|  |
| --- |
| Original Source Code  So basically, I borrowed the code from the MSP launchpad files, I only edited the main function within the second “program” because the rest of the button controllers were quite confusing to figure out in short time.  I created two different versions from the following source code. One that just changed the lights from toggling between green and red, to flashing both at the same time.  This is the original source code used to develop the other main function within the second project, the first version I just messed with one of the lines within the preApplication() function to make the lights flash just red and green. I was having quite a difficult time trying to create my own new project. I need to mess around more.  This code below is the entire program. It will toggle the LEDS between green and Red, on p1.3 press the buttons will shut off. On reset button press the will “reset”.  P1.0 is the RED LED  P1.6 is the Green LED |
| /\*  \* main.c  \*  \* MSP-EXP430G2-LaunchPad User Experience Application  \*  \* Copyright (C) 2011 Texas Instruments Incorporated - http://www.ti.com/  \*  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the  \* distribution.  \*  \* Neither the name of Texas Instruments Incorporated nor the names of  \* its contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS  \* "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT  \* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR  \* A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT  \* OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,  \* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT  \* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  \* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY  \* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  \* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE  \* OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*  \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* MSP-EXP430G2-LaunchPad User Experience Application  \*  \* 1. Device starts up in LPM3 + blinking LED to indicate device is alive  \* + Upon first button press, device transitions to application mode  \* 2. Application Mode  \* + Continuously sample ADC Temp Sensor channel, compare result against  \* initial value  \* + Set PWM based on measured ADC offset: Red LED for positive offset, Green  \* LED for negative offset  \* + Transmit temperature value via TimerA UART to PC  \* + Button Press --> Calibrate using current temperature  \* Send character '°' via UART, notifying PC  \*  \* Changes:  \*  \* 1.2 + Updated register naming conventions to reflect latest standard by TI  \* e.g.: CCR0 --> TACCR0, CCTL0 --> TACCTL0  \* + Changed method to capture TAR value into TACCR0 by using capture a  \* SW-triggered event. [Changing TACCR input from GND to VCC]  \* 1.1 + LED1 & LED2 labels changed so that Green LED(LED2) indicates sampled  \* temperature colder than calibrated temperature and vice versa  \* with Red LED (LED1).  \* + Turn off peripheral function of TXD after transmitting byte to  \* eliminate the extra glitch at the end of UART transmission  \* 1.0 Initial Release Version  \*  \* Texas Instruments, Inc.