DAC0 Continuous D/A Conversion

The DAC0 (DAC zero) module on the KL05 is a 12-bit digital-to-analog (D/A) converter. It converts an unsigned 12-bit value DAC0_DAT to an analog voltage DAC0_OUT (V_{out}) proportional to V_{in} according to the following formula.

$$V_{out} = \frac{1 + \text{DAC0\_DAT}}{4096} V_{in}$$

On the Freedom Board FRDM-KL05Z, DAC0_OUT from can be connected through KL05 port B pin 1 to I/O header J8 pin 2 (D1). (Note this pin is also used for the UART0_TX connection to the KL05Z virtual serial port through the OpenSDA USB connector, so the virtual serial port cannot be used if DAC0_OUT is connected to this pin.)

Using the DAC in a program for continuous D/A conversion relative to VDDA, (3.3 V), requires first configuring the DAC for non-buffered operation with VDDA as the reference voltage. Afterward, the 12-bit value DAC0_DAT written to the DAC data register is converted to an analog voltage DAC0_OUT in the range (0, 3.3] V, as indicated by the formula above. Whenever DAC0_DAT is changed, DAC0_OUT will change.

Program initialization

The following initializations are required for continuous D/A conversion with DAC0 (without output to KL05Z I/O header J8 pin 2).

- DAC0 module clock enabled in SIM_SCGC6: DAC0 = 1
- DAC0 DMA disabled and buffer disabled in DAC0_C1:
  DMAEN = 0; DACBFEN = 0
- DAC0 enabled with VDDA as reference voltage and read pointer interrupts disabled in DAC0_C0:
  DACEN = 1; DACRFS = 1; DACBTIEN = 0; DACBBIEN = 0

Assembly language code. The following assembly language code generates an analog output of 0.81 mV from DAC0. All symbols not defined in this code are from the RIT CMPE 250 include file MKL05Z4.s.

```
;DAC0_C0 symbol
DAC_C0_ENABLE               EQU  (DAC_C0_DACEN_MASK :OR: \
                                  DAC_C0_DACRFS_MASK)
;DAC0_C1 symbol
DAC_C1_BUFFER_DISABLED      EQU  0x00
;DAC0_DAT symbols
DAC_DATH_MIN                EQU  0x00
DAC_DATL_MIN                EQU  0x00

;Enable DAC0 module clock
            LDR   Ri,=SIM_SCGC6
            LDR   Rj,=SIM_SCGC6_DAC0_MASK
            LDR   Rk,[Ri,#0]        ;current SIM_SCGC6 value
            ORRS  Rk,Rk,Rj          ;only DAC0 bit changed (set)
            STR   Rk,[Ri,#0]        ;update SIM_SCGC6
;Load base address for DAC0
            LDR   Ri,=DAC0_BASE
;Set DAC0 DMA disabled and buffer disabled
            LDR   Rj,=DAC_C1_OFFSET
            MOVS  Rk,#DAC_C1_BUFFER_DISABLED
            STR   Rk,[Ri,Rj]
;Set DAC0 enabled with VDDA as reference voltage and read pointer interrupts disabled
            LDR   Rj,=DAC_C0_OFFSET
            MOVS  Rk,#DAC_C0_ENABLE
            STRB  Rk,[Ri,Rj]
;Set DAC0 output voltage at minimum value
            MOVS  Rj,#DAC_DATL_MIN
            STRB  Rj,[Ri,#DAC0_DAT0L_OFFSET]
            MOVS  Rj,#DAC_DATH_MIN
            STRB  Rj,[Ri,#DAC0_DAT0H_OFFSET]
```

