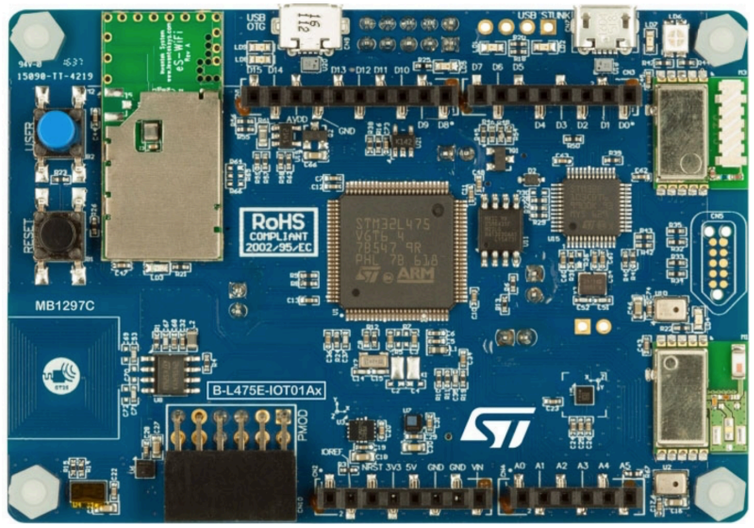
Embedded Systems Hardware Interfacing

ADC and DAC

Norman McEntire



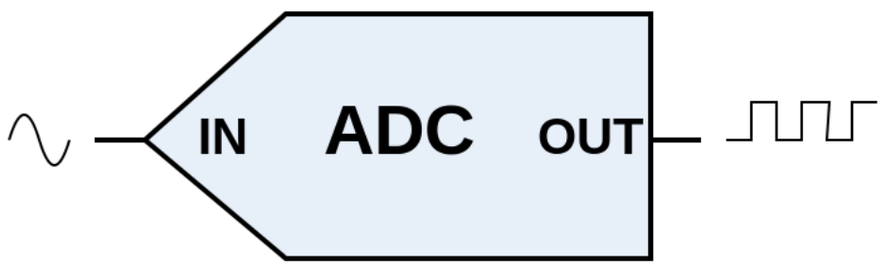
Contents

- 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 discreet 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 discreet values produced over an analog range
- Example: 8-bits give 256 different values

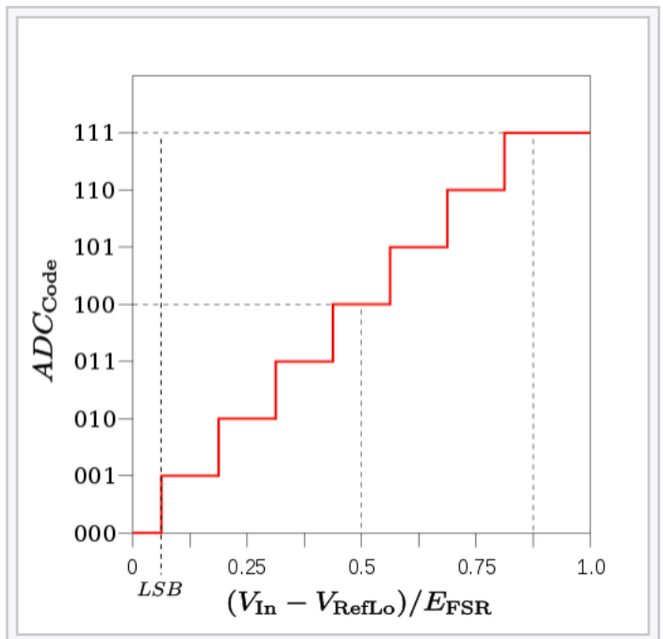


Fig. 1. An 8-level ADC coding scheme.

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 = 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

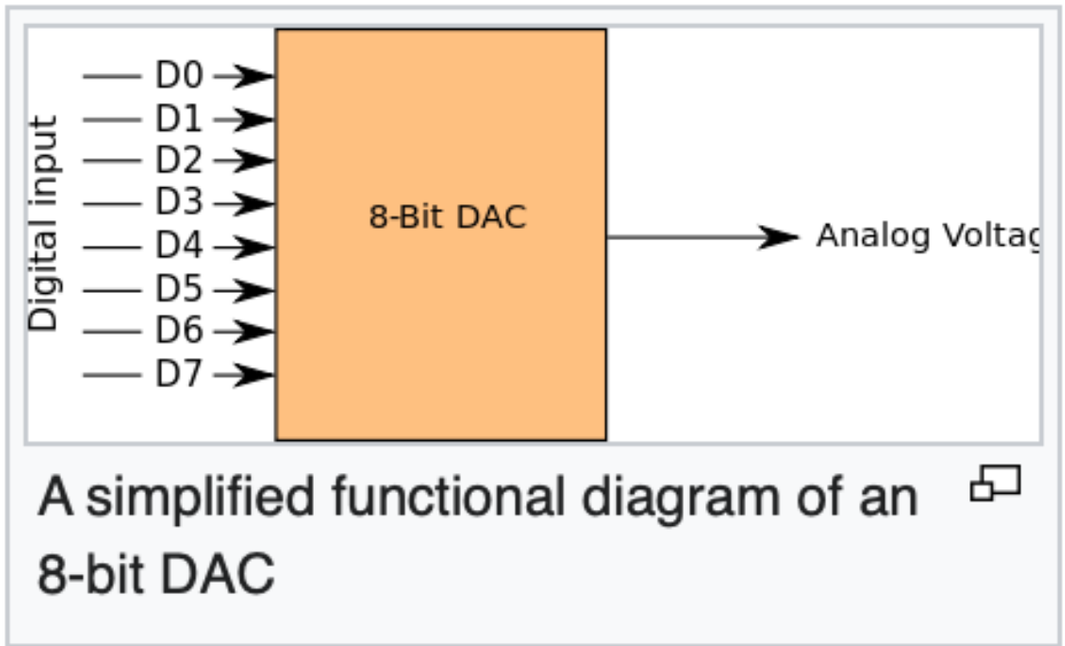
ADC Concepts
Singled Ended and Differential

