ECE 3849 D2022
Real-Time Embedded Systems
Lab 3: Advanced I/O

Adam Grabowski, Michael Rideout

April 26, 2022

Introduction

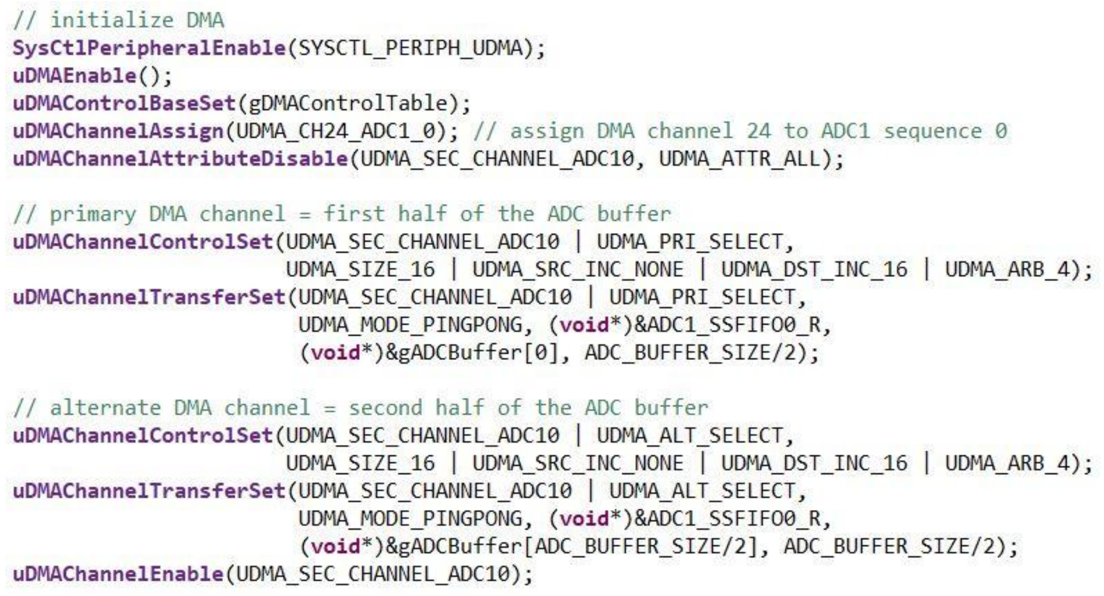
The purpose of this lab was to use a PWM signal to play an audio file and to optimize the ADC functionality from the previous labs by using DMA. To do this, we used DMA in place of the previous ISRs and measured the resulting CPU load. The second thing we added was the ability to display the measured frequency of the received signal.

Discussion and Results

This section describes our implementation of each major lab component in the signoff.

Challenge #1

To implement the first section of the lab, we replaced ISR functionality from the previous lab with DMA. Initialization code was added to ADC_Init with two halves of the ADC buffer corresponding to two different DMA channels. This way the CPU can use one buffer while DMA uses the other, with the two alternating repeatedly. With DMA functionality replacing what was previously handled by ISRs, we then measured the CPU load on the board to prove that the new method was more efficient. The buffering functionality also had to be adjusted to match the DMA usage.

