samples are often sampled at minimum rate required - the Nyquist rate
- With oversampling, sampling is higher than minimum required Nyquist rate
- Oversampling is typically used in audio frequency ADCs
 - Required sampling rate is very low compared to digital logic speed
 - Example: Audio sampling at 44.1KHz

ADC Concepts

Singled Ended and Differential

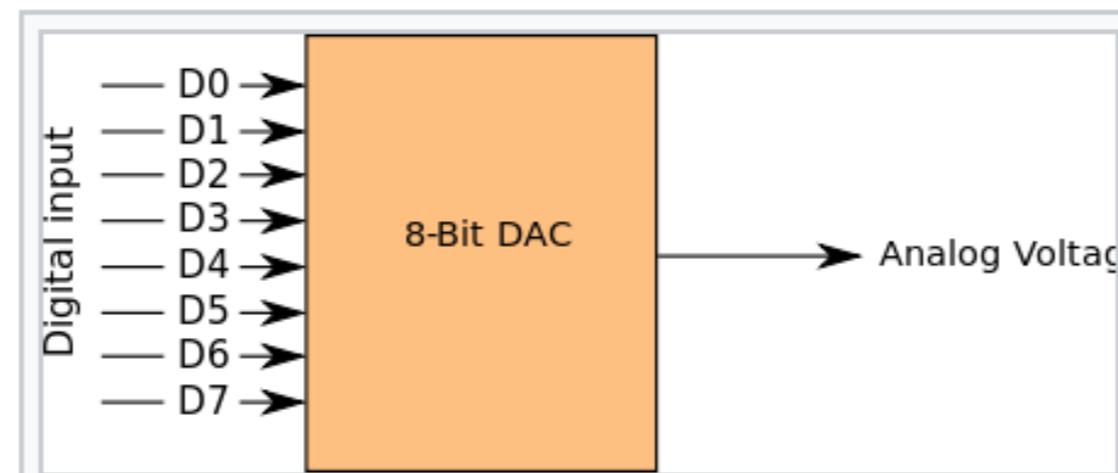
- Single Ended input has single signal
 - + signal
 - More susceptible to noise
- Differential Input has two signal
 - + signal
 - - signal
 - Less susceptible to noise
 - Noise cancels out between + and - signals

ADC Concepts

Interrupts

- ADC conversion can be free running
- Or can be interrupt driven

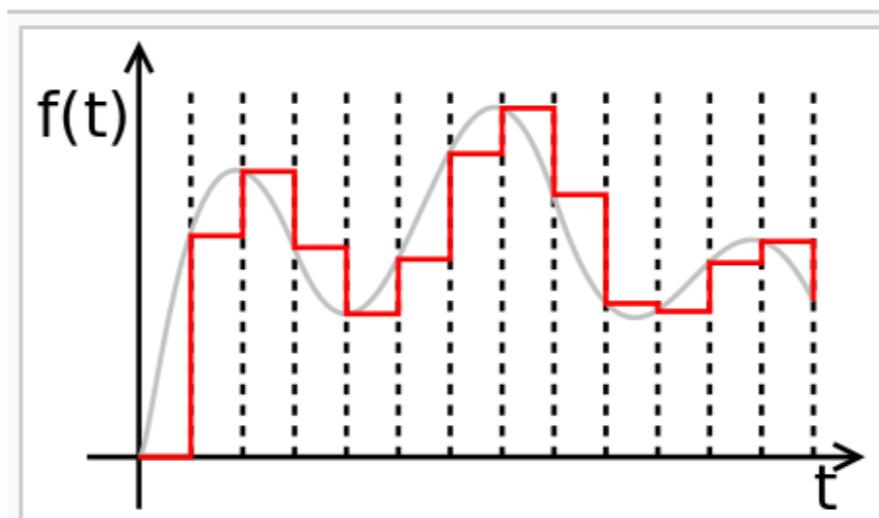
DAC



A simplified functional diagram of an
8-bit DAC

DAC Concepts

- DAC - Digital to Analog Conversion
 - Convert digital signal to analog signal
 - DACs convert discrete time and discrete values into continuous time and continuous values



Data Sheet

STM32I475

STM32L475 Data Sheet



STM32L475xx

**Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS,
up to 1MB Flash, 128 KB SRAM, USB OTG FS, analog, audio**

Datasheet - production data

- Rich analog peripherals (independent supply)
 - 3x 12-bit ADC 5 Msps, up to 16-bit with hardware oversampling, 200 μ A/Msp
 - 2x 12-bit DAC output channels, low-power sample and hold

3.15	Analog to digital converter (ADC)	41
3.15.1	Temperature sensor	41
3.15.2	Internal voltage reference (VREFINT)	42
3.15.3	VBAT battery voltage monitoring	42
3.16	Digital to analog converter (DAC)	42

STM32L475

Connectivity USB OTG 1x SD/SDIO/MMC, 3x SPI, 3x I ² C, 1x CAN, 1x Quad SPI, 5x USART + 1 x ULP UART, 1 x SWP	ARM® Cortex®-M4 CPU 80 MHz FPU MPU ETM	Timers 17 timers including: 2 x 16-bit advanced motor control timers 2 x ULP timers 7 x 16-bit-timers 2 x 32-bit timers
Digital TRNG, 2 x SAI, DFSDM (8 channels)	DMA ART Accelerator™ Up to 1-Mbyte Flash with ECC Dual Bank 128-Kbyte RAM	Analog 3x 16-bit ADC, 2 x DAC, 2 x comparators, 2 x Op amps 1 x Temperature sensor
I/Os Up to 114 I/Os Touch-sensing controller		Parallel Interface FSMC 8-/16-bit (TFT-LCD, SRAM, NOR, NAND)

3.15

Analog to digital converter (ADC)

The device embeds 3 successive approximation analog-to-digital converters with the following features:

- 12-bit native resolution, with built-in calibration
- 5.33 Msps maximum conversion rate with full resolution
 - Down to 18.75 ns sampling time
 - Increased conversion rate for lower resolution (up to 8.88 Msps for 6-bit resolution)
- Up to 24 external channels, some of them shared between ADC1 and ADC2, or ADC1, ADC2 and ADC3.
- 5 internal channels: internal reference voltage, temperature sensor, VBAT/3, DAC1_OUT1 and DAC1_OUT2.
- One external reference pin is available on some package, allowing the input voltage range to be independent from the power supply
- Single-ended and differential mode inputs

- Highly versatile digital interface
 - Single-shot or continuous/discontinuous sequencer-based scan mode: 2 groups of analog signals conversions can be programmed to differentiate background and high-priority real-time conversions
 - Handles two ADC converters for dual mode operation (simultaneous or interleaved sampling modes)
 - Each ADC support multiple trigger inputs for synchronization with on-chip timers and external signals
 - Results stored into 3 data register or in RAM with DMA controller support
 - Data pre-processing: left/right alignment and per channel offset compensation
 - Built-in oversampling unit for enhanced SNR
 - Channel-wise programmable sampling time
 - Three analog watchdog for automatic voltage monitoring, generating interrupts and trigger for selected timers
 - Hardware assistant to prepare the context of the injected channels to allow fast context switching

3.15.1 Temperature sensor

The temperature sensor (TS) generates a voltage V_{TS} that varies linearly with temperature.

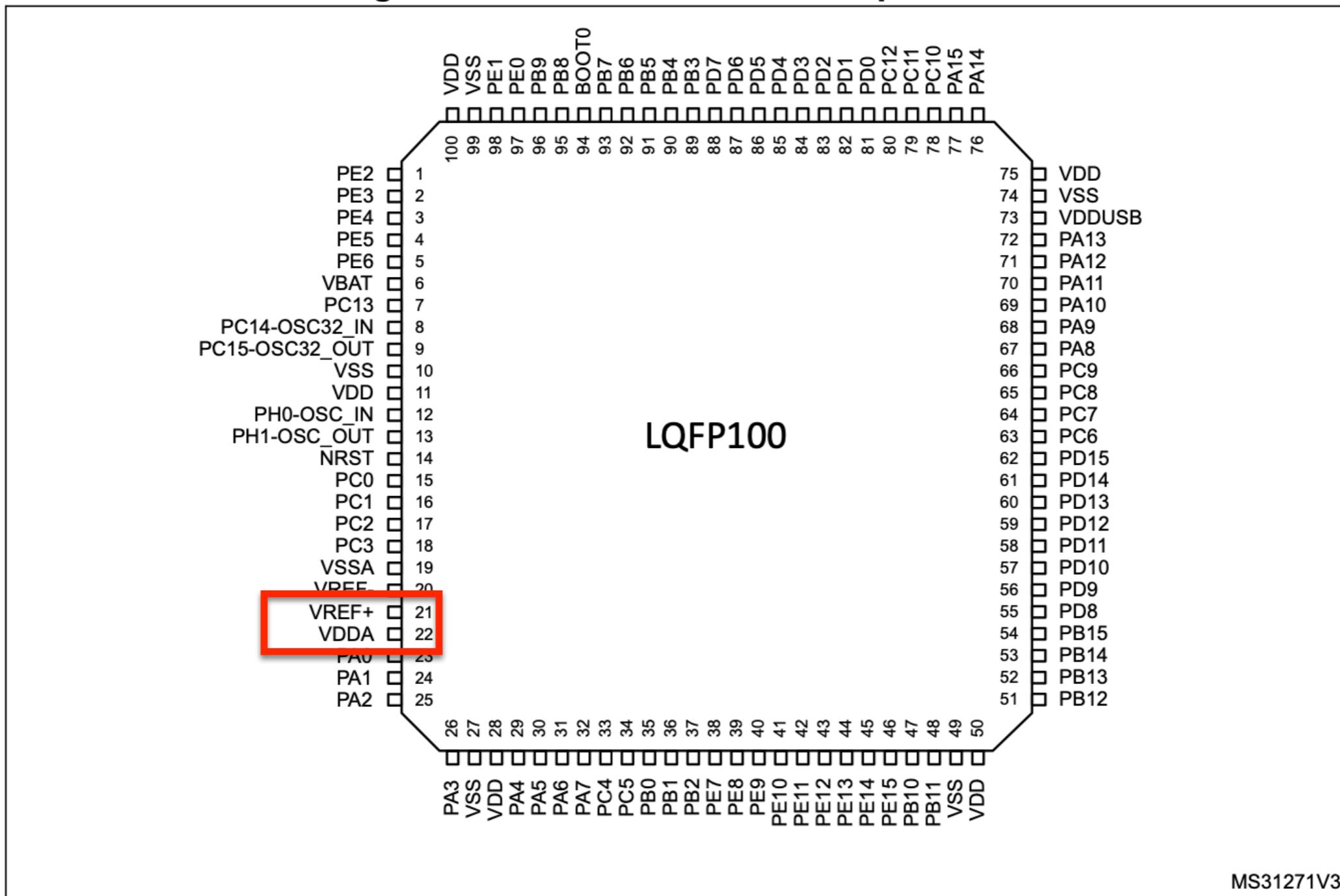
The temperature sensor is internally connected to the ADC1_IN17 and ADC3_IN17 input channels which is used to convert the sensor output voltage into a digital value.

