Lab 3 – ADC and DAC conversion verification

EE390 – Smart Sensor Systems

ECET Department

Minnesota State University, Mankato

Colin Roskos

2/10/2023

**Purpose:**

This lab is designed to introduce the use the ADC system through the use of the DAC1 to ADC internal channel.

**Program Description:**

The program consists of initializing components, system clock, DAC, ADC, USART;

System Clock, USART: 80MHz, USART1 115200 8bit 1sb

DAC1, GPIOA, MCR = 1 for normal mode output to pin and internal channel

ADC; Connect AHB2 to ADC2, wakeup ADC from deep power down mode, turn on the ADC voltage regulator – this takes TADCVREG\_STUP = 20us so a delay of 20 us is called. The TSEN (Temperature Sensor Enable register) is set to 0 indicating internal mode. Clear MDMA, set clock mode to 1 – synchronous clock mode, sample time set to 6.5 ADC clock cycles, and set to software trigger right aligned and 12-bit resolution – default 0’s in CFGR register.

The main program is straight forward with the value written to the DAC and the ADC2->DR being read and displayed to USART. The USART programs are implemented to have features closer to printf for ease of string printing.

**Program List:**

// author defined libraries

main.c

main.h

dma.c

dma.h

sysclk.c

sysclk.h

system.c

system.h

adc.c

adc.h

// external libraries

stm32l475xx.h

stdint.h

stdlib.h

**Discussion:**

This lab was fairly straightforward, with the only caveat being the implementation of the MCR register mode for the DAC causing a hangup in implementation. Otherwise the ADC when implemented by the rules pointed out in the STM32L4x5 reference manual and slides needed no real changes, and the only difficulty was in tracking down the channel 17 enable register (listed as CH17SEL but the 475 library listing it as TSEN). The output is reflected in Figure 1

**LAB3 CODE**

**MAIN.C**

/\*

\* Lab 3 | DAC output | Colin Roskos

\* Minnesota State University, Mankato

\* EE390 Smart Sensor Systems

\* --------------------------------

\*

\* Write a program to configure DAC1 and ADC2 to function properly.

\* For DAC1, feed the data to be converted using 12-bit right-aligned

\* format. The DAC1 output is the channel 17 input of ADC2. Set VREF

\* to 3.3 V (connect to VDD).

\*

\*/

#include "main.h"

static uint16\_t dacValue;

static uint16\_t adcValue;

int main(void)

{

sysClkInit();

initUSART1();

initDAC1();

initADC2();

dacValue = 200;

for(;;)

{

/\*

if (!pollUSART1(pRecBuff)){}

else if (!inputValidation(pRecBuff))

{

putsUSART1(USER\_INPUT\_DENY);

putsUSART1(USER\_INPUT\_REQUEST);

}

else

{

\*/

char tempDACBuff[32] = {'\0'};

char tempADCBuff[32] = {'\0'};

dacOutput(dacValue);

putsUSART1(CLEAR\_LINE);

putsUSART1(UP\_LINE);

putsUSART1(CLEAR\_LINE);

uinttstr(tempDACBuff, dacValue);

insertsUSART1(MESSAGE\_DAC\_CONVERSION, tempDACBuff);

wait\_ms\_80MHz(500); // wait half second

adcValue = getADC2();

uinttstr(tempADCBuff, adcValue);

insertsUSART1(MESSAGE\_ADC\_CONVERSION, tempADCBuff);

wait\_ms\_80MHz(500); // wait half second

dacValue = (dacValue < 4000) ? dacValue + 200 : 0;

//}

}

return 1;

}

uint32\_t inputValidation(char \* pMsgBuff)

{

uint16\_t ichar = 0; //character index

char nChar = pMsgBuff[ichar];

dacValue = 0;

while(nChar == ASCII\_EOL || ichar++ < 5) // cannot be above 4 values long, 0xFFF

{

if (nChar == ASCII\_EOL) return 1; // if all values checked

dacValue = (dacValue \* 10) + uint2char(nChar);

if (dacValue > 0xFFF) return 0;

nChar = pMsgBuff[ichar];

}

return 0; // false

}

**DAC.C**

void initDAC1()