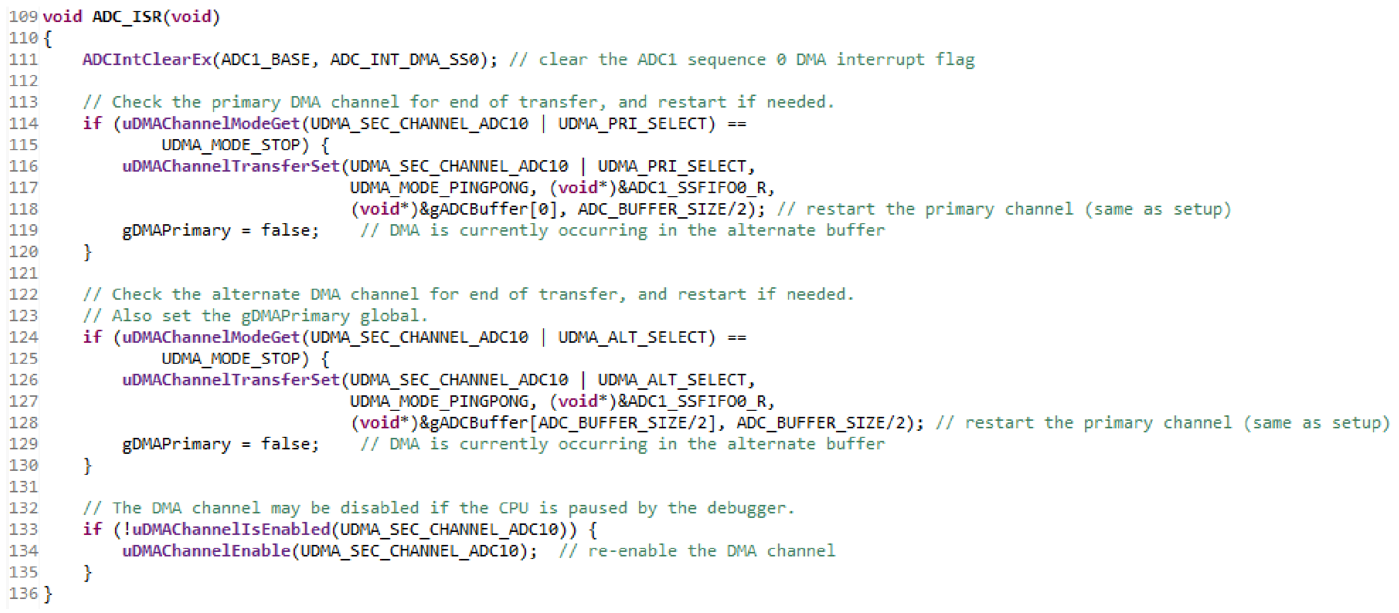


```
// initialize DMA
SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
uDMAEnable();
uDMAControlBaseSet(gDMAControlTable);
uDMAChannelAssign(UDMA_CH24_ADC1_0); // assign DMA channel 24 to ADC1 sequence 0
uDMAChannelAttributeDisable(UDMA_SEC_CHANNEL_ADC10, UDMA_ATTR_ALL);

// primary DMA channel = first half of the ADC buffer
uDMAChannelControlSet(UDMA_SEC_CHANNEL_ADC10 | UDMA_PRI_SELECT,
                      UDMA_SIZE_16 | UDMA_SRC_INC_NONE | UDMA_DST_INC_16 | UDMA_ARB_4);
uDMAChannelTransferSet(UDMA_SEC_CHANNEL_ADC10 | UDMA_PRI_SELECT,
                       UDMA_MODE_PINGPONG, (void*)&ADC1_SSFIFO0_R,
                       (void*)&gADCBuffer[0], ADC_BUFFER_SIZE/2);

// alternate DMA channel = second half of the ADC buffer
uDMAChannelControlSet(UDMA_SEC_CHANNEL_ADC10 | UDMA_ALT_SELECT,
                      UDMA_SIZE_16 | UDMA_SRC_INC_NONE | UDMA_DST_INC_16 | UDMA_ARB_4);
uDMAChannelTransferSet(UDMA_SEC_CHANNEL_ADC10 | UDMA_ALT_SELECT,
                       UDMA_MODE_PINGPONG, (void*)&ADC1_SSFIFO0_R,
                       (void*)&gADCBuffer[ADC_BUFFER_SIZE/2], ADC_BUFFER_SIZE/2);
uDMAChannelEnable(UDMA_SEC_CHANNEL_ADC10);
```