The sensor provides good linearity but it has to be calibrated to obtain good overall accuracy of the temperature measurement. As the offset of the temperature sensor varies from chip to chip due to process variation, the uncalibrated internal temperature sensor is suitable for applications that detect temperature changes only.

V_{BAT} battery voltage monitoring

This embedded hardware feature allows the application to measure the V_{BAT} battery voltage using the internal ADC channel ADC1_IN18 or ADC3_IN18. As the V_{BAT} voltage may be higher than V_{DDA} , and thus outside the ADC input range, the V_{BAT} pin is internally connected to a bridge divider by 3. As a consequence, the converted digital value is one third the V_{BAT} voltage.

Figure 6. STM32L475Vx LQFP100 pinout⁽¹⁾



1. The above figure shows the package top view.

	Pin type	
I/O structure	S	Supply pin
	I	Input only pin
	I/O	Input / output pin
	FT	5 V tolerant I/O
		3.6 V tolerant I/O
		Dedicated BOOT0 pin
		Bidirectional reset pin with embedded weak pull-up resistor

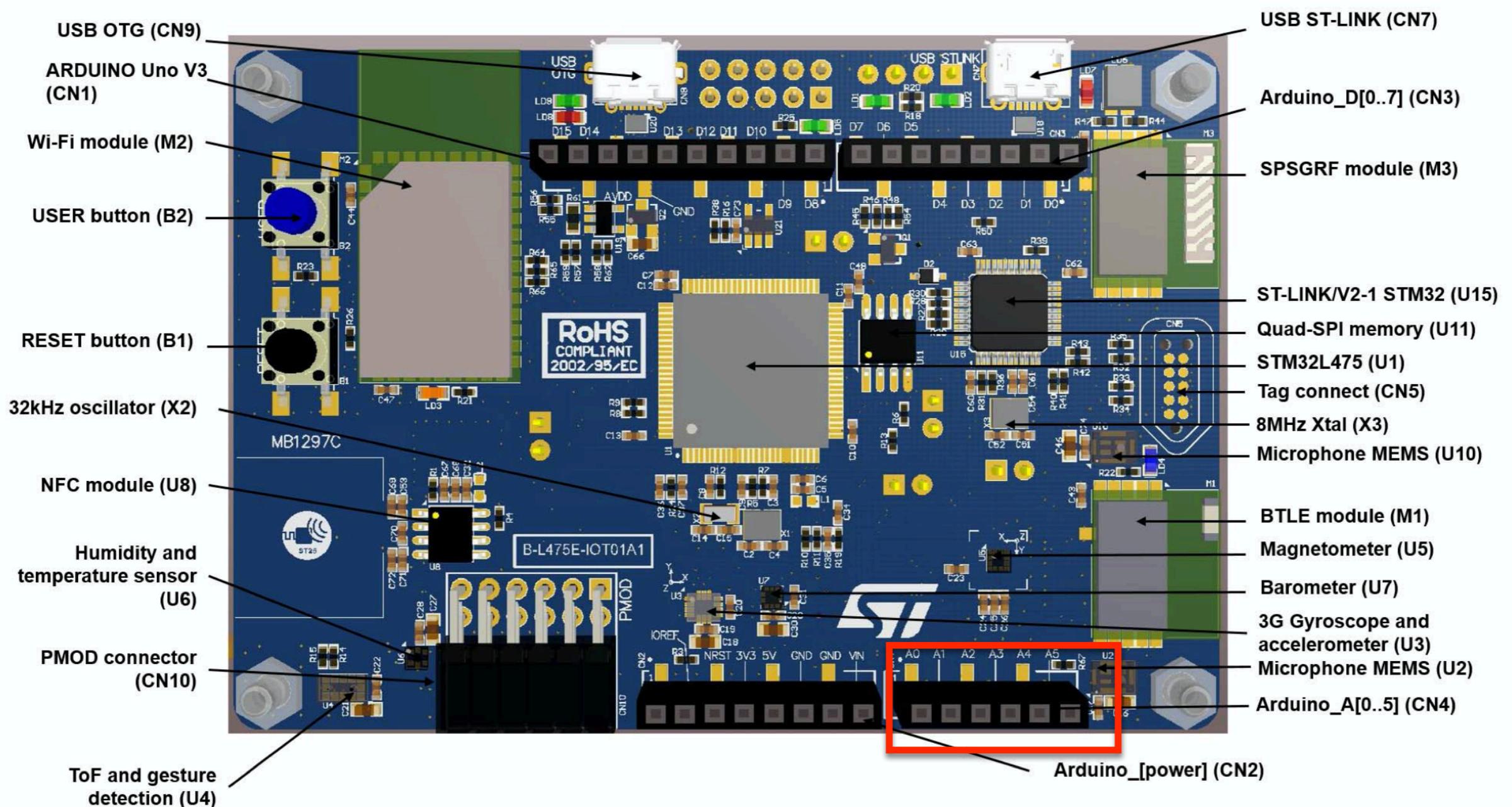
Pin Number	Pin name (function after reset)	Pin type	I/O structure	Notes	Pin functions	
					Alternate functions	Additional functions
LQFP64	LQFP100					
-	11	VDD	S	-	-	-
5	12	PH0-OSC_IN (PH0)	I/O	FT	- EVENTOUT	OSC_IN
6	13	PH1-OSC_OUT (PH1)	I/O	FT	- EVENTOUT	OSC_OUT
7	14	NRST	I/O	RST	-	-
8	15	PC0	I/O	FT_fa	- LPTIM1_IN1, I2C3_SCL, DFSDM1_DATIN4, LPUART1_RX, LPTIM2_IN1, EVENTOUT	ADC123_IN1
9	16	PC1	I/O	FT_fa	- LPTIM1_OUT, I2C3_SDA, DFSDM1_CKIN4, LPUART1_TX, EVENTOUT	ADC123_IN2
10	17	PC2	I/O	FT_a	- LPTIM1_IN2, SPI2_MISO, DFSDM1_CKOUT, EVENTOUT	ADC123_IN3
11	18	PC3	I/O	FT_a	- LPTIM1_ETR, SPI2_MOSI, SAI1_SD_A, LPTIM2_ETR, EVENTOUT	ADC123_IN4

