Cris Droguett

Professor Summerville

EECE 558

December 17, 2021

Final Project Report

Executive Summary

Problem Description:

For my Final Project I wanted to incorporate using a DMA driver in some degree. It was one of the last concepts we learned in the semester and since we had not had any assignments working with DMA, I figured it would be good to learn it for my Final Project. My idea for my project was to generate a DMA request every time I pressed the reset button on the Freedom board. I would have the DMA grab 32-bit data from an array I created and then use an FSM to cycle through that data and display it on the LEDs.

Solution and Driver Design:

To solve this problem, I first had to read the KL25 Sub-Family Reference Manual to learn how to configure the DMA driver I was going to create. I first looked at Chapter 23 where the manual explains some of the important registers in the DMA memory map. The Source and Destination registers are used by the DMA to store the data that is being transferred. In my project the value being stored in the SAR were the values in the colors array. Then the DMA would take those values and store them in the rg_colors variable I had, which was set at the DAR. In the status register I had to set the BCR field which holds the number of bytes that needed to be transferred. In the DMA control register I needed to enable the ERQ field which allows the DMA to be triggered by a peripheral request. Since I wanted to send requests from the button, I needed this enabled. I enabled cycle stealing and source incrementing. Source incrementing allows for DMA to grab the next values in the array after each successful transfer. One of the most important fields was the Source and Destination size fields. I needed to set both fields to 0 so that they would have 32-bit values. I also enabled the EINT field which generates an interrupt after each successful trigger. I had the DMA configured but now I needed to connect it to my button. For that I needed to go to Chapter 22 to read about the DMAMUX. There is a field in the DMA MUX Channel Configuration register where you can set what source is routed to DMA channel. I knew I needed to get the button's source number and set it here but did not know how to get its source number. To get this I went to chapter 3 in the manual, specifically section 3.4.8.1. Here it lists every DMA MUX request source number. I found that Port A has a source number of 49 and since the button is configured using Port A I knew this was the number I needed. The last thing I needed to do was to go to the Port A pin control register and enable DMA requests. I set the IRQC to 1 which would generate the requests on the rising edge of any button press. Now every button press would generate a DMA request.

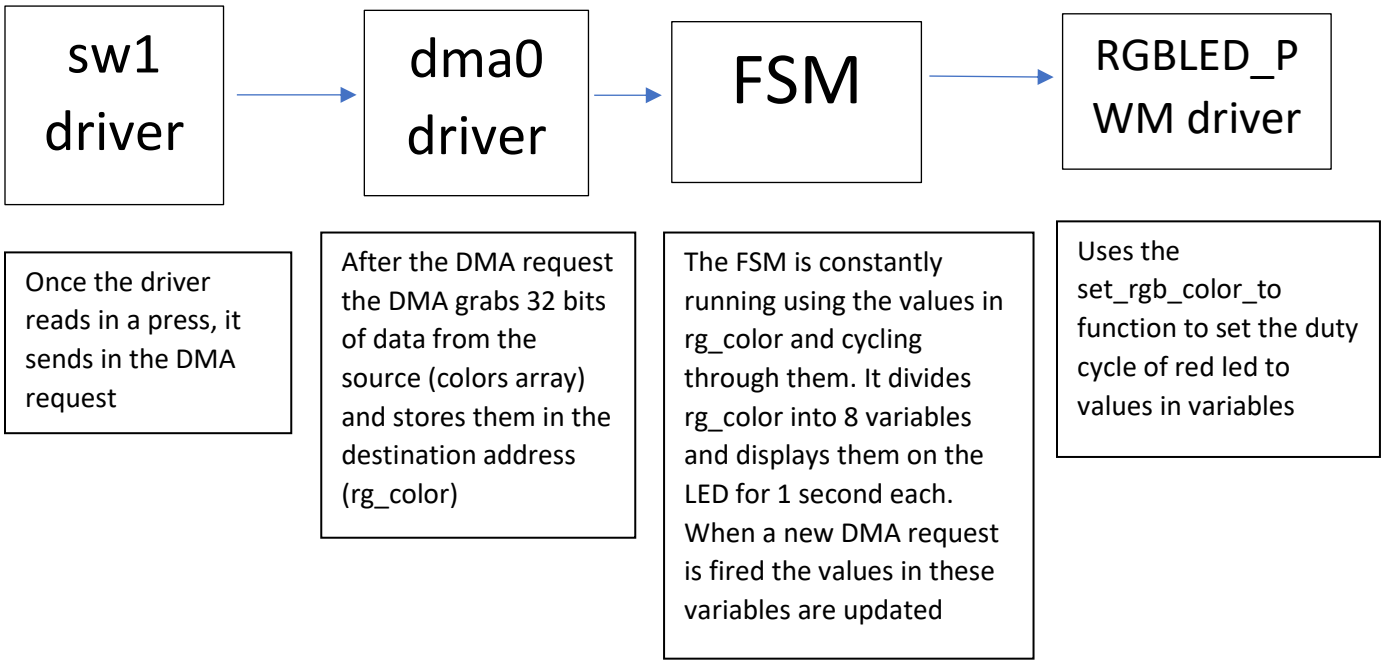
Limitations:

While trying to test values on my board I ran into an early problem with my design. I wanted the DMA to grab 32-bits worth of colors from my array and then display each of those colors of a duration. However, I forgot that a duty cycle for an LED must be 16 bits so realistically I could only get 2 colors per DMA

request. Instead, I figured out a way to get 8 color values per DMA request. My array of colors has 16-bit values in hex. For example, the first two values in my array that would be transferred would be 0x0357 and 0x8ACF. My idea was to store each hex value in one of eight variables and then display those. To accomplish this, I took the 32-bit transferred value that was in rg_color and right shifted accordingly for each variable until the values I wanted were the least four significant bits. For example, I wanted to store the hex value of 3 into color2 so I took rg_color and shifted it to the right by 24 bits. After this shift right the least four significant bits had the value of 0x3. I then masked off the rest of the variable so that the only values that weren't 0 were the least significant four bits. After doing this for each variable I had 0x0 in color1, 0x3 in color2, 0x5 in color3 and so on. However, these values are so small they would not even display on the LED. So, the final step was to bit shift the colors to the left by 12 so that they became 0x000, 0x3000, 0x5000 and so on. In my FSM I have a running counter and it cycles through displaying each of the 8 colors on the red LED. It displays each color for 1 second before switching to the next state and after displaying the last color it loops back to the first. I have filled my colors array with values that verify that my DMA was transferring my data correctly. The first 8 values in the array have the LED getting brighter for the 8 seconds before looping. The next values alternate the LED being off to very bright every other second and so on.

Detailed Design

The main drivers my project used were the sw1, dma0, and RGBLED_PWM drivers.



Appendix:

test_dma0.c

```
// Cris Droguett
// Embedded Systems Final Project

#include <stdint.h>
#include "systick.h"
#include "copwdt.h"
#include "dma0.h"
#include "sw1.h"
#include "RGBLED_PWM.h"

#define NUM_COLORS 32
int16_t colors[NUM_COLORS]={
        0x0357, 0x8ACF, 0x0F0F, 0x0F0F,
        0x3FFF, 0x7FFF, 0x0000, 0xFFFF,
        0xAABB, 0x5566, 0xFB85, 0x4321,
        0x0307, 0x0A0F, 0x2222, 0x7777,
        0xF000, 0x000F, 0xFF00, 0xFF00,
        0x1234, 0x5678, 0x9ABC, 0xDEF0,
        0x9382, 0xFAEB, 0x620F, 0x0690,
        0x0102, 0x0304, 0x0506, 0x0708
};
volatile uint32_t rg_color;

void dma0_fsm(){
        static uint16_t cntr = 0;

        uint16_t color1 = rg_color>>28;
```

```
uint16_t color2 = rg_color>>24;
uint16_t color3 = rg_color>>20;
uint16_t color4 = rg_color>>16;
uint16_t color5 = rg_color>>12;
uint16_t color6 = rg_color>>8;
uint16_t color7 = rg_color>>4;
uint16_t color8 = rg_color;

color1 &= 0x000F;
color2 &= 0x000F;
color3 &= 0x000F;
color4 &= 0x000F;
color5 &= 0x000F;
color6 &= 0x000F;
color7 &= 0x000F;
color8 &= 0x000F;

color1 = (color1 << 12);
color2 = (color2 << 12);
color3 = (color3 << 12);
color4 = (color4 << 12);
color5 = (color5 << 12);
color6 = (color6 << 12);
color7 = (color7 << 12);
color8 = (color8 << 12);

static enum {ST_COLOR1, ST_COLOR2, ST_COLOR3,ST_COLOR4, ST_COLOR5,ST_COLOR6,
ST_COLOR7, ST_COLOR8} state =ST_COLOR1;
//counter FSM
```

```
cntr ++;
if(cntr > 8000)
        cntr = 1;
//LED FSM
switch(state)
{
        default:
        case ST_COLOR1:
                if(cntr >= 1000)
                        state = ST_COLOR2;
                set_rgb_color_to(color1,0,0);
                break;
        case ST_COLOR2 :
                if(cntr >= 2000)
                        state = ST_COLOR3;
                set_rgb_color_to(color2,0,0);
                break;
        case ST_COLOR3:
                if (cntr >= 3000)
                        state = ST_COLOR4;
                set_rgb_color_to(color3,0, 0);
                break;
        case ST_COLOR4:
                if (cntr >= 4000)
                        state = ST_COLOR5;
                set_rgb_color_to(color4,0, 0);
                break;
        case ST_COLOR5:
                if(cntr >= 5000)
```

```
                    state = ST_COLOR6;
                set_rgb_color_to(color5,0,0);
                break;
            case ST_COLOR6 :
                if(cntr >= 6000)
                    state = ST_COLOR7;
                set_rgb_color_to(color6,0,0);
                break;
            case ST_COLOR7:
                if (cntr >= 7000)
                    state = ST_COLOR8;
                set_rgb_color_to(color7,0, 0);
                break;
            case ST_COLOR8:
                if (cntr >= 8000)
                    state = ST_COLOR1;
                set_rgb_color_to(color8,0, 0);
                break;
        }
}

void main()
{
        asm("CPSID i");
        configure_systick();
        configure_copwdt();
        configure_rgbled();
        configure_sw1();
        configure_dma0(colors, &rg_color, NUM_COLORS);
```

```
	asm("CPSIE i");


	while(1)
	{
		asm("WFI");
		if(!systick_has_fired())
			continue;
		dma0_fsm();
		feed_the_watchdog();
	}
}
```

