Power Conscious Sensor-based Remote Home Monitoring System through IOT

Ali Attaran **ENGR 844 Project**

Professor Dr. Xiaorong Zhang

Table of Contents

List of Figures

1 Introduction

Smart home monitoring IoT system is growing rapidly. With new research continuously revealing new and exciting appliances to add to our home monitoring systems, it is important for researchers to come up with new inventions to improve our quality of living by implementing smart monitoring systems in our urban living environment.

Fire hazards are among the most dangerous and unpredictable happenings in urban living. In the same time, there are many technologies that can be used to resolve these problems and more over support better living [1]. As a part of IoT, temperature monitoring system holds the promise of many new applications in the area of monitoring and control. Wireless sensor networks are an emerging technology consisting of small, low power, and low-cost devices that integrate limited computation, sensing and remote communication capabilities. This technology has enormous impact on fire emergency. The temperature sensor on TM4C123G Microcontroller Launchpad can be used to detect a dramatic change o-in temperature in its vicinity and report it to the user. But the on board temperature is not very reliable due to its poor accuracy. The TMP36 Temperature sensor is used in this project to detect the temperature and report to user on two platforms, LCD and PuTTY terminal on PC. In addition to that, to promote power saving living, a light sensor is used to detect the room's brightness and if the room is bright, turn on the LCD.

In recent times, development in computing and consumer electronics technologies have triggered Internet of Things (IOT) paradigm. IOT is described as enabler that links seamless objects surrounding the environment and performs some sort of message exchange among them. The IOT is a collection of objects that work jointly in order to serve consumer tasks in a federated manner [1]. It binds computational power to deliver data about the surrounding environments [2] [4]. These devices can be in form of bespoke sensors, appliances, embedded systems, and data analysis microchips.

IOT has been explored in many diverse applications; in this project, we focus on sensor-based home monitoring application of IOT [3]. The proposed system consists of light and temperature sensors which record the brightness and temperature of the room and report to the user's mobile phone. The user is able to send command text messages to the TM4C123G Launchpad to either turn the fan on, in case the temperature is higher than software programmed value and send the command to turn the light off in day time when the room is bright and turn the light off when the room is darker than a programmed ambient light amplitude.

The system consists of hardware and software parts. The TMP36 temperature sensor and TEMT6000 light sensor and the LCD are connected to TM4C123GH6PM Launchpad. The software for the system is written in embedded C. The system works as follows. It starts with the initialization of all variables needed by the software, then initializes the LCD to display that the system starts to work. After the initialization process, the system then enables the serial interrupt and ADC to read the temperature from TMP36 sensor and ambient light from TEMT6000. The LCD shows the temperature and light status every second and if the temperature is more than 70 C degrees, it triggers an alarm and turns on a fan close to the temperature sensor.

The ambient light sensor, TEMT6000, acts like a transistor, the greater the incoming light, the higher the analog voltage on the signal pin. If the ambient light of the indoor environment reaches some programmed amplitude, the Launchpad turns down the ambient light of the LED to save power. Otherwise the LED will be left on. This method is widely used for power conscious applications in automated power saving systems.

For the temperatures sensing, the analog temperature is sensed by the TMP36 temperature sensor. This temperature is then converted into digital format by the ADC in TM4C123G launchpad. The microcontroller receives this data and displays the temperature on 3 digital displays. The above process is repeated every second. The sensor's output voltage is linearly proportional to the Celsius (Centigrade) temperature. It is chosen for its low output impedance, linear output, and precise inherent calibration that make interfacing very easy [4] [5]. The user can monitor the sensors' record on the PC as well as LCD through UART and PuTTY.

2 System architecture

Temperature and light monitoring system follows the existing smart home monitoring systems. Firstly, the temperature sensor displays the temperature on LCD and PuTTY terminal on PC every second and if the temperature is higher than the programmed threshold temperature, it will display "ALERT" message on both platforms. Secondly, the light sensor only triggered when there is a change on light of the room. The flow chart of the code is shown in Fig.1 to 4.