User Manual

STM32L Discovery Kit IoT Node

Use of GPIO

Board View - Buttons and LEDs



Use of ADC

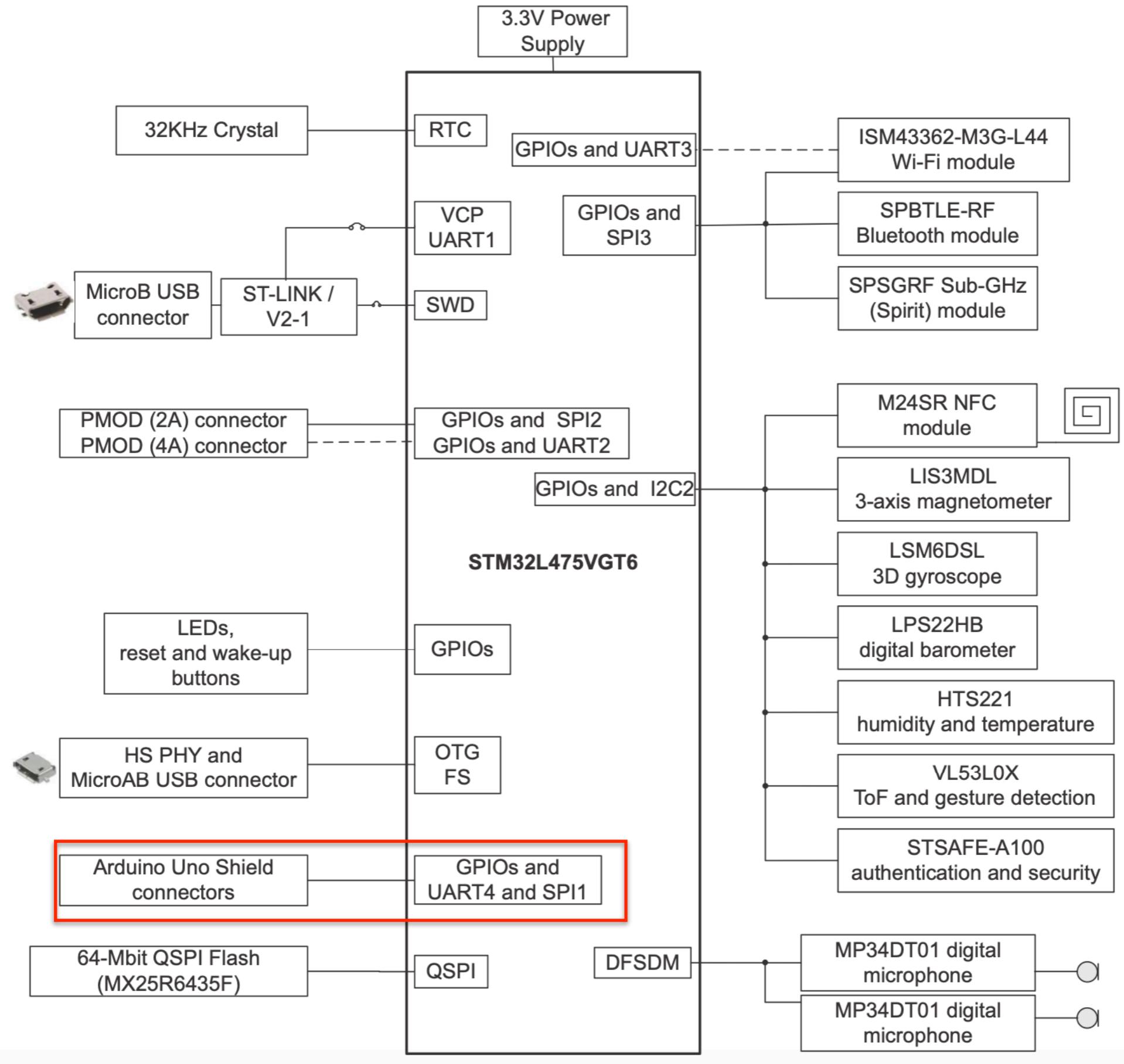
Functional Description

CN4	1	A0	ARD.A0-ADC	PC5	ADC
	2	A1	ARD.A1-ADC	PC4	ADC
	3	A2	ARD.A2-ADC	PC3	ADC
	4	A3	ARD.A3-ADC	PC2	ADC
	5	A4	ARD.A4-ADC	PC1	ADC / I2C3_SDA
	6	A5	ARD.A5-ADC	PC0	ADC / I2C3_SCL

