Appendix B: Example C Program

```
void disable_A9_interrupts (void);
void set_A9_IRQ_stack (void);
void config_GIC (void);
void config_KEYs (void);
void enable_A9_interrupts (void);

/*****************************************************************************
 * This program demonstrates use of interrupts with C code.  The program
 * responds to interrupts from the pushbutton KEY port in the FPGA.
 *
 * The interrupt service routine for the KEYs indicates which KEY has
 * been pressed on the display HEX0.
 ****************************************************************************/
int main(void)
{
   disable_A9_interrupts ();   // disable interrupts in the A9 processor
   set_A9_IRQ_stack ();        // initialize the stack pointer for IRQ mode
   config_GIC ();              // configure the general interrupt controller
   config_KEYs ();             // configure KEYs to generate interrupts

   enable_A9_interrupts ();    // enable interrupts in the A9 processor

   while (1)                   // wait for an interrupt
      ;
}

/* setup the KEY interrupts in the FPGA */
void config_KEYs()
{
   volatile int * KEY_ptr = (int *) 0xFF200050;   // KEY base address

   *(KEY_ptr + 2) = 0xF;    // enable interrupts for all four KEYs
}
```

```
/* This file:
 * 1. defines exception vectors for the A9 processor
 * 2. provides code that sets the IRQ mode stack, and that dis/enables
 *    interrupts
 * 3. provides code that initializes the generic interrupt controller
*/
void pushbutton_ISR (void);
void config_interrupt (int, int);

// Define the IRQ exception handler
void __attribute__ ((interrupt)) __cs3_isr_irq (void)
{
   // Read the ICCIAR from the CPU Interface in the GIC
   int interrupt_ID = *((int *) 0xFFFEC10C);

   if (interrupt_ID == 73)           // check if interrupt is from the KEYs
      pushbutton_ISR ();
   else
      while (1);                     // if unexpected, then stay here

   // Write to the End of Interrupt Register (ICCEOIR)
   *((int *) 0xFFFEC110) = interrupt_ID;
}

// Define the remaining exception handlers
void __attribute__ ((interrupt)) __cs3_reset (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_undef (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_swi (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_pabort (void)
{
    while(1);
}
```

```
void __attribute__ ((interrupt)) __cs3_isr_dabort (void)
{
    while(1);
}

void __attribute__ ((interrupt)) __cs3_isr_fiq (void)
{
    while(1);
}

/*
 * Turn off interrupts in the ARM processor
*/
void disable_A9_interrupts(void)
{
   int status = 0b11010011;
   asm("msr cpsr, %[ps]" : : [ps]"r"(status));
}

/*
 * Initialize the banked stack pointer register for IRQ mode
*/
void set_A9_IRQ_stack(void)
{
   int stack, mode;
   stack = 0xFFFFFFFF - 7;      // top of A9 onchip memory, aligned to 8 bytes
   /* change processor to IRQ mode with interrupts disabled */
   mode = 0b11010010;
   asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
   /* set banked stack pointer */
   asm("mov sp, %[ps]" : : [ps] "r" (stack));

   /* go back to SVC mode before executing subroutine return! */
   mode = 0b11010011;
   asm("msr cpsr, %[ps]" : : [ps] "r" (mode));
}

/*
 * Turn on interrupts in the ARM processor
*/
void enable_A9_interrupts(void)
{
   int status = 0b01010011;
   asm("msr cpsr, %[ps]" : : [ps]"r"(status));
}
```

```
/*
 * Configure the Generic Interrupt Controller (GIC)
*/
void config_GIC(void)
{
   config_interrupt (73, 1);    // configure the FPGA KEYs interrupt (73)

   // Set Interrupt Priority Mask Register (ICCPMR). Enable all priorities
   *((int *) 0xFFFEC104) = 0xFFFF;

   // Set the enable in the CPU Interface Control Register (ICCICR)
   *((int *) 0xFFFEC100) = 1;

   // Set the enable in the Distributor Control Register (ICDDCR)
   *((int *) 0xFFFED000) = 1;
}

/*
 * Configure registers in the GIC for an individual Interrupt ID.  We
 * configure only the Interrupt Set Enable Registers (ICDISERn) and
 * Interrupt Processor Target Registers (ICDIPTRn). The default (reset)
 * values are used for other registers in the GIC
*/
void config_interrupt (int N, int CPU_target)
{
   int reg_offset, index, value, address;

   /* Configure the Interrupt Set-Enable Registers (ICDISERn).
    * reg_offset = (integer_div(N / 32) * 4; value = 1 << (N mod 32) */
   reg_offset = (N >> 3) & 0xFFFFFFFC;
   index = N & 0x1F;
   value = 0x1 << index;
   address = 0xFFFED100 + reg_offset;
   /* Using the address and value, set the appropriate bit */
   *(int *)address |= value;

   /* Configure the Interrupt Processor Targets Register (ICDIPTRn)
    * reg_offset = integer_div(N / 4) * 4; index = N mod 4 */
   reg_offset = (N & 0xFFFFFFFC);
   index = N & 0x3;
   address = 0xFFFED800 + reg_offset + index;
   /* Using the address and value, write to (only) the appropriate byte */
   *(char *)address = (char) CPU_target;
}
```

```
/*****************************************************************
 * Pushbutton - Interrupt Service Routine
 *
 * This routine checks which KEY has been pressed. It writes to HEX0
 *****************************************************************/
void pushbutton_ISR( void )
{
   /* KEY base address */
   volatile int *KEY_ptr = (int *) 0xFF200050;
   /* HEX display base address */
   volatile int *HEX3_HEX0_ptr = (int *) 0xFF200020;
   int press, HEX_bits;

   press = *(KEY_ptr + 3);      // read the pushbutton interrupt register
   *(KEY_ptr + 3) = press;      // Clear the interrupt

   if (press & 0x1)             // KEY0
      HEX_bits = 0b00111111;
   else if (press & 0x2)        // KEY1
      HEX_bits = 0b00000110;
   else if (press & 0x4)        // KEY2
      HEX_bits = 0b01011011;
   else                         // press & 0x8, which is KEY3
      HEX_bits = 0b01001111;

   *HEX3_HEX0_ptr = HEX_bits;
   return;
}
```