William Zhang
ID: 260975150

Friday, November 11, 2022

ECSE 324 – Lab 2 Report

In lab 2, the basic I/O capabilities of the DE1-SoC computer are introduced. More specifically, the slider switches, pushbuttons, LEDs, 7-segment displays, and timers are used in this lab. To interact with the I/O components, assembly drivers are written. The first task is the implementation of a hex display using four seven segment displays, pushbuttons, switches and LEDs. The second task is the implementation of a stopwatch which counts in increments of 100 milliseconds. The third task is the modification of the stopwatch implemented in the second task so that it uses interrupts.

Task 1: Basic I/O

Problem

The first task of lab 1 consists of implementing an application which uses the last four slider switches to set the value of a number from 0 to 15. When the value is set, one of the four pushbuttons is then pressed and released to indicate which HEX display the number is displayed on. Also, asserting slider switch SW9 clears all the HEX displays. Since there are only four pushbuttons, only four hex displays can display the desired number. As such, the segments of the last two displays (HEX4 and HEX5 should all be turned on during the process. For the HEX displays, three subroutines are implemented, notably HEX_clear_ASM, HEX_flood_ASM, and HEX_write_ASM. For the pushbuttons, seven subroutines are written, and the subroutines for the slider switches and the LEDs are given in the instructions.

Solution

The application can be divided into four steps: using the slider switches as inputs, controlling the LEDs, using the pushbuttons as inputs and using the inputs to determine the behaviour of the HEX displays.

The slider switches parallel port consists of a 10-bit read-only data register which is mapped to address 0xFF200040. The Red LED parallel port contains a 10-bit data register mapped to address 0xFF200000. The pushbuttons, on the other hand, are connected to a parallel port which has 3 4-bit registers. These registers have addresses 0xFF200050, 0xFF200058 and 0xFF20005C. The data register is read-only and provides the values of the switches. The other two registers are the Interruptmask register and the Edgecapture register. The HEX displays, on the other hand, are connected to two parallel ports. Each parallel port comprises a 32-bit write-only register data register. The two parallel ports are mapped to addresses 0xFF200020 and 0xFF200030. The first parallel port contains the data pertaining to HEX0 to HEX3, where each block of 8 bits has 7 bits for the HEX register and one unused bit. The second parallel port contains the data for HEX4 and HEX5.

The first step uses the read_slider_switches_ASM subroutine to read the value from the memory location where the slider switches data is stored and store it in R0. The second step uses the write_LEDs_ASM subroutine to write the value in R0 to the LEDs memory location, thus turning them on or off. The third step consists of using the read_PB_edgecp_ASM and PB_clear_edgecp_ASM subroutines to use the pushbuttons as inputs. Five other subroutines are implemented for the pushbuttons, but they are not used in this task. The read_PB_edgecp_ASM subroutine reads the pushbuttons Edgecapture register and returns the indices of the pushbuttons that have been pressed and then released. It is important to mention that the indices of the pushbuttons are encoded based on a one-hot encoding scheme where PB0 is 0b1, PB1 is 0b10, PB2 is 0b100 and PB3 is 0b1000. The indices are displayed in hexadecimal form. On the other hand, the PB_clear_edgecp_ASM subroutine clears the pushbuttons Edgecapture register by calling the read_PB_edgecp_ASM subroutine and then storing the data in R0 in the Edgecapture register to clear it. The fourth step consists of using the HEX_flood_ASM to turn on all the segments of HEX4 and HEX5. It also uses HEX_write_ASM to write an integer value between 0-15 in the HEX displays indices passed as inputs. The fourth step can also use HEX_clear_ASM to turn off all the segments of the HEX displays passed in the argument.

The application begins by moving the value #0x30 into an argument register. The value corresponds to the indices of HEX4 and HEX5 together. HEX_flood_ASM is then called to turn on all the segments of those displays. The slider switches are then read, and the LEDs are written to in order to light them up according to the switches pressed. Then, read_PB_edgecp_ASM is called to get the indices of the displays that need to be written to. Using the arguments obtained by calling read_slider_switches_ASM and read_PB_edgecp_ASM, HEX_write_ASM is then called to write to the displays.

The Edgecapture register is then cleared. The slider switches are read again, and their value is compared to 0b1000000000 (the index of switch 9) using TST to see if switch 9 is pressed. If that is the case, the clear branch is called. In the clear branch, the index of all the hex displays are stored, then HEX_clear_ASM is called. Finally, b _start is called to loop back.

Improvements

A possible improvement is the separation of the initialization code from the rest of the code. The _start loop initializes the HEX4 and HEX5 displays every time the code loops around which makes the program less efficient. Also, the task 1 code calls disable_PB_INT_ASM, a subroutine which is not needed in this task. Thus, the program is not optimized and makes calls that are not relevant to the task.

Task 2: Timers

Problem

The second task of the lab is to create a stopwatch using the ARM A9 private timer, pushbuttons, and HEX displays. The stopwatch counts in increments of 100 milliseconds using the ARM A9 private timer. The milliseconds are displayed on HEX0, the seconds on HEX1 and HEX 2, the minutes on HEX3 and HEX4, and the hours on HEX5. The pushbuttons are used to start, stop and reset the stopwatch. An endless loop is used to poll the pushbutton edgecapture register and the "F" bit from the ARM A9 private timer interrupt status register.