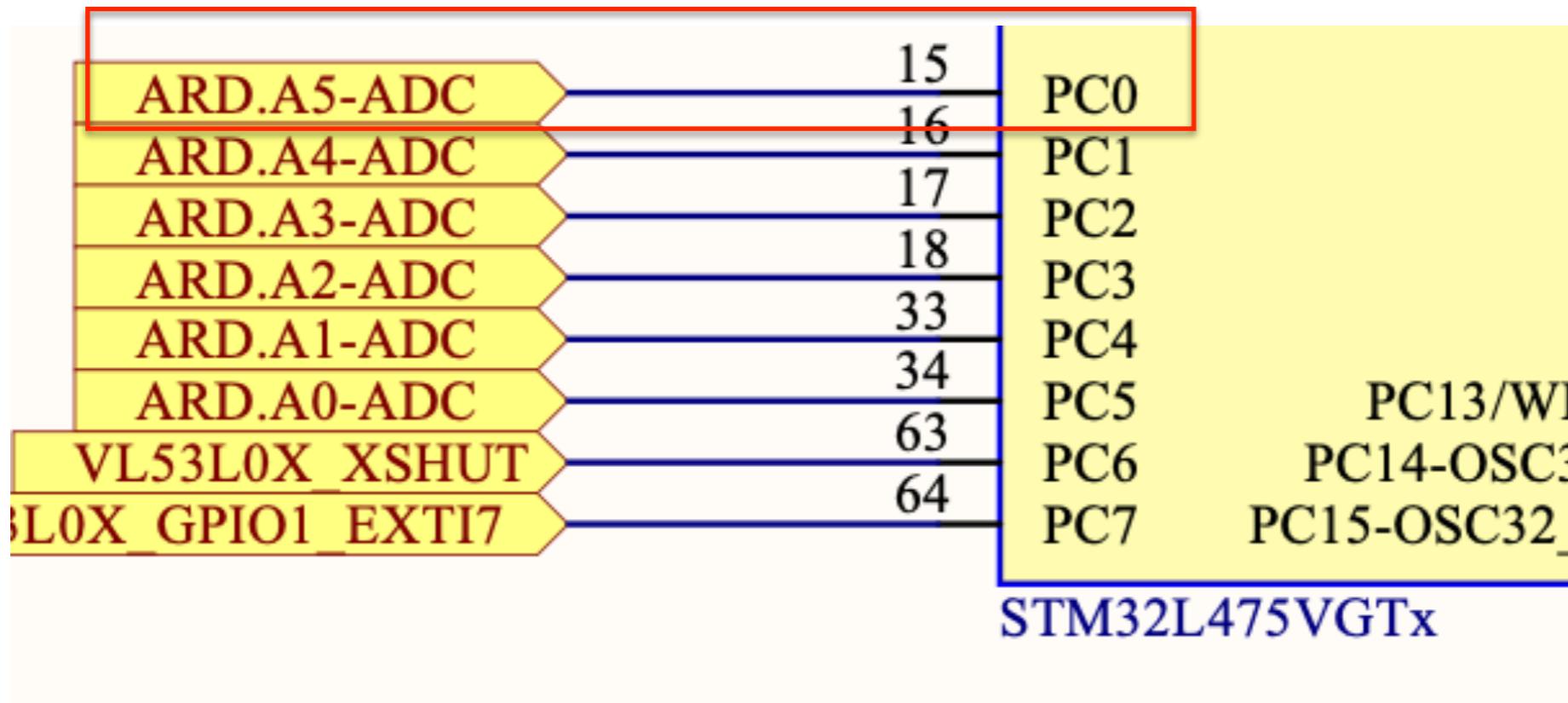
Schematics

STM324L Discovery Kit

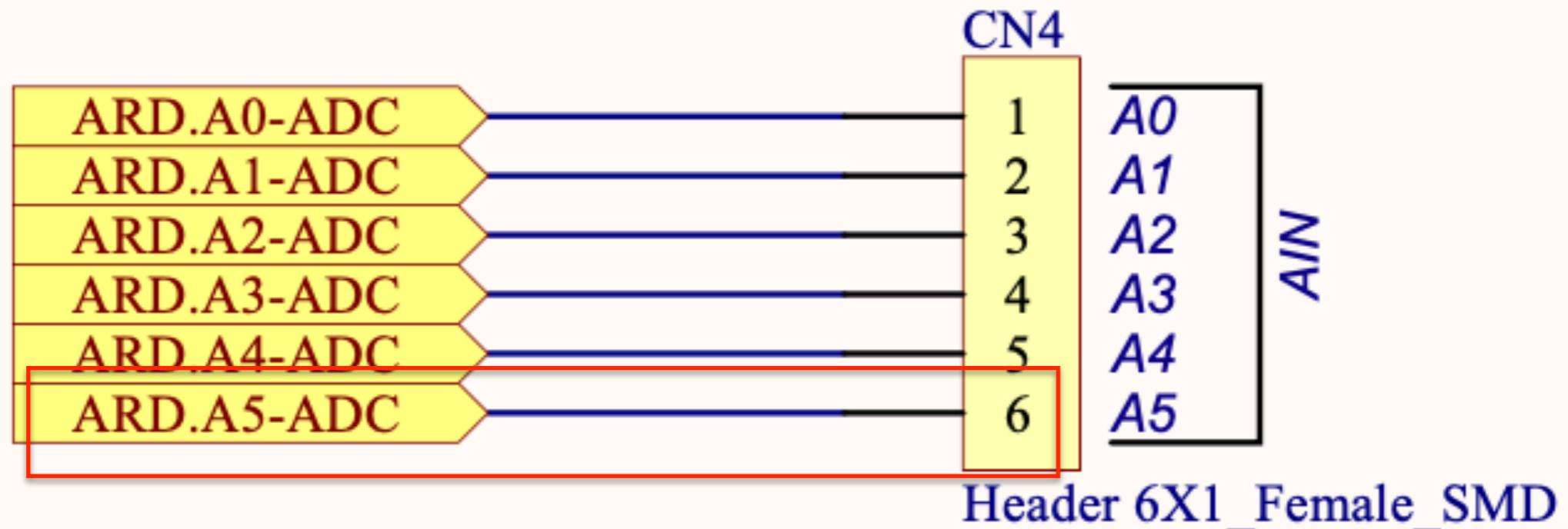
IoT Node



STM32L Discovery Kit IoT Node

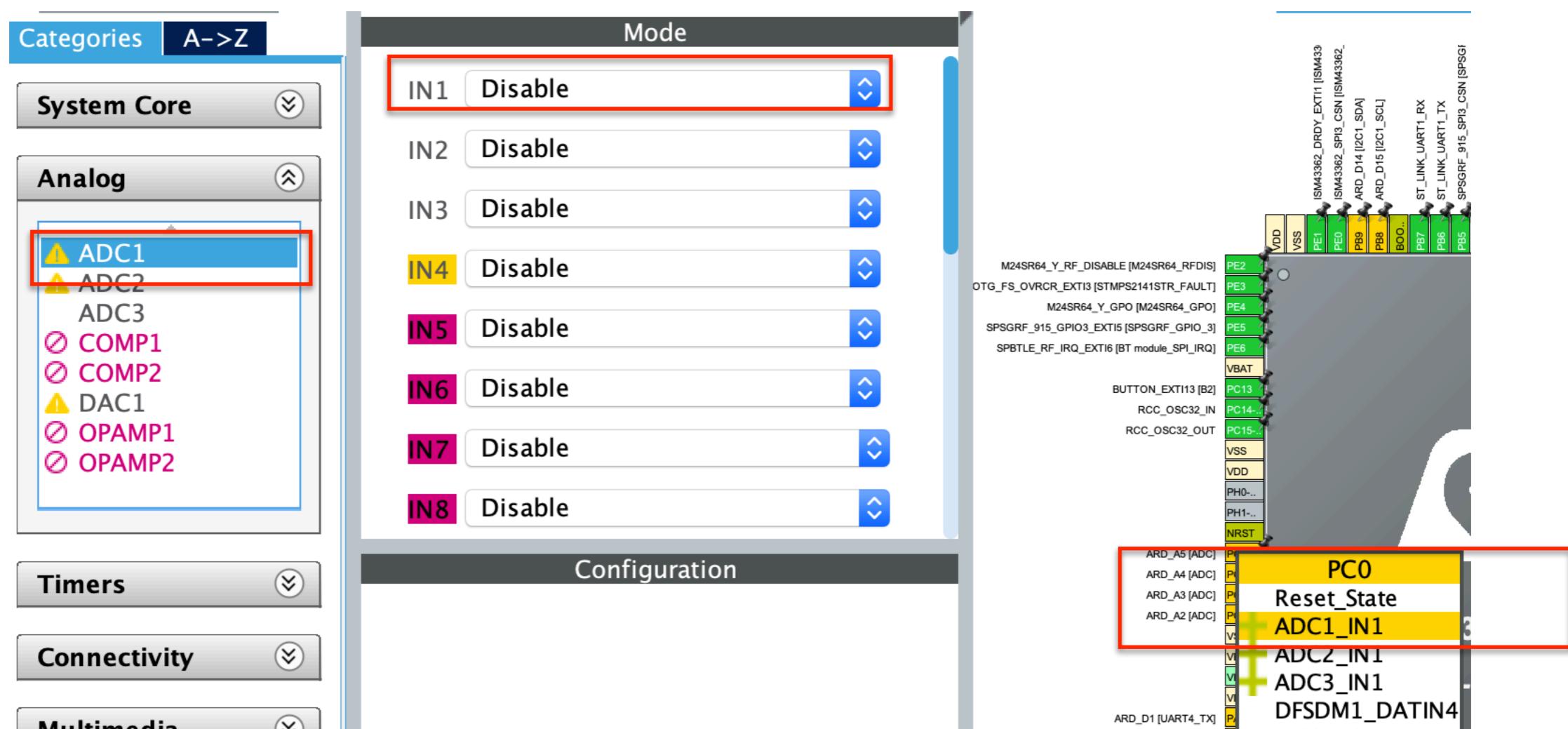


STM32L Discovery Kit IoT Node

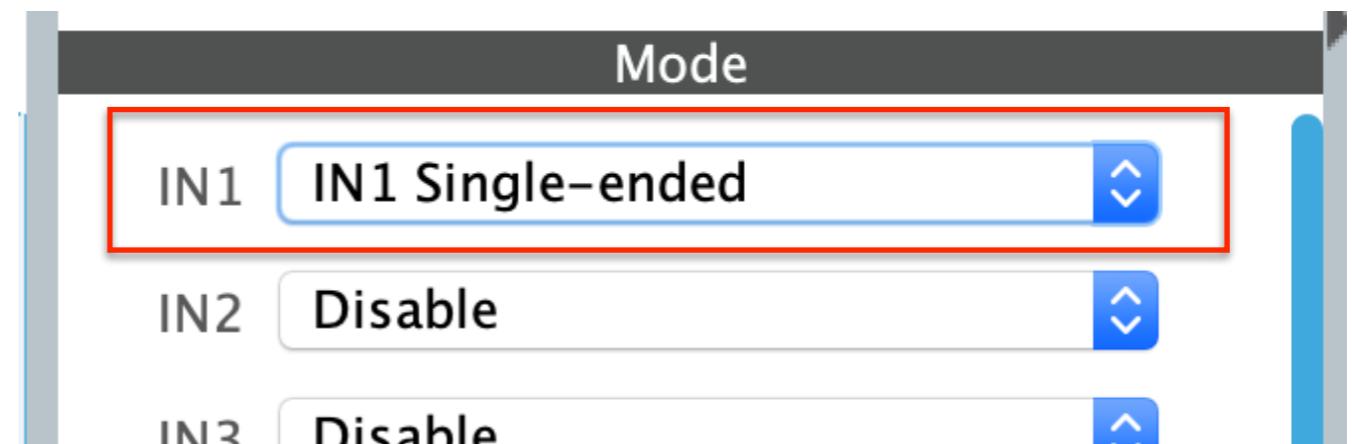
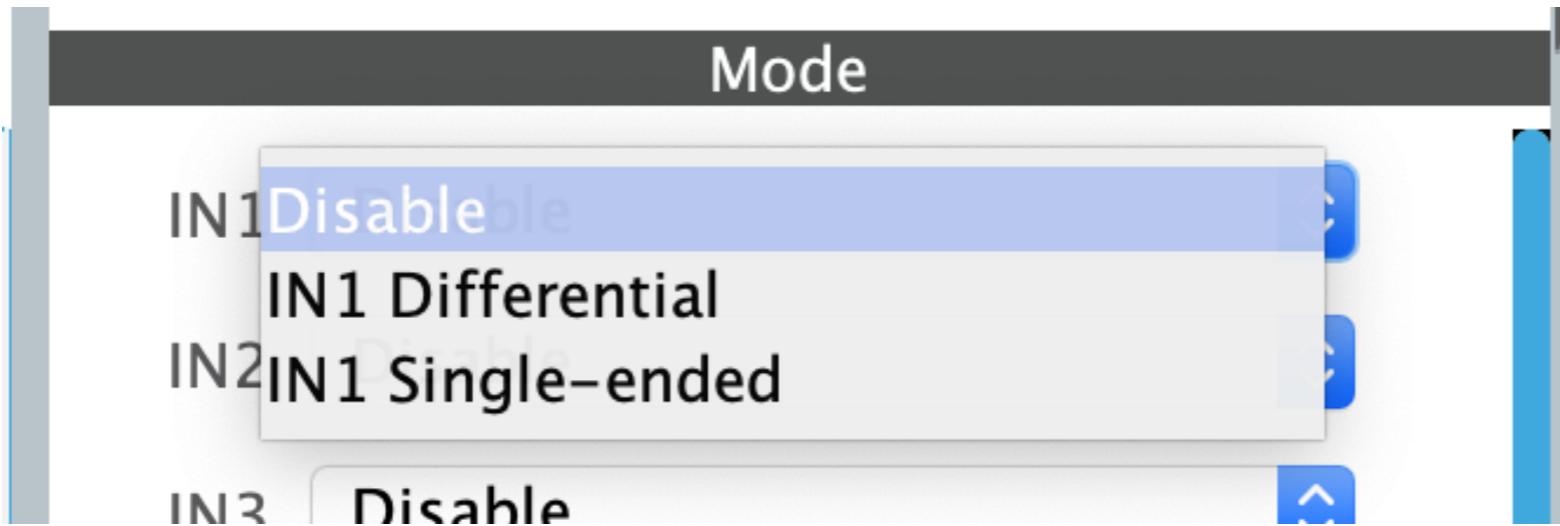


