Overview

This lab contained two sections. First we are going to use the output hardware of Timer A to implement a useful application: Pulse Width Modulation (PWM). On some microcontrollers and DSPs (Digital Signal Processors), Pulse Width Modulation is already built-in as a peripheral in its own, but that is rare. Therefore it is important to know how PWM is produced on a typical MCU using a timer. In this document, we are going to start with an introduction to Pulse Width Modulation, PWM, and Timer A’s output mode before we start the experiments.

Second, we are going to study the capture mode of the timer and use it for frequency measurements.

Section 1 – Pulse Width Modulation

Pulse Width Modulation modulates a signal by producing constant amplitude digital pulses whose pulse width (duty-cycle, duration) is proportional to the signal’s amplitude at a given sample time. PWM is used in communications and generating control signals among many other applications.
Fig. 1

Two main parameters are associated with PWM: resolution and sampling period. Resolution is the number of distinct output pulse widths you can have. Each output pulse width corresponds to a signal value/amplitude. Therefore resolution is the number of distinct signal values you can modulate.

Sampling time Ts gives the time interval between sampling points. In Fig. 1 sampling time is denoted as Ts. At every Ts, PWM produces a pulse width proportional to the signal (sine wave in the figure) value.

Later we are going to inspect how the two parameters (Ts and resolution) are set up using timer A.

**Output Modes of Timer A**

In the previous lab we toggled a pin in timer interrupt routines to generate various frequencies. In this lab we are going to explore the output module of Timer A to generate pulses at fixed intervals and varying widths to produce PWM waves.
You can find the schematic of the output module in Timer A handout as well as in the family guide. We include part of the schematic below:

As you can see the output mode is handled by hardware. This means that you can generate signals at the output without the need to use ISRs to toggle pins and spend CPU computing cycles when you use this mode.

Output pins of Timer A output module are called TAx.y. “x” is the timer number. In G2553 we have two timers TimerA_0 and TimerA_1 so x is 0 or 1. “y” is the capture compare register. y is either 0, 1 or 2. A list of available output modes and their description are summarized below. Notice that there are two types of output modes, one only depends on the capture compare register selected (CCR0, CCR1, CCR2), like modes 0, 1, 4 and 5. When the timer counts up to the selected CCR register value, the corresponding output pin is set, reset or toggled. Name of these modes simply follows the operation: “Output”, “Set”, “Toggle” and “Reset”.

The other type of output mode is related to the selected capture compare register (CCR1 or CCR2) and the special CCR0 register. The output changes when the timer reaches the selected CCR register and change again when it reaches the CCR0 value. Examples are modes 2, 3, 6 and 7. In the table you can also see the names of these modes as: “Toggle/Reset”, “Set/Reset”, “Toggle/Set” and “Reset/Set”. In these modes we use the output modules of CCR1 and CCR2, it is not useful to use the output mode of CCR0.

All these built-in options provide you with a lot of flexibility.
PWM Generation

Fig. 2 below illustrates how to generate PWM using Timer A and Reset/Set output mode for CCR1.
1) In this illustration we set the timer to **up mode**. In the up mode, TAR continues to increase at each Timer A clock tick to the value stored in TACCR0 and goes back to 0 again. Here we are using TACCR1 to store the signal value at the sample points.

2) In the figure the output is set to 1 when TAR reaches TACCR0 and the output is reset to 0 when TAR reaches the current value stored in TACCR1. From the table we can see that the corresponding behavior is obtained when the output mode is set to 7, i.e., the Reset/Set mode. Therefore this case we should set the output mode to 7 (in TACCTL1). The corresponding output pin is TA0.1 (on Pin 1.2).

3) The signal value is stored in TACCR1 and is updated every Ts. The update of TACCR1 can be done in either CCR0 or TAR interrupt.

4) Note that it takes CPU clock cycles to update the TACCR1 value in the interrupt service routine. Therefore if the signal value is very small, we may miss the signal value when modulating. This problem can be solved by offsetting the signal or using the up-down down. More methods are available based on the signal source and additional hardware attached to them.