Solution

The solution is implemented using three new subroutines as well as the subroutines from task 1. The three subroutines are used to control the timers: ARM_TIM_config_ASM, ARM_TIM_read_INT_ASM, and ARM_TIM_clear_INT_ASM. ARM_TIM_config_ASM is used to configure the timer. The timer requires two arguments that are passed in A1 and A2. A1 passes the initial count value of the down counter, and A2 passes the configuration bits. The configuration bits consist of three bits I, A and E, which are stored in the control register. When I is set to 1, a processor interrupt is generated when the timer reaches 0. It is set to 0 as it interrupts are not used in this task. The A bit controls the automation of the timer. If the bit is set to 1, the timer reloads the value in the Load register and continues decrementing. If the bit is set to 0, the timer stops. The E bit controls the timer. If E is set to 1, the timer is started and if E is set to 0, the timer is stopped. When the timer's count value decrements to 0, the "F" bit in the interrupt status register is set to 1. ARM_TIM_read_INT_ASM is used to read the "F" value of the interrupt status register and ARM_TIM_clear_INT_ASM is used to clear the "F" value. It can be cleared by writing 0x1 into the register.

The program starts by moving the value 0x1312D00 into R1. This value corresponds to 20000000. Since the frequency of the timer is 200 MHz, 0.1 seconds corresponds to 20000000/200 MHZ. Then, the registers used for the program are reset to 0 (r3, r5-12) and the value 0b011 is moved into r2. This value corresponds to the timer configuration bits. ARM_TIM_config_ASM is then called and the program branches into the timerloop branch. In the timerloop, the program checks the timer values to see if any of them are at the highest possible value (60 minutes and 60 seconds, 0xA milliseconds) and resets their values if that is the case. The program then writes the values into the respective displays.

The next branch in the program is checkloop, which reads the edgecapture register. If both the stop and start pushbuttons are released, the program clears the edgecapture register and branches back to the timerloop. If only the start pushbutton is released, nothing happens and the edgecapture register is cleared. If the reset pushbutton is pressed, _start is called to reset the timer. If the stop pushbutton is pressed, the program branches back to checkloop. If no pushbuttons are pressed but the "F" bit of the interrupt status register is 1, the counter is incremented and the "F" bit is cleared. The program then branches to timerloop.

Improvements

An improvement that could be made to the program includes fixing the HEX displays. Right now, the displays are refreshed and written to each time the program passes through the timerloop. Instead, the program could be writing to the displays if a change happens or if a display value is maximized. For example, the program compares the values of each register to the maximum value of that register and changes it if necessary. After going through all the registers associated with displays, the program then writes to all the registers. Instead, when comparing, if the comparison is successful, the register would then directly write to the displays.

Task 3: Interrupts

Problem

The third task of the lab is to modify the stopwatch application from task 2 to use interrupts. The two interrupts that are to be implemented are the one for the ARM A9 private timer and the one for the pushbuttons.

Solution

The solution is implemented by modifying the _start branch and by adding new branches to the program to implement the interrupts. The _start branch activates the interrupts for the pushbuttons and the private timer by calling the enable_PB_INT_ASM and ARM_TIM_config_ASM subroutines. The IDLE branch implements the stopwatch function. The SERVICE_IRQ branch reads the ICCIAR from the CPU interface and checks which interrupt has occurred, then calls the corresponding ISR. If the ID is not recognized, the program branches to UNEXPECTED. The CONFIG_GIC subroutine configures the two interrupts by passing their IDs to r0 and calling CONFIG_INTERRUPT. CONFIG_INTERRUPT configures the registers in the GIC for an individual interrupt. The KEY_ISR interrupt service routine for the pushbuttons writes the content of the pushbuttons edgecapture register in the PB_int_flag memory and clears the interrupts. The ARM_TIM_ISR subroutine writes the value “1” into the tim_int_flag memory when an interrupt is received and clears the interrupt.

The program starts by setting up the stack pointers for IRQ and SVC processor modes. If then configures the ARM A9 timer and enables the IRQ interrupts in the processor. The program then branches into IDLE. The IDLE branch first loads the value of the pushbuttons interrupt register and compares it to certain values to stop, start or reset the timer. The program then checks if there is a timer interrupt and increments the counter by 1 if there is one. The program then checks the value of each display to see if they are at their maximum possible value and resets their value if that is the case. Finally, the values are written back to the displays and the IDLE program branches to itself.

Improvements

My code does not function properly as there is a clobbered register in the program. However, the timer can function if that clobbered register error is ignored. However, the program does not make appropriate use of the KEY_ISR and ARM_TIM_ISR subroutines as the memory address contents are stored in a register and compared in the program before those two subroutines are called. Thus, those subroutines only clear the interrupt bits. The subroutines should be called at the start of the IDLE branch and the ldr/str instructions that do the subroutine’s work should be removed. The same issue that occurs in task 2 can also be fixed in this task as the code was only modified to use interrupts.