EXPERIMENT 8 - INTERRUPT PROGRAMMING USING MSP432

LAB CODE AND TITLE - 19CCE283 EMBEDDED COMPUTING LAB
EXPERIMENT NUMBER - 8
DATE - 14/06/2022

* AIMS:
Design and develop an embedded computing platform to perform GPIO port and UART serial port interrupt programming using an MSP432 microcontroller.

A) GPIO port Interrupt Programming:
1. Disable global IRQs in the main function.
2. Configure functionality of P1.1 and P1.4 as simple I/O and direction as input for switch.
3. Enable P1.1 and P1.4 pull resistor; Pull up/down is selected by P1→ OUT.
4. Make interrupt trigger on high-to-low transition and clear pending interrupt flags. Enable interrupt from P1.1 and P1.4.
5. Configure functionality of P2.0 – P2.2 as simple I/O and direction as output for tri-color LEDs. Turn all three LEDs OFF.
6. Set priority to 3 in NVIC, enable interrupt in NVIC and global enable IRQs. Toggle the red LED (2.0) continuously.
7. Switch 1 is connected to P1.1 pin and switch 2 is connected to P1.4 pin. Both of them trigger PORT1 interrupt. Toggle green LED (P2.1) three times. clear the interrupt flag before return.
8. Delay n milliseconds (3 MHz CPU clock). Do nothing for 1 ms.

→ C PROGRAM CODE:

```
/* www.microDigitalEd.com
 * P6_1 : Toggle red LED on P2.0 continuously. Upon pressing either
 * SW1 or SW2, the green LED of P2.1 should toggle for three times.
 * main program toggles red LED while waiting for interrupt for
 * SW1 or SW2. When green LED is blinking, the red LED ceases to
 * blink.
```

```
* Tested with Keil 5.20 and MSP432 Device Family Pack V2.2.0
* on XMS432P401R Rev C.
*/

#include "msp.h"

void delayMs(int n);

int main(void) {
    __disable_irq();  /* global disable IRQs */

    /* configure P1.1, P1.4 for switch inputs */
    P1->SEL1 &= ~0x12; /* configure P1.1, P1.4 as simple I/O */
    P1->SEL0 &= ~0x12;
    P1->DIR &= ~0x12;  /* P1.1, P1.4 set as input */
    P1->REN |= 0x12;   /* P1.1, P1.4 pull resistor enabled */
    P1->OUT |= 0x12;   /* Pull up/down is selected by P1->OUT */
    P1->IES |= 0x12;   /* make interrupt trigger on high-to-low
                          transition */
    P1->IFG = 0;       /* clear pending interrupt flags */
    P1->IE |= 0x12;    /* enable interrupt from P1.1, P1.4 */

    /* configure P2.2-P2.0 for tri-color LEDs */
    P2->SEL1 &= ~7;  /* configure P2.2-P2.0 as simple I/O */
    P2->SEL0 &= ~7;

    P2->DIR |= 7;  /* P2.2-2.0 set as output */
    P2->OUT &= ~7; /* turn all three LEDs off */

    NVIC_SetPriority(PORT1_IRQn, 3); /* set priority to 3 in NVIC */
    NVIC_EnableIRQ(PORT1_IRQn);      /* enable interrupt in NVIC */
    __enable_irq();                  /* global enable IRQs */
```

```
    /* toggle the red LED (P2.0) continuously */
    while (1) {
      P2->OUT |= 0x01;
      delayMs(500);
      P2->OUT &= ~0x01;
      delayMs(500);
    }
}

/* SW1 is connected to P1.1 pin, SW2 is connected to P1.4
 * Both of them trigger PORT1 interrupt */
void PORT1_IRQHandler(void) {
    int i;
    volatile int readback;
    /* toggle green LED (P2.1) three times */
    for (i=0; i<3; i++) {
      P2->OUT |= 0x02;
      delayMs(500);
      P2->OUT &= ~0x02;
      delayMs(500);
    }
    P1->IFG &= ~0x12; /* clear the interrupt flag before return */
}

/* delay n milliseconds (3 MHz CPU clock) */
void delayMs(int n) {
    int i, j;

    for (j=0; j<n; j++)
      for (i=750; i>0; i--); /* do nothing for 1 ms */
}
```

b) UART Serial Port Interrupt Programming :