ADC API Data Structures

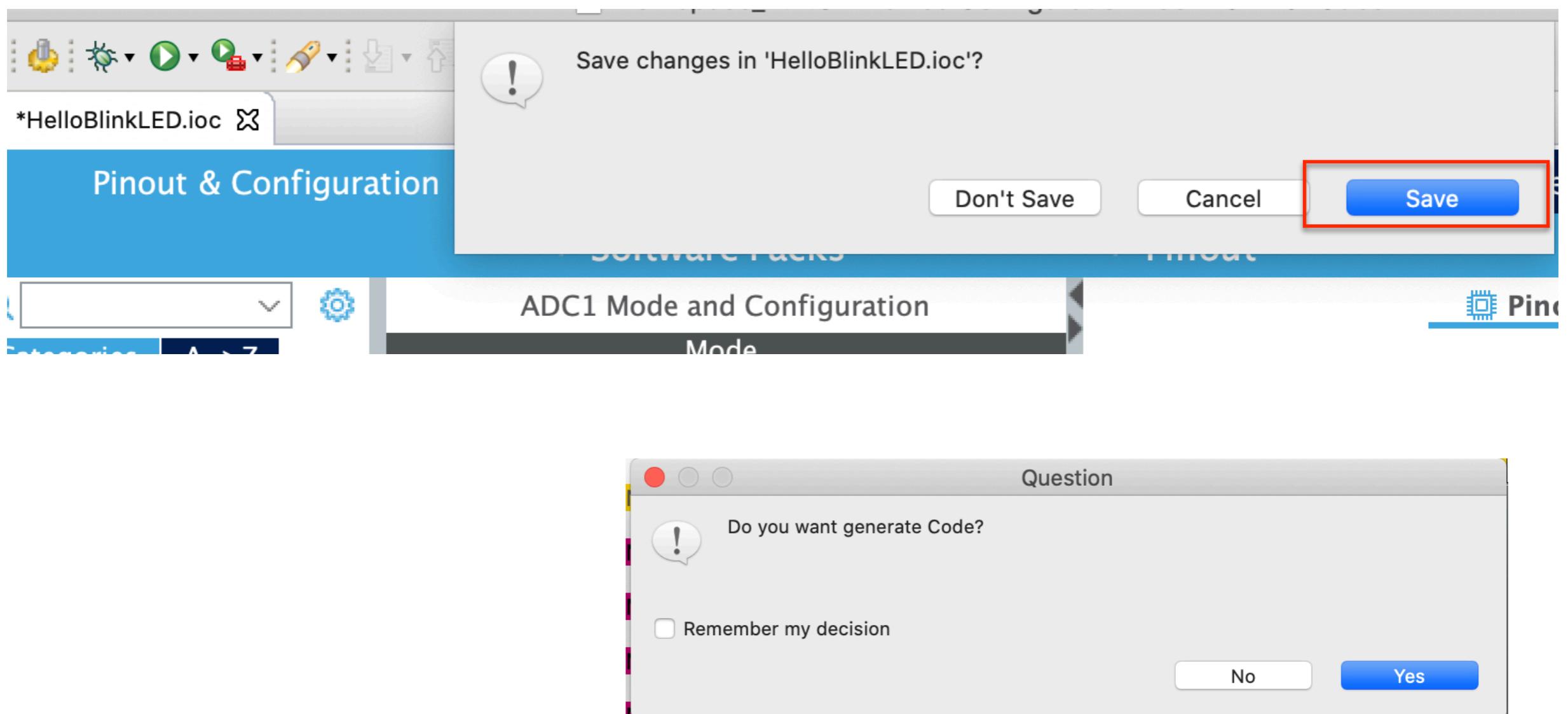
Default Project Has ADC Disabled



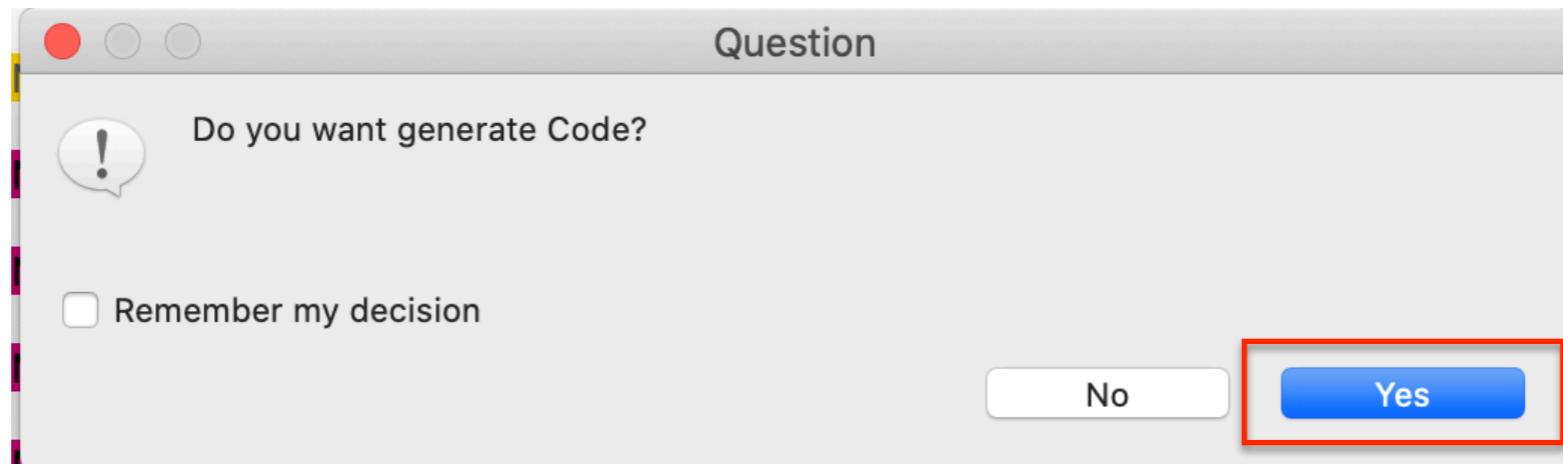
Default Project Has ADC Enable As Single Ended



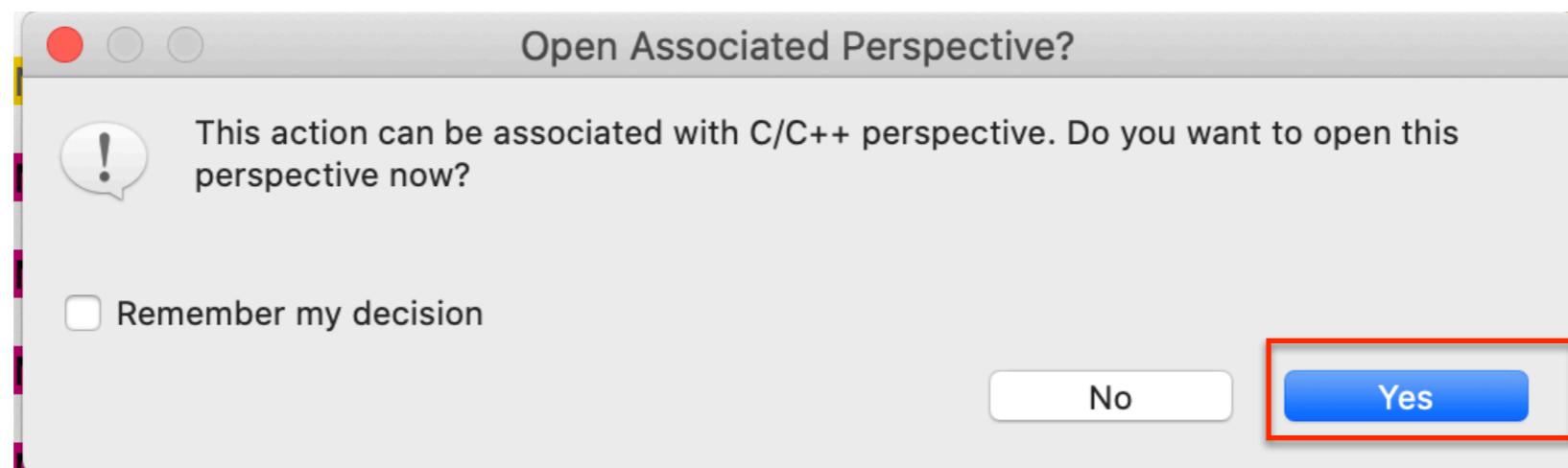
Save Changes Should Add HAL ADC Code to your project