Figure 1: DMA Initialization



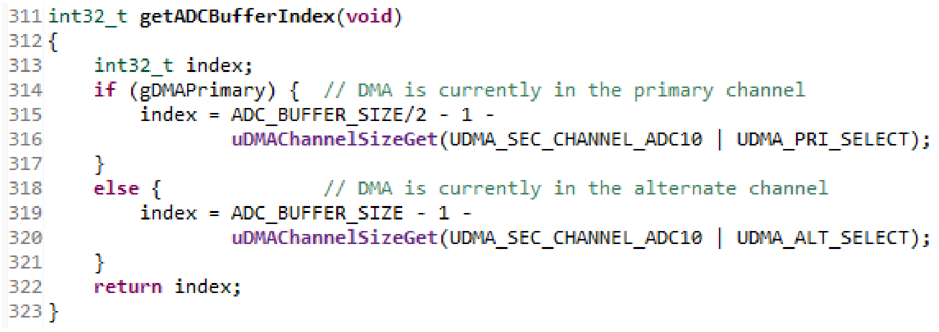
```c
void ADC_ISR(void)
{
    ADCIntClearEx(ADC1_BASE, ADC_INT_DMA_SS0); // clear the ADC1 sequence 0 DMA interrupt flag

    // Check the primary DMA channel for end of transfer, and restart if needed.
    if (uDMAChannelModeGet(UDMA_SEC_CHANNEL_ADC10 | UDMA_PRI_SELECT) ==
            UDMA_MODE_STOP) {
        uDMAChannelTransferSet(UDMA_SEC_CHANNEL_ADC10 | UDMA_PRI_SELECT,
                               UDMA_MODE_PINGPONG, (void*)&ADC1_SSFIFO0_R,
                               (void*)&gADCBuffer[0], ADC_BUFFER_SIZE/2); // restart the primary channel (same as setup)
        gDMAPrimary = false;    // DMA is currently occurring in the alternate buffer
    }

    // Check the alternate DMA channel for end of transfer, and restart if needed.
    // Also set the gDMAPrimary global.
    if (uDMAChannelModeGet(UDMA_SEC_CHANNEL_ADC10 | UDMA_ALT_SELECT) ==
            UDMA_MODE_STOP) {
        uDMAChannelTransferSet(UDMA_SEC_CHANNEL_ADC10 | UDMA_ALT_SELECT,
                               UDMA_MODE_PINGPONG, (void*)&ADC1_SSFIFO0_R,
                               (void*)&gADCBuffer[ADC_BUFFER_SIZE/2], ADC_BUFFER_SIZE/2); // restart the primary channel (same as setup)
        gDMAPrimary = false;    // DMA is currently occurring in the alternate buffer
    }

    // The DMA channel may be disabled if the CPU is paused by the debugger.
    if (!uDMAChannelIsEnabled(UDMA_SEC_CHANNEL_ADC10)) {
        uDMAChannelEnable(UDMA_SEC_CHANNEL_ADC10);  // re-enable the DMA channel
    }
}
```

Figure 2: ADC Interrupt Service Routine



```c
int32_t getADCBufferIndex(void)
{
    int32_t index;
    if (gDMAPrimary) {  // DMA is currently in the primary channel
        index = ADC_BUFFER_SIZE/2 - 1 -
                uDMAChannelSizeGet(UDMA_SEC_CHANNEL_ADC10 | UDMA_PRI_SELECT);
    }
    else {              // DMA is currently in the alternate channel
        index = ADC_BUFFER_SIZE - 1 -
                uDMAChannelSizeGet(UDMA_SEC_CHANNEL_ADC10 | UDMA_ALT_SELECT);
    }
    return index;
}
```

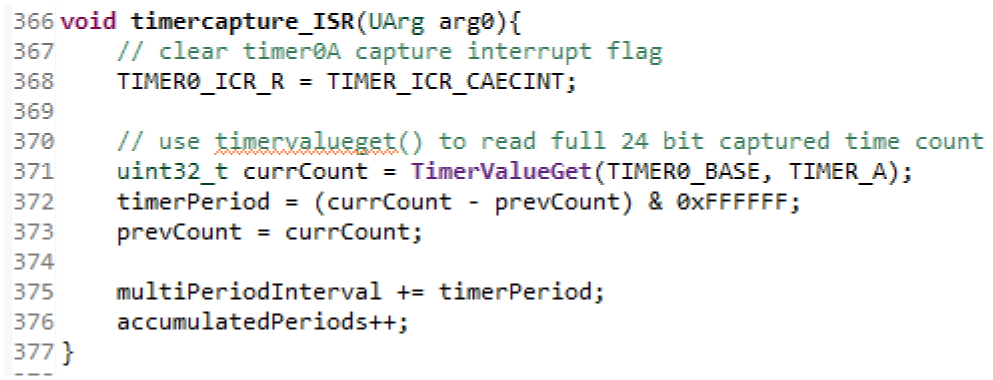
Figure 3: ADC Buffer Index Computation

ADC Configuration	Sampling Rate	CPU Load	ISR Relative Deadline
Single- sample ISR	1	0.77	1 us
DMA	1	0.018	1024 us
DMA	2	0.025	2048 us

As shown in the table above, using the DMA reduced the CPU load greatly. Now, there is plenty of CPU load available for other real-time tasks. The ISR relative deadline is equal to the period of the ISR which was determined by inverting the sampling rate and dividing by the sample size: 1 / (sampling rate / buffer size).

