Interrupts

Interrupts for UART input via the keyboard is enabled prewritten files provided by texas instruments.

In this project the interrupt status is obtained from the predefined function “ROM\_UARTIntStatus(UART0\_BASE, true)” line 449 of pong.c , the function is defined in the “rom.h” file. Where the function wait for UART0\_BASE (address 0x4000C000), defined in hw\_memmap.h line 61 to contain data, that is a character from the keyboard.

With this design the system is interrupt handler stores any character that is passed from the keyboard. In order to deal with the interrupt, after an interrupt has occurred it is cleared with the “ROM\_UARTIntClear” function defined in the rom.h file, line 6681.

The function “ROM\_UARTCharsAvail(UART0\_BASE)” is true if there is any data in the receive FIFO. These values are put into the declared variable “value” in pong.c. If the charatcers match any of the characters in the switch statement the “g\_pX\_move\_dir”, where X is the 1 or 2 for the paddles, is assigned a constant value to determine the direction of the ball on rebound from the paddles.

Enabling the clocks for UART0 using sysctl\_RCGCUART and the GPIO port clock using sysctl\_RCGCGPIO in this project has been implemented through:

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); // line 579 in pong.c

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); // line 580 in pong.c

This implementation of using interrupts does not show the asm implementation, which usually first of all disables all system interrupts (CPSID I) in the main program before Initialisng GPIO and UART peripherals using system control registers. Where register values are saved to the stack. Interrupt types edge or level trigger as wel as high/low for edge and rising/falling or both are set.

Interrupts in asm would then be enabled for Port A, by enabling the clock for GPIOA using the SYSCTL\_RCGCGPIO register, then enabling alternate function for pins 0 and 1 using the base address of GPIO A and GPIO\_AFSEL register offset and logical ORRing bits 0 and 1 for UART0 and then selecting the alternate function for, that is the UART using GPIO\_PCTL offsetand assigning the he value for pins 0 and 1.

In this project this has been implemented as:

GPIOPinConfigure(GPIO\_PA0\_U0RX); // line 585 in pong.c

GPIOPinConfigure(GPIO\_PA1\_U0TX); // line 586 in pong.c

ROM\_GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); // line 587 in pong.c

Finally enabling digital access for input and output pins using the digital enable register offset GPIO\_DEN.

In this project this has beesn implemented through:

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0) // Line 579 in pong.c

ROM\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA) // Line 580 in pong.c

In this project the enable all interrupts CPSIE is done before configuring Port A as UART and UART interrupts:

ROM\_IntMasterEnable(); // line 583 in pong.c

In the C implementation of interrupts using the functions:

ROM\_IntEnable(INT\_UART0);

ROM\_UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT);

There is no explicit renaming of the associated interrupt vector table name at the associated position as done in ASM with vector tables. In C setting the UART interrupts using the provided functions associates the UART0 interrupt to the associated interrupt number.