ECE 2049 LECTURE 14 PART 2: PRACTICE EXAM

ECE2049 -- Homework 7 / Exam 2 Review
Clocks, Timers, ADCs, and Operating Modes

Submission notes: For homework credit you must complete problem 1 and any one other problem from this document—we will review all of the problems in class!

Problem 1: Alkaline AAA batteries have a capacity of 1100 mAh. A certain embedded system uses the MSP430F5529 powered by 2 AAA batteries in series (i.e. Vcc = 3V). How long can the system run if the MSP430F5529 is always in active mode? How long can it run if the system is in LPM0 84% of the time?

You may assume the system is running at the default clock frequencies. If you need to make any further assumptions, state them clearly.

BATTERY CAPACITY: 1100mA·h $V_{CC}=3V$

(i) ALWAYS IN ACTIVE MODE?

ASSUME TYPICAL VALUES.

$I_{AM} = 0.36mA$

$$\text{RUNTIME} = \frac{1100mA\cdot h}{0.36mA} = 3055 \text{ HOURS}$$

(ii) LPM0 FOR 84% OF TIME?
AM FOR REST

$I_{LPM0} = 83\mu A$ (ASSUME 25°C)

DANGER: MIXED UNITS!

$$I_{AVG} = (.84)(83\mu A) + (1-.84)(.36mA)$$
$$= 122.3\mu A$$

$$\text{RUNTIME} = \frac{1100mA\cdot h}{122.3\mu A} \approx 8639 \text{ HOURS}$$

$$\approx \boxed{359 \text{ DAYS}}$$

ECE2049 -- PRACTICE EXAM #2
Clocks, Timers, and Digital I/O

Study HW3, Class Notes, Davies Ch 2.6, 5.8, 8, 9.2-3, 9.7,
MSP43F5529 User's Guide Ch 5, 17, 28
Work all problems with your note sheet first THEN look at solutions!

1. Answer the questions below completely. (25 pts)

a. What are the default frequencies for ACLK, MCLK and SMCLK on the MSP430F5529 after power up? What is the purpose of these various clock signals (i.e. What are they used for)?

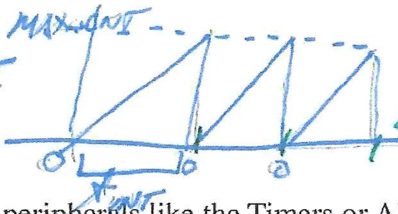
ACLK = 32768 Hz
SMCLK = 1.048576 MHz] CAN BE USED BY PERIPHERALS
MCLK = 1.048576 MHz – CONTROLS CPU EXECUTION

b. How does the CPU know where to go on an interrupt? How does it find the correct ISR?

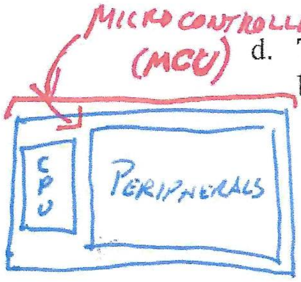
THE INTERRUPT VECTOR TABLE MAPS EACH HARDWARE INTERRUPT (EX. TIMER A2, I/O PORT 1, SPI, ETC) TO THE ADDRESS OF ITS ISR IN CODE MEMORY.

c. Explain the operation of a Timer in "up mode" (ie. What happens to the timer count? When is an interrupt triggered?).

IN UP MODE, THE TIMER COUNTS FROM 0-MAX_CNT.



INTERRUPTS OCCUR WHEN COUNT GOES FROM MAX_CNT → 0



d. True or [✓ False]: The operation of peripherals like the Timers or ADC12 causes a big drain on CPU speed. Why or why not?

FALSE! TIMERS + THE ADC ARE HARDWARE PERIPHERALS – THEY OPERATE INDEPENDENTLY OF THE CPU!

e. True or [✓ False]: Interrupt Service Routines (ISRs) should be kept short because the CPU will stop executing them after 255 instructions.

FALSE! ISRS CAN TAKE A LONG TIME, BUT YOU WILL HAVE ISSUES IF IT DOES NOT FINISH BEFORE THE NEXT ONE ARRIVES.

2a.