- 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

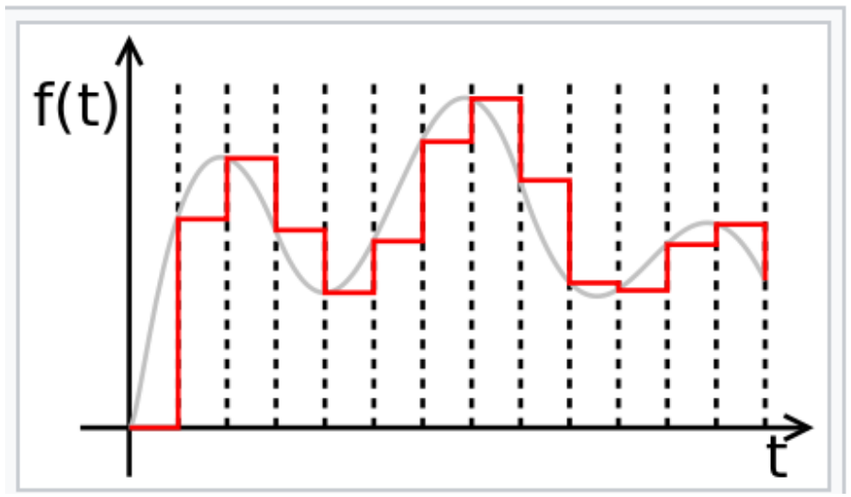
DAC



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

STM32l475

STM32L475 Data Sheet

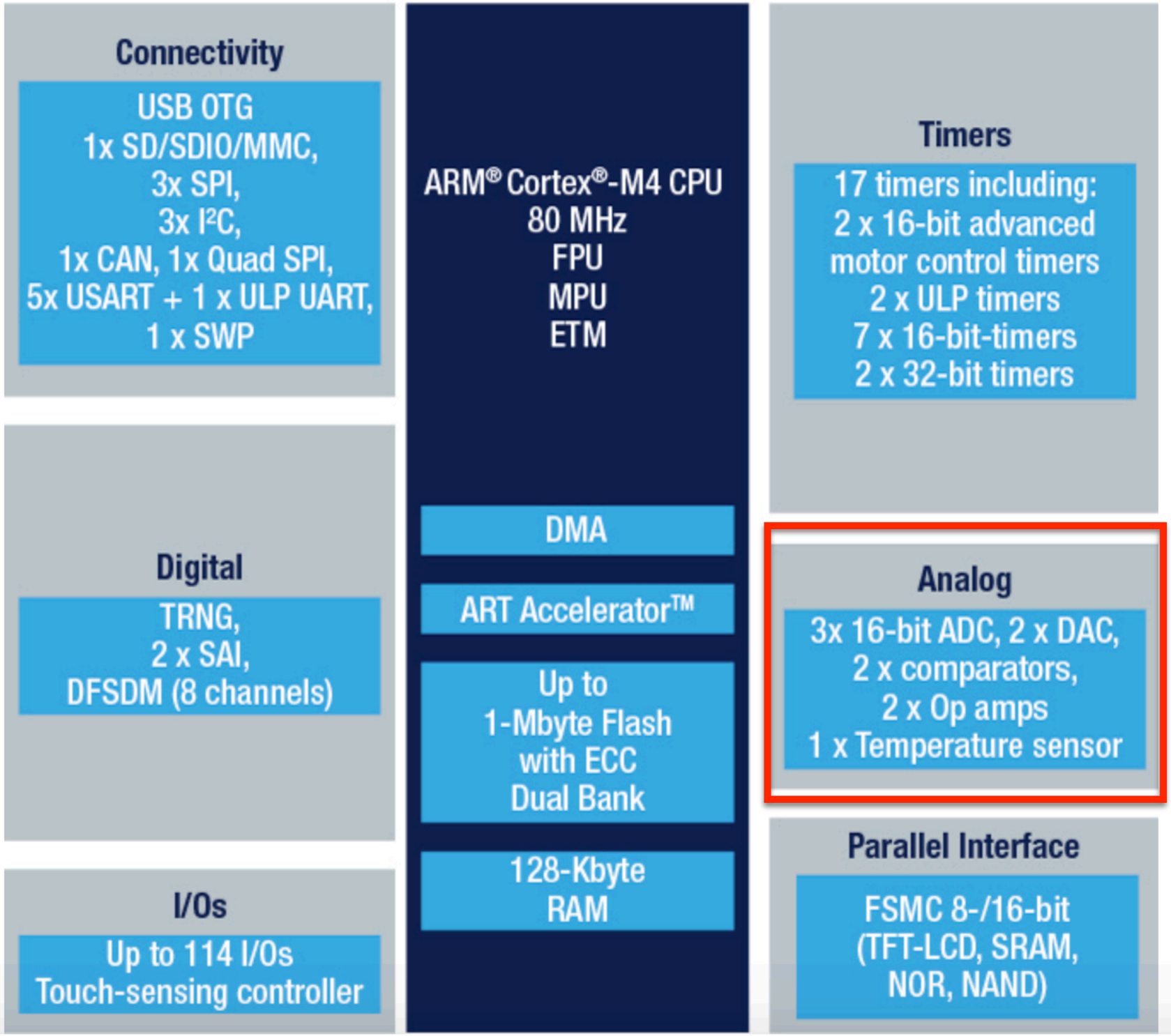
STM32L475xx

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 µA/Msps
  - 2x 12-bit DAC output channels, low-power sample and hold

STM32L475



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.

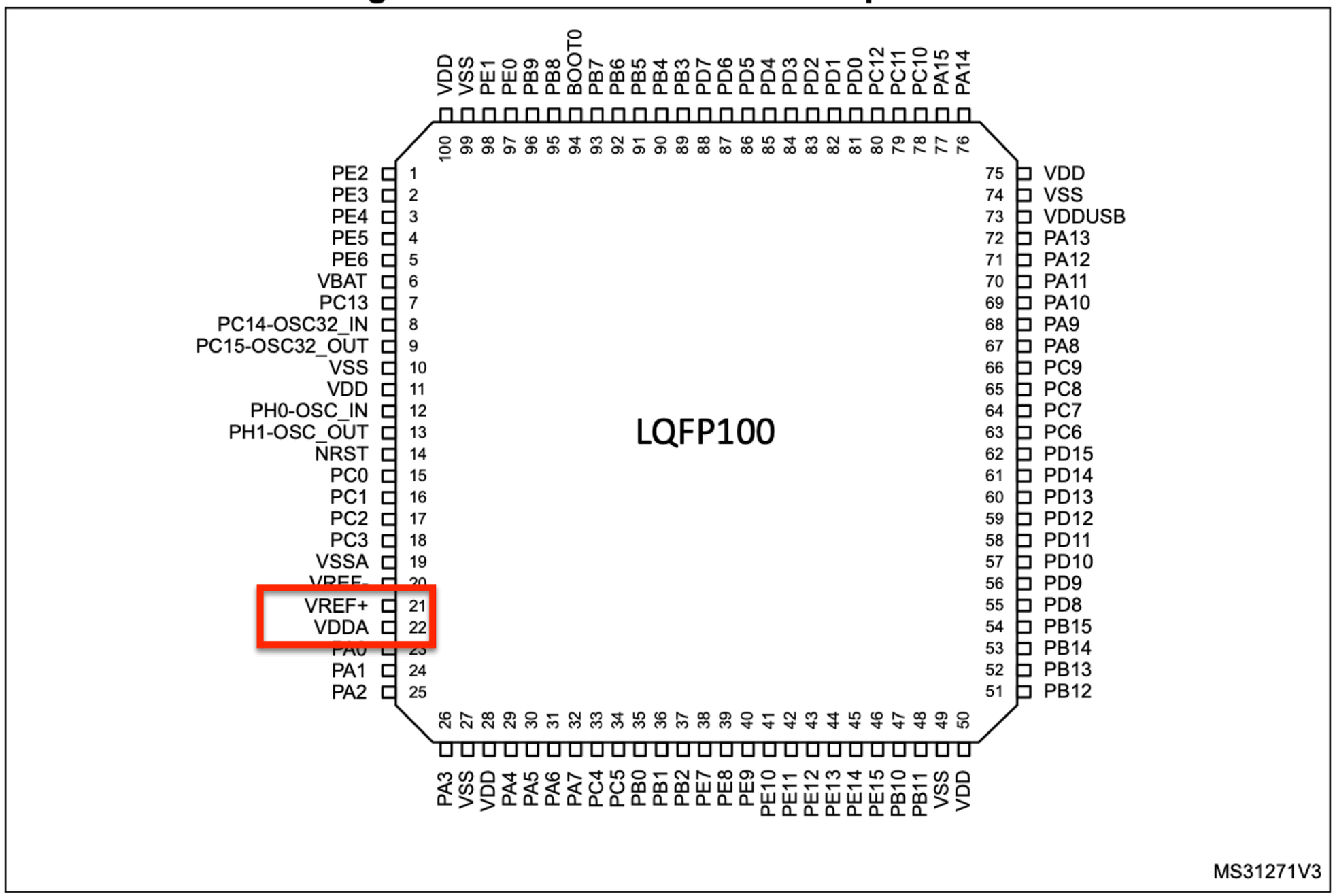
The temperature sensor is internally connected to the ADC1_IN17 and ADC3_IN17 input channels which is used to convert the sensor output voltage into a digital value.

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.

3.15.3 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 VBAT 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.