Challenge #2

The second part of the lab saw the creation of a capture mode timer that measured the period of the inputted square wave. A hardware interrupt was used for each capture, and the period measurements were converted to frequencies and displayed on the board. The interrupt, shown below, reads the timer count and compares it to the timer value from the last time the interrupt was run, giving it the current period of the signal. It also keeps track of how many periods there have been and sums them to keep track of the interval of multiple periods. With this information, we can do some simple calculations to change the periods to frequencies and display them on the board.

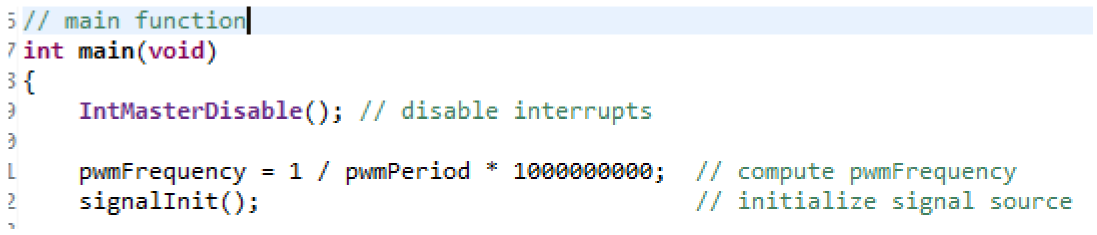


```
void timercapture_ISR(UArg arg0){
    // clear timer0A capture interrupt flag
    TIMER0_ICR_R = TIMER_ICR_CAECINT;

    // use timervalueget() to read full 24 bit captured time count
    uint32_t currCount = TimerValueGet(TIMER0_BASE, TIMER_A);
    timerPeriod = (currCount - prevCount) & 0xFFFFFF;
    prevCount = currCount;

    multiPeriodInterval += timerPeriod;
    accumulatedPeriods++;
}
```

Figure 4: Capture ISR



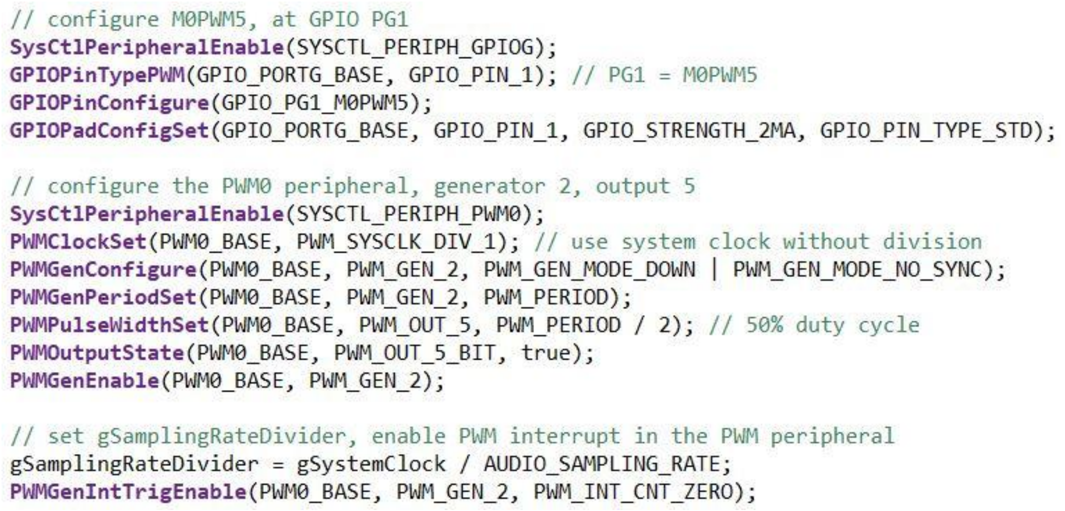
```
// main function
int main(void)
{
    IntMasterDisable(); // disable interrupts

    pwmFrequency = 1 / pwmPeriod * 1000000000;  // compute pwmFrequency
    signalInit();                               // initialize signal source
```

