Lab 2 – DAC with DMA Transfer

EE390 – Smart Sensor Systems

ECET Department

Minnesota State University, Mankato

Colin Roskos

2/10/2023

**Purpose:**

This lab is designed to introduce the use of DMA (Direct Memory Access) and its use in Memory-to-Peripheral direct access with a timer trigger.

**Program Description:**

The program was designed to take a sine wave broken into 60 points, converted to a 12-bit integer with -1 converting to 0x000, and 1 converting to 0xFFF.

The program as written initializes the sine wave in a generic manner, with a length of integer of size 16-bits (uint16\_t).

The system clock is initialized to 80MHz. DAC1 is initialized, enabling GPIOA, enabling DAC, enabling timer input for DMA1 on DAC1, enable DMA1 on DAC1.

The system timer 6 is initialized to use a prescaler of 8x making each clock cycle a time of .1us. the ARR is set to 10 \* uwait (4), the counter is reset to 0, the DIER register UIE interrupt is enabled, the Master slave configuration is set to 0x2, and the counter is enabled.

The DMA is enabled by selecting the clock source and enabling the clock (AHB1) for DMA1 output. Selecting the channel 3 buffer to b’0110, TIM6\_UP/DAC1. Setting the DMA1 memory address to the sine buffer, the destination to the DAC1 Right Justification register (DAC1->DHR12R1). The counter for the DMA loop is set to the sine length (60), with the DMA set to High Priority, 16-bit (memory and destination), memory incrementing, circular, out from memory to peripheral, and enabling dma.

This allows the DMA to output the signal.

**Program List:**

// author defined libraries

main.c

main.h

dma.c

dma.h

sysclk.c

sysclk.h

system.c

system.h

// external libraries

stm32l475xx.h

math.h

stdint.h

stdlib.h

**Discussion:**

The lab was successful, with the largest struggles being the initial confusion of clarity in the board layout (PA4 being labelled D7). Otherwise the signal is cleanly displayed with a signal rise time at sine(0) being approximately 180ns allowing for the signal to resolve easily in the 4us window and the signal change time to be less than 5% of the held output. The signal does ‘stick’ at the peak of the sine wave (see figures 1, 3) which may have solutions which may need investigation. More sticking points were the problems that come with global variables, static globals, volatile and scope of variables. These problems only caused problems with the defining of the sine wave, but also are a source of warnings in the main.c code – this is a future correction that is needed.

Note, with further investigation on the value sticking at DAC high, modifying the sine equation to use 0.98 of the sine height (Sine(theta)\*0.98) solves the hanging values at the max output. (Figure 4) This does lose the lowest and highest 50mv of the range, but allows for a cleaner sine wave. In this experiment, I also increased the steps to 240 (4x) and reduced the step duration to 1us (1/4x).

**NOTE: Figures are at the end of this document**

**LAB2 CODE**

**MAIN.C**

/\*

\* Lab 2 | DAC with DMA Transfer | Colin Roskos

\* Minnesota State University, Mankato

\* EE390 Smart Sensor Systems

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

\*

\* Write a program to generate a sine wave (output from PA4 pin). Divide

\* a cycle of the sine wave into 60 points. Use DMA transfer to send the

\* data to be converted to the DAC1. Use external trigger (TIM6) to

\* trigger the DMA transfer every 4 ?s. The DMA must be configured to

\* circular mode.

\*

\*/

#include "main.h"

void initClk(void);

static uint16\_t sineWave[SINE\_SIZE];

static uint16\_t returnVal = SUCCESS;

**int main()**

{

returnVal = 1;

memSine(sineWave, SINE\_SIZE); // setup sine wave buffer with values

sysClkInit(); // init system clock to 80MHz, HSI

initDAC();

initTimer6TriggerDMA(4); // initialize trigger with a wait of 4 us

initDMAM2P(sineWave, &(DAC1->DHR12R1), SINE\_SIZE); // init DMA from mem SineWave to DAC1 12 bit right justified

while(returnVal == SUCCESS)

{

returnVal = DMAcheck(); // A while loop with a TODO DMA check

}

return returnVal;

}

/\*

\* memSine

\* inputs: data: pointer to 16 bit data array, size: size of array

\*

\* Creates a 12-bit sine wave with /size/ number of datapoints

\*/

**void memSine(uint16\_t\* data, uint16\_t size)**

{

double inter;

for(int i = 0 ; i < size ; i++)

{

inter = (((double) i)/((float)size)) \* 2.0 \* M\_PI;

inter = (1.0 + sin(inter))/2.0;

data[i] = (uint16\_t)(inter \* 0xfff);

}

}

SYSCLK.C

**void system\_clock\_init\_HSI\_80MHz(void)**

{

//system\_clock\_init\_MSI(); // cannot set HSI until it is not used as clock.