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **#include** "msp430g2553.h"  **#define** LED1 BIT0  **#define** LED2 BIT6  **#define** LED\_DIR P1DIR  **#define** LED\_OUT P1OUT  **#define** BUTTON BIT3  **#define** BUTTON\_OUT P1OUT  **#define** BUTTON\_DIR P1DIR  **#define** BUTTON\_IN P1IN  **#define** BUTTON\_IE P1IE  **#define** BUTTON\_IES P1IES  **#define** BUTTON\_IFG P1IFG  **#define** BUTTON\_REN P1REN  **#define** TXD BIT1 // TXD on P1.1  **#define** RXD BIT2 // RXD on P1.2  **#define** APP\_STANDBY\_MODE 0  **#define** APP\_APPLICATION\_MODE 1  **#define** TIMER\_PWM\_MODE 0  **#define** TIMER\_UART\_MODE 1  **#define** TIMER\_PWM\_PERIOD 2000  **#define** TIMER\_PWM\_OFFSET 20  **#define** TEMP\_SAME 0  **#define** TEMP\_HOT 1  **#define** TEMP\_COLD 2  **#define** TEMP\_THRESHOLD 5  // Conditions for 9600/4=2400 Baud SW UART, SMCLK = 1MHz  **#define** Bitime\_5 0x05\*4 // ~ 0.5 bit length + small adjustment  **#define** Bitime 13\*4//0x0D  **#define** UART\_UPDATE\_INTERVAL 1000  **unsigned** **char** BitCnt;  **unsigned** **char** applicationMode = APP\_STANDBY\_MODE;  **unsigned** **char** timerMode = TIMER\_PWM\_MODE;  **unsigned** **char** tempMode;  **unsigned** **char** calibrateUpdate = 0;  **unsigned** **char** tempPolarity = TEMP\_SAME;  **unsigned** **int** TXByte;  /\* Using an 8-value moving average filter on sampled ADC values \*/  **long** tempMeasured[8];  **unsigned** **char** tempMeasuredPosition=0;  **long** tempAverage;  **long** tempCalibrated, tempDifference;  **void** **InitializeLeds**(**void**);  **void** **InitializeButton**(**void**);  **void** **PreApplicationMode**(**void**); // Blinks LED, waits for button press  **void** **ConfigureAdcTempSensor**(**void**);  **void** **ConfigureTimerPwm**(**void**);  **void** **ConfigureTimerUart**(**void**);  **void** **Transmit**(**void**);  **void** **InitializeClocks**(**void**);  **void** **main**(**void**)  {  **unsigned** **int** uartUpdateTimer = UART\_UPDATE\_INTERVAL;  **unsigned** **char** i;  WDTCTL = WDTPW + WDTHOLD; // Stop WDT  InitializeClocks();  InitializeButton();  InitializeLeds();  PreApplicationMode(); // Blinks LEDs, waits for button press  /\* Application Mode begins \*/  applicationMode = APP\_APPLICATION\_MODE;  ConfigureAdcTempSensor();  ConfigureTimerPwm();  **\_\_enable\_interrupt**(); // Enable interrupts.  /\* Main Application Loop \*/  **while**(1)  {  ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start  **\_\_bis\_SR\_register**(CPUOFF + GIE); // LPM0 with interrupts enabled  /\* Moving average filter out of 8 values to somewhat stabilize sampled ADC \*/  tempMeasured[tempMeasuredPosition++] = ADC10MEM;  **if** (tempMeasuredPosition == 8)  tempMeasuredPosition = 0;  tempAverage = 0;  **for** (i = 0; i < 8; i++)  tempAverage += tempMeasured[i];  tempAverage >>= 3; // Divide by 8 to get average  **if** ((--uartUpdateTimer == 0) || calibrateUpdate )  {  ConfigureTimerUart();  **if** (calibrateUpdate)  {  TXByte = 248; // A character with high value, outside of temp range  Transmit();  calibrateUpdate = 0;  }  TXByte = (**unsigned** **char**)( ((tempAverage - 630) \* 761) / 1024 );  Transmit();  uartUpdateTimer = UART\_UPDATE\_INTERVAL;  ConfigureTimerPwm();  }  tempDifference = tempAverage - tempCalibrated;  **if** (tempDifference < -TEMP\_THRESHOLD)  {  tempDifference = -tempDifference;  tempPolarity = TEMP\_COLD;  LED\_OUT &= ~ LED1;  }  **else**  **if** (tempDifference > TEMP\_THRESHOLD)  {  tempPolarity = TEMP\_HOT;  LED\_OUT &= ~ LED2;  }  **else**  {  tempPolarity = TEMP\_SAME;  TACCTL0 &= ~CCIE;  TACCTL1 &= ~CCIE;  LED\_OUT &= ~(LED1 + LED2);  }  **if** (tempPolarity != TEMP\_SAME)  {  tempDifference <<= 3;  tempDifference += TIMER\_PWM\_OFFSET;  TACCR1 = ( (tempDifference) < (TIMER\_PWM\_PERIOD-1) ? (tempDifference) : (TIMER\_PWM\_PERIOD-1) );  TACCTL0 |= CCIE;  TACCTL1 |= CCIE;  }  }  }  **void** **PreApplicationMode**(**void**)  {  LED\_DIR |= LED1 + LED2;  LED\_OUT |= LED1; // To enable the LED toggling effect  LED\_OUT &= ~LED2;  BCSCTL1 |= DIVA\_1; // ACLK/2  BCSCTL3 |= LFXT1S\_2; // ACLK = VLO  TACCR0 = 1200; //  TACTL = TASSEL\_1 | MC\_1; // TACLK = SMCLK, Up mode.  