With an understanding of how PWM signals are generated in the **up mode**, we can understand how resolution and sampling period are setup:

Resolution is the value stored in TACCR0

Sampling period Ts is the Timer A count period = TACCR0*\(T_C\) (\(T_C\) is the Timer A Clock) Sampling rate \(f_s\) (PWM frequency) is \(1/T_s\).

\[
T_s/T_C = TACCR0, \quad f_C/f_s = TACCR0
\]

As an example, if you are given a design requirement to generate a 12 kHz PWM (\(f_s = 1/T_s\)) of a signal with 4096 (2^12 bits) resolution, you must have a 50 MHz clock (\(T_C\)). However the G2553 microcontroller does not have such a high frequency clock, so we must compromise and either decrease the sampling frequency \(1/T_s\) to keep the resolution or decrease the resolution to keep the sampling frequency. This example should give you a feel for the constrains required to generate PWM.
Connect any Output Pin to the LED on Launchpad

We have two LEDs available on the Launchpad: LED1 (Red) and LED2 (Green). By default LED1 is connected to P1.0 and LED2 is connected to P1.6. The connections between the output pins P1.0 and P1.6 to the LEDs are established by two jumpers. The location of the two jumpers are marked out by the yellow square in the figure below.

If you remove one of the jumper, you will see two pins exposed. The connection is as shown in the figure below:
You can see that the pins closer to the LEDs (marked out by yellow color) are hardwired to the LEDs. If you take a jumper wire and connect any output pins to the yellow pins, you can establish the connection.

*Make sure you do not throw your jumpers away!*

- Please make sure you understand the content of the document *TimerAModule* before proceed with lab experiments.
- Please attach the code you write in this lab at the end of your lab report. The code should be nicely commented for you to use in later labs. We encourage you to write modular programs by using subroutines. Missing course code will cost you points in your lab report grades.

**Experiment 1 – DC Pulse Width Modulation**

We are going to test the output module configuration by first generating pulse width modulation of a DC signal on output TA0.1 (on Pin 1.2).

*Connect SMCLK to DCO and set the DCO frequency to 1MHz*

*Set Timer A0 input divider to be divide by 8*

*Configure your Timer A0 to count in up mode*

*Set resolution to be $2^{10} = 1024$ by storing 1024 in TA0CCR0*

*Store value 50 into TA0CCR1 in compare mode. We are simulating a DC value of 50 (note that no DC signal is connected as input)*

*Configure your output mode as Reset/Set by setting OUTMOD bits to the corresponding value in TA0CCTL1 register.*

Connect TA0.1 pin out to either one of the LEDs using a jumper wire and observe the result via the intensity of the LED. Our eyes will behave like a low pass filter when we see the lighting of LED.

Change the value in TA0CCR1 to 900. Comment on the difference related to the LED.
Answer the question:

1) What will happen if you are in up/down mode? Will you still have PWM? What about in continuous mode?
2) Based on the values stored in CCR0 and CCR1 registers what is the duty cycle of the generated square wave?

Duty cycle = width of a pulse / period of the square wave

**Experiment 2 – Sine Wave PWM Generation**

In this experiment we are going to generate Pulse Width Modulation of an array of data stored in memory on output TA0.1 (BIT2). We will **connect TA0.1 pin to either one of the LEDs using a jumper wires** and observe the result via the intensity of the LED.

For the signal to modulate we will use a table consisting of half a period of a sine wave. As a result we expect to see the brightness of the LED light to vary from dim to bright and then back to dim in the behavior of a half period of a sine wave. The sine wave table will be inserted in the memory of the microcontroller by loading a “.dat” file into the memory space (RAM or ROM?). The data file (as well as the MATLAB file that generated it) will be provided to you on the lab’s websites.

Here are some requirements:

1) **The sine table contains 256 = 0x100 word long data points**
2) Use a resolution of 1024 as shown in previous experiment
3) Choose a proper clock frequency to have the flashing period around 4 seconds (You do not need to reach exactly 4 second)
4) Update TACCR1 value when timer register reaches TACCR0 value: use TACCR0 interrupt to update value

Setup PWM generation based on requirements 1-5. What did you observe on the LED?

