```
void Init_Port2 (void) {
P2SEL0= GPIO_SET; //Sets all port pins to GP I/O
P2SEL1= GPIO_SET; // Sets all port pins to GP I/O
P2SEL0 &=~USB_TXD;// Sets to UCAOTXD
P2SEL1 |=USB_TXD;// Sets to UCAOTXD
P2DIR |=USB_TXD;// Sets USB_TXD to output

P2DIR=INPUT_SET;  //Sets all of port pins to input
P2SEL0 &=~USB_RXD;// Sets to UCAORXD
P2SEL1 |=USB_RXD;// Sets to UCAORXD
P2SEL0 &=~SPI_SCK; // Sets to UCBOCLK
P2SEL1 |=SPI_SCK;// Sets to UCBOCLK
//P2SEL0 &=~CPU_TXD; //Sets to UCA1TXD
//P2SEL1 |=CPU_TXD;//Sets to UCA1TXD
//P2SEL0 &=~CPU_RXD;// Sets to UCA1RXD
//P2SEL1 |=CPU_RXD;//Sets to UCA1RXD
P2SEL0 &=~PIN2_7;// Sets to GP I/0
P2SEL1 &=~PIN2_7;// Sets to GP I/0

P2DIR=INPUT_SET;
P2DIR |= CPU_TXD;
P2DIR |= SPI_SCK;

P2DIR |= CPU_TXD;
P2DIR |= PIN2_7;

P2OUT=OUTPUT_SET;
P2OUT|=SPI_SCK;


P2REN =LOW_SET;
P2REN |=SPI_SCK;// Enable pull up resistor
}
```

```
void Init_Port2 (void) {
P2SEL0= GPIO_SET; //Sets all port pins to GP I/O
P2SEL1= GPIO_SET; // Sets all port pins to GP I/O
P2SEL0 &=~USB_TXD;// Sets to UCAOTXD
P2SEL1 |=USB_TXD;// Sets to UCAOTXD
P2DIR |=USB_TXD;// Sets USB_TXD to output

P2DIR = INPUT_SET;  //Sets all of port pins to input
P2SEL0 &=~USB_RXD;// Sets to UCAORXD
P2SEL1 |=USB_RXD;// Sets to UCAORXD
P2SEL0 &=~SPI_SCK; // Sets to UCBOCLK
P2SEL1 |=SPI_SCK;// Sets to UCBOCLK
//P2SEL0 &=~CPU_TXD; //Sets to UCA1TXD
//P2SEL1 |=CPU_TXD;//Sets to UCA1TXD
//P2SEL0 &=~CPU_RXD;// Sets to UCA1RXD
//P2SEL1 |=CPU_RXD;//Sets to UCA1RXD
P2SEL0 &=~PIN2_7;// Sets to GP I/0
P2SEL1 &=~PIN2_7;// Sets to GP I/0

P2DIR = INPUT_SET;
P2DIR |= CPU_TXD;
P2DIR |= SPI_SCK;

P2DIR |= CPU_TXD;
P2DIR |= PIN2_7;

P2OUT = OUTPUT_SET;
P2OUT|=SPI_SCK;


P2REN =LOW_SET;
P2REN |=SPI_SCK;// Enable pull up resistor
}
```

```
//-----------------------------------------------------------------------
// Beginning of the "While" Operating System
//-----------------------------------------------------------------------
  while(ALWAYS) {                         // Can the Operating system run
    if(slow_input_down){
      slow_input_down = NONE;             // No need to check for changes in commands
      if(control_state[CONTROL_STATE_2] & BLINK_LED){  // Determine if LED should blink
   PJOUT ^= LED1;                         // Change LED 2 to indicate operation
      }
    }
// Switches_Process();                    // Check for switch state change
   Display_ADC();                         // Displays the values in the ADC
// five_msec_sleep(MSEC_5_DELAY);         // This will provide a 5 msec delay

    }
}

//------------------------------------------------------------------------------
```

```
//-----------------------------------------------------------------------
// Beginning of the "While" Operating System
//-----------------------------------------------------------------------
  while(ALWAYS) {                         // Can the Operating system run
    if(slow_input_down){
      slow_input_down = NONE;             // No need to check for changes in commands
      if(control_state[CONTROL_STATE_2] & BLINK_LED){  // Determine if LED should blink
   PJOUT ^= LED1;                         // Change LED 2 to indicate operation
      }
    }
// Switches_Process();                    // Check for switch state change
   Display_ADC();                         // Displays the values in the ADC
// five_msec_sleep(MSEC_5_DELAY);         // This will provide a 5 msec delay

    }
}

//------------------------------------------------------------------------------
```

```
// TimerA0 0 Interrupt handler
#pragma vector = TIMER0_A0_VECTOR
 __interrupt void Timer0_A0_ISR(void){

  TA0CCR0 += TA0CCR0_INTERVAL; // Add Offset to TACCR0

  if(DELAY_5MSEC){
   DELAY_5MSEC--;
  }

  if(switch_states & SW1_DEBOUNCE){
    count_debounce_SW1++; //Add to debounce count
    if(count_debounce_SW1 >= SOME_NUMBER_OF_MILLISECONDS){
