Ali Tejani
Caroline Yao
Amt3639
Chy253

Lab 2 Prep

2. Questions
   a. What is the purpose of all the DCW statements?
      i. To store the addresses of the port F data registers in memory.
   b. The main program toggles PF1. Neglecting interrupts for this part, estimate how fast PF1 will toggle.
      i. The code given takes 6 instructions, so PF1 will toggle every 150 nanoseconds.
   c. What is in R0 after the first LDR is executed? What is in R0 after the second LDR is executed?
      i. After the first LDR is executed, R0 has the address of Port F, 0x40025000. After the second LDR is executed, R0 has the contents of the memory at 0x40025008.
   d. How would you have written the compiler to remove an instruction?
      i. In the first LDR, store the address of Port F in R1 instead of R0, so it would not have to copy the address from memory twice.
   e. 100-Hz ADC sampling occurs in the Timer0 ISR. The ISR toggles PF2 three times. Toggling three times in the ISR allows you to measure both the time to execute the ISR and the time between interrupts. See Figure 2.1. Do these two read-modify write sequences to Port F create a critical section? If yes, describe how to remove the critical section? If no, justify your answer?
      i. No, the program uses bit specific addressing to change the value of PF1 and PF2. Since the two sequences read and write to different addresses, there is no critical section.

```
// ******** ADCTestMain.c **************
// Ali Tejani and Caroline Yao
// amt3639 and chy253
// Creation Date: 1/31/2017
// Possible main program to test the lab 2
// Runs on TM4C123
// Uses ST7735.c LCD.
// Lab section: Tue/Thur 12:30 - 2 PM
// TA: Lavanya
// Last Revision: 2/1/2017

// center of X-ohm potentiometer connected to PE3/AIN0
// bottom of X-ohm potentiometer connected to ground
// top of X-ohm potentiometer connected to +3.3V
#include <stdint.h>
#include "fixed.h"
#include "ST7735.h"
#include "ADCSWTrigger.h"
#include "../inc/tm4c123gh6pm.h"
#include "PLL.h"

#define PF2             (*((volatile uint32_t *)0x40025010))
#define PF1             (*((volatile uint32_t *)0x40025008))
void DisableInterrupts(void);           // Disable interrupts
void EnableInterrupts(void);            // Enable interrupts
long StartCritical (void);              // previous I bit, disable interrupts
void EndCritical(long sr);              // restore I bit to previous value
void WaitForInterrupt(void);                    // low power mode
void CalculateTimeJitter(void);         // calculate time jitter once dumps are full
void PlotPMF(void);                                                     // create pmf plot once dumps
are full

// size of dumps
const int DUMP_SIZE = 1000;
// counter to increment through dumps
volatile uint32_t Count;
// time dump
volatile uint32_t TimeDump[DUMP_SIZE];
// ADC value dump
volatile uint32_t ADCDump[DUMP_SIZE];


#define PF4   (*((volatile uint32_t *)0x40025040))

// Subroutine to wait 10 msec
// Inputs: None
// Outputs: None
// Notes: ...
void DelayWait10ms(uint32_t n){uint32_t volatile time;
  while(n){
    time = 727240*2/91;  // 10msec
    while(time){
	  	time--;
    }
    n--;
  }
}

// Subroutine to wait for switch to go to next screen
void Pause(void){
  while(PF4==0x00){
    DelayWait10ms(10);
  }
  while(PF4==0x10){
    DelayWait10ms(10);
  }
}
// This debug function initializes Timer0A to request interrupts
// at a 100 Hz frequency.  It is similar to FreqMeasure.c.