TACCTL1 = CCIE + OUTMOD\_3; // TACCTL1 Capture Compare  TACCR1 = 600;  **\_\_bis\_SR\_register**(LPM3\_bits + GIE); // LPM0 with interrupts enabled  }  **void** **ConfigureAdcTempSensor**(**void**)  {  **unsigned** **char** i;  /\* Configure ADC Temp Sensor Channel \*/  ADC10CTL1 = INCH\_10 + ADC10DIV\_3; // Temp Sensor ADC10CLK/4  ADC10CTL0 = SREF\_1 + ADC10SHT\_3 + REFON + ADC10ON + ADC10IE;  **\_\_delay\_cycles**(1000); // Wait for ADC Ref to settle  ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start  **\_\_bis\_SR\_register**(CPUOFF + GIE); // LPM0 with interrupts enabled  tempCalibrated = ADC10MEM;  **for** (i=0; i < 8; i++)  tempMeasured[i] = tempCalibrated;  tempAverage = tempCalibrated;  }  **void** **ConfigureTimerPwm**(**void**)  {  timerMode = TIMER\_PWM\_MODE;  TACCR0 = TIMER\_PWM\_PERIOD; //  TACTL = TASSEL\_2 | MC\_1; // TACLK = SMCLK, Up mode.  TACCTL0 = CCIE;  TACCTL1 = CCIE + OUTMOD\_3; // TACCTL1 Capture Compare  TACCR1 = 1;  }  **void** **ConfigureTimerUart**(**void**)  {  timerMode = TIMER\_UART\_MODE; // Configure TimerA0 UART TX  TACCTL0 = OUT; // TXD Idle as Mark  TACTL = TASSEL\_2 + MC\_2 + ID\_3; // SMCLK/8, continuous mode  P1SEL |= TXD + RXD; //  P1DIR |= TXD; //  }  // Function Transmits Character from TXByte  **void** **Transmit**()  {  BitCnt = 0xA; // Load Bit counter, 8data + ST/SP  /\* Simulate a timer capture event to obtain the value of TAR into the TACCR0 register \*/  TACCTL0 = CM\_1 + CCIS\_2 + SCS + CAP + OUTMOD0; //capture on rising edge, initially set to GND as input // clear CCIFG flag  TACCTL0 |= CCIS\_3; //change input to Vcc, effectively rising the edge, triggering the capture action  **while** (!(TACCTL0 & CCIFG)); //allowing for the capturing//updating TACCR0.  TACCR0 += Bitime ; // Some time till first bit  TXByte |= 0x100; // Add mark stop bit to TXByte  TXByte = TXByte << 1; // Add space start bit  TACCTL0 = CCIS0 + OUTMOD0 + CCIE; // TXD = mark = idle  **while** ( TACCTL0 & CCIE ); // Wait for TX completion  }  // Timer A0 interrupt service routine  **#pragma** vector=TIMER0\_A0\_VECTOR  \_\_interrupt **void** **Timer\_A** (**void**)  {  **if** (timerMode == TIMER\_UART\_MODE)  {  TACCR0 += Bitime; // Add Offset to TACCR0  **if** (TACCTL0 & CCIS0) // TX on CCI0B?  {  **if** ( BitCnt == 0)  {  P1SEL &= ~(TXD+RXD);  TACCTL0 &= ~ CCIE ; // All bits TXed, disable interrupt  }  **else**  {  TACCTL0 |= OUTMOD2; // TX Space  **if** (TXByte & 0x01)  TACCTL0 &= ~ OUTMOD2; // TX Mark  TXByte = TXByte >> 1;  BitCnt --;  }  }  }  **else**  {  **if** (tempPolarity == TEMP\_HOT)  LED\_OUT |= LED1;  **if** (tempPolarity == TEMP\_COLD)  LED\_OUT |= LED2;  TACCTL0 &= ~CCIFG;  }  }  **#pragma** vector=TIMER0\_A1\_VECTOR  \_\_interrupt **void** **ta1\_isr**(**void**)  {  TACCTL1 &= ~CCIFG;  **if** (applicationMode == APP\_APPLICATION\_MODE)  LED\_OUT &= ~(LED1 + LED2);  **else**  LED\_OUT ^= (LED1 + LED2);  }  **void** **InitializeClocks**(**void**)  {  BCSCTL1 = CALBC1\_1MHZ; // Set range  DCOCTL = CALDCO\_1MHZ;  BCSCTL2 &= ~(DIVS\_3); // SMCLK = DCO = 1MHz  }  **void** **InitializeButton**(**void**) // Configure Push Button  {  BUTTON\_DIR &= ~BUTTON;  BUTTON\_OUT |= BUTTON;  BUTTON\_REN |= BUTTON;  BUTTON\_IES |= BUTTON;  BUTTON\_IFG &= ~BUTTON;  BUTTON\_IE |= BUTTON;  }  **void** **InitializeLeds**(**void**)  {  LED\_DIR |= LED1 + LED2;  LED\_OUT &= ~(LED1 + LED2);  }  /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* Port