TIMER A2

TA2CTL = TASSEL_1 + MC_1 + ID_0

$$TASSEL\_1 \rightarrow \text{CLOCK SOURCE} = ACLK = 32768\,Hz$$

$$MC\_1 \rightarrow \text{UP MODE}$$

$$ID\_0 \rightarrow \text{DIVIDE BY 1}$$

TA2CCR0 = 16383

TA2CCTL0 = CCIE

SINCE UP MODE

$$t_{INT} = \frac{(\overset{\text{KNOW}}{MAX\_CNT+1})}{f_{CLK} \leftarrow \text{KNOW}}$$

$$= \frac{(16383+1)}{32768} = \frac{16384}{32768} = \boxed{0.5s}$$

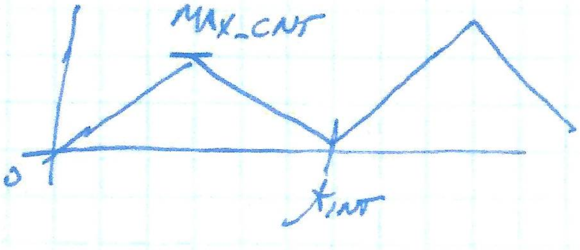
b.

CONTINUOUS MODE

IN CONTINUOUS MODE, ALWAYS COUNT TO MAXIMUM TIMER VALUE.

$$t_{INT} = \frac{2^{16}}{f_{CLK}} = \frac{2^{16}}{32768} \frac{\text{TICKS}}{\text{TICKS/SEC}} = \frac{65536}{32768} = 2.0_{SEC}$$

UP-DOWN MODE!



$$t_{INT} = \frac{(2 * MAX\_CNT)}{f_{CLK}}$$

$$t_{INT} = \frac{(2 * 24000)}{\cancel{3278}\ 32768} = \frac{48000\ \text{TICKS}}{32768\ \text{TICKS/SEC}}$$

$$\approx \boxed{1.4648\ SEC}$$

2c. SET REGISTERS TO USE SMCLK, TO COUNT INTERVALS ~~OF~~ 25ms.

FOR MORE ON TIMERS, SEE HW6 SOLUTIONS, AND LECTURE 13.

(IN ADDITION TO HW5 AND TIMER LECTURES)

$f_{SMCLK} = 1.048576\ MHz$

$t_{INT} = \frac{MAX\_CNT+1}{f_{CLK}}$

t_{INT} ↑ KNOW f_{CLK} ← KNOW

1ms = 0.001s

25ms = 0.025s

$25ms = \frac{MAX\_CNT+1}{1048576\ Hz}\ \frac{TICKS}{TICKS/SEC}$

$MAX\_CNT+1 = \lceil 26214.4 \rceil \approx 26214$

$MAX\_CNT =$ [illegible] $\boxed{26213}$ TICKS

d. FAST OR SLOW?

HOW LONG UNTIL OFF BY 0.025s (25ms)?

REPORTED TIME: TIME YOU INTENDED ISR TO EXECUTE = 25ms

ACTUAL TIME: PLUG MAX-CNT INTO EQUATION + FIND t_{INT} AGAIN

$\text{ACTUAL } t_{INT} = \frac{26213\ TICKS + 1}{1048576\ TICKS/SEC} \approx .02499962$ SECONDS

T_{INT} ACTUAL < T_{INT}, REPORTED, ∴ TIMER IS **FAST**

HOW LONG UNTIL OFF BY 0.025 SEC?

$\text{TOTAL DIFF} = \left|(x \text{ INTERRUPTS})(t_{INT,REPORTED} - t_{INT,ACTUAL})\right|$

$0.025s = \left|(x)(0.025 - 0.02499962\ldots)\right|$

$x = 65535.999 \approx 65536$ INTERRUPTS UNTIL OFF BY 25ms.

(65536 INTERRUPTS)(SEC/INT) = $\boxed{1638.4\ SEC}$

Problem 3: Accelerometers are used for a variety of applications including determining orientation, detecting vibrations, or as components of navigation systems. Accelerometers provide output in g's, which is acceleration relative to the force of gravity. Let's say we want to use accelerometer as single-axis tilt sensor to measure the orientation of a device. We can use the accelerometer readings to measure tilt as shown in the examples below:

Tilt = 0°
Reading: 1g

Tilt = 45°
Reading: 0.707g

Tilt = 90°
Reading: 0g

SENSOR UNITS

Tilt = 180°
Reading: -1g

An older version of the ECE2049 lab boards had a AXDL335 3-axis accelerometer, which provides three output signals X_{Out}, Y_{Out}, and Z_{Out} as analog outputs with output voltages that correspond to the acceleration along each axis. For this problem, we only need to read the output on the z-axis, which provides enough information to measure tilt (as shown in the examples).