void Timer0A_Init100HzInt(void){
  volatile uint32_t delay;
  DisableInterrupts();
  // **** general initialization ****
  SYSCTL_RCGCTIMER_R |= 0x01;      // activate timer0
  delay = SYSCTL_RCGCTIMER_R;      // allow time to finish activating
  TIMER0_CTL_R &= ~TIMER_CTL_TAEN; // disable timer0A during setup
  TIMER0_CFG_R = 0;                // configure for 32-bit timer mode
  // **** timer0A initialization ****
                                   // configure for periodic mode
  TIMER0_TAMR_R = TIMER_TAMR_TAMR_PERIOD;
  TIMER0_TAILR_R = 799999;         // start value for 100 Hz interrupts
  TIMER0_IMR_R |= TIMER_IMR_TATOIM;// enable timeout (rollover) interrupt
  TIMER0_ICR_R = TIMER_ICR_TATOCINT;// clear timer0A timeout flag
  TIMER0_CTL_R |= TIMER_CTL_TAEN;  // enable timer0A 32-b, periodic, interrupts
  // **** interrupt initialization ****
                                   // Timer0A=priority 2
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x40000000; // top 3 bits
  NVIC_EN0_R = 1<<19;              // enable interrupt 19 in NVIC
}

// Interrupt handler that reads ADC value and stores in dump
void Timer0A_Handler(void){
  TIMER0_ICR_R = TIMER_ICR_TATOCINT;    // acknowledge timer0A timeout
  PF2 ^= 0x04;                          // profile
  PF2 ^= 0x04;                          // profile
  ADCDump[Count] = ADC0_InSeq3();                // Add ADC value to array
  TimeDump[Count] = TIMER1_TAR_R;                // Add time to array
  Count+=1;                                      // increment counter
  PF2 ^= 0x04;                          // profile
}

// Initializes timer to store value into time dump
void Timer1_Init(void){
  SYSCTL_RCGCTIMER_R |= 0x02;   // 0) activate TIMER1
  TIMER1_CTL_R = 0x00000000;    // 1) disable TIMER1A during setup
  TIMER1_CFG_R = 0x00000000;    // 2) configure for 32-bit mode
  TIMER1_TAMR_R = 0x00000002;
                             // 3) configure for periodic mode, default down-count settings
  TIMER1_TAILR_R = 0xFFFFFFFF;  // 4) reload value
  TIMER1_TAPR_R = 0;            // 5) bus clock resolution
  TIMER1_ICR_R = 0x00000001;    // 6) clear TIMER1A timeout flag
  //TIMER1_IMR_R = 0x00000001;    // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0xFFFF00FF)|0x00008000; // 8) priority 4
// interrupts enabled in the main program after all devices initialized
// vector number 37, interrupt number 21
  //NVIC_EN0_R = 1<<21;           // 9) enable IRQ 21 in NVIC
  TIMER1_CTL_R = 0x00000001;    // 10) enable TIMER1A
}

// initialize GPIO, Timers, and LCD
void init(void) {
  PLL_Init(Bus80MHz);                  // 80 MHz
  SYSCTL_RCGCGPIO_R |= 0x20;            // activate port F
  ADC0_InitSWTriggerSeq3_Ch9();         // allow time to finish activating
  Timer0A_Init100HzInt();               // set up Timer0A for 100 Hz interrupts
        Timer1_Init();
        // set up Timer1
  ST7735_InitR(INITR_REDTAB);                                         // set up LCD
  GPIO_PORTF_DIR_R |= 0x06;             // make PF2, PF1 out (built-in LED)
  GPIO_PORTF_AFSEL_R &= ~0x06;          // disable alt funct on PF2, PF1
  GPIO_PORTF_DEN_R |= 0x06;             // enable digital I/O on PF2, PF1
                                // configure PF2 as GPIO
  GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFFF00F)+0x00000000;
  GPIO_PORTF_AMSEL_R = 0;               // disable analog functionality on PF
  PF2 = 0;                                       // turn off LED
  Count = 0;
                // Start Counter at 0
  EnableInterrupts();
```

```
}

int main(void){
        init();

        while(Count < DUMP_SIZE) {                              // wait for dumps to fill
                WaitForInterrupt();
        }
        DisableInterrupts();                    // make sure no more data is being added to dumps

        while(1) {
                CalculateTimeJitter();                          // Display time jitter
                Pause();
                PlotPMF();                                      // display pmf
                Pause();
        }
}

// Calculates time jitter from dumps and displays to LCD
void CalculateTimeJitter() {
        uint32_t maxTime = TimeDump[0] - TimeDump[1];                   // initialize variables
        uint32_t minTime = TimeDump[0] - TimeDump[1];

        for(uint32_t i = 1; i < DUMP_SIZE - 1; i += 1) {        // find each time difference
                uint32_t nextTime = TimeDump[i] - TimeDump[i + 1];
                                                        // and compare to current max  and min
                if(nextTime > maxTime) { maxTime = nextTime; }
                if(nextTime < minTime) { maxTime = nextTime; }
        }

        uint32_t timeJitter = maxTime - minTime;                // compute time jitter
        char* output = "        0 ns";
        for(uint32_t i = 9; timeJitter > 0; i += 1) {           // Display Jitter
                output[i] = '0' + timeJitter % 10;
                timeJitter = timeJitter / 10;
        }
        ST7735_FillScreen(ST7735_BLACK);
        ST7735_SetCursor(0,0);
        ST7735_OutString("Time Jitter: \r");
        ST7735_OutString(output);
}

// stores the occurrences of each adc value in the dump
static uint32_t Occurrences[4096];

// Creates and displays the graph of the PMF to the LCD
```

```
void PlotPMF() {
    for(uint32_t i = 0; i < DUMP_SIZE; i += 1) {
                                                // compute the number of occurrences
        uint32_t ADCValue = ADCDump[i];
        Occurrences[ADCValue] += 1;
    }
    ST7735_PlotClear(0,100);                    // Clear graph
    ST7735_FillScreen(ST7735_BLACK);
    ST7735_SetCursor(0,0);
    ST7735_OutString("PMF");
    for(int i = 0; i < 128; i += 1) {           // display graph
        for(int j = 0; j < 32; j += 1) {        // display every value in graph
            ST7735_PlotLine(Occurrences[i*32 + j]);
        }
        ST7735_PlotNext();
    }
}
```