Interrupt for Button Press  \* 1. During standby mode: to exit and enter application mode  \* 2. During application mode: to recalibrate temp sensor  \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/  **#pragma** vector=PORT1\_VECTOR  \_\_interrupt **void** **PORT1\_ISR**(**void**)  {  BUTTON\_IFG = 0;  BUTTON\_IE &= ~BUTTON; /\* Debounce \*/  WDTCTL = WDT\_ADLY\_250;  IFG1 &= ~WDTIFG; /\* clear interrupt flag \*/  IE1 |= WDTIE;  **if** (applicationMode == APP\_APPLICATION\_MODE)  {  tempCalibrated = tempAverage;  calibrateUpdate = 1;  }  **else**  {  applicationMode = APP\_APPLICATION\_MODE; // Switch from STANDBY to APPLICATION MODE  **\_\_bic\_SR\_register\_on\_exit**(LPM3\_bits);  }  }  // WDT Interrupt Service Routine used to de-bounce button press  **#pragma** vector=WDT\_VECTOR  \_\_interrupt **void** **WDT\_ISR**(**void**)  {  IE1 &= ~WDTIE; /\* disable interrupt \*/  IFG1 &= ~WDTIFG; /\* clear interrupt flag \*/  WDTCTL = WDTPW + WDTHOLD; /\* put WDT back in hold state \*/  BUTTON\_IE |= BUTTON; /\* Debouncing complete \*/  }  // ADC10 interrupt service routine  **#pragma** vector=ADC10\_VECTOR  \_\_interrupt **void** **ADC10\_ISR** (**void**)  {  **\_\_bic\_SR\_register\_on\_exit**(CPUOFF); // Return to active mode  } |
| So changing this function to this version will make the green and red LED’s flash together.  void PreApplicationMode(void)  {    LED\_DIR |= LED1 + LED2;  LED\_OUT &= ~LED2;  BCSCTL1 |= DIVA\_1; // ACLK/2  BCSCTL3 |= LFXT1S\_2; // ACLK = VLO  TACCR0 = 1200; //  TACTL = TASSEL\_1 | MC\_1; // TACLK = SMCLK, Up mode.  TACCTL1 = CCIE + OUTMOD\_3; // TACCTL1 Capture Compare  TACCR1 = 600;  \_\_bis\_SR\_register(LPM3\_bits + GIE); // LPM0 with interrupts enabled  } |
| As for this part, Changing the main function.  It changes the blink speed of the LEDS  First loop turns on p1.6(green LED) and turns off p.10(Red LED)  Second loop does the opposite of this.  Finally the last loop turns on both LEDS and blinks them three times.  #include <msp430g2231.h>  int main(void){  WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer  P2DIR &= 0x00;  P2OUT &= 0x00;  P1DIR |= 0x01; // Set P1.0 to output direction  P1DIR |= 0x41; // Set P1.0 & P1.6 to output direction  // Port 1 RED Port 6 GREEN  volatile unsigned int i;  unsigned int j;  unsigned int k;  // HEX Chart  // 0000 0 1000 8  // 0001 1 1001 9  // 0010 2 1010 A  // 0011 3 1011 B  // 0100 4 1100 C  // 0101 5 1101 D  // 0110 6 1110 E  // 0111 7 1111 F  //P1OUT &= 0xBE; // 1011 1110 Turn P1.6 off & P1.0 off leaving others intact  //P1OUT &= 0xFF; // 1111 1111 Turn P1.6 on & P1.0 on leaving others intact  P1OUT &= 0xFE; // 1111 1110 Turn P1.6 on & P1.0 off leaving others intact  //P1OUT &= 0xBF; // 1011 1111 Turn P1.6 off & P1.0 on leaving others intact    //blinks P1.6 or the green 6 times and turns on P1.6 and turns off P1.0  for (j=8;j>0;j--){  for (k=0;k<12;k++){  P1OUT ^= 0xFE;  i = 10000;  do{  (i--);  }  while (i != 0);  }  }  //blinks P1.0 or the red 6 times and turns on P1.0 and turns off P1.6  for (k=0;k<12;k++){  P1OUT ^= 0xBF;  for (j=8;j>0;j--){  i = 10000;  do{  (i--);  }  while (i != 0);  }  }  //turns off P1.6 and P1.0  //keeps P1.6 and P1.0 off for time ii would taketo blink three times  for (k=0;k<6;k++){  P1OUT ^= 0xBE;  for (j=8;j>0;j--) {  i = 10000;  do{  (i--);  }  while (i != 0);  }  }  //turns on P1.6 and P1.0  //blinks both red and green 3 times  for (k=0;k<6;k++){  P1OUT ^= 0xFF;  for (j=8;j>0;j--){  i = 10500;  do{  (i--);  }  while (i != 0);  }  }  // Turn P1.6 and P1.0 off leaving others intact  P1OUT &= 0xBE;  } |