dma0.c

```
// Cris Droguett
// Embedded Systems Final Project
#include "dma0.h"
#include <stdint.h>
#include <MKL25Z4.h>
volatile static void *gfrom;
volatile static uint16_t glength;
void configure_dma0(void *from, void *to, uint8_t length)
{
	SIM->SCGC6 |= SIM_SCGC6_DMAMUX_MASK;
	SIM->SCGC7 |= SIM_SCGC7_DMA_MASK;
	DMA0 -> DMA[0].DCR =
		DMA_DCR_ERQ_MASK |
		DMA_DCR_CS_MASK  |
		DMA_DCR_SINC_MASK|
		DMA_DCR_SSIZE(0) |
		DMA_DCR_DSIZE(0) |
```

```
                DMA_DCR_EINT_MASK;
        DMA0->DMA[0].SAR = (uint32_t) from;
        DMA0->DMA[0].DAR = (uint32_t) to;
        DMA0->DMA[0].DSR_BCR = DMA_DSR_BCR_BCR(2*length);
        DMAMUX0->CHCFG[0] = DMAMUX_CHCFG_SOURCE(49) | DMAMUX_CHCFG_ENBL_MASK;

        glength = 1;
        gfrom = from;
        NVIC_SetPriority(DMA0_IRQn,2);
        NVIC_ClearPendingIRQ(DMA0_IRQn);
        NVIC_EnableIRQ(DMA0_IRQn);
}

void DMA0_IRQHandler()
{
        DMA0->DMA[0].SAR = (uint32_t) gfrom;
        DMA0->DMA[0].DSR_BCR = DMA_DSR_BCR_BCR(2*glength) | DMA_DSR_BCR_DONE_MASK;
}
```

dma0.h

```
// Cris Droguett
// Embedded Systems Final Project
#ifndef DMA0_H
#define DMA0_H
#include <stdint.h>

volatile static void *gfrom;
volatile static uint16_t glength;
void configure_dma0(void *from, void *to, uint8_t length);
void DMA0_IRQHandler();
```

```
#endif
```