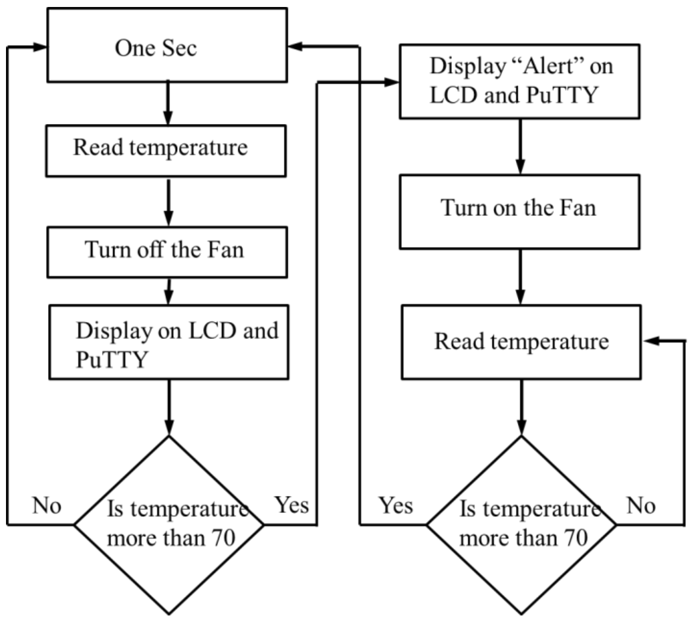


Figure 1. Temperature sensor flowchart

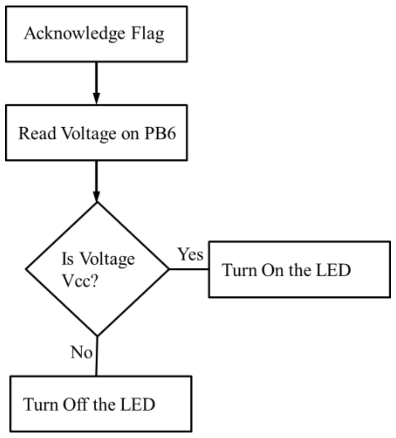


Figure 2. Light sensor flowchart

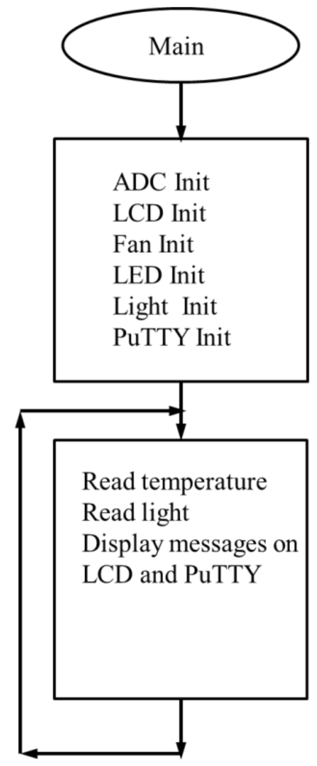


Figure 3. Main flow chart

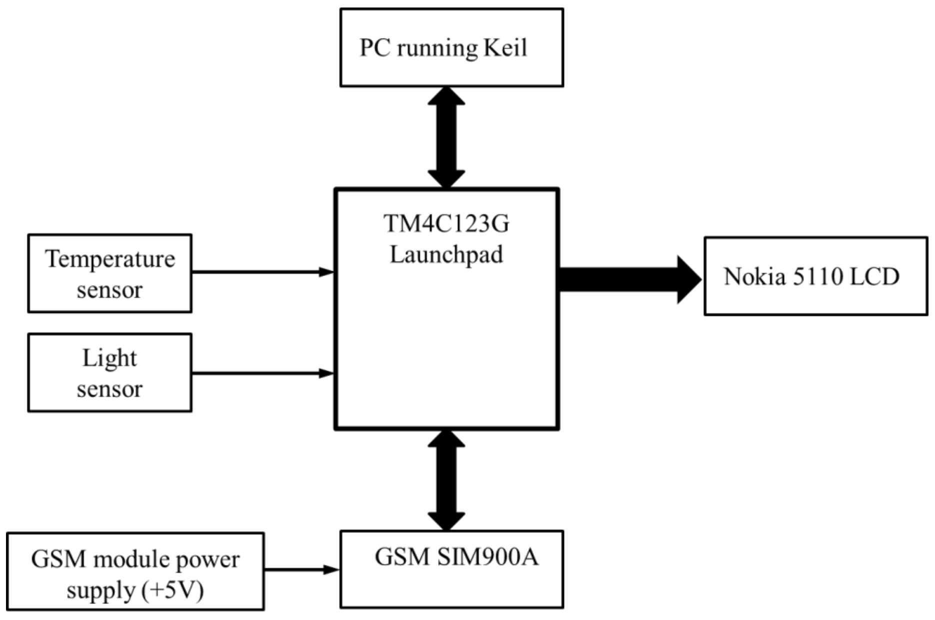


Figure 4. Overall scheme of the project

3 Light Dependent Resistor (LDR) sensor

This sensor works by sensing the intensity of light in its environment. The sensor that can be used to detect light is an LDR. The LDR gives out an analog voltage when connected to Vcc (3.3V), which varies in magnitude in direct proportion to the input light intensity on it. That is, the greater the intensity of light, the greater the corresponding voltage from the LDR will be. Since the LDR gives out an analog voltage, it is connected to the input pin (PB6) on the launchpad. When there is sufficient light in its environment or on its surface, its resistance is 4kΩ and its analog output voltage is 3V or logic high. When the light is off, its analog output voltage is 0V and its resistance is 2MΩ.

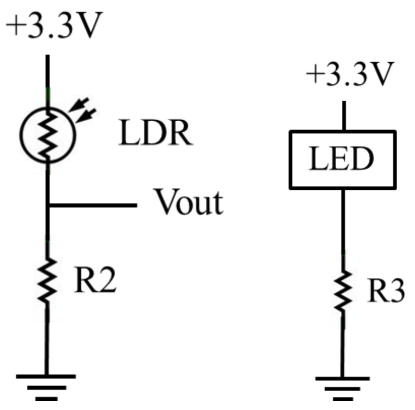


Figure 5. LDR and LED interface

R2 is chosen to b 39k Ω to satisfy the Vout requirement.

LED needs around 1.8V and 10mA to turn on. So R3 is calculated to be 150 Ω.

3.1 Launchpad connection

The Launchpad connections for LED and LDR are shown below.