      switch_states &= ~SW1_DEBOUNCE;
      enable_switch_SW1();
    }
  }

  if(switch_states & SW2_DEBOUNCE){
    count_debounce_SW2++;
    if(count_debounce_SW2 >= SOME_NUMBER_OF_MILLISECONDS){
      switch_states &= ~SW2_DEBOUNCE;
      enable_switch_SW2();
    }
  }
  if((!(switch_states & SW1_DEBOUNCE)) && (!(switch_states & SW2_DEBOUNCE))){
     TA0CCTL0 &= ~CCIE;                     // CCR0 enable interrupt
  }

}
```

```
// TimerA0 0 Interrupt handler
#pragma vector = TIMER0_A0_VECTOR
 __interrupt void Timer0_A0_ISR(void){

  TA0CCR0 += TA0CCR0_INTERVAL; // Add Offset to TACCR0

  if(DELAY_5MSEC){
    DELAY_5MSEC--;
  }

  if(switch_states & SW1_DEBOUNCE){
    count_debounce_SW1++; //Add to debounce count
    if(count_debounce_SW1 >= SOME_NUMBER_OF_MILLISECONDS){
      switch_states &= ~SW1_DEBOUNCE;
      enable_switch_SW1();
    }
  }

  if(switch_states & SW2_DEBOUNCE){
    count_debounce_SW2++;
    if(count_debounce_SW2 >= SOME_NUMBER_OF_MILLISECONDS){
      switch_states &= ~SW2_DEBOUNCE;
      enable_switch_SW2();
    }
  }
  if((!(switch_states & SW1_DEBOUNCE)) && (!(switch_states & SW2_DEBOUNCE))){
      TA0CCTL0 &= ~CCIE;     // CCR0 disable interrupt
  }

}
```

```
case 12:
   // Need this to change the ADC10INCH_x value.
      ADC10CTL0 &= ~ADC10ENC;             // Toggle ENC bit.

      switch (ADC_Channel++){
        case Right_Detector:
          ADC10MCTL0 = ADC10INCH_1;       // Next channel A1
          ADC_Right_Detector = ADC10MEM0; // Read Channel A0
          break;
        case Left_Detector:
          ADC10MCTL0 = ADC10INCH_3;       // Next channel A3
          ADC_Left_Detector = ADC10MEM0;  // Read Channel A1
          break;
        case Thumbwheel:
          ADC10MCTL0 = ADC10INCH_11;      // Next channel A11
          ADC_Thumb = ADC10MEM0;          // Read Channel A3
          break;
        case CHANNEL_A10:
          ADC10MCTL0 = ADC10INCH_10;      // Next channel A10
          ADC_Temp = ADC10MEM0;           // Read Channel A10
          break;
        case CHANNEL_A11:
          ADC10MCTL0 = ADC10INCH_0;       // Next channel A0
          ADC_Bat = ADC10MEM0;            // Read Channel A11
          ADC_Channel=NONE;
          break;
        default:
        break;
      }
      ADC10CTL0 |= ADC10ENC | ADC10SC;    // Start next sample.
      break;
```

```
case 12:
   // Need this to change the ADC10INCH_x value.
      ADC10CTL0 &= ~ADC10ENC;             // Toggle ENC bit.

      switch (ADC_Channel++){
        case Right_Detector:
          ADC10MCTL0 = ADC10INCH_1;       // Next channel A1
          ADC_Right_Detector = ADC10MEM0; // Read Channel A0
          break;
        case Left_Detector:
          ADC10MCTL0 = ADC10INCH_3;       // Next channel A3
          ADC_Left_Detector = ADC10MEM0;  // Read Channel A1
          break;
        case Thumbwheel:
          ADC10MCTL0 = ADC10INCH_11;      // Next channel A11
          ADC_Thumb = ADC10MEM0;          // Read Channel A3
          break;
        case CHANNEL_A10:
          ADC10MCTL0 = ADC10INCH_10;      // Next channel A10
          ADC_Temp = ADC10MEM0;           // Read Channel A10
          break;
        case CHANNEL_A11:
          ADC10MCTL0 = ADC10INCH_0;       // Next channel A0
          ADC_Bat = ADC10MEM0;            // Read Channel A11
          ADC_Channel=NONE;
          break;
        default:
        break;
      }
       ADC10CTL0 |= ADC10ENC | ADC10SC;    // Start next sample.
      break;
```

```
switch (time_slice){
  case 20:
    // Do the 200 msec stuff
  case 10:
    // Do the 100 msec stuff
  case 15:
  case  5:
    // Do the 50 msec stuff
  case 19:
  case 18:
  case 17:
  case 16:
  case 14:
  case 13:
  case 12:
  case 11:
  case  9:
  case  8:
  case  7:
  case  6:
  case  4:
  case  3:
  case  2:
  case  1:
    // Do the 10 msec stuff
    break;
  default:
    break;
}
```