**Short Answer Questions**

1. Why cannot we use CCR2 output of Timer A0 to observe the output waveform using our microcontroller?
2. Once a signal is converted to PWM, it can be converted back to the original signal by using an analog filter. What type of filter (low pass, high pass or band pass) would be needed for this conversion, explain. What cutoff frequency should your filter have?
Section 2 Capture Mode

In this lab we will experiment with the capture mode of Timer A. In this mode we connect an input signal to a Capture Compare Input (CCI) pins (CCIxA, CCIxB etc. in the figure below). When a rising or falling edge (or both) event occurs at a Capture Compare Input, the current value in TAR (Timer A Register) gets recorded into the Capture Compare Register and an interrupt flag CCIFG is set. In other words this mode records the “time” (value in the TAR) in the CCR when an event (rising edge, falling edge etc.) happens at a Capture Compare Input. Thus this mode is often used to measure the width of pulses, time periods of signals (i.e., frequency) etc.

In this experiment we are going to use the capture mode to measure the time period (therefore frequency) of an external signal originating from a signal generator. We can record the “time” (TAR value) of consecutive rising edges of an incoming square wave signal to determine the time period of the square wave. This method can be used, for example, to measure the rotational speed of a motor shaft by measuring the time difference between pulses if the shaft is fitted with sensors that produce pulses.

In this lab we will pay particular attention to the accuracy of the measured time period (or frequency). Different methods will be adopted to deal with high and low frequency measurements respectively.

We are going to start with a summary of the capture mode and then proceed on to the experiments.
Capture Mode

On the partial block diagram of Timer A module above, you can see that the capture mode has four different inputs (marked out in the blue square) called CCIxA (Capture Compare Input x A), CCIxB (Capture Compare Input x B), GND and VCC (we will not deal with GND and VCC in this lab). Recall that for each Timer A we have three Capture Compare Registers (CCRs), therefore the value of x is 0, 1 or 2. GND and VCC, as their name imply, are the internal ground the supply voltages of the MCU. For most of the CCR modules CCIxA and CCIxB are connected to pins of the MCU so that you can connect an external signal to those pins. However CCI0B and CCI1B for Timer0 are not connected to external pins of the MCU, as you can see from the table gotten from the device specific sheet of your MCU (table below). The table lists all the available options. Notice (parts highlighted in red) that CCI0B of Timer0 is hardwired internally to the ACLK and CCI1B of Timer0 is hardwired internally to the output of the Comparator (Comparator is a peripheral on your MCU, you will get to use it in a later lab). This means that CCI0B and CCI1B for Timer0 are not accessible to external signals. Also note that for Timer1 all the CCIxA and CCIxB inputs are connected to external pins of the MCU and none of the inputs are hardwired to an internal peripheral.

<table>
<thead>
<tr>
<th>Timer</th>
<th>CC Input Name</th>
<th>Description</th>
<th>Port and Pin Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer0</td>
<td>CCI0A</td>
<td>TA0.0</td>
<td>P1.1</td>
</tr>
<tr>
<td></td>
<td>CCI0B</td>
<td>ACLK</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td>CCI1A</td>
<td>TA0.1</td>
<td>P1.2</td>
</tr>
<tr>
<td></td>
<td>CCI1B</td>
<td>CAOUT</td>
<td>/</td>
</tr>
</tbody>
</table>
In order to use the capture mode, you need to

1) Setup Timer A Clock
2) Select capture input source
3) Enable capture mode
4) Select capture logic (rising edge/falling edge or both)
5) Chose synchronous/asynchronous (choose synchronous in this lab)
6) Setup output modes and interrupts

**Experiment 3 – Low frequency measurements**

The high/low frequency measurements we are referring here are compared with the Timer A Clock frequency. In this experiment we are going to measure frequencies that are lower than \(1/T_{TACLK}\).

Idea of time period measurement is to capture consecutive rising edges/falling edge of the signal connected to CCIxA in this experiment as shown in the figure below.

Recall that in capture mode you are able to record the TAR value at the rising/falling edges. Assume that CCR reads \(N_1\) at the first rising edge, and reads \(N_2\) at the next rising edge (i.e. after a period), then the period \(T_{signal}\) is:
\[ T_{\text{signal}} = (N2 - N1) * T_{\text{TACLK}} \]

Determine the values of \( N2 - N1 \) in your ISR (Interrupt Service Routine). Use your own logic in the ISR to make sure that you measure exactly one period. Here is one way you can implement this logic:

Use a register or variable as a flag. Set the value of this flag to 0 at the start of the program. In your ISR, somewhere in the end, toggle this flag. Before the toggling code construct an if statement based on the value of the flag as shown in the pseudo code below:

```pseudo
if (flag == 0)
{
    Measure CCR. This is N1. Store in a register or variable;
}
else // flag == 1
{
    Measure CCR. This is N2. Calculate N2 - N1 and store this value in a register or variable.
}
Toggle flag
```

As stated before, for this method to be accurate we need \( f_{TACLK} \) to be much larger than \( f_{\text{signal}} \).