sw1.c

```
// Cris Droguett
// Embedded Systems Final Project
#include <sw1.h>
#include <MKL25Z4.h>
#include <stdbool.h>
#define SW1_LOCATION 20

void configure_sw1()
{
	SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
	PORTA->PCR[SW1_LOCATION] =
		PORT_PCR_PE(1)|
		PORT_PCR_PS(1)|
		PORT_PCR_PFE(0)|
		PORT_PCR_DSE(0)|
		PORT_PCR_SRE(0)|
		PORT_PCR_MUX(1)|
		PORT_PCR_IRQC(1)|
		PORT_PCR_ISF(1);
	PTA->PDDR &= ~(1<<SW1_LOCATION);
}

_Bool sw1_is_pressed(){
	return !(PTA->PDIR & (1<<SW1_LOCATION));
}
```

```
_Bool sw1_is_not_pressed(){
	return !sw1_is_pressed();
}
//enum press_type {NO_PRESS,SHORT_PRESS,LONG_PRESS};
enum press_type switch_press_duration(){
	static uint16_t cntr = 0;
	static enum
{ST_NO_PRESS,ST_DEBOUNCE_PRESS,ST_SHORT_PRESS,ST_DEBOUNCE_RELEASE1,ST_LONG_PRESS,ST_
DEBOUNCE_RELEASE2} state = ST_NO_PRESS;
	switch(state) {
		default:
		case ST_NO_PRESS:
			cntr = 0;
			if(sw1_is_pressed())
			{
				state = ST_SHORT_PRESS;
				cntr = 1;
			}
			else
				state = ST_NO_PRESS;
			break;
		case ST_DEBOUNCE_PRESS:
			if(sw1_is_pressed() && (cntr>=5))
			    state = ST_SHORT_PRESS;
			else if(sw1_is_pressed() && (cntr<5))
				state = ST_DEBOUNCE_PRESS;
			else if(sw1_is_not_pressed())
				state = ST_NO_PRESS;
			cntr++;
```

```
            break;
    case ST_SHORT_PRESS:
            cntr ++;
            if(sw1_is_pressed() && (cntr >= 1499))
                    state = ST_LONG_PRESS;
            else if(sw1_is_pressed() && (cntr < 1499))
                    state = ST_SHORT_PRESS;
            else if(sw1_is_not_pressed())
            {
                    cntr = 0;
                    state = ST_DEBOUNCE_RELEASE1;
                    return SHORT_PRESS;
            }
            break;
    case ST_LONG_PRESS:
            cntr = 0;
            if(sw1_is_not_pressed())
            {
                    state = ST_DEBOUNCE_RELEASE2;
                    return LONG_PRESS;
            }
            break;
    case ST_DEBOUNCE_RELEASE1:
            cntr++;
            if(sw1_is_not_pressed() && cntr >= 5)
                    state = ST_NO_PRESS;
            else
                    state = ST_DEBOUNCE_RELEASE1;
            break;
```

```
                case ST_DEBOUNCE_RELEASE2:
                        cntr++;
                        if(sw1_is_not_pressed() && cntr >= 5)
                                state = ST_NO_PRESS;
                        else
                                state = ST_DEBOUNCE_RELEASE2;
                        break;
        }
                return NO_PRESS;
}
```

sw1.h

```
// Cris Droguett
// Embedded Systems Final Project
#ifndef SW1_H
#define SW1_H

#include <stdbool.h>

void configure_sw1();
_Bool sw1_is_pressed();
_Bool sw1_is_not_pressed();
enum press_type {NO_PRESS,SHORT_PRESS,LONG_PRESS};
enum press_type switch_press_duration();
#endif
```