I. Main Function -

① Disable global IRQs and call UART0 initialization function.
② Configure P2.2 - P2.0 as simple I/O and set direction as output for tri-color LEDs.
③ Set priority to 4 in NVIC, enable interrupt in NVIC and global enable IRQs.
④ Declare an empty infinitely running while loop.

II. Continuous Transmission of a character Using UART -

① Put in reset mode and disable oversampling.
② Set 8-bits data - No Parity - 1 stop bit for transmission. In asynchronous mode, first LSB and then MSB should be set, followed by SMCLK (00).
③ Enabled EUSCI_A0 logic is held in reset state (81).
④ Configure functionality of P1.2, P1.3 as UART pins.
⑤ Take UART out of reset mode.
⑥ Enable receive interrupt.

III. Reception of a Character Using UART -

① Read the received character and set the LEDs.
② Interrupt flag is cleared by reading RXBUF.

→ C PROGRAM CODE :-

```
/* www.microDigitalEd.com
 * P6_3.c UART0 Receive Interrupt Handling
 *
 * This program modifies p4_2.c to use interrupt to handle the UART0
 * receive. It receives any key from terminal emulator (TeraTerm) of
 * the host PC to the UART0 on MSP432 LaunchPad. The UART0 is
 * connected to the XDS110 debug interface on the LaunchPad and it
 * has a virtual connection to the host PC COM port.
```

```
* Launch a terminal emulator (TeraTerm) on a PC and hit any key.
* The tri-color LEDs are turned on or off according to the key
* received.
*
* By default the subsystem master clock is 3MHz.
* Setting EUSCI_A0 -> BRW = 26 with oversampling disabled yields
* 115200 Baud.
*
* Tested with Keil 5.20 and MSP432 Device Family Pack V2.2.0
* on XMS432P401R Rev C.
*/
#include "msp.h"

void UART0_init(void);

int main(void) {
  __disable_irq();

  UART0_init();

  /* initialize P2.2-P2.0 for tri-color LEDs */
  P2->SEL1 &= ~7;  /* configure P2.2-P2.0 as simple I/O */
  P2->SEL0 &= ~7;
  P2->DIR |= 7;    /* P2.2-P2.0 set as output */

  NVIC_SetPriority(EUSCIA0_IRQn, 4); /* set priority to 4 in NVIC */
  NVIC_EnableIRQ(EUSCIA0_IRQn);      /* enable interrupt in NVIC */
  __enable_irq();                    /* global enable IRQs */

  while(1) {
  }
}
```

```
void UART0_init(void) {
    EUSCI_A0 -> CTLW0 |= 1;        /* put in reset mode for config */
    EUSCI_A0 -> MCTLW = 0;         /* disable oversampling */
    EUSCI_A0 -> CTLW0 = 0x0081;    /* 1 stop bit, no parity, SMCLK,
                                      8-bit data */
    EUSCI_A0 -> BRW = 26;          /* 3000000/115200 = 26 */
    P1 -> SEL0 |= 0x0C;            /* P1.3, P1.2 for UART */
    P1 -> SEL1 &= ~0x0C;
    EUSCI_A0 -> CTLW0 &= ~1;       /* take UART out of reset mode */
    EUSCI_A0 -> IE |= 1;           /* enable receive interrupt */
}

void EUSCIA0_IRQHandler(void) {
    P2 -> OUT = EUSCI_A0 -> RXBUF; /* Read the receive char and
                                      set the LEDs */
                                   /* Interrupt flag is cleared by
                                      reading RXBUF */
}
```

4) GPIO - UART SERIAL PORT INTERRUPT PROGRAMMING :

I. Main Function -

① Disable global IRQs and call UART0 initialization function.
② Configure P2.2 - P2.0 as simple I/O and set direction as output for tri-color LEDs.
③ Set priority to 4 in NVIC, enable interrupt in NVIC and global enable IRQs.
④ Declare an empty infinitely running while loop.