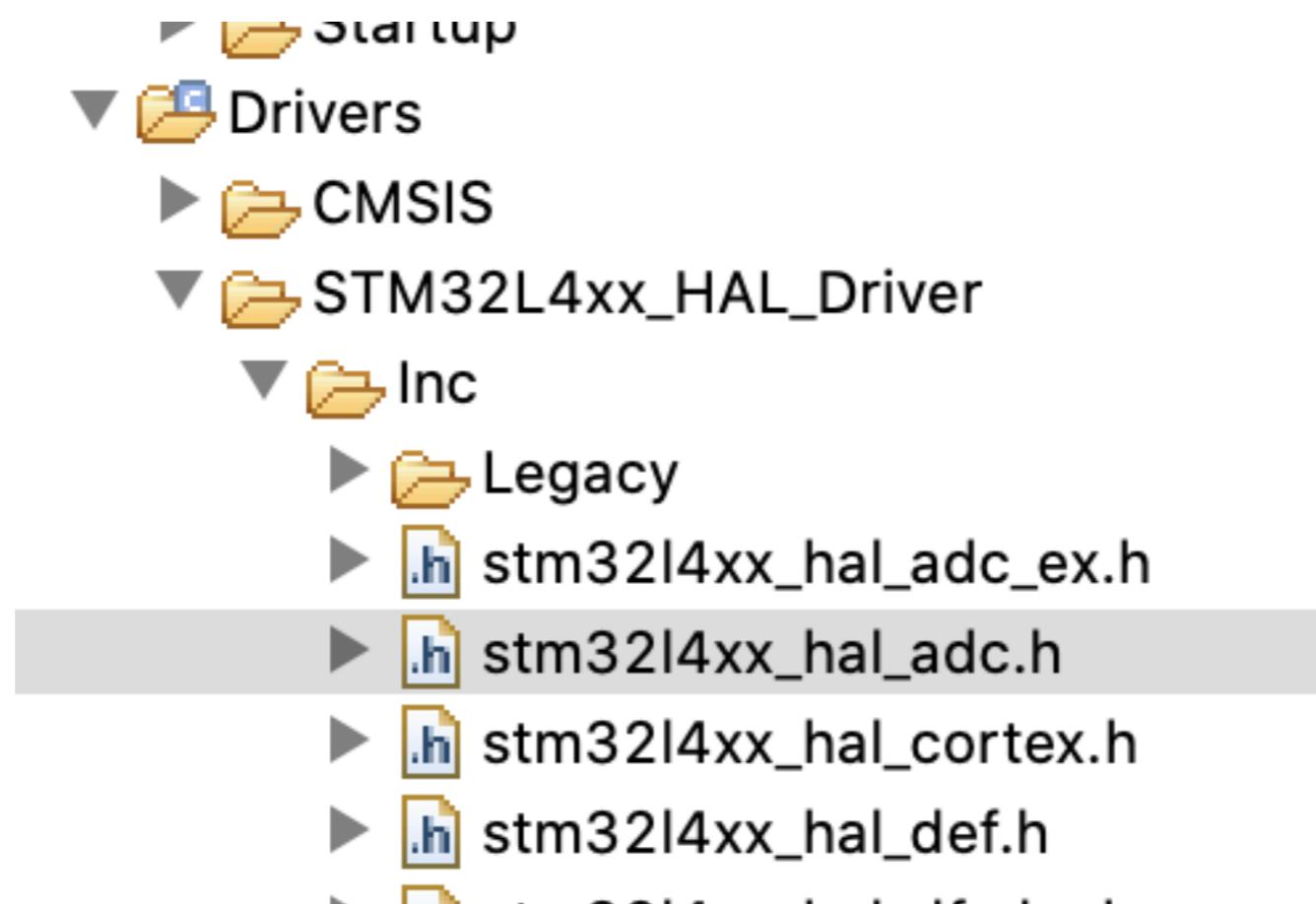
Generate Code



Yes Change to C/C++ Perspective



stm32l4xx_hal_adc.h



stm32l4xx_hal_adc.h

ADC_OversamplingTypeDef

```
47/*
48     * @brief ADC group regular oversampling structure definition
49     */
50typedef struct
51 {
52     uint32_t Ratio;                                /*!< Configures the oversampling ratio.  
This parameter can be a value of @ref ADC_HAL_EC_OVS_RATIO */
53
54     uint32_t RightBitShift;                         /*!< Configures the division coefficient for the Oversampler.  
This parameter can be a value of @ref ADC_HAL_EC_OVS_SHIFT */
55
56     uint32_t TriggeredMode;                          /*!< Selects the regular triggered oversampling mode.  
This parameter can be a value of @ref ADC_HAL_EC_OVS_DISCONT_MODE */
57
58     uint32_t OversamplingStopReset;                 /*!< Selects the regular oversampling mode.  
The oversampling is either temporary stopped or reset upon an injected  
sequence interruption.  
If oversampling is enabled on both regular and injected groups, this parameter  
is discarded and forced to setting "ADC_REGOVERSAMPLING_RESUMED_MODE"  
(the oversampling buffer is zeroed during injection sequence).  
This parameter can be a value of @ref ADC_HAL_EC_OVS_SCOPE_REG */
59
60
61 } ADC_OversamplingTypeDef;
62
63
64
65
66
67
68
69 }
70 }
```

stm32l4xx_hal_adc.h

ADC_InitTypeDef

```
86     */
87 typedef struct
88 {
89     uint32_t ClockPrescaler;          /*!< Select ADC clock source (synchronous clock derived from APB clock or asynchronous cl
90     This parameter can be a value of @ref ADC_HAL_EC_COMMON_CLOCK_SOURCE.
91     Note: The ADC clock configuration is common to all ADC instances.
92     Note: In case of usage of channels on injected group, ADC frequency should be lower
93     AHB clock frequency /3 for resolution 8 bits, AHB clock frequency /2 for resol
94     Note: In case of synchronous clock mode based on HCLK/1, the configuration must be e
95     if the system clock has a 50% duty clock cycle (APB prescaler configured insid
96     must be bypassed and PCLK clock must have 50% duty cycle). Refer to reference
97     Note: In case of usage of asynchronous clock, the selected clock must be preliminary
98     Note: This parameter can be modified only if all ADC instances are disabled. */
99
100    uint32_t Resolution;           /*!< Configure the ADC resolution.
101     This parameter can be a value of @ref ADC_HAL_EC_RESOLUTION */
102
103    uint32_t DataAlign;            /*!< Specify ADC data alignment in conversion data register (right or left).
104     Refer to reference manual for alignments formats versus resolutions.
105     This parameter can be a value of @ref ADC_HAL_EC_DATA_ALIGN */
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178    ADC_OversamplingTypeDef Oversampling;   /*!< Specify the Oversampling parameters.
179     Caution: this setting overwrites the previous oversampling configuration if
180
181
182 #if defined(ADC_CFGR_DFSDMCFG) &&defined(DFSDM1_Channel0)
183     uint32_t DFSDMConfig;                  /*!< Specify whether ADC conversion data is sent directly to DFSDM.
184     This parameter can be a value of @ref ADC_HAL_EC_REG_DFSDM_TRANSFER.
185     Note: This parameter can be modified only if there is no conversion is ongoing (both
186
187 #endif /* ADC_CFGR_DFSDMCFG */
188 } ADC_InitTypeDef;
```

stm32l4xx_hal_adc.h

ADC_ChannelConfTypeDef

```
200  */
201 @typedef struct
202 {
203     uint32_t Channel;           /*!< Specify the channel to configure into ADC regular group.  
This parameter can be a value of @ref ADC_HAL_EC_CHANNEL  
Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for */
204
205     uint32_t Rank;             /*!< Specify the rank in the regular group sequencer.  
This parameter can be a value of @ref ADC_HAL_EC_REG_SEQ_RANKS  
Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by  
the new channel setting (or parameter number of conversions adjusted) */
206
207     uint32_t SamplingTime;    /*!< Sampling time value to be set for the selected channel.  
Unit: ADC clock cycles  
Conversion time is the addition of sampling time and processing time  
(12.5 ADC clock cycles at ADC resolution 12 bits, 10.5 cycles at 10 bits, 8.5 cycles at 8 bits, 6.5 cycles at 6 bits).  
This parameter can be a value of @ref ADC_HAL_EC_CHANNEL_SAMPLINGTIME  
Caution: This parameter applies to a channel that can be used into regular and/or injected group.  
It overwrites the last setting.  
Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor),  
sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time :  
Refer to device datasheet for timings values. */
208
209
210     uint32_t SingleDiff;      /*!< Select single-ended or differential input.  
In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (negative input).  
Only channel 'i' has to be configured, channel 'i+1' is configured automatically.  
This parameter must be a value of @ref ADC_HAL_EC_CHANNEL_SINGLE_DIFFENDING  
Caution: This parameter applies to a channel that can be used in a regular and/or injected group.  
It overwrites the last setting.  
Note: Refer to Reference Manual to ensure the selected channel is available in differential mode.  
Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.  
Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).  
If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case  
of another parameter update on the fly) */
211
212
213     uint32_t OffsetNumber;    /*!< Select the offset number  
This parameter can be a value of @ref ADC_HAL_EC_OFFSET_NB  
Caution: Only one offset is allowed per channel. This parameter overwrites the last setting. */
214
215     uint32_t Offset;          /*!< Define the offset to be subtracted from the raw converted data.  
Offset value must be a positive number.  
Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0x1FFF.  
Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled  
without continuous mode or external trigger that could launch a conversion). */
216
217
218 } ADC_ChannelConfTypeDef;
```

ADC API Functions

stm32l4xx_hal_adc.h

Polling Mode

```
1/41
1742 * Blocking mode: Polling */
1743 AL_StatusTypeDef      HAL_ADC_Start(ADC_HandleTypeDef *hadc);
1744 AL_StatusTypeDef      HAL_ADC_Stop(ADC_HandleTypeDef *hadc);
1745 AL_StatusTypeDef      HAL_ADC_PollForConversion(ADC_HandleTypeDef *hadc, uint32_t Timeout);
1746 AL_StatusTypeDef      HAL_ADC_PollForEvent(ADC_HandleTypeDef *hadc, uint32_t EventType, uint32_t Timeout);
1747
```

stm32l4xx_hal_adc.h

Interrupt Mode

```
1/4/  
1748 * Non-blocking mode: Interruption */  
1749 AL_StatusTypeDef      HAL_ADC_Start_IT(ADC_HandleTypeDef *hadc);  
1750 AL_StatusTypeDef      HAL_ADC_Stop_IT(ADC_HandleTypeDef *hadc);  
1751
```

stm32l4xx_hal_adc.h

DMA Mode

```
1751
1752 * Non-blocking mode: DMA */
1753 AL_StatusTypeDef      HAL_ADC_Start_DMA(ADC_HandleTypeDef *hadc, uint32_t *pData, uint32_t Length);
1754 AL_StatusTypeDef      HAL_ADC_Stop_DMA(ADC_HandleTypeDef *hadc);
1755
```

stm32l4xx_hal_adc.h

Callbacks for Interrupt and DMA Modes

```
1/58
1759 * ADC IRQHandler and Callbacks used in non-blocking modes (Interruption and DMA) */
1760 oid HAL_ADC_IRQHandler(ADC_HandleTypeDef *hadc);
1761 oid HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc);
1762 oid HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef *hadc);
1763 oid HAL_ADC_LevelOutOfWindowCallback(ADC_HandleTypeDef *hadc);
1764 oid HAL_ADC_ErrorCallback(ADC_HandleTypeDef *hadc);
1765 **
```

