Karime Saad - Tarang Khandpur

ks38728 - tk8435

EE 445L

**Lab 2 Preparation: Performance Debugging**

**Questions:**

1. What is the purpose of all the DCW statements?

* DCW (Data Constant Word) – allocates 1 or more halfwords of memory. Since a word on the CortexM4 is 32 bits so DCW will allocate 16 bits of memory. Most computers are byte addressable so each of the 4 bytes in the word on the CortexM4 has its own contiguous address. The address of each line increases by 2 or 4 (because each instruction needs 2 bytes, or 4 bytes ). (ex. EOR r0,r0 #0x02 needs 4 bytes).
* The DCW 0xE608 reserves a halfword and stores the value E608(16 bits –> 2 bytes –> +2 addresses) – even if the contents require only 1 byte)

1. The main program toggles PF1. Neglecting interrupts for this part, estimate how fast PF1 will toggle.

* PF1 will take 150ns since the main program does 6 instructions of Assembly code that each take approximately 25ns.

1. What is in R0 after the first LDR is executed? What is in R0 after the second LDR is executed?

* The first LDR loads the address of PF1 (0x40025000) and the second LDR loads the value of PF1 (either 0x00 or 0x02).

1. How would you have written the compiler to remove an instruction?

* Under the assumption that the meaning of this question is to ascertain how to remove the redundant load operation generated in the Assembly code, one possibility would be to:

Make the compiler write DCW 0x5008 instead of DCW 0x5000, then

LDR r0,[pc,#24]

EOR r0,r0,#0x02

LDR r1, [pc,#16]

STR r0, [r1,#0x08]

This effectively removes the necessity to offset by x08.

1. 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?

* No. because, PF1 is not read or written to when the PF2 is being modified and PF1 and PF2 have been defined separately with their own addresses. A critical error would occur if there was variable manipulation during a concurrent interrupt. However, since PF1 and PF2 are being written to separate specific addresses, they will not converge or result in a critical error.

**main.c**

#include <stdint.h>

#include "ADCSWTrigger.h"

#include "../inc/tm4c123gh6pm.h"

#include "PLL.h"

#include "Timer1.h"

#include "ST7735.h"

#define PF2 (\*((volatile uint32\_t \*)0x40025010))

#define PF1 (\*((volatile uint32\_t \*)0x40025008))

#define PMF\_MAX\_SIZE 4096

#define ARR\_SIZE 1000

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 CalculateJitter(void);

void CalculatePMF(void);

void CalculateXAxis(void);

void CalculateYAxis(void);

void DrawPMF(void);

void ResetScreen(void);

void PortF\_Init(void);

volatile uint32\_t ADCvalue;

uint32\_t timeStamps[ARR\_SIZE] = {0};

uint32\_t adcValues[ARR\_SIZE] = {0};

uint32\_t currIndex = 0;

uint32\_t jitter = 0;

uint32\_t pmf[PMF\_MAX\_SIZE] = {0};

uint32\_t pmfMinX = 0;

uint32\_t pmfMaxX = 0;

uint32\_t pmfMinY = 0;

uint32\_t pmfMaxY = 0;

uint32\_t calculating = 1;

// 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

}

void Timer0A\_Handler(void){

TIMER0\_ICR\_R = TIMER\_ICR\_TATOCINT; // acknowledge timer0A timeout

PF2 ^= 0x04; // profile

PF2 ^= 0x04; // profile

ADCvalue = ADC0\_InSeq3();

PF2 ^= 0x04; // profile

if(currIndex < ARR\_SIZE){

timeStamps[currIndex] = TIMER1\_TAR\_R;

adcValues[currIndex] = ADCvalue;

currIndex ++;

}

}

int main(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

PortF\_Init();

Timer1\_Init();

ResetScreen();

EnableInterrupts();

while(1){

while(currIndex < ARR\_SIZE){

PF1 ^= 0x02; // toggles when running in main

}

DisableInterrupts();

CalculateJitter();

CalculatePMF();

DrawPMF();

}

}

void CalculateJitter(void){

uint32\_t smallestTimeDiff = timeStamps[0] - timeStamps[1];

uint32\_t largestTimeDiff = timeStamps[0] - timeStamps[1];

uint32\_t delta = 0;

for(uint32\_t i=1; i<ARR\_SIZE - 1; i++){

delta = timeStamps[i - 1] - timeStamps[i];

if(delta < smallestTimeDiff){

smallestTimeDiff = delta;

}

if(delta > largestTimeDiff){

largestTimeDiff = delta;

}

}

jitter = smallestTimeDiff - largestTimeDiff;

}

void CalculatePMF(void){

CalculateXAxis();

CalculateYAxis();

}

void CalculateXAxis(void){

pmfMinX = adcValues[0];

pmfMaxX = adcValues[0];

for(uint32\_t i=0; i<ARR\_SIZE; i++){

if(adcValues[i] < pmfMinX){

pmfMinX = adcValues[i];

}

if(adcValues[i] > pmfMaxX){

pmfMaxX = adcValues[i];

}

}

}

void CalculateYAxis(void){

pmfMinY = pmf[adcValues[0]];

pmfMaxY = pmf[adcValues[0]];

for(uint32\_t i=0; i<ARR\_SIZE; i++){

//Get range for Y axis: minY and maxY

if(adcValues[i] < pmfMinY){

pmfMinY = adcValues[i];

}

if(adcValues[i] > pmfMaxY){

pmfMaxY = adcValues[i];

}

//Add occurence of ADC value

pmf[adcValues[i]] += 1;

}

}

void DrawPMF(void){

for(uint32\_t x = pmfMinX; x<pmfMaxX; x++){

if(adcValues[x] != 0){

ST7735\_DrawFastVLine(adcValues[x], 20, pmf[adcValues[x]], ST7735\_WHITE);

}

}

calculating = 0;

}

void ResetScreen(void){

ST7735\_InitR(INITR\_REDTAB);

ST7735\_FillScreen(ST7735\_BLACK);

ST7735\_SetCursor(0,0);

}

void PortF\_Init(void){

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

}

**Timer1.c**

// Timer1.c

// Runs on LM4F120/TM4C123

// Use TIMER1 in 32-bit periodic mode to request interrupts at a periodic rate

// Daniel Valvano

// May 5, 2015

/\* This example accompanies the book

"Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015

Program 7.5, example 7.6

Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu

You may use, edit, run or distribute this file

as long as the above copyright notice remains

THIS SOFTWARE IS PROVIDED "AS IS". NO WARRANTIES, WHETHER EXPRESS, IMPLIED

OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.

VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,

OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.

For more information about my classes, my research, and my books, see

http://users.ece.utexas.edu/~valvano/

\*/

#include <stdint.h>

#include "../inc/tm4c123gh6pm.h"

void (\*PeriodicTask)(void); // user function

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* TIMER1\_Init \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Activate TIMER1 interrupts to run user task periodically

// Inputs: task is a pointer to a user function

// period in units (1/clockfreq)

// Outputs: none

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-1; // 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

}

void Timer1A\_Handler(void){

TIMER1\_ICR\_R = TIMER\_ICR\_TATOCINT;// acknowledge TIMER1A timeout

}