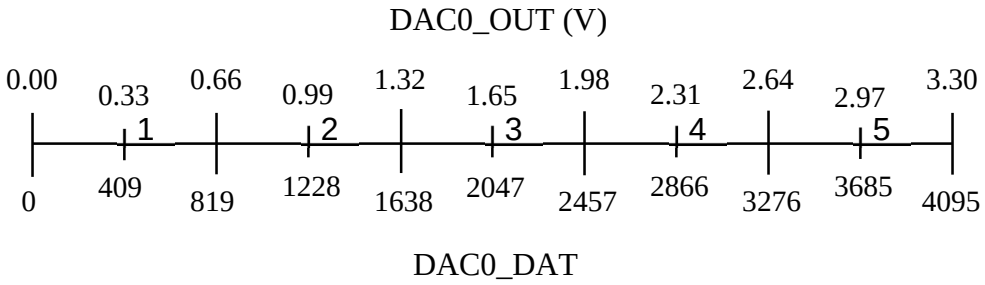
C code. The following C code generates an analog output of 0.81 mV from DAC0 that can be accessed through port E pin 30 connected to KL05Z I/O header J4 pin 11. All symbols not defined in this code are from the Keil Freescale include file MKL05Z4.h.

```
/* DAC0_C0 symbol */
#define DAC_C0_ENABLE (DAC_C0_DACEN_MASK |
                       DAC_C0_DACRFS_MASK)
/* DAC0_C1 symbol */
#define DAC_C1_BUFFER_DISABLED (0x00u)
/* DAC0_DAT symbols */
#define DAC_DATL_MIN (0x00u)
#define DAC_DATH_MIN (0x00u)

/* Enable DAC0 module clock */
SIM->SCGC6 |= SIM_SCGC6_DAC0_MASK;
/* Set DAC0 DMA disabled and buffer disabled */
DAC0->C1 = DAC_C1_BUFFER_DISABLED;
/* Set DAC0 enabled with VDDA as reference voltage */
/* and read pointer interrupts disabled            */
DAC0->C0 = DAC_C0_ENABLE;
/* Set DAC0 output voltage at minimum value */
DAC0->DAT[0].DATL = DAC_DATL_MIN;
DAC0->DAT[0].DATH = DAC_DATH_MIN;
```

Program change of DAC0 output voltage

To change the value output by DAC0, the program should change the 12-bit value in in the DAC0 data register, (bits 11–8 in DAC0_DATH bits 3–0, and bits 7–0 in DAC0_DATL bits 7–0). If *n* distinct analog values are required for an application, an approach that results in maximum separation of the analog values is to divide the range into *n* segments. In lab, five analog values are needed, and the figure below shows the analog range divided into five equal segments with endpoints and midpoints values indicated.



Division of digital input domain and analog output range into five equal segments

Since the analog value output by DAC0 over the possible analog range of (0.0, 3.3] V is proportional to the digital value input over its corresponding domain of [0, 4095], maximum separation of analog values will occur if the digital values input are also as far apart as possible over the domain. The following constants create a lookup table in ROM that consists of the digital value at the midpoint of the each segment in the figure above.

```
;DAC0
DAC0_BITS   EQU  12
DAC0_STEPS  EQU  4096
;LED
LED_LEVELS  EQU  5
;ROM lookup table of digital values for conversion to analog
            EXPORT  DAC0_table_0  ;make available to C program
DAC0_table_0
DAC0_table
            DCW     (DAC0_STEPS / (LED_LEVELS * 2))
            DCW     ((DAC0_STEPS * 3) / (LED_LEVELS * 2))
            DCW     ((DAC0_STEPS * 5) / (LED_LEVELS * 2))
            DCW     ((DAC0_STEPS * 7) / (LED_LEVELS * 2))
            DCW     ((DAC0_STEPS * 9) / (LED_LEVELS * 2))
```

Assembly language code. The following assembly language code could be used to set DAC0_OUT to the voltage at the midpoint of segment two in the preceding figure.

```
   MOVS   Ri,#5          ;Desired segment
   SUBS   Ri,Ri,#1       ;Corresponding look-up table entry index
   LSLS   Ri,Ri,#1       ;Convert to byte index
   LDR    Rj,=DAC0_table
   LDRH   Rj,[Rj,Ri]     ;Corresponding look-up table entry for digital value out
   LDR    Rk,=DAC0_BASE
   STRB   Rj,[Rk,#DAC0_DATL_OFFSET]      ;Output low byte to DAC0
   LSRS   Rj,Rj,#8                       ;High byte of digital value out
   STRB   Rj,[Rk,#DAC0_DATJ_OFFSET]      ;Output low byte to DAC0
```

C code. The following code accesses the assembly language ROM lookup table to set DAC0_OUT to the voltage at the midpoint of segment two in the preceding figure.

```
/* Prototype to enable access to ROM table from assembly language object */
extern const UInt16 DAC0_table_0; /* Provided in header file */
/* Initialization for C assess to assembly language ROM table */
const UInt16 *DAC0_table = &DAC0_table_0;

int    Segment;

   Segment = 2;  /* Desired segement of voltage range */
/* Set DAC0 output voltage to midpoint of desired segment of voltage range */
DAC0->DAT[0].DATL = (UInt8) (DAC0_table [Segment - 1] & 0xFF);
DAC0->DAT[0].DATH = (UInt8) (DAC0_table [Segment - 1] >> 8);
```