Code Segments

ADC Polling

HAL_ADCEx_Calibration_Start()

```
110
117 */
118     * @brief  Perform an ADC automatic self-calibration
119     *
120     * Calibration prerequisite: ADC must be disabled (execute this
121     * function before HAL_ADC_Start() or after HAL_ADC_Stop() ).  

122     * @param hadc      ADC handle
123     * @param SingleDiff Selection of single-ended or differential input
124     * This parameter can be one of the following values:
125     *          @arg @ref ADC_SINGLE_ENDED      Channel in mode input single ended
126     *          @arg @ref ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
127     * @retval HAL status
128 */
128 HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef *hadc, uint32_t SingleDiff)
129 {
```

HAL_ADC_Start()

```
1205
1206 /**
1207  * @brief  Enable ADC, start conversion of regular group.
1208  * @note   Interruptions enabled in this function: None.
1209  * @note   Case of multimode enabled (when multimode feature is available):
1210  *         if ADC is Slave, ADC is enabled but conversion is not started,
1211  *         if ADC is master, ADC is enabled and multimode conversion is started.
1212  * @param hadc ADC handle
1213  * @retval HAL status
1214 */
1215 HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef *hadc)
```

HAL_ADC_PollForConversion()

```
1385
1386 /**
1387  * @brief Wait for regular group conversion to be completed.
1388  * @note ADC conversion flags EOS (end of sequence) and EOC (end of
1389  * conversion) are cleared by this function, with an exception:
1390  * if low power feature "LowPowerAutoWait" is enabled, flags are
1391  * not cleared to not interfere with this feature until data register
1392  * is read using function HAL_ADC_GetValue().
1393  * @note This function cannot be used in a particular setup: ADC configured
1394  * in DMA mode and polling for end of each conversion (ADC init
1395  * parameter "EOCSelection" set to ADC_EOC_SINGLE_CONV).
1396  * In this case, DMA resets the flag EOC and polling cannot be
1397  * performed on each conversion. Nevertheless, polling can still
1398  * be performed on the complete sequence (ADC init
1399  * parameter "EOCSelection" set to ADC_EOC_SEQ_CONV).
1400  * @param hadc ADC handle
1401  * @param Timeout Timeout value in millisecond.
1402  * @retval HAL status
1403 */
1404 HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef *hadc, uint32_t Timeout)
1405 }
```

HAL_ADC_GetValue()

```
2194
2195 /*
2196     * @brief  Get ADC regular group conversion result.
2197     * @note   Reading register DR automatically clears ADC flag EOC
2198     *         (ADC group regular end of unitary conversion).
2199     * @note   This function does not clear ADC flag EOS
2200     *         (ADC group regular end of sequence conversion).
2201     * Occurrence of flag EOS rising:
2202     *     - If sequencer is composed of 1 rank, flag EOS is equivalent
2203     *       to flag EOC.
2204     *     - If sequencer is composed of several ranks, during the scan
2205     *       sequence flag EOC only is raised, at the end of the scan sequence
2206     *       both flags EOC and EOS are raised.
2207     * To clear this flag, either use function:
2208     *     in programming model IT: @ref HAL_ADC_IRQHandler(), in programming
2209     *     model polling: @ref HAL_ADC_PollForConversion()
2210     *     or @ref __HAL_ADC_CLEAR_FLAG(&hadc, ADC_FLAG_EOS).
2211     * @param hadc ADC handle
2212     * @retval ADC group regular conversion data
2213 */
2214 uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef *hadc)
2215 r
```

ADC Polling

Code Segment

```
137
138 // Calibrate ADC
139 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
140
141 while (1)
142 {
143     /* USER CODE END WHILE */
144
145     /* USER CODE BEGIN 3 */
146     HAL_GPIO_TogglePin(LED3_WIFI_LED4_BLE_GPIO_Port, LED3_WIFI_LED4_BLE_Pin);
147     HAL_Delay(1000); //500 msec = 1 sec
148
149     // ADC Start Conversion
150     HAL_ADC_Start(&hadc1);
151
152     // Pool for results - Timeout is 10us
153     HAL_ADC_PollForConversion(&hadc1, 10);
154
155     uint16_t value = HAL_ADC_GetValue(&hadc1);
156
157     // Send value to console
158     char buf[100];
159     sprintf(buf, "%u ", value);
160
161     HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
162
163
164 }
```

Code Segments

ADC Interrupts

HAL_ADC_Start_IT

```
1704
1705 /**
1706     * @brief Enable ADC, start conversion of regular group with interruption.
1707     * @note Interruptions enabled in this function according to initialization
1708     * setting : EOC (end of conversion), EOS (end of sequence),
1709     * OVR overrun.
1710     * Each of these interruptions has its dedicated callback function.
1711     * @note Case of multimode enabled (when multimode feature is available):
1712     * HAL_ADC_Start_IT() must be called for ADC Slave first, then for
1713     * ADC Master.
1714     * For ADC Slave, ADC is enabled only (conversion is not started).
1715     * For ADC Master, ADC is enabled and multimode conversion is started.
1716     * @note To guarantee a proper reset of all interruptions once all the needed
1717     * conversions are obtained, HAL_ADC_Stop_IT() must be called to ensure
1718     * a correct stop of the IT-based conversions.
1719     * @note By default, HAL_ADC_Start_IT() does not enable the End Of Sampling
1720     * interruption. If required (e.g. in case of oversampling with trigger
1721     * mode), the user must:
1722     *   1. first clear the EOSMP flag if set with macro __HAL_ADC_CLEAR_FLAG(hadc, ADC_FLAG_EOSM)
1723     *   2. then enable the EOSMP interrupt with macro __HAL_ADC_ENABLE_IT(hadc, ADC_IT_EOSMP)
1724     * before calling HAL_ADC_Start_IT().
1725     * @param hadc ADC handle
1726     * @retval HAL status
1727 */
1728 HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef *hadc)
```

ADC Interrupt Driven Code Segment - Part 1

ADC1 Mode and Configuration

Mode

IN1 IN1 Single-ended

IN2 Disable

IN3 Disable

IN4 Disable

IN5 Disable

IN6 Disable

IN7 Disable

IN8 Disable

IN9 Disable

IN10 Disable

Configuration

Reset Configuration

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

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

ADC Interrupt Driven Code Segment - Part 2

```
154 // Calibrate ADC
155 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
156
157 while (1)
158 {
159     /* USER CODE END WHILE */
160
161     /* USER CODE BEGIN 3 */
162     HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
163     HAL_Delay(1000); //500 msec = 1 sec
164
165     // ADC Start Conversion
166     //HAL_ADC_Start(&hadc1);
167     HAL_ADC_Start_IT(&hadc1);
168     //HAL_ADC_Start_DMA(&hadc1, (uint32_t *) &adc_value, 1);
169
170     // Pool for results - Timeout is 10ms
171     //HAL_ADC_PollForConversion(&hadc1, 10);
172
173     //uint16_t value = HAL_ADC_GetValue(&hadc1);
174
175     // Send value to console
176     //char buf[100];
177     //snprintf(buf, sizeof(buf), "%u ", value);
178
179     //HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
180
181
182 }
183 /* USER CODE END 3 */
184
185 }
```

ADC Interrupt Driven Code Segment - Part 3

The image shows a file browser on the left and a code editor on the right. The file browser displays a project structure with files like main.c, stm32l4xx_hal_msp.c, and stm32l4xx_it.c. The code editor shows a C code snippet for the ADC1_2_IRQHandler function.

```
192
193  **** STM32L4xx Peripheral Interrupt Handlers ****
194  * Add here the Interrupt Handlers for the used peripherals.
195  * For the available peripheral interrupt handler names,
196  * please refer to the startup file (startup_stm32l4xx.s).
197  ****
198
199
200  **
201      * @brief This function handles ADC1 and ADC2 interrupts.
202
203 void ADC1_2_IRQHandler(void)
204 {
205     /* USER CODE BEGIN ADC1_2_IRQHandler 0 */
206
207     /* USER CODE END ADC1_2_IRQHandler 0 */
208     HAL_ADC_IRQHandler(&hadc1);
209     /* USER CODE BEGIN ADC1_2_IRQHandler 1 */
210
211     /* USER CODE END ADC1_2_IRQHandler 1 */
212 }
```

ADC Interrupt Driven Code Segment - Part 4

```
2259 */
2260 void HAL_ADC_IRQHandler(ADC_HandleTypeDef *hadc)
2261 {
2262     uint32_t overrun_error = 0UL; /* flag set if overrun occurrence has to be considered as
2263     uint32_t tmp_isr = hadc->Instance->ISR;
2264     uint32_t tmp_iер = hadc->Instance->IER;
2265     uint32_t tmp_adc_inj_is_trigger_source_sw_start;
2266     uint32_t tmp_adc_reg_is_trigger_source_sw_start;
2267     uint32_t tmp_cfgr;
2268 #if defined(ADC_MULTIMODE_SUPPORT)
2269     const ADC_TypeDef *tmpADC_Master;
2270     uint32_t tmp_multimode_config = LL_ADC_GetMultimode(__LL_ADC_COMMON_INSTANCE(hadc->Instanc
2271 #endif
2272
2273     /* Check the parameters */
2274     assert_param(IS_ADC_ALL_INSTANCE(hadc->Instance));
2275     assert_param(IS_ADC_EOC_SELECTION(hadc->Init.EOCSelection));
2276
2277     /* ====== Check End of Sampling flag for ADC group regular ===== */
2278     if (((tmp_isr & ADC_FLAG_EOSMP) == ADC_FLAG_EOSMP) && ((tmp_iер & ADC_IT_EOSMP) == ADC_IT_EOSMP))
2279     {
2280         /* Update state machine on end of sampling status if not in error state */
2281         if ((hadc->State & HAL_ADC_STATE_ERROR_INTERNAL) == 0UL)
2282         {
2283             /* Set ADC state */
2284             SET_BIT(hadc->State, HAL_ADC_STATE_REG_EOSMP);
2285         }
2286
2287         /* End Of Sampling callback */
2288 #if (USE_HAL_ADC_REGISTER_CALLBACKS == 1)
2289         hadc->EndOfSamplingCallback(hadc);
2290 #else
2291         HAL_ADCEx_EndOfSamplingCallback(hadc);
2292 #endif /* USE_HAL_ADC_REGISTER_CALLBACKS */
2293     }
2294 }
```