{

RCC->AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN;

NOP; // delay for unlocking GPIOA

RCC->APB1ENR1 |= RCC\_APB1ENR1\_DAC1EN;

NOP; // delay for unlocking DAC1

GPIOA->MODER |= GPIO\_MODER\_MODE4; // set mode to analog

/\*DAC Channel 1 in Normal mode

000: DAC Channel 1 is connected to external pin with Buffer enabled

001: DAC Channel 1 is connected to external pin and to on chip peripherals with Buffer

enabled

010: DAC Channel 1 is connected to external pin with Buffer disabled

011: DAC Channel 1 is connected to on chip peripherals with Buffer disabled

\*/

DAC->MCR = DAC\_MCR\_MODE1\_0; // set both DAC channel to normal mode with buffer enabled

DAC1->CR |= 1; // enable DAC channel 1

}

void dacOutput(uint16\_t outputValue)

{

DAC->DHR12R1 = outputValue;

}

**ADC.C**

void initADC2(void)

{

RCC->AHB2ENR |= RCC\_AHB2ENR\_ADCEN;

NOP; // delay for unlocking ADC

// Channel 17 input ADC2, this is the internal channel connected to DAC1

ADC2->CR &= ~ADC\_CR\_DEEPPWD; // get out of deep power down mode

ADC2->CR |= ADC\_CR\_ADVREGEN; // enable ADC voltage regulator

wait\_us\_80MHz(40); // wait for regulator to be stabilized ; T\_ADCVREG\_STUP = 20us

// Set ADC2\_CCR CH17SEL

ADC123\_COMMON->CCR &= ~(ADC\_CCR\_TSEN\_Msk); // TSEN, Temp Sensor

ADC123\_COMMON->CCR &= ~(ADC\_CCR\_CKMODE\_Msk + ADC\_CCR\_MDMA\_Msk);

ADC123\_COMMON->CCR |= ADC\_CCR\_CKMODE\_0;

ADC2->ISR |= ADC\_ISR\_ADRDY; // write 1 to ISR ADRDY, clears ADC2\_ISR

ADC2->CR |= ADC\_CR\_ADEN; // enable ADC2

while(!(ADC2->ISR & ADC\_ISR\_ADRDY)); // wait for ADC2 to become ready

// channel 17 diffsel is forced single ended mode

ADC2->SQR1 = ADC\_SQR1\_L\_0 +

ADC\_SQR1\_SQ1\_4 + ADC\_SQR1\_SQ1\_0;

/\*

000: 2.5 ADC clock cycles

001: 6.5 ADC clock cycles

010: 12.5 ADC clock cycles

011: 24.5 ADC clock cycles

100: 47.5 ADC clock cycles

101: 92.5 ADC clock cycles

110: 247.5 ADC clock cycles

111: 640.5 ADC clock cycles

\*/

ADC2->SMPR2 |= 0x1 << (7\*3); // set channel 17 sample time to 640.5 ADC clock cycles

ADC2->CFGR = 0; // software trigger, right aligned, 12-bit resolution

return;

}

uint16\_t getADC2(void)

{

ADC2->CR |= ADC\_CR\_ADSTART; // start a new A/D conversion

wait\_us\_80MHz(9); // 1us ~= 80 clock cycles

while(!(ADC2->ISR & ADC\_ISR\_EOC)); // wait for ADC to complete

return (uint16\_t) ADC2->DR;

}

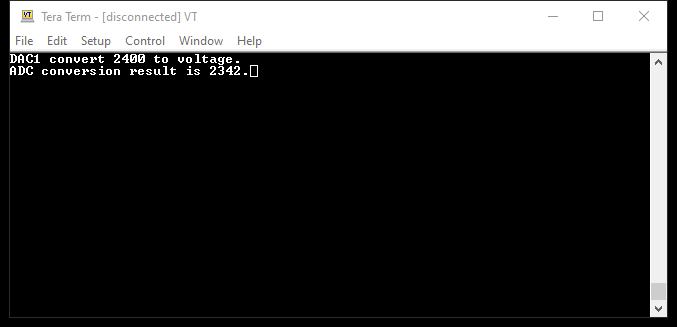


Figure : USART1 output