For this problem, assume that V_{CC} for the sensor is 3V. V_{CC} on the MSP30 is still 3.3V.

- **a.** The datasheet for the ADXL335 is provided on the course website. The datasheet assumes that the chip has a source voltage of 3V. Using this assumption, what is the measurement range (in g) and sensitivity (ie. resolution, in mV/g) of the ADXL335?
- **b.** Write an equation to express the voltage output ~~voltage~~ of the sensor (on the z axis) in terms of acceleration (in g). (Hint: You will also need to the *bias voltage*, or the voltage at a reading of 0g based on the datasheet.)
- **c.** What ADC12 reference voltage would you select in order to measure acceleration over the full measurement range of the sensor?
- **d.** What voltage would the sensor output for a tilt of 45 degrees? Using the reference voltage you selected in part (c), what is the ADC12 output for this voltage?
- **e.** Assume that the ADC12 has already been configured using the reference voltage you selected. Write a C function `calc_tilt` that converts the ADC12 output code to a tilt angle from 0 to 180 degrees—you can assume that the standard library functions in `math.h` are available.
  (For this part, you can also assume that the ADC12 output is always within ±1g.)

3. ADC PROBLEM: ADXL335

RANGE ± 3.6g

RESOLUTION 300mV/g

$AX+B$

$V_{OUT} = (300 mV/g)(INPUT) + 1.5V$

$V_{OUT} = (300 mV/g)(Z_{OUT}) + 1.5V$

CHECK MEASUREMENT RANGE.

$+3.6g$: $(300 mV/g)(3.6) + 1.5V =$ 2.58V

$-3.6g$ $(300 mV/g)(-3.6) + 1.5V =$ 420mV

$\boxed{.420V - 2.58V}$

2. REFERENCE VOLTAGE?

$V_{REF-} = 0V$ $V_{REF+} = 1.5V, 2.5V, \boxed{3.3V}$

2.58V > 2.5V, SO PICK 3.3V TO COVER WHOLE RANGE OF SENSOR.

IF MEASUREMENT RANGE IS ONLY ±1g...

$-1g \Rightarrow .2V$

$+1g \Rightarrow 1.8V$

IN THIS CASE COULD PICK 2.5V.

d. FIND V_{OUT}, ADC CODE FOR. TILT OF 45°!

$Z_{OUT} = 0.707g$ (FROM PROB. DESCRIPTION)

$$V_{OUT} = (360\,mV/g)(.707g) + 1.5V$$

$$= 1.7121V$$

FOR OUR CONFIG:

K = 12

REF = (0V, 3.3V)

$$CODE = \left\lfloor \left(\frac{V_{IN} - V_{REF-}}{V_{REF+} - V_{REF-}} \right) (2^K - 1) \right\rfloor$$

$$= \left\lfloor \frac{1.7121V}{3.3V} (2^{12} - 1) \right\rfloor$$

$$= \lfloor 2124.56 \rfloor = \boxed{2124}$$

$$CODE = \frac{V_{IN}}{3.3V} (2^{12} - 1)$$

$$CODE = V_{IN} \left(\frac{2^{12} - 1}{3.3V} \right)$$

$$V_{IN} = CODE \left(\frac{3.3V}{2^{12} - 1} \right)$$

FOR THE NEXT PART, WE NEED TO REWRITE EQUATIONS SO WE CAN USE IN OUR CODE:

