CSE3215 Embedded Systems Laboratory

Lab3 – Reaction Time Measurement

Introduction

Human reaction time is a parameter of interest in many psychological and physiological studies of the effects of drugs, stress, attention etc. In order to measure reaction time an experimenter presents a stimulus to the subject (for example a flashing light or a beep) and measures the time that passes until the subject reacts (for example pushes a button). Tests measuring the reaction time to single audio or visual stimuli can measure the "vigilance" of the central nervous system. More complex reaction test involving different stimuli can measure the speed of the decision making and response selection process.

Objective

The objective of this lab is to provide you with experience with microcontroller peripheral and I/O interfacing. To accomplish this you will design a "complex" reaction timer based on the Dragon12 board. The reaction timer is complex in the sense that although only a single visual stimulus is used the subject must still make a decision in determining the correct response.

Design Specifications

The reaction timer measures and displays the reaction time of a subject's response to a visual stimulus. The reaction timer uses LEDS PB7 and PB0 as the visual stimulus and pushbuttons SW2 and SW5 as the response mechanisms.

1. The reaction timer must measure reaction times up to 9999+/-1ms.

2. Measurements are displayed on the 7-segment display with leading zeros blanked i.e. display " 67" and not "0067".

3. The system starts up in an idle state with the power-up message "P000" displayed on the 7-segment display while waiting for the first reaction time measurement to be initiated.

4. Pushbutton SW3 is used to start a new reaction time measurement. Actuating SW3 blanks the 7-segment display; the measurement itself does not start until the release of SW3. Upon the release of SW3 a random stimulus onset delay (500ms to 3sec) is started. After the stimulus onset delay is complete the stimulus LEDS (PB7 or PB0) are pulsed for 100ms. The system then waits for a press of the response buttons (SW2 or SW5). Following the press of the response buttons the measured reaction time for this trial is displayed on the 7-segment display. If a valid response is not received within 9999ms after the stimulus LEDS are pulsed then the

measurement is discarded and the value “E001” is displayed indicating an error condition. The system then waits for a new reaction time measurement request.

5. The onset delay time must be random and within the range of 500ms to 3sec.

6. The choice of which stimulus LEDS to display for each measurement must be random and either PB7 or PB0.

7. Pushbuttons SW2 and SW5 are used as the subjects response mechanism. Which button(s) used during a measurement is dependent on the stimulus LEDS used for that measurement. Use SW2 to record responses to illumination of LED PB7 and use SW5 to record responses to illumination of LED PB0.

8. The reaction time is defined as the time interval between the initial display of the LEDS PB7 or PB0 to the actuation (falling edge) of the correct pushbuttons, SW2 or SW5.

Pre Lab

1. How many timers are available on the MC9S12DP256B microcontroller?

2. What other feature is incorporated into the MC9S12DP256 timer module that maybe of use in measuring time intervals?

3. Where is the interrupt vector table located in the MC9S12DP256B’ s memory map? What happens to this vector table when using D-Bug12?

4. What is the interrupt vector address for the Real-Time Interrupt (RTI) when using the MC9SDP256B with and without D-Bug12?

5. Develop your reaction timer according to the design specifications in:
   - Assembler using the _CSE3215_ABS_ASM stationery for absolute assembler projects.
   - Assembler for absolute or the _CSE3215_RELOC_ASM stationery for re-locatable assembler projects.
   - **C using the _CSE3215_C_MS stationery. (preferred)**

   A complete design and compiled implementation is required prior to starting the lab.

Hints

It may be possible to generate pseudo random numbers using the timer modules 16-bit main timer register TCNT, you may have to mask certain bits to achieve the necessary results but for the purposes of this lab the level of “randomness” obtained is adequate.

Reference Material

Dragon12 Schematics
Dragon12 Schematics.pdf

Reference Guide For D-Bug12 Version 4.x.x
DB12RG4 April 04, 2005.pdf

MC9S12DP256 Advanced information
MC9S12DP256B.pdf

Procedure

1. Download and debug your program on the Dragon12.

2. Use the oscilloscope and or logic analyzer to help you debug your program and verify your reaction time measurements.

Evaluation

1. Prelab (including preliminary code) 20%
2. Lab demonstration, in-lab explanations and answers, debug and test approach 60%
3. Program/design documentation (code should be well documented but no report is required – submit electronically within 24 hours on Prism) 20%

Demonstrate your program to the T.A showing that it conforms to the design specifications. Part of this demonstration must be a measurement of the reaction time using both the oscilloscope and logic analyzer.

Appendix A: Declaring Interrupt Routines

There are two things one must remember when setting up interrupt service routines (ISR): 1. You must use the *"rti"* return from interrupt instruction not the *"rts"* return from subroutine instruction to return from the ISR. 2. The address of the ISR must be stored in the correct vector address for the interrupt source.

To declare a function in C/C++ as an interrupt service routine (ISR) use the keyword *interrupt* as part of the functions return type as follows:

```
#pragma CODE_SEG __NEAR_SEG NON_BANKED
        interrupt void MyISR( void )
        {
                your code
        }
#pragma CODE_SEG DEFAULT
```

Note: This function has a return type of void and does not accept any parameters. The keyword *interrupt* informs the compiler that this is an ISR and as such the compiler will use the correct *rti* instruction and not the *rts* instruction to return from the ISR.

To place the ISR in the correct vector address edit the isrVectors.c file placing the proto-type for your ISR in the "External ISR Function prototypes" section as follows:

```
/*****************************************************************************
 External ISR Function prototypes
*****************************************************************************/
interrupt void MyISR( void );
```

Then within the *const IsrFunc _vectab[] @0x3e00* structure replace the null 0 with the name of your ISR in the line with the description of the appropriate interrupt source. The @0x3e00 tells the linker to place this structure at address 0x3e00. Below shows how to set the *MyISR* routine in the correct vector address for PORTH.

```
(IsrFunc)0,                     /* Mod Down Cnt Underflow 0xFFCA */
(IsrFunc)MyISR,                 /* PortH Interrupt 0xFFCC       */
(IsrFunc)0,                     /* PortJ Interrupt 0xFFCE       */
```

Note: Do not edit this structure any further. If you do want to make use of an interrupt then replace the name of the ISR with the null 0.

Declaring a sub-routine in Assembler as an (ISR) is the same as that of a subroutine that does not retrieve parameters via the stack. Just remember to return from the subroutine using the “rti” instruction and not the “rts” instruction.

```
MyISR:
        your code
        rti
```

You must manually store the address of your ISR in the proper vector address as part of your code. As an example to set your ISR to service interrupts generated by PORTH one can do the following:

```
movw #MyISR, $3E4C
```

Note: Do not forget to initialize the stack in assembler based applications.