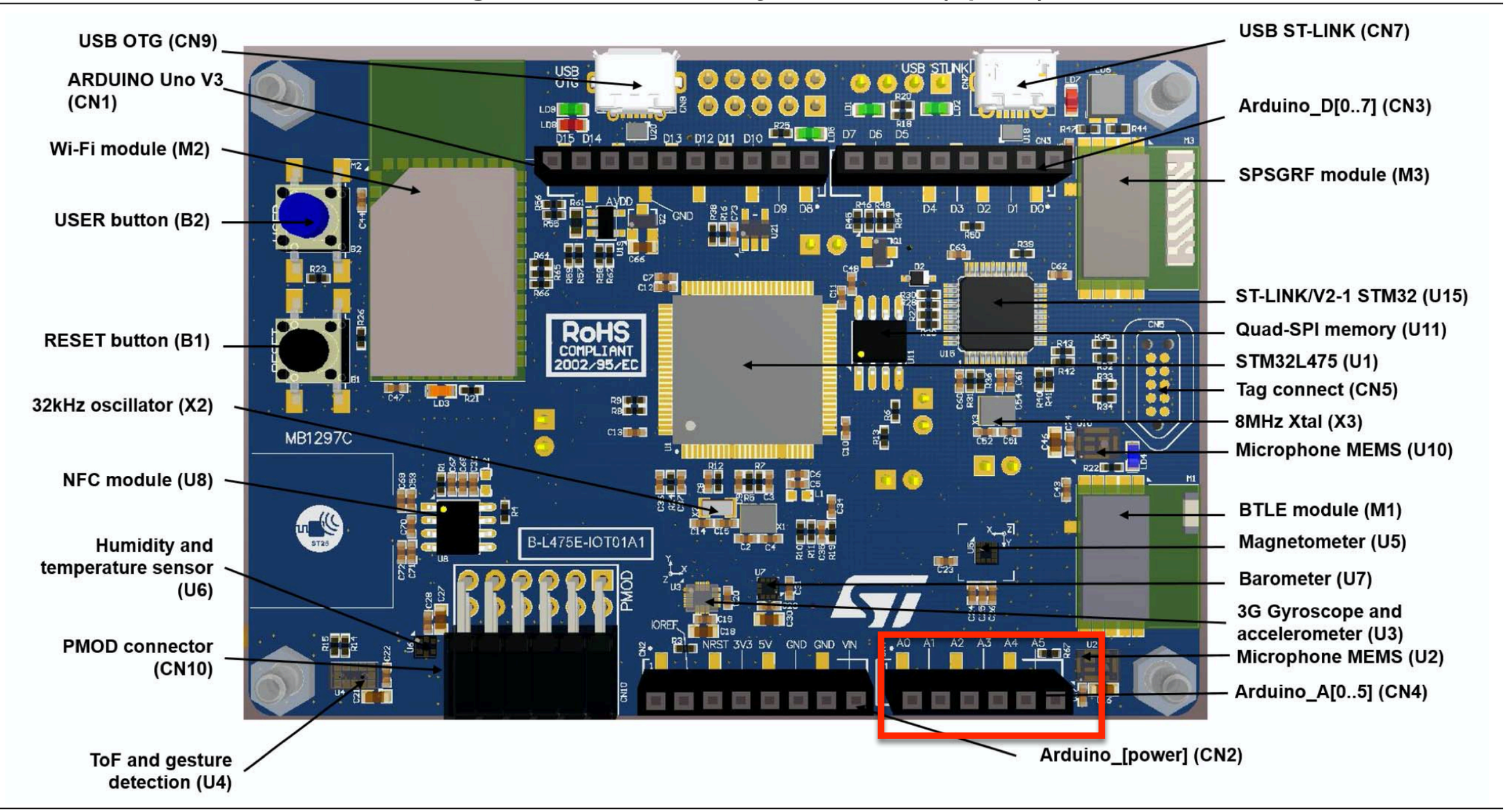
	reset is the same as the actual pin name	
Pin type	S	Supply pin
	I	Input only pin
	I/O	Input / output pin
I/O structure	FT	5 V tolerant I/O
	TT	3.6 V tolerant I/O
	B	Dedicated BOOT0 pin
	RST	Bidirectional reset pin with embedded weak pull-up resistor

Pin Number		Pin name (function after reset)	Pin type	I/O structure	Notes	Pin functions	
LQFP64	LQFP100					Alternate functions	Additional functions
-	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

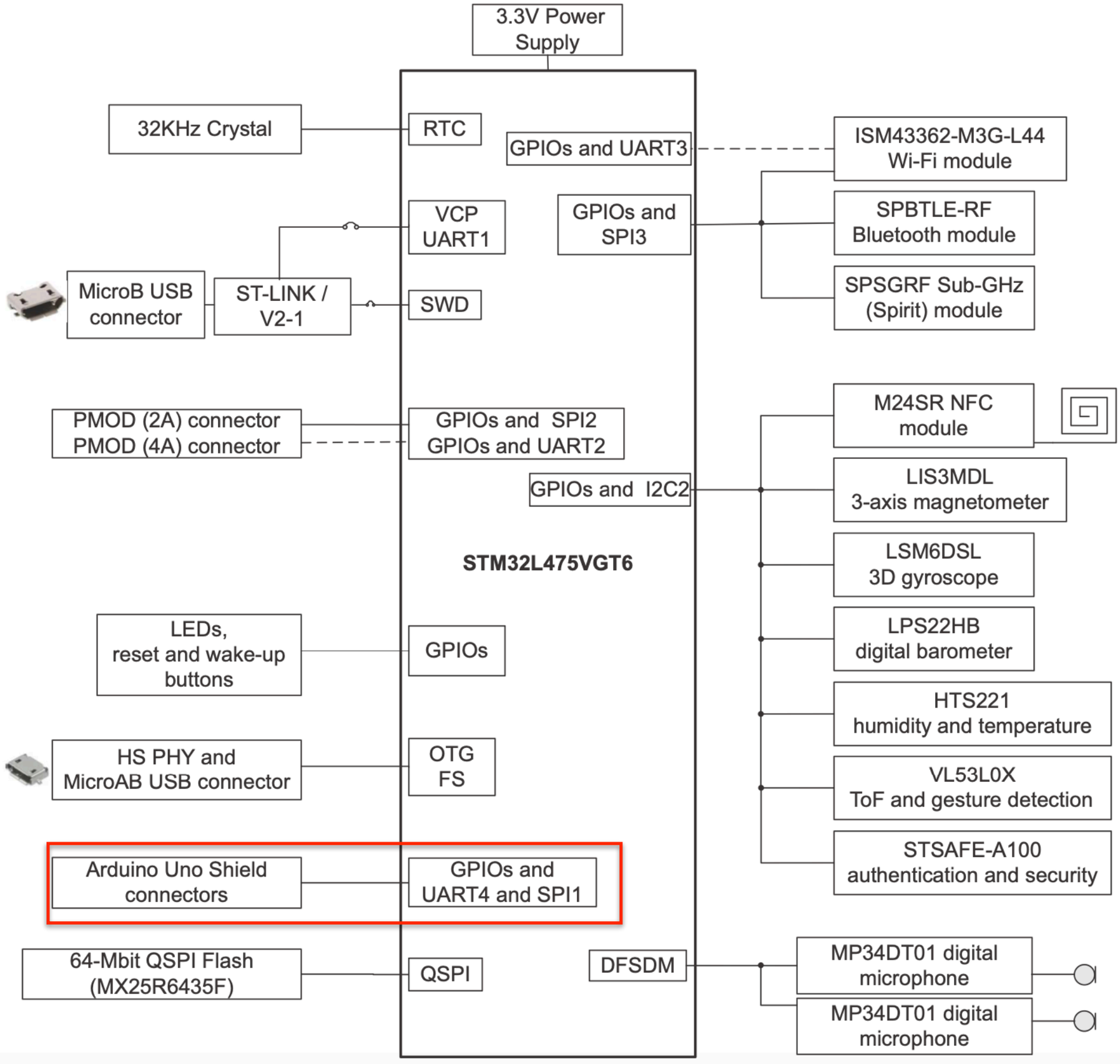


Use of ADC
Functional Description

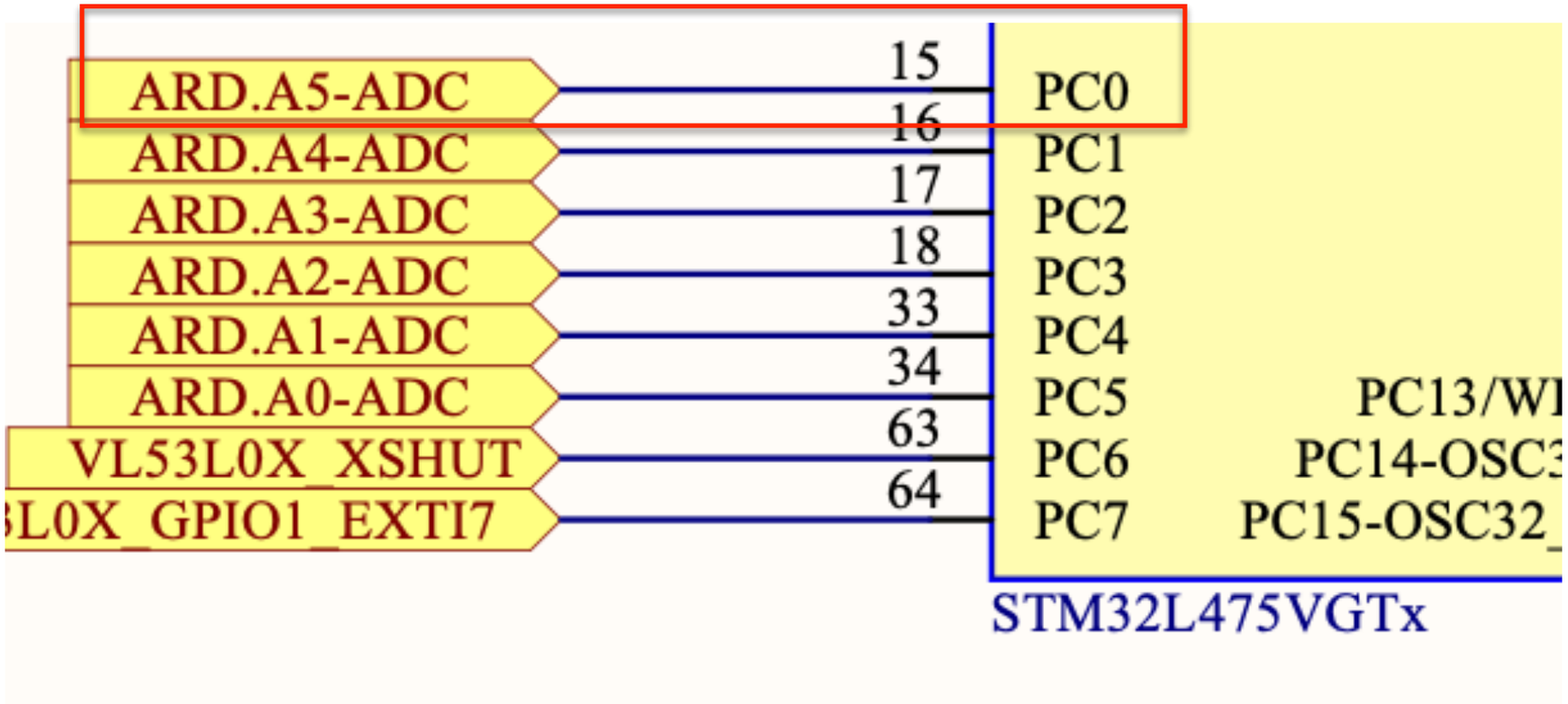
CN4	1	A0	ARD.A0-ADC	PC5	ADC
	2	A1	ARD.A1-ADC	PC4	ADC
	3	A2	ARD.A2-ADC	PC3	ADC
	4	A3	ARD.A3-ADC	PC2	ADC
	5	A4	ARD.A4-ADC	PC1	ADC / I2C3_SDA
	6	A5	ARD.A5-ADC	PC0	ADC / I2C3_SCL

