**Chapter 6: Analog Interfacing**

**Quantization**

**Quantization:** Process of selecting a discrete digital value to represent an analog value

**Sampling:** Process of converting a continuous-time signal to a series of discrete-time samples

Digital electronics work with binary values, so the number of discrete output values is typically a power of 2

* The resolution describes the number of bits (B) used to hold the output value
* 8-bit code -> 2^8=256 possible output values
* Note that the value of *n* does not represent an exact voltage, but instead a range of voltages
* As the resolution B increases, the quantization becomes more accurate
  + The maximum quantization error is typically half of the voltage range
* **Forward Transfer Function**:
* **Inverse Transfer Functions:**

|  |  |
| --- | --- |
|  | Converted Code |
|  | Sampled Input Voltage |
|  | Upper Voltage Reference |
|  | Lower Voltage Reference |
|  | Number of bits of resolution in ADC |

Where,

**Sampling**

A sampled signal is a discrete time representation of a continuous time signal; that is, it is a series of individual samples

* The sampling rate determines how often an input is measured or how often an output is updated
* If any energy at a frequency of or higher, those signal components will appear in the sampled signal at different(lower) frequencies, distorting the sampled signal with aliasing

**Aliasing:** Distortion of signal resulting from sampling at too low a frequency

* Ways to prevent aliasing:
  + Choose to be more than twice the frequency of the highest signal frequency of interest,
  + Use a low-pass/anti-aliasing filter to remove/attenuate the signal components above

**Digital-to-Analog Conversion**

A digital-to-analog converter (DAC) generates an analog output signal based on the digital input value

* Output signal may be a voltage or a current depending on the type of DAC

A transfer function defines the relationship between the digital input value and the analog output voltage

* For a DAC with a lower reference voltage of 0 V, and an upper reference voltage of and bits of resolution, the general transfer function is:

**Converter Architectures**

* Two common DAC architectures:
  + Resistor ladder
    - An **N-bit resistor ladder** used resistors of equal value connected in series between the upper and lower reference voltages
    - These resistors form a voltage divider with equally spaced voltages at the taps
  + R-2R ladder
    - An **R-2R resistor ladder** used resistors of one values (R) and resistors of twice that value (2R)
* Regardless of type of DAC, an amplifier is typically used to buffer the output signal
* Also possible to create analog output with timer peripheral in PWM mode coupled with a lower pass filter

**Kinetis KL25Z DAC**

* The KL25Z has a 12-bit DAC
  + Two upper references
  + Lower reference connected to ground
  + Amplifier buffers output signal
* Control register DACx\_C0 controls various aspects of the DAC **(Pg. 158)**
  + DAC is enabled by writing a 1 to DACEN in DACx\_C0
  + Output buffer’s power consumption can be reduced by writing a one to LPEN in DACx\_C0 at the cost of increasing the output’s response time
  + Upper reference can be connected to one of the two sources ( or ) using a mux controlled by DACRFS field in DACx\_C0
* Output data for the DAC is 12 bits long and is stored in the DACDAT register
  + Upper 4 bits written to DACx\_DAT0H
  + Lower byte written to DACx\_DAT0L
* Transfer function for this DAC includes an offset of 1 added to n. This allows the output voltage to range from to
* DAC also offers a buffered output mode
  + New data can be written to DACx\_DATA0H and DACx\_DATA0L, but the DAC continues to generate the old output voltage until it receives a trigger signal from a hardware timer or a software write to the DACSWTRG field in DACx\_C0

Example:

1. #define DAC\_POS (30)
3. void Init\_DAC(void) {
4. // Enable clock to DAC and Port E
5. SIM->SCGC6 |= SIM\_SCGC6\_DAC0\_MASK;
6. SIM->SCGC5 |= SIM\_SCGC5\_PORTE\_MASK;
8. // Select analog for pin
9. PORTE->PCR[DAC\_POS] &= ~PORT\_PCR\_MUX\_MASK;
10. PORTE->PCR[DAC\_POS] |= PORT\_PCR\_MUX(0);
12. // Disable buffer mode
13. DAC0->C1 = 0;
14. DAC0->C2 = 0;
16. // Enable DAC, select VDDA as reference voltage
17. DAC0->C0 = DAC\_C0\_DACEN\_MASK | DAC\_C0\_DACRFS\_MASK;
18. }
20. void Triangle\_Output(void) {
21. int i = 0;
22. int change = 0;
24. while (1) {
25. DAC0->DAT[0].DATL = DAC\_DATL\_DATA0(i);
26. DAC0->DAT[0].DATH = DAC\_DATH\_DATA1(i >> 8);
28. i += change;
29. if (i == 0) {
30. change = 1;
31. } else {
32. change = -1;
33. }
34. }
35. }