II. Continuous Transmission of a character using UART -

① Put in reset mode and disable oversampling.
② Set 8-bits data - no Parity - 1 stop bit for transmission. In asynchronous mode, first LSB and then MSB should be set, followed by SMCLK (00).

③ Enabled EUSCI_A0 logic is held in reset state (SI). Configure functionality of P1.2, P1.3 as UART pins.
④ Take UART out of reset mode. Enable receive interrupt.

D. Reception of a character using UART -
① read the received character and set the LEDs.
② Interrupt flag is cleared by reading RXBUF.
③ Toggle green LED (P2.1) two times, if input = aaa; else,
④ Toggle red LED (P2.0) three times, if input = bbbbb; else
⑤ Toggle the red and green LED continuously.

→ C PROGRAM CODE -

```
#include "msp.h"
#include <string.h>

void UART0_init(void);
void delayMs(int n);
char ch, str[5], case1[3] = "aaa", case2[5] = "bbbbb";
int i=0;

int main(void) {
    _disable_irq();

    UART0_init();

    /* initialize P2.2 - P2.0 for tri-color LEDs */
    P2->SEL1 &= ~7; /* configure P2.2-P2.0 as simple I/O */
    P2->SEL0 &= ~7;
    P2->DIR |= 7;  /* P2.2-2.0 set as output */

    NVIC_SetPriority(EUSCIA0_IRQn, 4); /* set priority to 4 in NVIC */
    NVIC_EnableIRQ(EUSCIA0_IRQn);      /* enable interrupt in NVIC */
    _enable_irq();                     /* global enable IRQs */
```

```
    while (1) {
    }
}

void UART0_init (void) {
    EUSCI_A0 -> CTLW0 |= 1;          /* put in reset mode for config */
    EUSCI_A0 -> MCTLW = 0;           /* disable oversampling */
    EUSCI_A0 -> CTLW0 = 0x0081;      /* 1 stop bit, no parity, SMCLK, 8-bit data */
    EUSCI_A0 -> BRW = 26;            /* 3000000/115200 = 26 */
    P1 -> SEL0 |= 0x0C;              /* P1.3, P1.2 for UART */
    P1 -> SEL1 &= ~0x0C;
    EUSCI_A0 -> CTLW0 &= ~1;         /* take UART out of reset mode */
    EUSCI_A0 -> IE |= 1;             /* enable receive interrupt
}

void EUSCIA0_IRQHandler (void) {
    while (ch != '\n')
    {
      ch = EUSCI_A0 -> RXBUF;  /* read the receive char and set the LEDs */
                               /* interrupt flag is cleared by reading RXBUF */
      str[i] = ch;
      i++;
    }

    if (strcmp (str, case1) == 0) {
     /* toggle green LED (P2.1) two times */
     for (i=0; i<2; i++)
     {
       P2 -> OUT |= 0x02;
       delayms (500);
       P2 -> OUT &= ~0x02;
       delayms (500);
     }
}
```

```
else if (strcmp (str, case2) == 0) {
  /* toggle red LED (P2.0) three times */
  for (i=0; i<3; i++)
  {
    P2->OUT |= 0x01;
    delayMs (500);
    P2->OUT &= ~0x01;
    delayMs (500);
  }
}

else {
  /* toggle the red LED (P2.0) and green LED (P2.1) continuously */
  P2->OUT |= 0x01;
  delayMs (500);
  P2->OUT &= ~0x01;
  delayMs (500);

  P2->OUT |= 0x02;
  delayMs (500);
  P2->OUT &= ~0x02;
  delayMs (500);
}
}

/* delay n milliseconds (3 MHz CPU clock) */
void delayMs (int n) {
  int i, j;

  for (j=0; j<n; j++)
    for (i=150; i>0; i--); /* do nothing for 1 ms */
}
```