Schematics

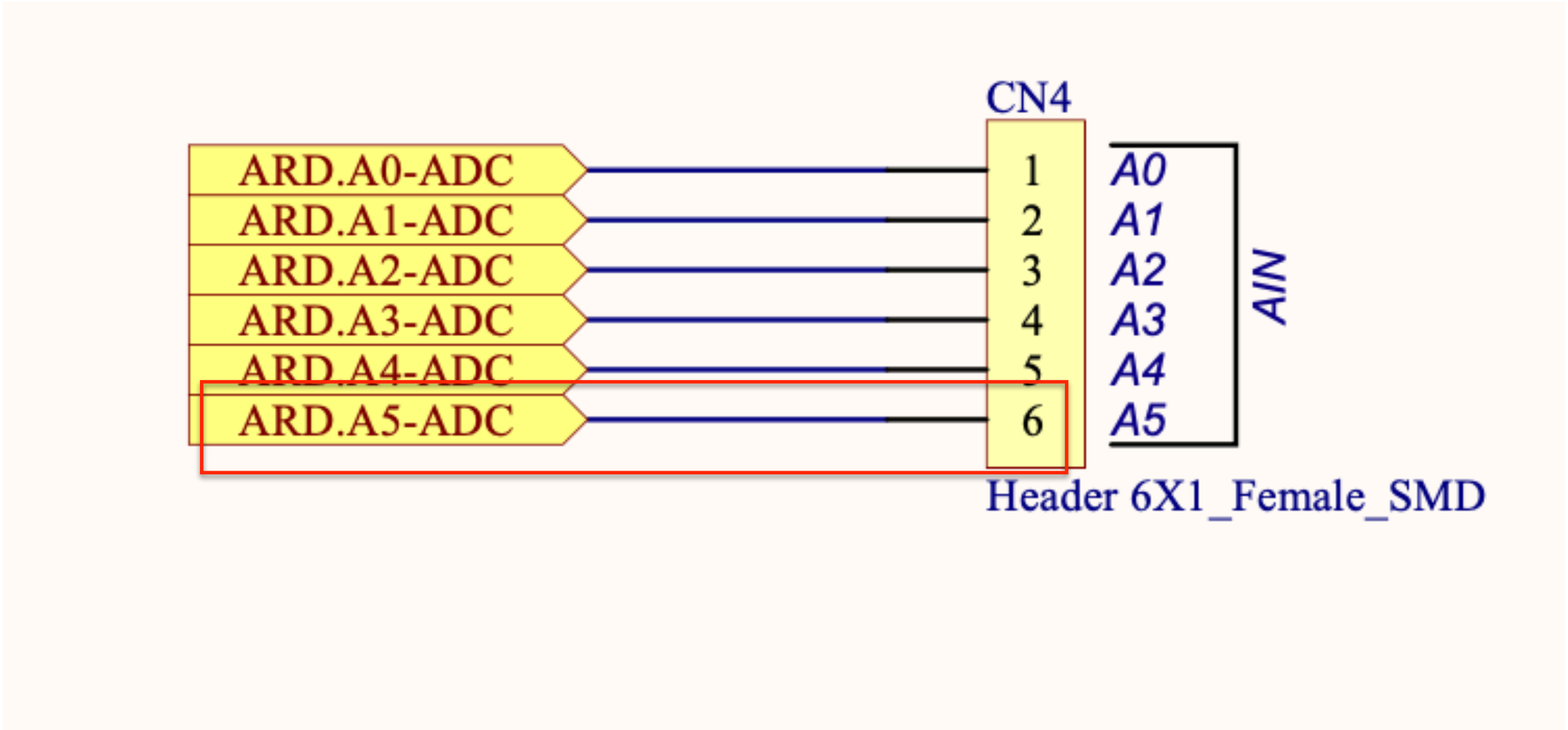
STM324L Discovery Kit IoT Node



STM32L Discovery Kit IoT Node



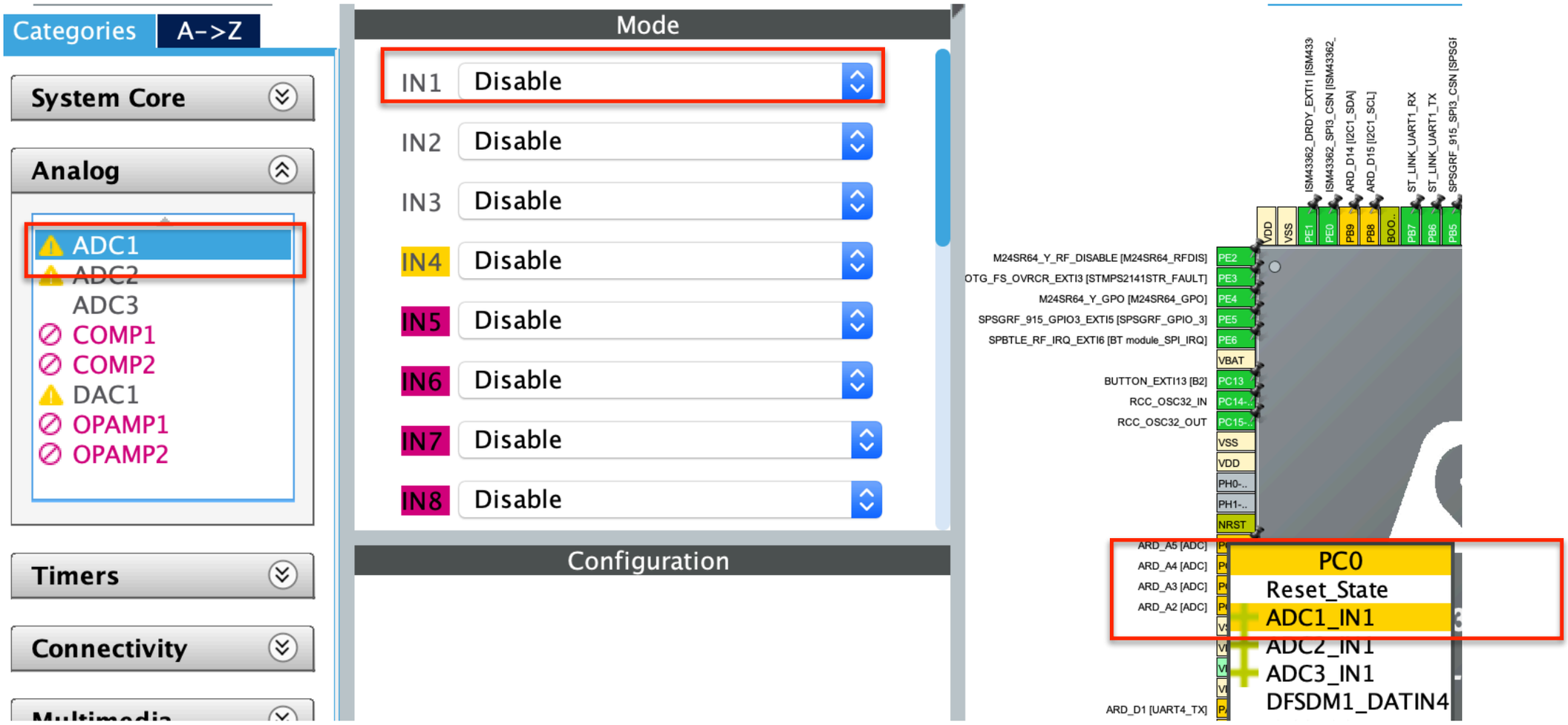
STM32L Discovery Kit IoT Node



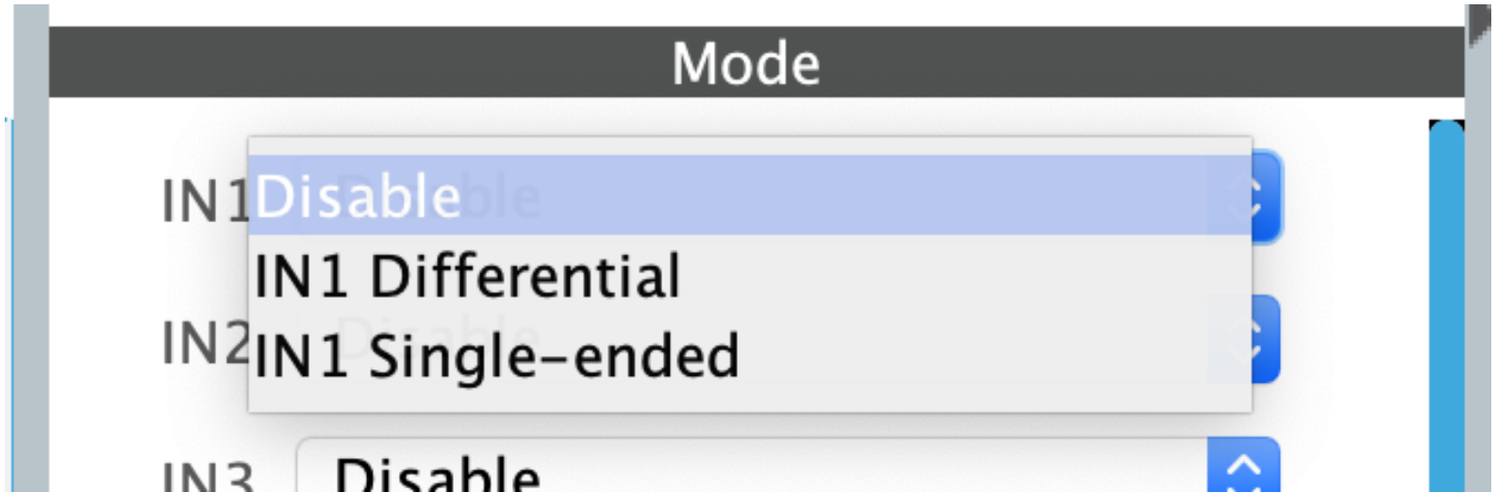