**Analog Comparator**

**Comparator:** Circuit which compares two analog inputs to identify larger value

**Kinetis KL25Z Comparator** **(Pg. 161)**

* In order to enable the comparator, the clock gating must be enabled by writing a one to the CMP bit in the SIM\_SCGC4 register, and then writing one to the comparator enable bit (EN) in CMPx\_CR1
* Each comparator input can be connected to one of eight possible signals (Table 6.1)
  + The CMPx\_MUXCR register contains the fields PSEL and MSEL to control which signals are connected to the plus and minus inputs
* In order to use the comparator’s 6-bit DAC, the DAC must be enabled by writing a 1 to the DACEN in CMPx\_DACCR
  + Writing a value of n to VOSEL field of CMPx\_DACCR results in a DAC output voltage of:
  + VRSEL controls whether the DAC’s upper reference voltage is connected to or
* The comparator’s output signal CMP0\_OUT (COUT of CMPx\_SCR) can drive a digital output pin
  + To do this, set the OPE bit in CMPx\_CR1 to 1 and configure the appropriate pin control register multiplexer as shown table 6.2 (Pg. 163)

Example:

1. void Init\_Comparator(void) {
2. // Enable clock to comparator
3. SIM->SCGC4 |= SIM\_SCGC4\_CMP\_MASK;
5. // Enable comparator
6. CMP0->CR1 = CMP\_CR1\_EN\_MASK;
8. // Select input channels
9. // Plus: channel 5 on Port E bit 29
10. // Minus: CMP DAC is channel 7
11. CMP0->MUXCR = CMP\_MUXCR\_PSEL(5) | CMP\_MUXCR\_MSEL(7);
13. // Enable DAC, set reference voltage at 1.85V (64 \* 1.85/3.3 = 36)
14. CMP0->DACCR = CMP\_DACCR\_DACEN\_MASK | CMP\_DACCR\_VOSEL(36);
16. // Enable interrupt for comparator on both edges
17. CMP0->SCR = CMP\_SCR\_IEF\_MASK | CMP\_SCR\_IER\_MASK;
19. // Configure Interrupt
20. NVIC\_SetPriority(CMP0\_IRQn, 128);
21. NVIC\_ClearPendingIRQ(CMP0\_IRQn);
22. NVIC\_EnableIRQ(CMP0\_IRQn);
23. }
25. void CMP0\_IRQHandler(void) {
26. // Set break point here to observe operation
27. if (CMP0->SCR & CMP\_SCR\_CFR\_MASK) { // rising
28. // light green LED
29. Control\_RGB\_LEDs(0, 1, 0);
30. } else if (CMP0->SCR & CMP\_SCR\_CFF\_MASK) { // falling
31. // light red LED
32. Control\_RBG\_LEDs(1, 0, 0);
33. }
34. // Clear flags, keep interrupt on both edges enabled
35. CMP0->SCR = CMP\_SCR\_IEF\_MASK | CMP\_SCR\_IER\_MASK | CMP\_SCR\_CFR\_MASK | CMP\_SCR\_CFF\_MASK;
36. }

**Analog-to-Digital Conversion**

**Converter Architectures:**

* Flash
  + A B-bit ADC can be built out of analog comparators operating in parallel, each with a different reference voltage
  + Resulting B-Bit code is created with digital logic that encodes the output bits of the comparators
  + This is called a flash architecture because it is really fast
    - delay is only comparator delay + digital encoder delay
    - expensive; requires a lot of comparators, with each additional bit doubling the number of comparators needed
* Successive Approximation
  + Uses a single comparator to make a series of comparisons, changing its reference voltage for each
  + Performs a binary search to quantize the input (see Figure 6.17 on Pg. 165)
  + Not as fast as flash ADC, but only requires one comparator
    - Circuit size is small and does not grow as quickly as flash ADC when increasing resolution
    - Adding one bit of resolution does slow down conversion slightly as it requires an additional comparison to produce the output

**Inputs**

* ADCs often include an input multiplexer to allow a single ADC to select one of the multiple input channels to measure
* A sample an hold circuit can be used to sample the input signal and then hold it fixed during the conversion time
* A single-ended signal represents information with the voltage difference between the signal and ground
* Differential signals represents information with the voltage difference between two signals, neither of which is ground (this can reduce noise)

**Triggering**

* ADCs typically include two types of triggers: software and hardware
  + Software triggers requires software to write a value to a specific ADC control register to start the conversion
  + Hardware trigger requires a hardware signal to be asserted by a circuit, whether outside the MCU or within it
    - Hardware timer could generate a signal every millisecond to trigger the ADC operation
* ADC indicates the conversion has been completed by setting a flag in an ADC status register; possibly also signaling an interrupt request
  + Conversion is available in digital form in an ADC result register