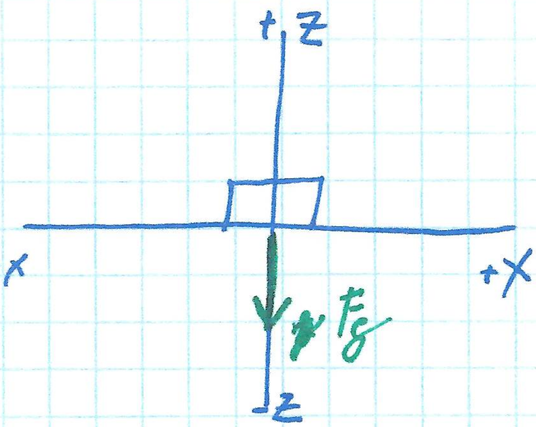
$$V_{OUT} = (300\,mV/g)(F_Z) + 1.5V$$

$$F_Z = \frac{V_{OUT} - 1.5}{(360\,mV/g)}$$

~~ADD~~ ON FINDING THE TILT ANGLE

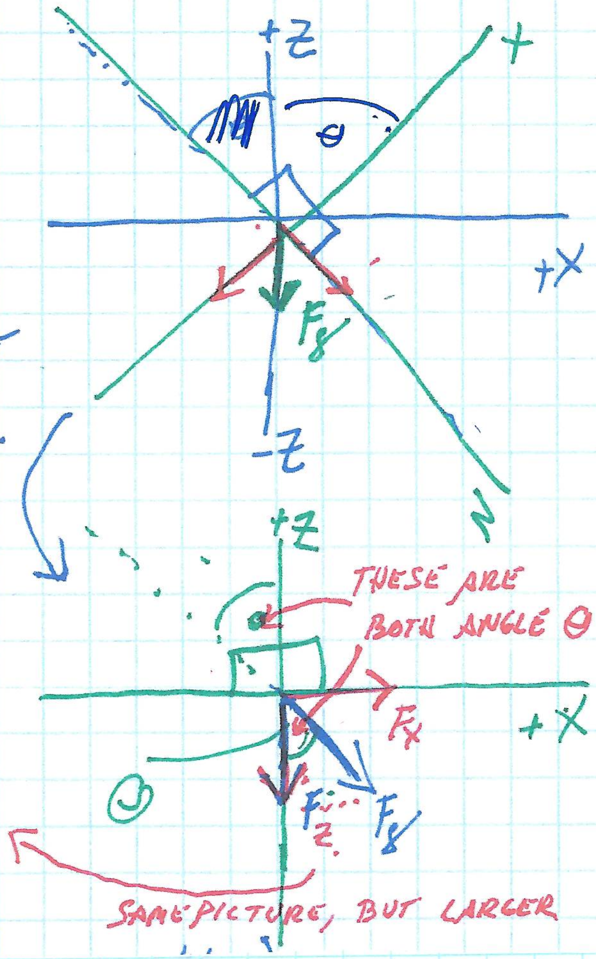
①. AT REST, THE SENSOR JUST READS THE FORCE DUE TO GRAVITY (1g).

AT REST, THE SENSOR WILL READ THIS AS 1g ON THE Z-AXIS.



② IF WE TILT THE SENSOR, BY SOME ANGLE θ, THE FORCE OBSERVED ON THE Z-AXIS CHANGES.

WE CAN MORE EASILY VISUALIZE THIS BY REDRAWING OUR COORDINATE SYSTEM IN TERMS OF WHAT THE SENSOR SEES (GREEN AXES).

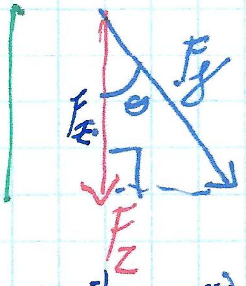


③

NEED: TILT ANGLE

KNOW: F_Z, ~~F_X~~

$F_g = 1g$



$$\theta = \cos^{-1}\left(\frac{F_Z}{\cancel{F_g}}\right) = \cos^{-1}(F_Z)$$

SO TO FIND THE TILT ANGLE, WE CAN JUST COMPUTE:

$$\boxed{\theta = \cos^{-1}(F_Z)}$$

8

Part e

```
#include <math.h>

// Equation constants
// Optional:  this example uses the 'f' suffix on decimal numbers
// to make absolutely sure the compiler interprets the constant as a
// float (this isn't strictly required, but is often a good idea)
#define VOLTS_PER_BIT      (0.0007326007f) // (3.0/4095)
#define VOLTS_PER_G        (0.300f)
#define ZERO_OFFSET_VOLTS (1.5f)

float calc_tilt(unsigned int adc_code)
{
  float v_adc = ((float)adc_code) * VOLTS_PER_BIT;
  float gees = (v_adc - ZERO_OFFSET_VOLTS)/VOLTS_PER_G;

  // Convert to tilt angle
  // acos() returns angle in radians, so need to convert to degrees
  // This example uses acosf() which is just a version of acos that
  // operates on single-precision floats rather than double-precision.
  // M_PI is the math.h constant for pi.
  float degrees = acosf(gees) * (M_PI/180.0f);
  return degrees;
}
```

EXTRA PROBLEM

TIMER COUNT EXAMPLE

E_1 - READ DATA EVERY 200 ms FROM SOME SENSOR

E_2 - SEND DATA OUT EVERY 5 SECONDS

a. SELECT A TIMER PERIOD?