ADC Interrupt Driven Code Segment - Part 5

This code in stm32l4xx_hal_adc.c

```
2600  */
2601 _weak void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
2602 {
2603     /* Prevent unused argument(s) compilation warning */
2604     UNUSED(hadc);
2605
2606     /* NOTE : This function should not be modified. When the callback is needed,
2607            function HAL_ADC_ConvCpltCallback must be implemented in the user file.
2608     */
2609 }
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3588
3589
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3598
3599
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3688
3689
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3698
3699
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3788
3789
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3798
3799
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3888
3889
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3898
3899
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3988
3989
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3998
3999
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4098
4099
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4
```

ADC Interrupt Driven Code Segment - Part 6

This code in main.c

```
/8
79/* Private user code -----
80/* USER CODE BEGIN 0 */
81
82void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
83
84    uint16_t value = HAL_ADC_GetValue(&hadc1);
85
86    // Send value to console
87    char buf[100];
88    sprintf(buf, "%u ", value);
89
90    HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
91}
92
93/* USER CODE END 0 */
```

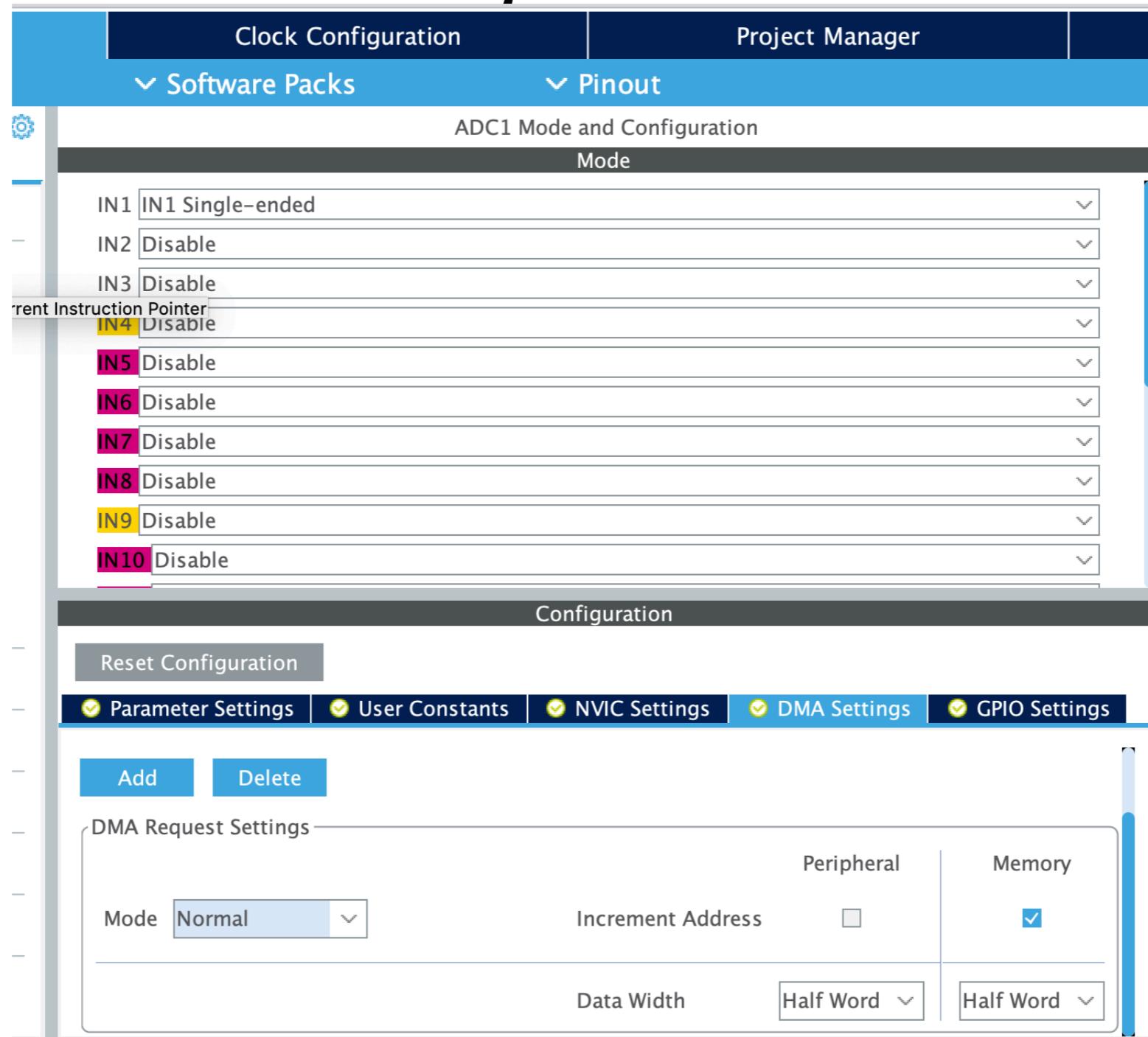
Code Segments

ADC DMA

HAL_ADC_Start_DMA

```
...
1975 /**
1976  * @brief Enable ADC, start conversion of regular group and transfer result through DMA.
1977  * @note Interruptions enabled in this function:
1978  *       overrun (if applicable), DMA half transfer, DMA transfer complete.
1979  *       Each of these interruptions has its dedicated callback function.
1980  * @note Case of multimode enabled (when multimode feature is available): HAL_ADC_Start_DMA()
1981  *       is designed for single-ADC mode only. For multimode, the dedicated
1982  *       HAL_ADCEx_MultiModeStart_DMA() function must be used.
1983  * @param hadc ADC handle
1984  * @param pData Destination Buffer address.
1985  * @param Length Number of data to be transferred from ADC peripheral to memory
1986  * @retval HAL status.
1987 */
1988 HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef *hadc, uint32_t *pData, uint32_t Length)
1989 }
```

ADC DMA Driven Code Segment - Part 1



ADC DMA Driven

Code Segment - Part 2

This code in main.c

```
80
81 /* Private user code -----
82 /* USER CODE BEGIN 0 */
83
84 static uint16_t adc_value;
85
86 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
87
88     //uint16_t value = HAL_ADC_GetValue(&hadc1);
89
90     // Send value to console
91     char buf[100];
92     snprintf(buf, sizeof(buf), "%u ", adc_value);
93
94     HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
95 }
96
```

ADC DMA Driven

Code Segment - Part 3

```
153
154 // Calibrate ADC
155 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
156
157 while (1)
158 {
159     /* USER CODE END WHILE */
160
161     /* USER CODE BEGIN 3 */
162     HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port, LED3_WIFI__LED4_BLE_Pin);
163     HAL_Delay(1000); //500 msec = 1 sec
164
165     // ADC Start Conversion
166     //HAL_ADC_Start(&hadc1);
167     //HAL_ADC_Start_IT(&hadc1);
168     HAL_ADC_Start_DMA(&hadc1, &adc_value, 1);
169
170     // Pool for results - Timeout is 10us
171     //HAL_ADC_PollForConversion(&hadc1, 10);
172
173     //uint16_t value = HAL_ADC_GetValue(&hadc1);
174
175     // Send value to console
176     //char buf[100];
177     //snprintf(buf, sizeof(buf), "%u ", value);
178
179     //HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
180
181
182 }
183 /* USER CODE END 3 */
184 }
```

ADC Hands-On Project

ADC Hands-On Project

Overview

- The goal of this project is to give you hands-on experience with ADC
 - Using STM32L4 Discovery Kit IoT Node
 - Using STM32Cube IDE
- To confirm your experience, you will create a **PDF document** that you will submit for grading
 - The PDF document will capture the major steps you perform to complete this project
 - See example PDF posted with assignment for example PDF format

ADC Hands-On Project

User Stories

- User Story 1 - Connect ARD-A0 to 1.5VDC Battery
 - Read using ADC Polling Mode
- User Story 2 - Connect ARD-A0 to 1.5VDC Battery
 - Read using ADC Interrupt Mode
- User Story 3 - Connect ARD-A0 to 1.5VDC Battery
 - Read using ADC DMA Mode
- User Story 4 - Compare the results of the ADC values from User Stories 1-3. Are the values the same?
- User Story 5 - Run the tests without calibration enabled - do the results change?
 - Try results with and without calling
 - HAL_ADCEx_Calibration_Start(&hdc1, ADC_SINGLE_ENDED)

WARNING! Do not
Connect voltage greater
Than 3.3VDC to the
ADC Pin!

Summary

- Concepts
- Data Sheet - STM32L475
- User Manual - UM2153 - STM32L Discovery Kit for IoT
- Schematics - STM32L Discovery Kit for IoT
- API - STM32L HAL
 - Data Structures
 - Functions
- Hands-On Project