ADC API

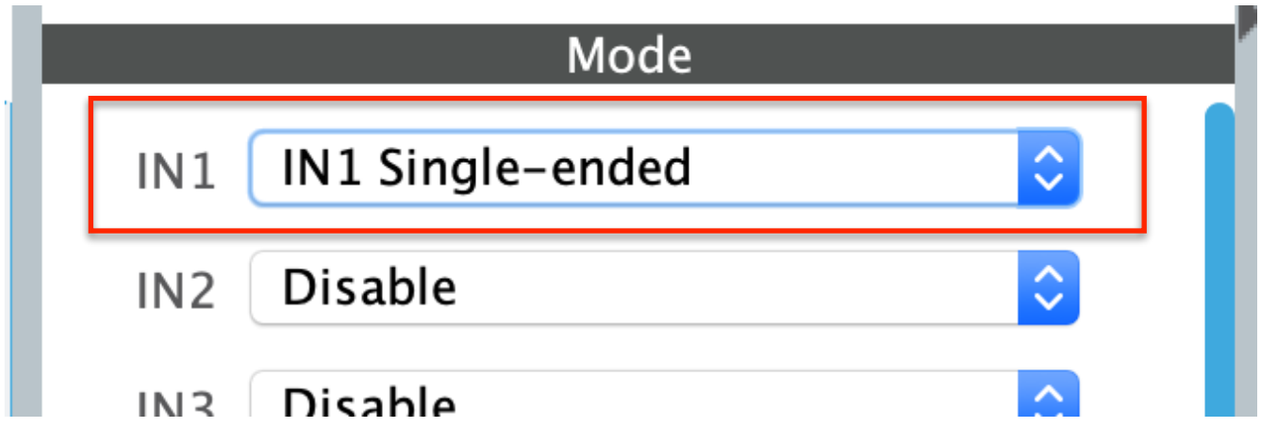
Data Structures

Default Project Has ADC Disabled

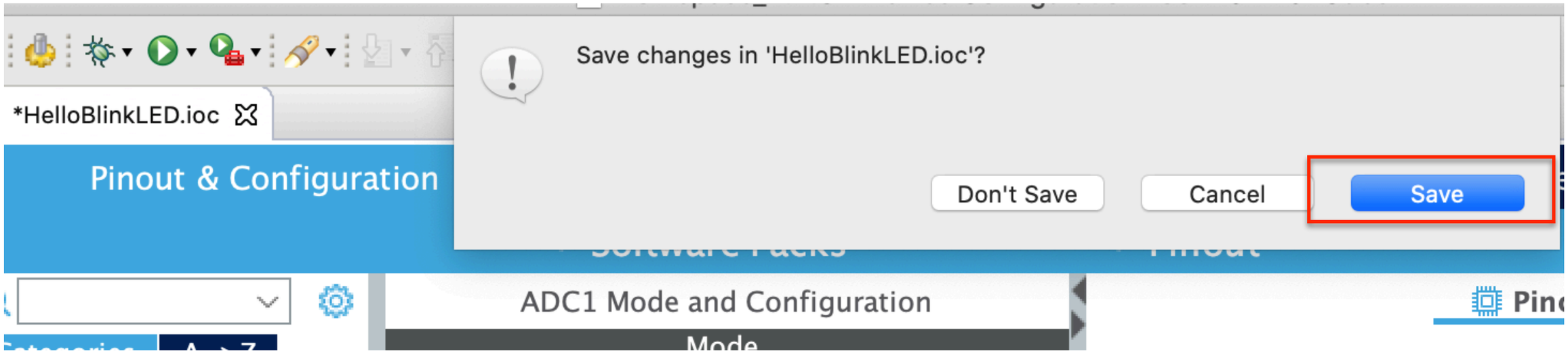


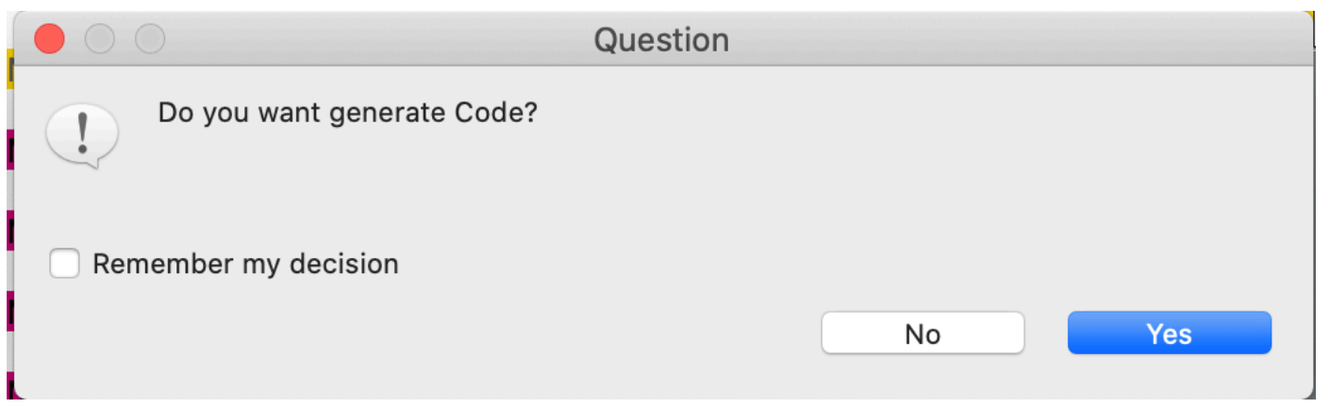
Default Project Has ADC Enable As Single Ended