**Kinetis KL25Z ADC**

* Analog input data is routed through an input mux to the SAR converter
* A trigger signal starts the conversion process
* Control sequencer steps through a series of activities at a rate determined by the clock signal
* Output of SAR converter may be processed before being placed into a result register
* ADC may generate an interrupt when conversion completes
* Compare logic can be used to discard the results in/outside of a specified range
* **Analog Inputs:**
  + input mux can select one of 24 single-ended inputs or one of 4 pairs of differential inputs
    - input channel is selected by the ADCH field in the ADCx\_SC1n register
    - differential input mode is selected by setting the DIFF bit in ADCx\_SC1n
    - Several special MUX inputs:
      * ch26 = on-chip temperature sensor
      * ch27 = on-chip fixed 1.0 V reference (band gap reference)
      * ch29 = connected to the high-reference voltage ()
      * ch30 is connected to the low-reference voltage ()
* **Voltage references:**
  + ADC uses two reference voltages: and
    - These can be selected from two pairs of reference voltages and or and using the REFSEL bit of the ADCx\_SC2 register
    - On the KL25Z, is connected to the analog supply voltage which is nominally 3.3V
* **SAR Converter**
  + Supports various resolutions from 8 to 16 bits in signed or unsigned formats
    - Specified by the MODE field of ADCx\_CFG1
  + Other conversion options also available
    - Low-power; reduces maximum clock speed. Set bit ADLPC1 in ADC\_CFG1 to enable
    - Longer sampling time; reduces input noise. Set ADLSMP in ADCx\_CFG1 to enable long samples, and then use the ADLSTS field in ADCx\_CFG2 to add from 6 to 20 ADCK cycles to each conversion sample time
    - Continuous conversion; back-to-back conversions. Enabled by setting ADCO in ADCx\_SC3
* **Conversion Trigger**
  + ADTRG bit in ADCx\_SC2 register selects either software or hardware trigger
    - Software trigger consists of writing to SC1A to start conversion
    - Hardware triggering consists of starting the conversion with a specific hardware signal that indicates when an event has occurred. These include timer signals, comparator output, and an external trigger signal
      * Trigger source is selected using the ADC0TRGSEL field of the register SIM\_SOPT7
* **Conversion Clock**
  + Conversion clock signal ADCK determines how quickly the ADC samples and then converts the input data
    - Depending on configuration, a sample can take from 4 to 26 ADCK cycles and a conversion can take from 20 to 71 ADCK cycles
  + ADCK signal has frequency restrictions:
    - must be between 1 and 18 MHz when the ADC is operating with the resolution of up to 13 bits, or between 2 and 12 MHz for higher resolutions
  + Four possible inputs to conversion clock. Selected with the ADICLK field of the ADCx\_CFG1 register
    - Bus clock
    - Bus clock divided by two
    - ADACK, a local clock that can keep running when the rest of the CPU is stopped
    - ALTCLK, an alternate clock source
  + Input clock is divided by a factor specified by a code in the ADIV field of register ADCx\_CFG1
* **Conversion Completion and ISR**
  + Completed conversion can be indicated in two ways:
    - control hardware sets COCO bit in ADCx\_SC1x to 1
    - ADC can generate an interrupt if AIEN in SC1 is set
      * ADC ISR is ADC0\_IRQHandler
  + Next conversion will occur automatically if continuous conversion mode is set (ADCO in ADCx\_SC3)
* **Special Output Processing**
  + Can remove offsets determined during ADC calibration
  + Can average multiple conversion results automatically if bit AVGE in ADCx\_SC3 is set
  + Can format output data, performing justification and extension to create a 16-bit result
  + Compare function can detect conversion results that exceed a certain threshold or range and then generate an interrupt
    - Threshold comparison value located in ADCx\_CV1
    - Range comparison value located in ADCx\_CV1 and ADCx\_CV2
    - Comparison performed selected with the ACFGT and ACREN fields of the ADCx\_SC2 reg

Examples:

1. #define ADC\_POS (20)
3. void Init\_ADC(void) {
4. // Enable clock to ADC and Port E
5. SIM->SCGC6 |= SIM\_SCGC6\_ADC0\_MASK;
6. SIM->SCGC5 |= SIM\_SCGC5\_PORTE\_MASK;
8. // Select analog for pin
9. PORTE->PCR[ADC\_POS] &= ~PORT\_PCR\_MUX\_MASK;
10. PORTE->PCR[ADC\_POS] |= PORT\_PCR\_MUX(0);
12. // Low power configuration, long sample time, 16-bit single-ended conversion
13. // Bus clock input
14. ADC0-CFG1 = ADC\_CFG1\_ADLPC\_MASK | ADC\_CFG1\_ADLSMP\_MASK | ADC\_CFG1\_MODE(3) |
15. ADC\_CFG1\_ADICLK(0);
17. // Software Trigger, compare function disabled, DMA disabled
18. // Voltage references VREFH and VREFL
19. ADC0-SC2 = ADC\_SC2\_REFSEL(0);
20. }
22. float Measure\_Temperature(void) {
23. float n;
24. float temp;
26. ADC0->SC1[0] = 0x00; // Start conversion on channel 0
28. // Wait for conversion to finish
29. while (!(ADC0->SC1[0] & ADC\_SC1\_COCO\_MASK)) {}
31. // Read result, convert to floating point
32. n = (float) ADC0->R[0];
34. // Calculate temperature using polynomial equation
35. // Assumes ADC is in 16 bit mode and has VREF = 3.3V
36. temp = -36.9861 + n\*(0.0155762 + n \* (-1.43216E-06 + n \*...) (see pg. 173 if needed)
38. return temp;
39. }