→ CIRCUIT DIAGRAM :-

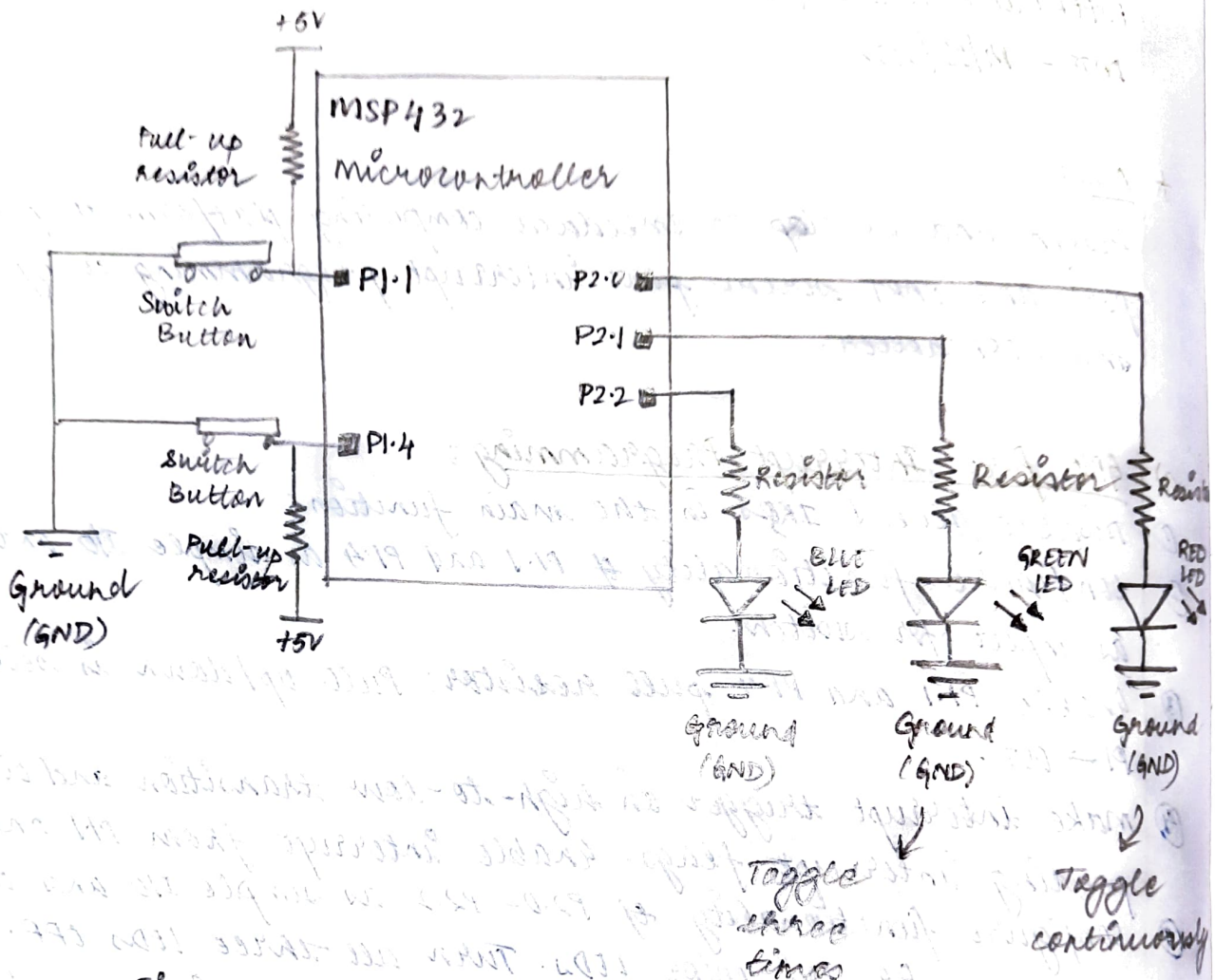

Figure 1 - GPIO Port Interrupt Programming

→ CIRCUIT DIAGRAM :-

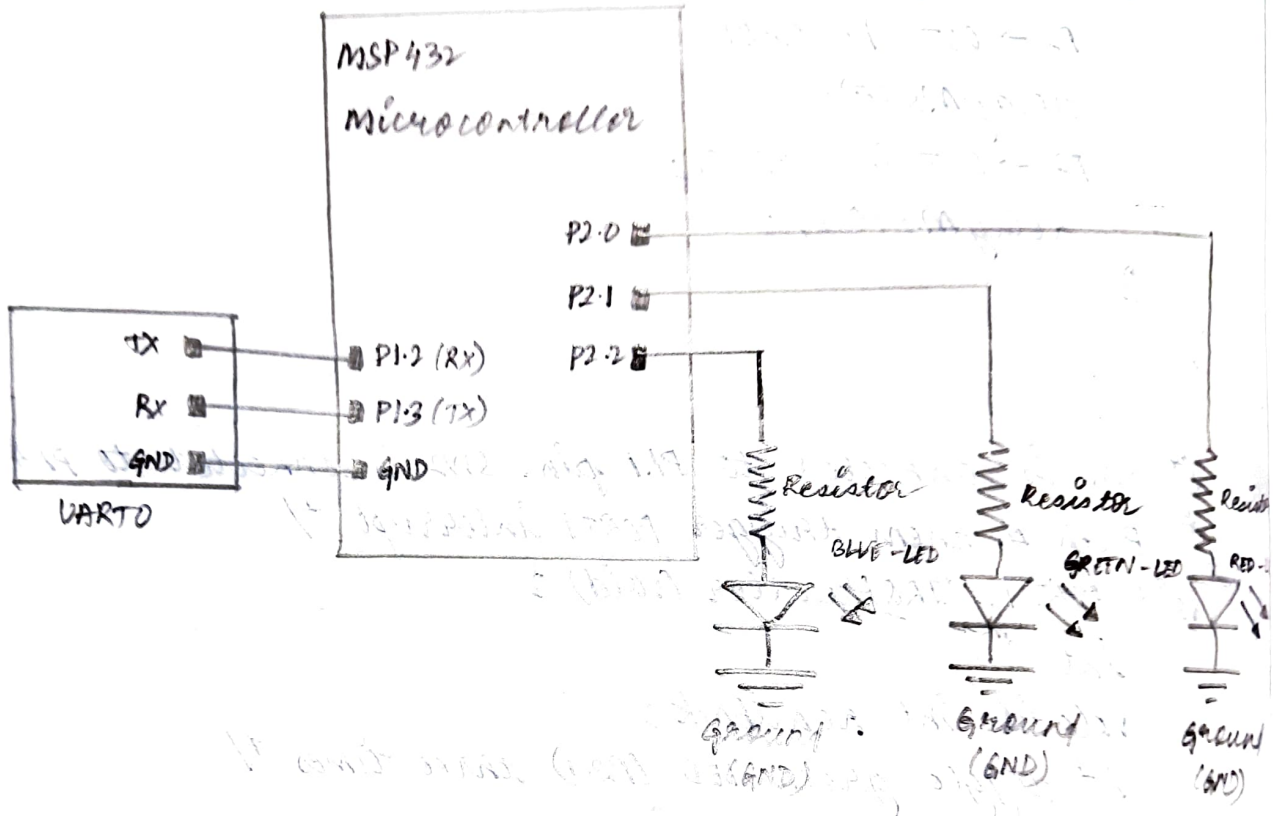