If we want to be able to measure a frequency as low as 0.1Hz, what clock configuration should you use? Note that we want to calculate the signal period by using subtraction \( N2 - N1 \); for the subtraction to work properly in two’s compliment, \( N2 - N1 \) cannot be larger than \( 2^{16} - 1 \). Therefore the maximum \( T_{TACLK} \) we can use is limited by the lowest frequency we need to measure (0.1Hz = 10sec). We have the following constrains:

\[
(2^{16} - 1) * T_{TACLK} < T_{\text{signal}}
\]

\[
f_{TACLK} < 6553Hz
\]
In this lab, we are going to choose $f_{\text{TACLK}}$ of 5000Hz. Think about the DCO frequencies we have experimented in Basic Clock Module lab, how are you going to reach $f_{\text{TACLK}} = 5000\text{Hz}$? What DCO frequency are you going to use? (Recall that SMCLK and TimerA both have pre-dividers)

- You need to justify your clock frequency selection in the lab report.
- Please measure the actual clock frequency under your setting to be able to correctly evaluate your measurement

Use Timer A0 for this experiment and set it in continuous mode.

Connect the input signal to CCIXA where “x” stands for the CCR register number 0-2 (remember to configure the appropriate pin). Which CCR register should you choose and why?

Configure corresponding CCR to be used in capture mode. Select capture source to CCIXA by setting CCIS bits in TA0CCTLx register.

Enable interrupts so that an interrupt is thrown whenever there is a rising edge event at CCIOA.

Use a signal generator to send in a square wave signal with a minimum of 0 volts and a maximum of no more than 3.5 volts. Measure the frequency of this signal using the method and accuracy outlined above. Repeat the experiment for 1Hz, 100Hz, 1kHz, 5kHz. Compare your experimental values with the actual readings on the signal generator. Also calculate the percentage error in your measurement.

How to inspect your results: (Very Important)

Since our microcontroller does not support run time debugging (the CCS debugger cannot update values while the microcontroller is running), we need to set a break point to check our results.

The break point should be set after we have performed the subtraction N2 – N1. However we will need to manually clear the flag CCIFG in the TAxCCTLx register in order for the break point to work properly even if you are using CCR0 (normally we do not need
to clear this flag manually). This is because while your program is paused, the next rising edges of the incoming signal are still being captured by hardware (flag will be raised) causing a wrong TAR value to be registered at wrong time.

**Discussion Questions**

There is a limit on how high a frequency you can measure using the method given above: \( f_{\text{signal}} << f_{\text{TACLK}} \). To extend this range of measurement we need to switch to higher clock frequencies at appropriate time. Therefore this condition translates to \( f_{\text{signal}} << \text{highest clock frequency of the MCU} \). But what if the frequency to measure is much higher than the frequency of the highest clock frequency that the MCU can offer? i.e.,

\[ f_{\text{signal}} >> \text{clock frequency}. \]

In this case we need to follow an alternate method for frequency measurement. To illustrate the method we will use a clock with a low frequency, for example the ACLK, although ACLK is not recommended for timing.

In this method we will drive the timer (TAR) with the input signal whose frequency we want to measure, and we will connect ACLK to the Capture Compare Register module. An illustration of this method is shown in the figure below.

![Diagram](image.png)

So now we are measuring how many cycles of the signal are contained in one period of ACLK. The period of the signal can be found via:

\[ T_{\text{TACLK}} = (N2 - N1) \times T_{\text{signal}} \]

Answer the following questions:
1) Which timer and which capture compare register can be used for the high frequency measurement described above using the Launchpad? Give your reasoning.

2) How high a frequency can you measure (assuming all internal timing constraints can be satisfied) by using just the ACLK frequency?