// Enable High Speed Internal Clock (HSI = 16 MHz)

RCC->CR |= ((uint32\_t)RCC\_CR\_HSION);

// wait until HSI is ready

while ( (RCC->CR & (uint32\_t) RCC\_CR\_HSIRDY) == 0 );

PWR->CR1 &= ~PWR\_CR1\_VOS;

PWR->CR1 |= PWR\_CR1\_VOS\_0;

while(PWR->SR2 & PWR\_SR2\_VOSF);

FLASH->ACR |= FLASH\_ACR\_LATENCY\_4WS; /\* Frequency is above 64MHz, 4 WS required \*/

FLASH->ACR |= FLASH\_ACR\_PRFTEN; /\* prefetch enable \*/

RCC->CR &= ~(RCC\_CR\_PLLON);

while(RCC->CR & RCC\_CR\_PLLRDY)

;

// choose HSI as PLL Input

RCC->PLLCFGR = (RCC\_PLLCFGR\_PLLSRC\_HSI + RCC\_PLLCFGR\_PLLN\_4 +

RCC\_PLLCFGR\_PLLN\_2 + RCC\_PLLCFGR\_PLLR\_0);

// Wait till PLL is used as system clock source

RCC->CR |= RCC\_CR\_PLLON;

while ((RCC->CR & RCC\_CR\_PLLRDY) == 0)

;

RCC->PLLCFGR = RCC\_PLLCFGR\_PLLREN;

// Select PLL as system clock source

RCC->CFGR &= ~(RCC\_CFGR\_SW + RCC\_CFGR\_HPRE);

RCC->CFGR |= (uint32\_t)RCC\_CFGR\_SW\_PLL + RCC\_CFGR\_HPRE\_1; //11: PLL oscillator used as system clock, divide clock by 1

while ((RCC->CFGR & (RCC\_CFGR\_SW\_PLL + RCC\_CFGR\_HPRE\_1)) == 0 ); //11: PLL oscillator used as system clock, divide clock by 1

SystemCoreClockUpdate();

}

**void initTimer6TriggerDMA(uint32\_t usecWait)**

{

if((RCC->APB1ENR1 & RCC\_APB1ENR1\_TIM6EN) == 0){

RCC->APB1ENR1|= RCC\_APB1ENR1\_TIM6EN; // enable APB1 for TIM6

NOP; // delay one clock cycle to keep RCC change

}

TIM6->PSC = 7; // prescale 8

TIM6->ARR = usecWait \* 10; // wait usecWait us

TIM6->CNT = 0;

TIM6->DIER = TIM\_DIER\_UIE;

TIM6->CR2 &= ~TIM\_CR2\_MMS\_Msk;

TIM6->CR2 |= TIM\_CR2\_MMS\_1;

TIM6->CR1 |= TIM\_CR1\_CEN;

}

DAC.C

**void initDAC()**

{

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

NOP; // delay for unlocking GPIOA

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

NOP; // delay for unlocking DAC1

// Setting DAC Control Register

// TSEL = 0 for Timer 6,

DAC->CR = DAC\_CR\_TEN1 + DAC\_CR\_DMAEN1;// TEN1 Timer enable Channel 1, Enable DMA for channel 1

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

DAC->MCR = 0; // set both DAC channel to normal mode with buffer enabled (also sets DAC2)

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

}

DMA.C

**void initDMAM2P(void \* DMAFromMemBuffer, void \* DMAToMemBuffer, uint16\_t buffSize)**

{

setDMA1ClkSource();

DMA1\_CSELR->CSELR &= ~(DMA\_CSELR\_C3S\_Msk);

DMA1\_CSELR->CSELR |= 0x6 << DMA\_CSELR\_C3S\_Pos; // b`0110 TIM6\_UP/DAC1

DMA1\_Channel3->CMAR = (uint32\_t) DMAFromMemBuffer;

DMA1\_Channel3->CPAR = (uint32\_t) DMAToMemBuffer;

DMA1\_Channel3->CNDTR = buffSize;

DMA1\_Channel3->CCR = DMA\_CCR\_PL\_1 + DMA\_CCR\_PL\_0 + // DMA High Priority

DMA\_CCR\_MSIZE\_0 + // 16-bit local memory size

DMA\_CCR\_PSIZE\_0 + DMA\_CCR\_MINC + // 16-bit destination memory size, increment

DMA\_CCR\_CIRC + DMA\_CCR\_DIR + DMA\_CCR\_EN; // circular, out from device, enable

}

**void setDMA1ClkSource(void)**

{

RCC->AHB1ENR |= RCC\_AHB1ENR\_DMA1EN;