$A_{INT} = ?$ MUST BE AT LEAST 200 ms

SO THAT E_1 CAN OCCUR AT DESIGNED RATE

b. SUPPOSE $A_{INT} = 25\,ms$

E_1 200 ms $\frac{200\,ms}{25\,ms} = 8$ INTERRUPTS

E_2 5000 ms $\frac{5000\,ms}{25\,ms} = 200$ INTERRUPTS

```c
/*************** EVENTS EXAMPLE **********************/
/***************  8 July 2021  **********************/
/****************************************************/

#include <msp430.h>

#include "peripherals.h"
#include "lecture.h"
#include "utils/test_runner.h"
#include "utils/ustdlib.h"

// Function Prototypes
void swDelay(char numLoops);
void runtimerA2(void);
void displayTime(unsigned long time);

// For this example, we have two "event" functions taht need to run
// at specific intervals
void event1(void); // Need to run every 200ms (every 8 ticks)
void event2(void); // Need to run every 5000ms (every 200 ticks)

// We can handle this in two ways--which one we would use in
// a particular scenario depends on how long each event takes to run:

// *Example 1*:  Assume both event1 and event2 can run in << t_INT
//    - If  we can do BOTH events in a shorter time than t_INT, then
//      we can call both events from the ISR!  This requires that both event
//      are done before the next t_INT
//      Ex.  What if event1 and event2 each take 1ms to run?
//           (1ms + 1ms) << 25ms => OK!

// *Example 2*:  Assume event2 takes a long time
//    - If event2 takes longer than 25ms to run, we can't put it inside the ISR
//      because then the ISR would not finish in time.  Instead, we need to
//      call event2 from main() where it can take longer.  We do this often
//      in lab for slow tasks like updating the LCD.

// (continued on next page)
```

```
48 volatile unsigned long time_count = 0;
49
50 #pragma vector=TIMER2_A0_VECTOR
51 __interrupt void TimerA2_ISR(void) // Runs every 25ms
52 {
53     time_count++; // Increments global counter of clock ticks
54
55     // Run event1 every 8 ticks
56     // Inside the ISR, we can periodically schedule an event like this
57     if ((time_count % 8) == 0) { // Runs every 8 ticks
58         event1();
59     }
60
61     // EXAMPLE 1 ONLY (if event1 runs in << 25ms, we can also schedule it
   here)
62 //    if ((time_count % 200) == 0) { // Runs every 200 ticks
63 //        event2();
64 //    }
65 }
66
67 // Main
68 void main(void)
69 {
70     unsigned long last_event2 = 0;
71     WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer.
72
73     runtimerA2(); // Configure timer to interrupt every 25ms
74     _enable_interrupt();
75
76     while (1)
77     {
78         // Example 2, method 1:  We *could* schedule event2 in main()
79         // in a similar way as in the ISR, but it might not work as we
   expect!
80         //
81         // The if condition (line 76) is only true at ticks
82         // 0, 200, 400, 600, ...
83         // If something else is going on in main() when timer_count == 200,
84         // (like event3)  event2 won't run for this interval!)
85 //        if ((time_count % 200) == 0) {
86 //            event2();
87 //        }
88
89         // (continued on next page)
90
91
92
93
```

```
        // Example 2, Better method
        // Instead of scheduling our event at specific values of time_count,
        // we can instead keep track of the last time event2 was run, and then
        // run the event after enough time has elapsed
        // Here, we store the last time event2 ran in last_event2.  If
        // If >= 200 ticks have elapsed since the last event2, we run event2
        // This is more reliable!
        if ((last_event2 - time_count) >= 200) {
            event2();
            last_event2 = time_count; // Record the current time of event2
        }

        // What if this other event takes 2s to run?
        event3();

        // ...
    }
}

void event1(void)
{
    // ...
}

void event2(void)
{
    // ...
}

// . . . Other demo functions omitted . . .

```