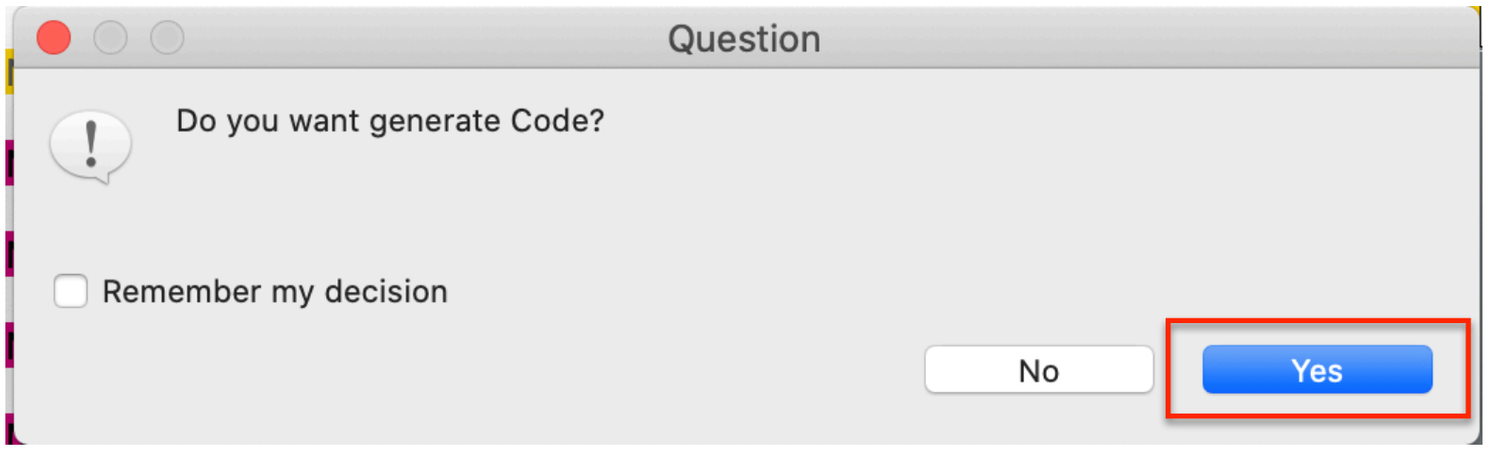


Save Changes Should Add HAL ADC Code to your project

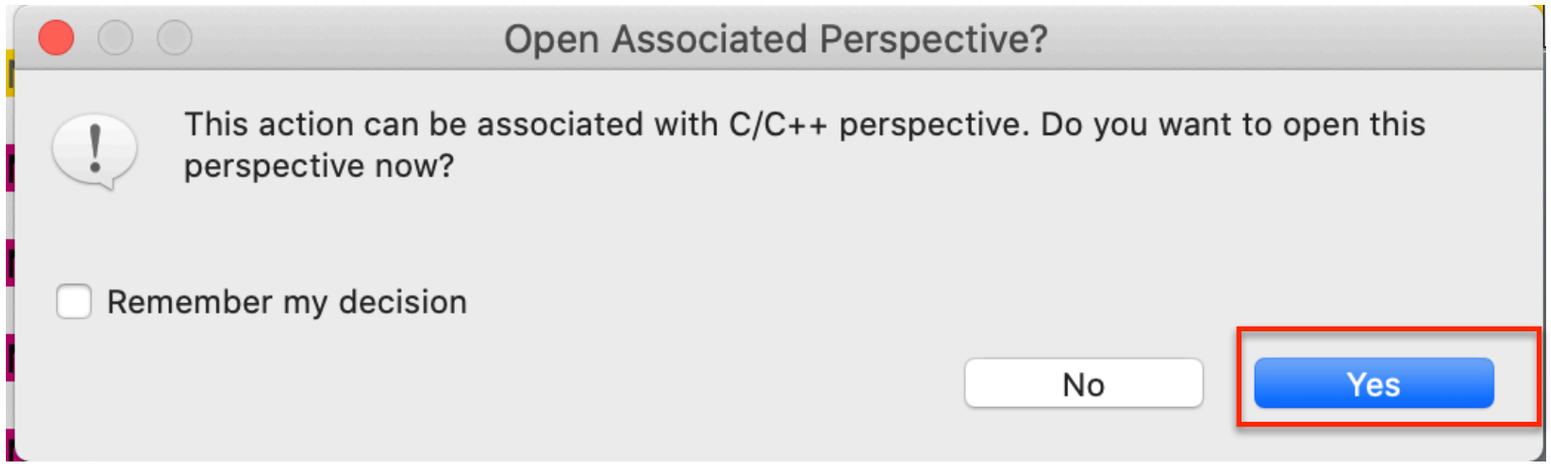




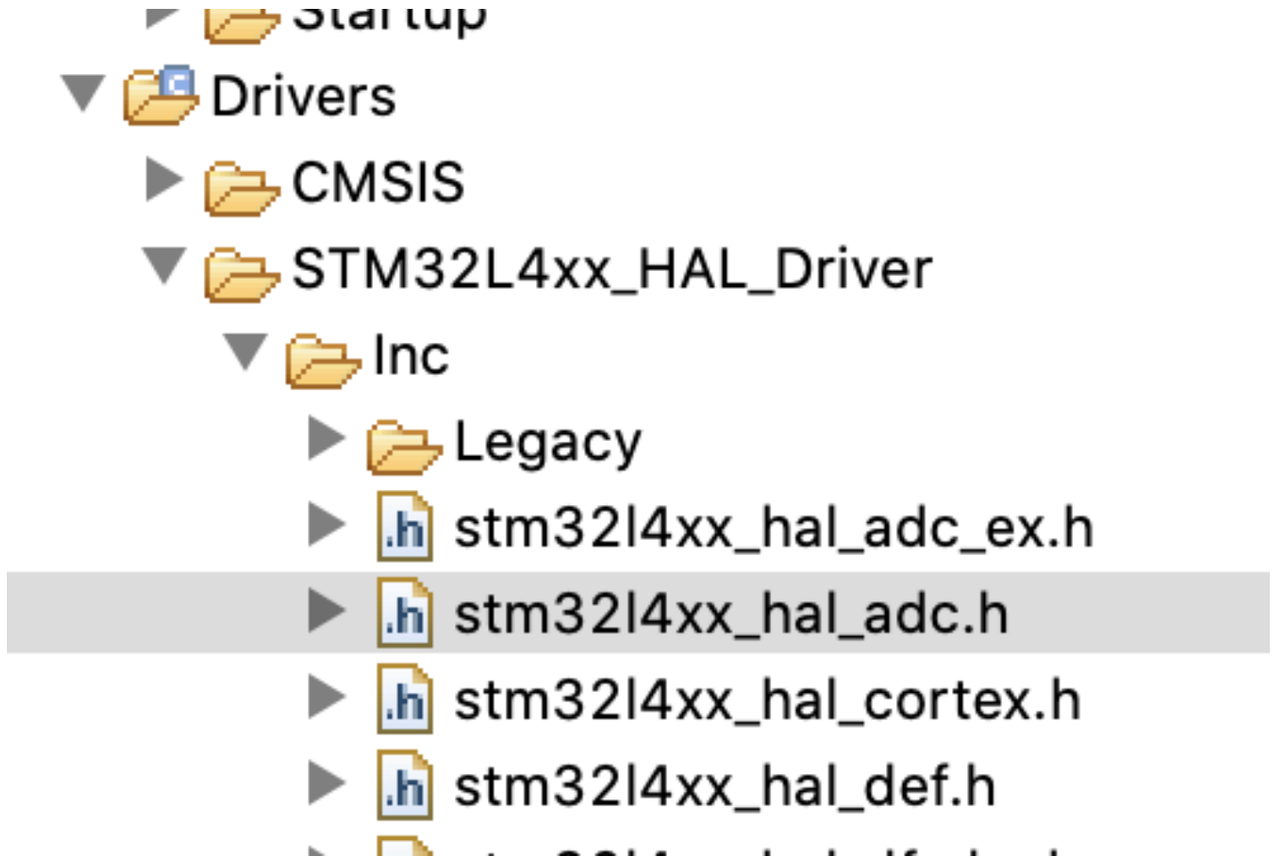
Generate Code



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

. . .

```
178
179   ADC_OversamplingTypeDef Oversampling;   /*!< Specify the Oversampling parameters.
180                                                Caution: this setting overwrites the previous oversampling configuration if
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;
189
```