RGBLED_PWM.c

```
// Cris Droguett
// Embedded Systems Final Project
#include <stdint.h>
#include <RGBLED_PWM.h>
```

```
#include <MKL25Z4.h>

#define RED_LED_LOC 18
#define GREEN_LED_LOC 19
#define BLUE_LED_LOC 1

void set_red_led_duty_cycle(uint16_t duty);
void set_green_led_duty_cycle(uint16_t duty);
void set_blue_led_duty_cycle(uint16_t duty);

void configure_rgbled()
{
	//configure red led port
	SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;
	PORTB->PCR[RED_LED_LOC] = PORT_PCR_MUX(3);
	//configure green led port
	SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;
	PORTB->PCR[GREEN_LED_LOC] = PORT_PCR_MUX(3);
	//configure blue led port
	SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK;
	PORTD->PCR[1] = PORT_PCR_MUX(4);
	PTD->PDDR |= (1<<1);
	//configure red and green led timer
	SIM->SCGC6 |= SIM_SCGC6_TPM2_MASK;
	TPM2->SC = TPM_SC_PS(0) |
		TPM_SC_CMOD(0) |
		TPM_SC_CPWMS(0);
	TPM2->MOD=0xFFFF;
	//configure blue led timer
```

```
SIM->SCGC6 |= SIM_SCGC6_TPM0_MASK;
TPM0->SC = TPM_SC_PS(0) |
        TPM_SC_CMOD(0) |
        TPM_SC_CPWMS(0);
TPM0->MOD=0xFFFF;

//configure red led channel
TPM2->CONTROLS[0].CnSC =  TPM_CnSC_MSB(1)
        | TPM_CnSC_MSA(0)
        | TPM_CnSC_ELSB(0)
        // TPM_CnSC_CHF(1);
        // TPM_CnSC_CHIE(0);
        // TPM_CnSC_DMA(0);
        | TPM_CnSC_ELSA(1);
//configure green led channel
TPM2->CONTROLS[1].CnSC =  TPM_CnSC_MSB(1)
        | TPM_CnSC_MSA(0)
        | TPM_CnSC_ELSB(0)
        // TPM_CnSC_CHF(1);
        // TPM_CnSC_CHIE(0);
        // TPM_CnSC_DMA(0);
        | TPM_CnSC_ELSA(1);
//configure blue led channel
TPM0->CONTROLS[1].CnSC =  TPM_CnSC_MSB(1)
        | TPM_CnSC_MSA(0)
        | TPM_CnSC_ELSB(0)
        // TPM_CnSC_CHF(1);
        // TPM_CnSC_CHIE(0);
        // TPM_CnSC_DMA(0);
```

```
                | TPM_CnSC_ELSA(1);
        //turn off leds at config
        turn_off_rgbled();
        //turn on timer clocks
        TPM0->SC |= TPM_SC_CMOD(1);
        TPM2->SC |= TPM_SC_CMOD(1);
}

void set_rgb_color_to( uint16_t red, uint16_t green, uint16_t blue)
{
        set_red_led_duty_cycle(red);
        set_green_led_duty_cycle(green);
        set_blue_led_duty_cycle(blue);
}

void turn_off_rgbled()
{
        set_red_led_duty_cycle(0);
        set_green_led_duty_cycle(0);
        set_blue_led_duty_cycle(0);
}
void set_red_led_duty_cycle(uint16_t duty)
{
        TPM2->CONTROLS[0].CnV = duty;
}

void set_green_led_duty_cycle(uint16_t duty)
{
        TPM2->CONTROLS[1].CnV = duty;
```

```
}

void set_blue_led_duty_cycle(uint16_t duty)
{
	TPM0->CONTROLS[1].CnV = duty;
}
```

RGBLED_PWM.h

```
// Cris Droguett
// Embedded Systems Final Project
#ifndef REDLED_PWM_H
#define REDLED_PWM_H
#include <stdint.h>

void set_red_led_duty_cycle(uint16_t duty);
void set_green_led_duty_cycle(uint16_t duty);
void set_blue_led_duty_cycle(uint16_t duty);
void configure_rgbled();
void set_rgb_color_to( uint16_t red, uint16_t green, uint16_t blue);
void turn_off_rgbled();

#endif
```

Copwdt.c

```
// Cris Droguett
// Embedded Systems Final Project
#include <copwdt.h>
#include <MKL25Z4.h>
void configure_copwdt(){
	SIM->COPC = SIM_COPC_COPT(3)|
		SIM_COPC_COPCLKS(1)|
```

```
                    SIM_COPC_COPW(0);
}

void feed_the_watchdog(){
        SIM->SRVCOP = 0x55;
        SIM->SRVCOP = 0xAA;
}
```

Copwdt.h

```
// Cris Droguett
// Embedded Systems Final Project
#ifndef COPWDT_H
#define COPWDT_H

void configure_copwdt();
void feed_the_watchdog();

#endif
```

Systick.c

```
// Cris Droguett
// Embedded Systems Final Project
#include <MKL25Z4.h>
#include <systick.h>
#include <stdbool.h>

static volatile  _Bool systick_interrupt_has_occured =0;
#define SYSTICK_PERIOD ( 1000 )
#define SYSTICK_TOP ( SYS_CLOCK / SYSTICK_PERIOD )

void configure_systick()
```

```
{
	SysTick->LOAD = SYSTICK_TOP;
	SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk |
			SysTick_CTRL_TICKINT_Msk   |
			SysTick_CTRL_ENABLE_Msk;
}
_Bool systick_has_fired()
{
	_Bool retval = systick_interrupt_has_occured;
	systick_interrupt_has_occured = 0;
	return retval;
}

void SysTick_Handler()
{
	systick_interrupt_has_occured = 1;
}
```

Systick.h

```
// Cris Droguett
// Embedded Systems Final Project
#ifndef SYSTICK_H
#define SYSTICK_H

#include <stdbool.h>
void configure_systick();
_Bool systick_has_fired();

#endif
```