Figure 2- UART Serial Port Interrupt Programming

→ CIRCUIT DIAGRAM :-

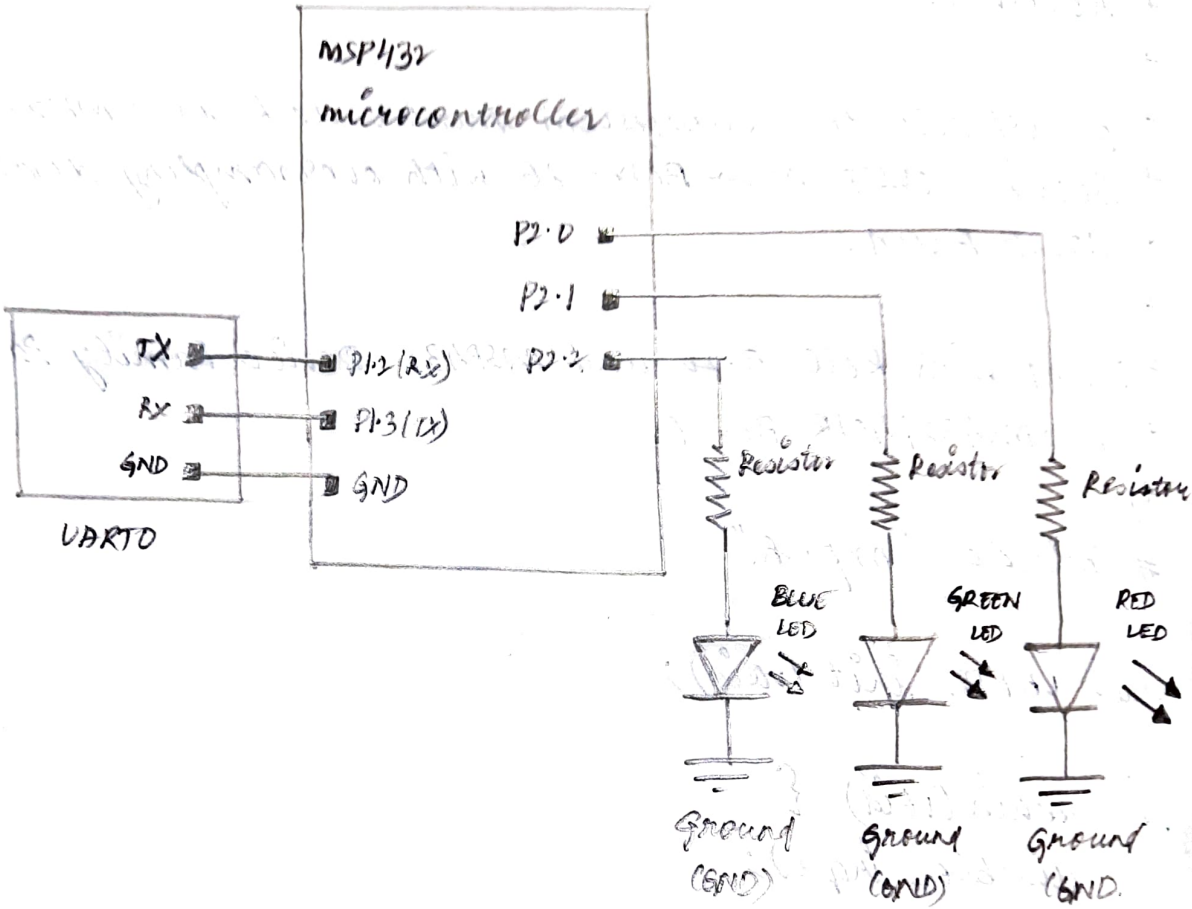

Figure 3 - SPI0- UART Serial Port Interrupt Programming

* RESULT:

Thus, designed and developed an embedded computing platform to perform GPIO port and UART serial port interrupt programming using an MSP430 microcontroller. All simulation results were verified successfully.