Figure 5: Frequency Calculation

Challenge #3

To play an audio file through the board, we used PWM. PWM signals can function as digital to analog converters because adjustments to a high frequency PWM signal can simulate rising and falling voltages, creating smooth analog waves. As shown below, the PWM initialization code was adjusted to create a PWM signal for this purpose. Also, a simple PWM ISR was made to periodically update the frequency of the signal to control the sound. An audio file was provided to be read, and by incrementing over it in real time and playing the converted frequency as an analog signal it was played.

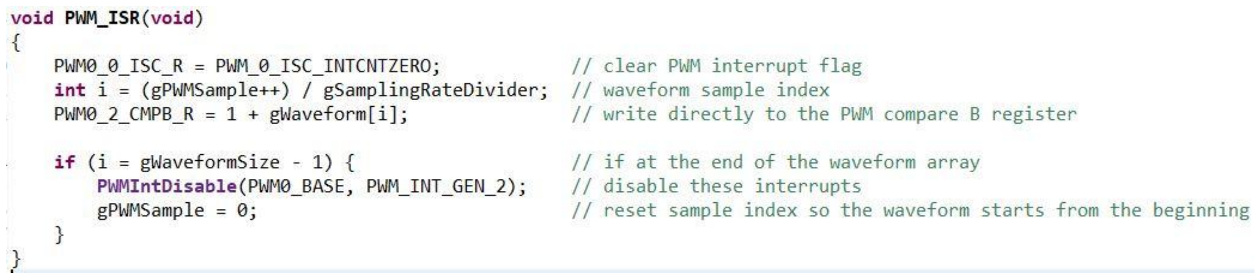


```
// configure M0PWM5, at GPIO PG1
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_1); // PG1 = M0PWM5
GPIOPinConfigure(GPIO_PG1_M0PWM5);
GPIOPadConfigSet(GPIO_PORTG_BASE, GPIO_PIN_1, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD);

// configure the PWM0 peripheral, generator 2, output 5
SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
PWMClockSet(PWM0_BASE, PWM_SYSCLK_DIV_1); // use system clock without division
PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
PWMGenPeriodSet(PWM0_BASE, PWM_GEN_2, PWM_PERIOD);
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, PWM_PERIOD / 2); // 50% duty cycle
PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, true);
PWMGenEnable(PWM0_BASE, PWM_GEN_2);

// set gSamplingRateDivider, enable PWM interrupt in the PWM peripheral
gSamplingRateDivider = gSystemClock / AUDIO_SAMPLING_RATE;
PWMGenIntTrigEnable(PWM0_BASE, PWM_GEN_2, PWM_INT_CNT_ZERO);
```

Figure 6: PWM Initialization



```
void PWM_ISR(void)
{
    PWM0_0_ISC_R = PWM_0_ISC_INTCNTZERO;             // clear PWM interrupt flag
    int i = (gPWMSample++) / gSamplingRateDivider;  // waveform sample index
    PWM0_2_CMPB_R = 1 + gWaveform[i];                // write directly to the PWM compare B register

    if (i = gWaveformSize - 1) {                     // if at the end of the waveform array
        PWMIntDisable(PWM0_BASE, PWM_INT_GEN_2);     // disable these interrupts
        gPWMSample = 0;                              // reset sample index so the waveform starts from the beginning
    }
}
```

Figure 7: PWM Interrupt Service Routine

Difficulties

Our team was not able to complete challenge 3 completely, as the audio does not play when the button is pressed. However, audio appears to play when the code is loaded onto the board as shown in the video.

Conclusions

While giving us a deeper understanding of how to use DMA, this lab also gave a clear example of how PWM can be used to simulate an analog signal. By varying the period of a digital signal, a voltage can be effectively raised and lowered, with 100 percent duty cycle being the actual voltage of the source and 0 percent obviously being 0 volts. Although it isn't perfect, this is a viable method of digital to analog conversion. Challenge number 1 also provided a clear example of how using DMA instead of interrupts can improve the efficiency of a system and free up CPU usage for other things.