stm32l4xx_hal_adc.h
ADC_ChannelConfTypeDef

```
201 typedef struct
202 {
203   uint32_t Channel;              /*!< Specify the channel to configure into ADC regular group.
204                                       This parameter can be a value of @ref ADC_HAL_EC_CHANNEL
205                                       Note: Depending on devices and ADC instances, some channels may not be available on device package pins. Refer to device datasheet for
206
207   uint32_t Rank;                 /*!< Specify the rank in the regular group sequencer.
208                                       This parameter can be a value of @ref ADC_HAL_EC_REG_SEQ_RANKS
209                                       Note: to disable a channel or change order of conversion sequencer, rank containing a previous channel setting can be overwritten by
210                                       the new channel setting (or parameter number of conversions adjusted) */
211
212   uint32_t SamplingTime;         /*!< Sampling time value to be set for the selected channel.
213                                       Unit: ADC clock cycles
214                                       Conversion time is the addition of sampling time and processing time
215                                       (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).
216                                       This parameter can be a value of @ref ADC_HAL_EC_CHANNEL_SAMPLINGTIME
217                                       Caution: This parameter applies to a channel that can be used into regular and/or injected group.
218                                             It overwrites the last setting.
219                                       Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor),
220                                             sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time s
221                                             Refer to device datasheet for timings values. */
222
223   uint32_t SingleDiff;           /*!< Select single-ended or differential input.
224                                       In differential mode: Differential measurement is carried out between the selected channel 'i' (positive input) and channel 'i+1' (nega
225                                                             Only channel 'i' has to be configured, channel 'i+1' is configured automatically.
226                                       This parameter must be a value of @ref ADC_HAL_EC_CHANNEL_SINGLE_DIFF_ENDING
227                                       Caution: This parameter applies to a channel that can be used in a regular and/or injected group.
228                                             It overwrites the last setting.
229                                       Note: Refer to Reference Manual to ensure the selected channel is available in differential mode.
230                                       Note: When configuring a channel 'i' in differential mode, the channel 'i+1' is not usable separately.
231                                       Note: This parameter must be modified when ADC is disabled (before ADC start conversion or after ADC stop conversion).
232                                             If ADC is enabled, this parameter setting is bypassed without error reporting (as it can be the expected behavior in case
233                                       of another parameter update on the fly) */
234
235   uint32_t OffsetNumber;         /*!< Select the offset number
236                                       This parameter can be a value of @ref ADC_HAL_EC_OFFSET_NB
237                                       Caution: Only one offset is allowed per channel. This parameter overwrites the last setting. */
238
239   uint32_t Offset;               /*!< Define the offset to be subtracted from the raw converted data.
240                                       Offset value must be a positive number.
241                                       Depending of ADC resolution selected (12, 10, 8 or 6 bits), this parameter must be a number between Min_Data = 0x000 and Max_Data = 0xF
242                                       0x3FF, 0xFF or 0x3F respectively.
243                                       Note: This parameter must be modified when no conversion is on going on both regular and injected groups (ADC disabled, or ADC enabled
244                                             without continuous mode or external trigger that could launch a conversion). */
245
246 } ADC_ChannelConfTypeDef;
```

ADC API

Functions

stm32l4xx_hal_adc.h
Polling Mode

```
1741
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

```
1747
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

```
1758
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()

```
117 /**
118   * @brief  Perform an ADC automatic self-calibration
119   *         Calibration prerequisite: ADC must be disabled (execute this
120   *         function before HAL_ADC_Start() or after HAL_ADC_Stop() ).
121   * @param  hadc       ADC handle
122   * @param  SingleDiff Selection of single-ended or differential input
123   *         This parameter can be one of the following values:
124   *           @arg @ref ADC_SINGLE_ENDED       Channel in mode input single ended
125   *           @arg @ref ADC_DIFFERENTIAL_ENDED Channel in mode input differential ended
126   * @retval HAL status
127   */
128 HAL_StatusTypeDef HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef *hadc, uint32_t SingleDiff)
129 {
```

HAL_ADC_Start()

```
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)
```

ADC **Polling** Code Segment

```
138   // Calabrate 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       snprintf(buf, sizeof(buf), "%u ", value);
160
161       HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
162
163
164   }
```

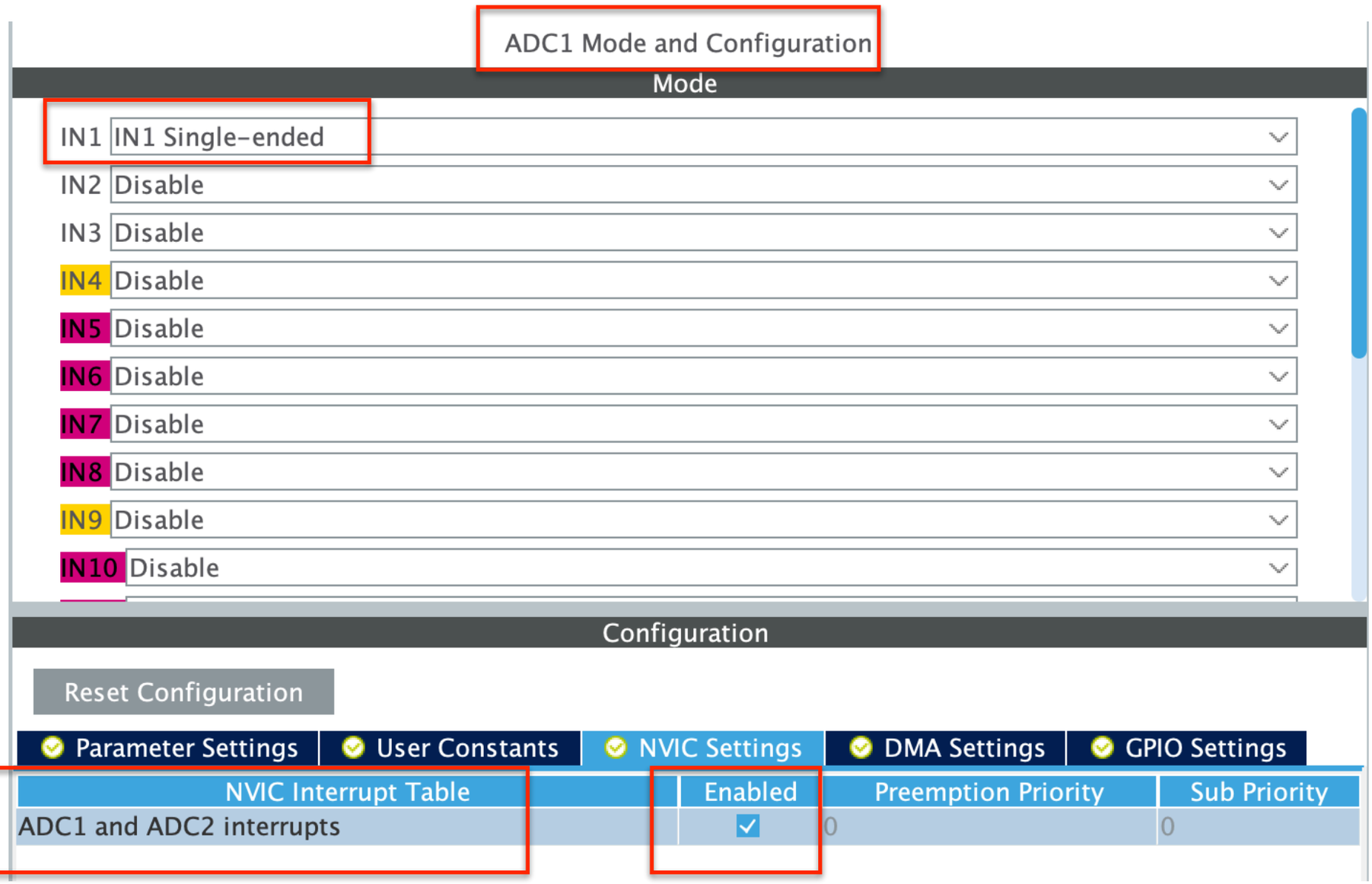
Code Segments

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)
```

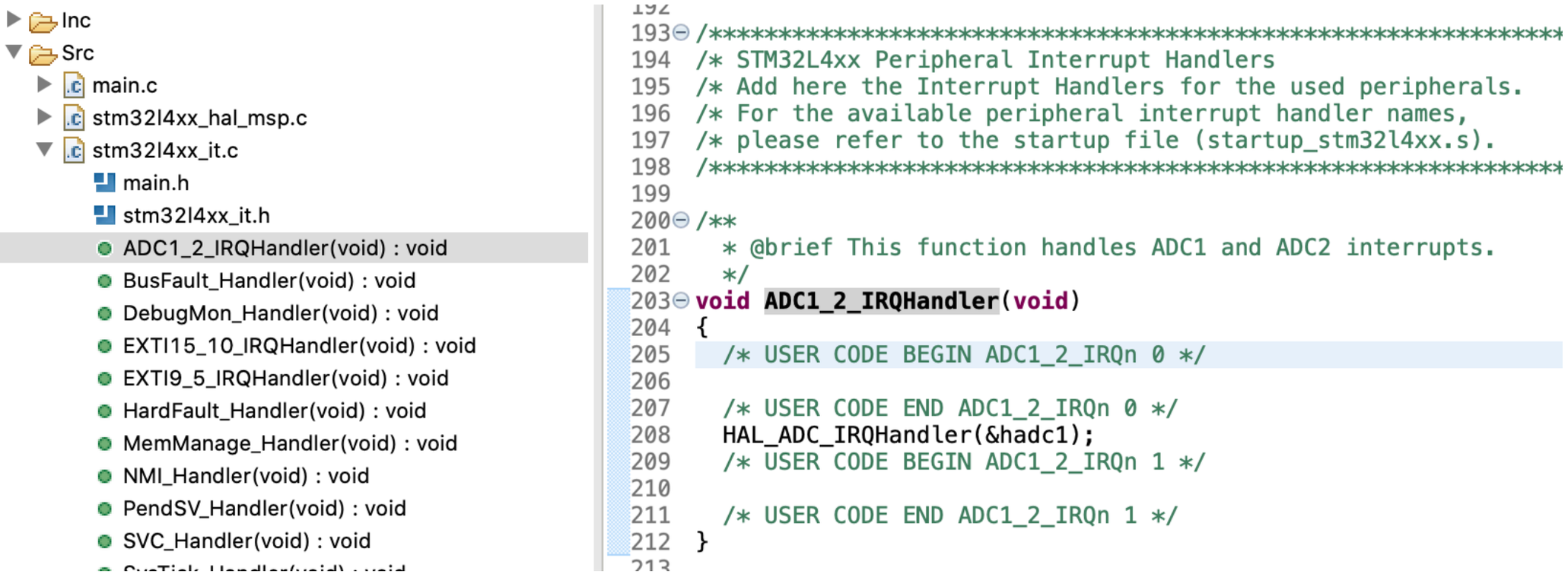
ADC **Interrupt** Driven Code Segment - Part 1