NOP; // delay one clock cycle to keep RCC change

}

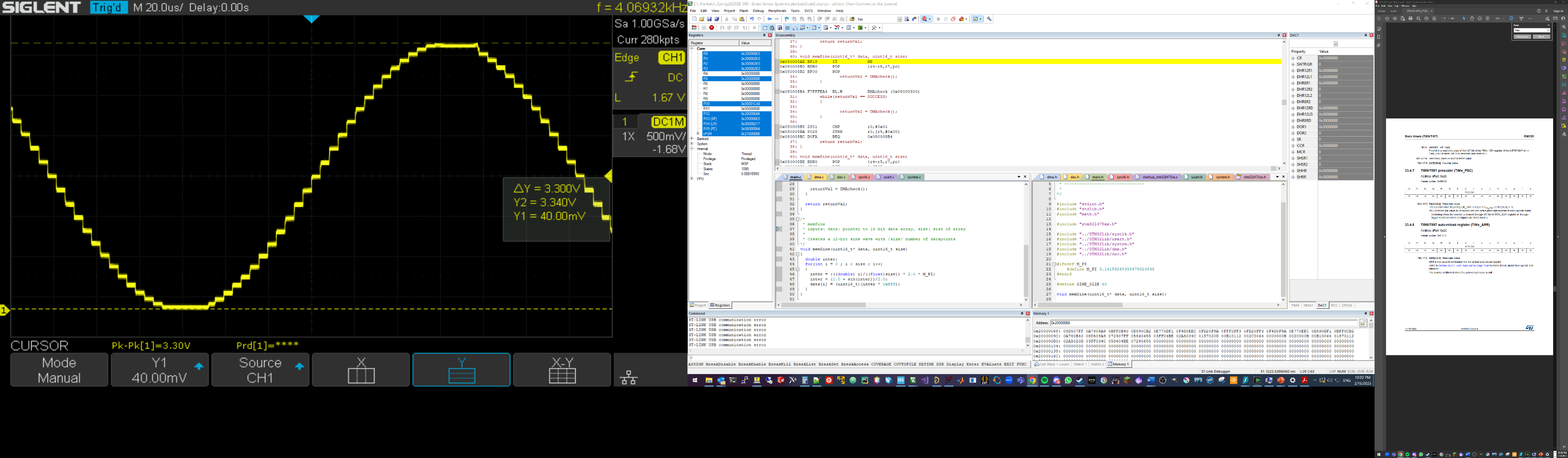


Figure : DAC1 output (PA4) with a sine amplitude of 3.3V

A screenshot of a computer

Description automatically generated with medium confidence

Figure : DAC1 output (PA4) with a step duration of 4us

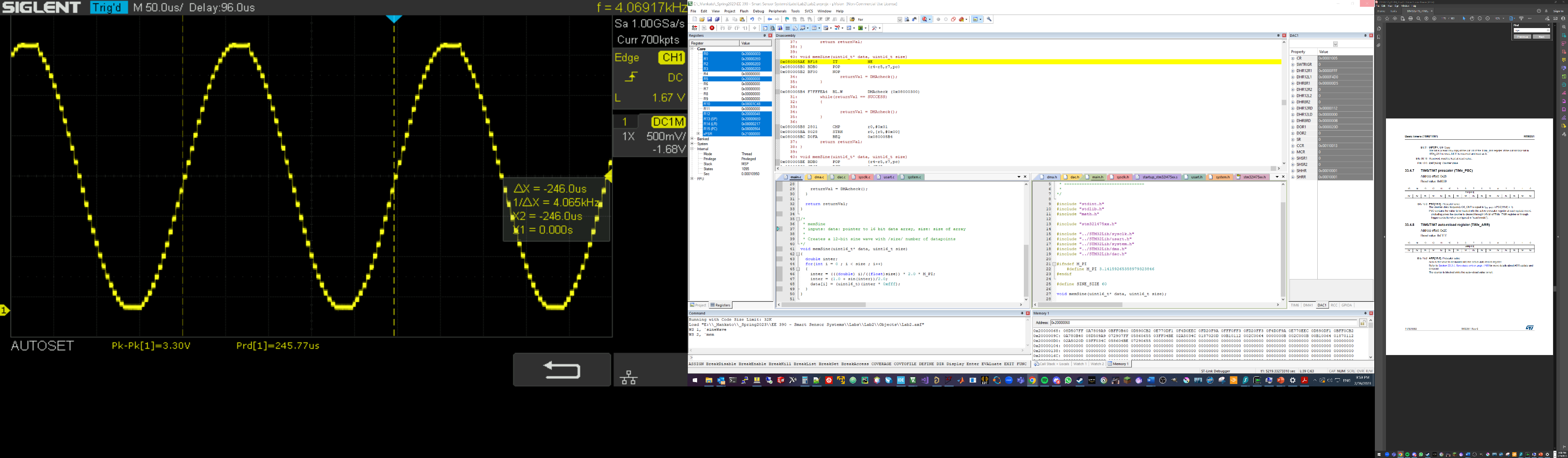


Figure : DAC output (PA4) with sine period of 246us

A screenshot of a computer

Description automatically generated with medium confidence

Figure : DAC lab continued