ADC **Interrupt** Driven Code Segment - Part 2

```
154
155   // Calibrate ADC
156   HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
157
158   while (1)
159   {
160     /* USER CODE END WHILE */
161
162     /* USER CODE BEGIN 3 */
163       HAL_GPIO_TogglePin(LED3_WIFI__LED4_BLE_GPIO_Port,LED3_WIFI__LED4_BLE_Pin);
164       HAL_Delay(1000); //500 msec = 1 sec
165
166       // ADC Start Conversion
167       //HAL_ADC_Start(&hadc1);
168       HAL_ADC_Start_IT(&hadc1);
169       //HAL_ADC_Start_DMA(&hadc1, (uint32_t *) &adc_value, 1);
170
171       // Pool for results - Timeout is 10ms
172       //HAL_ADC_PollForConversion(&hadc1, 10);
173
174       //uint16_t value = HAL_ADC_GetValue(&hadc1);
175
176       // Send value to console
177       //char buf[100];
178       //snprintf(buf, sizeof(buf), "%u ", value);
179
180       //HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
181
182
183   }
184   /* USER CODE END 3 */
185 }
186
```

ADC **Interrupt** Driven Code Segment - Part 3



ADC **Interrupt** Driven Code Segment - Part 4

```
2230    */
2231  void HAL_ADC_IRQHandler(ADC_HandleTypeDef *hadc)
2232  {
2233    uint32_t overrun_error = 0UL; /* flag set if overrun occurrence has to be considered a
2234    uint32_t tmp_isr = hadc->Instance->ISR;
2235    uint32_t tmp_ier = hadc->Instance->IER;
2236    uint32_t tmp_adc_inj_is_trigger_source_sw_start;
2237    uint32_t tmp_adc_reg_is_trigger_source_sw_start;
2238    uint32_t tmp_cfgr;
2239  #if defined(ADC_MULTIMODE_SUPPORT)
2240    const ADC_TypeDef *tmpADC_Master;
2241    uint32_t tmp_multimode_config = LL_ADC_GetMultimode(__LL_ADC_COMMON_INSTANCE(hadc->Ins
2242  #endif
2243
2244    /* Check the parameters */
2245    assert_param(IS_ADC_ALL_INSTANCE(hadc->Instance));
2246    assert_param(IS_ADC_EOC_SELECTION(hadc->Init.EOCSelection));
2247
2248    /* ========== Check End of Sampling flag for ADC group regular ========== */
2249    if (((tmp_isr & ADC_FLAG_EOSMP) == ADC_FLAG_EOSMP) && ((tmp_ier & ADC_IT_EOSMP) == ADC
2250    {
2251      /* Update state machine on end of sampling status if not in error state */
2252      if ((hadc->State & HAL_ADC_STATE_ERROR_INTERNAL) == 0UL)
2253      {
2254        /* Set ADC state */
2255        SET_BIT(hadc->State, HAL_ADC_STATE_REG_EOSMP);
2256      }
2257
2258      /* End Of Sampling callback */
2259  #if (USE_HAL_ADC_REGISTER_CALLBACKS == 1)
2260      hadc->EndOfSamplingCallback(hadc);
2261  #else
2262      HAL_ADCEx_EndOfSamplingCallback(hadc);
2263  #endif /* USE_HAL_ADC_REGISTER_CALLBACKS */
2264
```

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
```

ADC **Interrupt** Driven Code Segment - Part 6

This code in main.c

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {

	uint16_t value = HAL_ADC_GetValue(&hadc1);

	// Send value to console
	char buf[100];
	snprintf(buf, sizeof(buf), "%u ", value);

	HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
}

/* USER CODE END 0 */
```

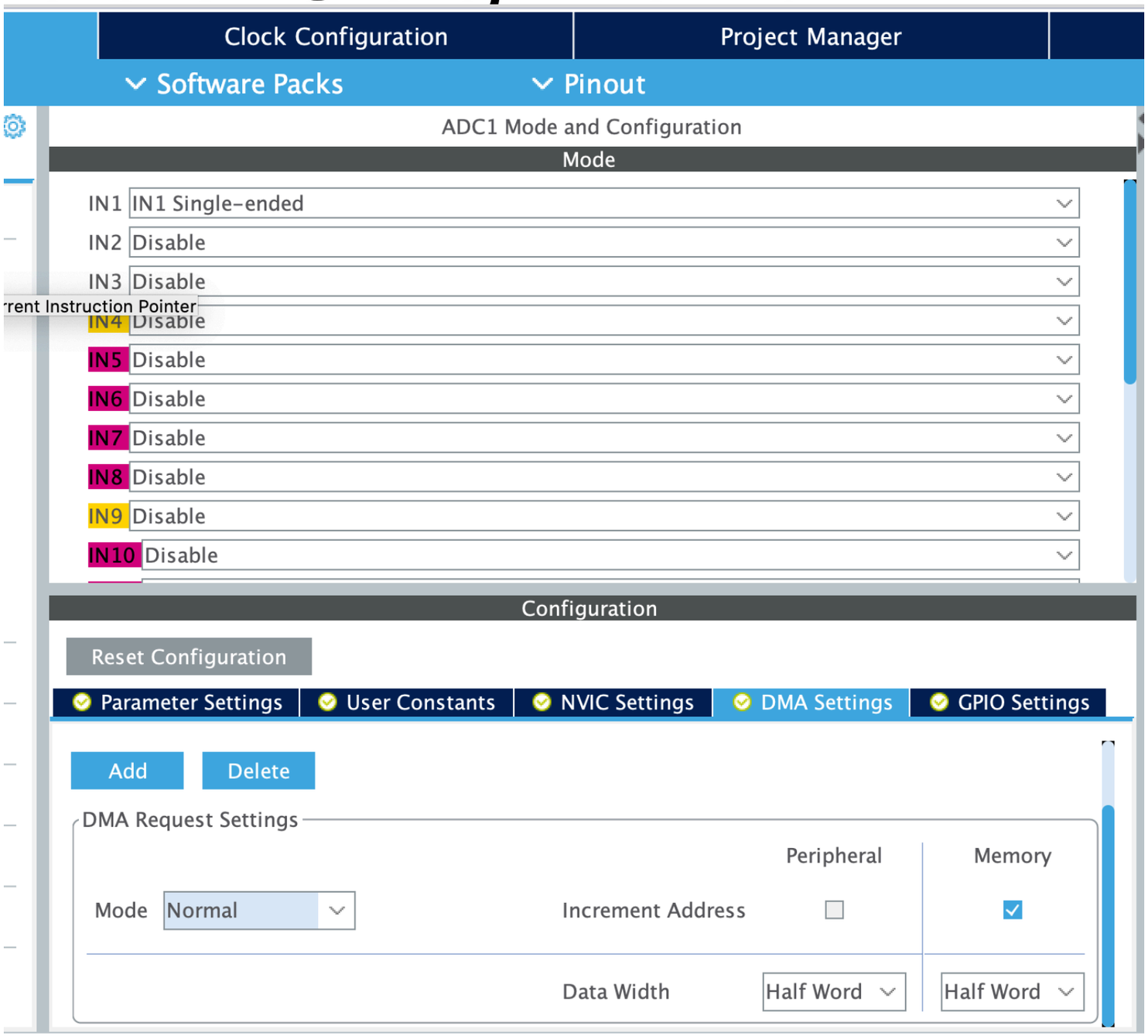
Code Segments

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)
```

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