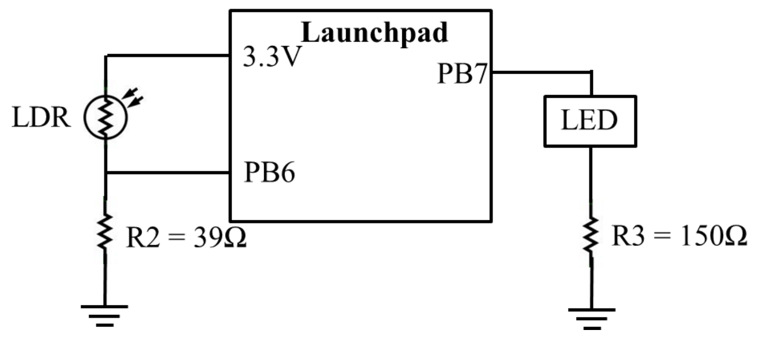


Figure 6. The Launchpad connections

3.2 Software: timer interrupt

To initialize the PB6 as input to read the light sensor, following steps are necessary:

```
void Light_Sensor_Interrupt_Init(void)

{

   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);       // Enable port B

   GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_6);  // Init PB6 as input

   GPIOIntDisable(GPIO_PORTB_BASE, GPIO_PIN_6);          // Disable interrupt for PB6 (in case it was
enabled)

   GPIOIntClear(GPIO_PORTB_BASE, GPIO_PIN_6);     // Clear pending interrupts for PB6

   GPIOIntRegister(GPIO_PORTB_BASE, light_sensor_handler);       // Register our handler function for
port B

   GPIOIntTypeSet(GPIO_PORTB_BASE, GPIO_PIN_6, GPIO_BOTH_EDGES); // Configure for rising or
falling edge trigger

   GPIOIntEnable(GPIO_PORTB_BASE, GPIO_PIN_6);    // Enable interrupt

}
```

And finally, to obtain the voltage at PB6, the void light_sensor_handler(void) is used to return the voltage at PB6, as shown here:

```
void light_sensor_handler(void)
{
   if(Light_Sensor_Get_Value()==1)
   {
                  Led_Turn_off();
         }
```

```
            else
   {
                        Led_Turn_on();
            }
```

The Timer0A interrupt handler is used as below:

```
void TIMER0A_intHandler(void)

{

   unsigned long ADCdata;

      ADCdata = read_temperature();   // Get current temperature

   newTemperatureData = true;  // Flag the main function that new data is ready

      TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);  // Clear the timer interrupt

}
```

To initialize the LED as output on pin PB7, following steps are necessary:

```
void Led_GPIO_Init(){ volatile unsigned long delay;

  SYSCTL_RCGC2_R |= 0x00000002;       // 1) B clock

  delay = SYSCTL_RCGC2_R;             // delay to allow clock to stabilize

  GPIO_PORTB_AMSEL_R &= ~0x80;        // 2) disable analog function

  GPIO_PORTB_PCTL_R &=~ 0xF0000000;   // 3) GPIO clear bit PCTL

  GPIO_PORTB_DIR_R |= 0x80;          // 4) PB7 Output,

  GPIO_PORTB_AFSEL_R &=~ 0x80;        // 5) no alternate function

  GPIO_PORTB_DEN_R |= 0x80;          // 6) enable digital pins PB7

}
```

To turn on the LED, the PB7 is set to 1 and to turn it off, it's cleared.

```
void Led_Turn_on(){

            GPIO_PORTB_DATA_R |= 0x00000080;

}
```

```
void Led_Turn_off(){

            GPIO_PORTB_DATA_R &=~ 0x00000080;

}
```

4 Temperature sensor

4.1 Hardware

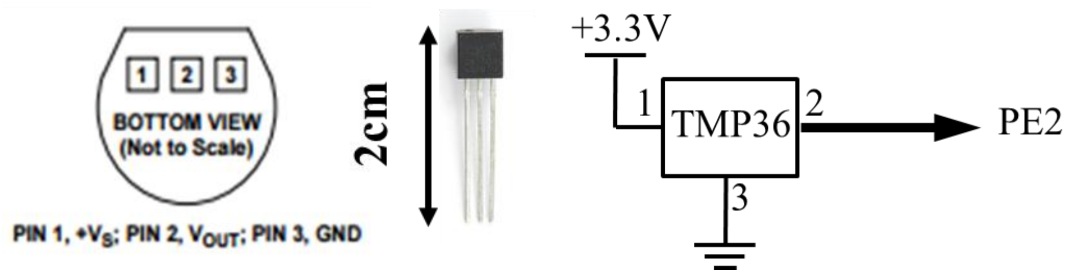


Figure 7. TMP36 sensor

This sensor is a typical temperature to voltage converter. It increments the output voltage at Pin 2, for every 1C degree, by 10mV. At 25C degree, the Pin 2 voltage is 750mV. So at 70C degrees, the output voltage at Pin 2 will be:

(70 – 25) x 10mV + 750mV = 1100mV

The Pin 2 is connected to the PE2 pin on the Launchpad.

4.2 Software

4.3 Reading the temperature from ADC

Assume that the analog temperature samples from the sensor are continuously varying as shown in Figure 1.

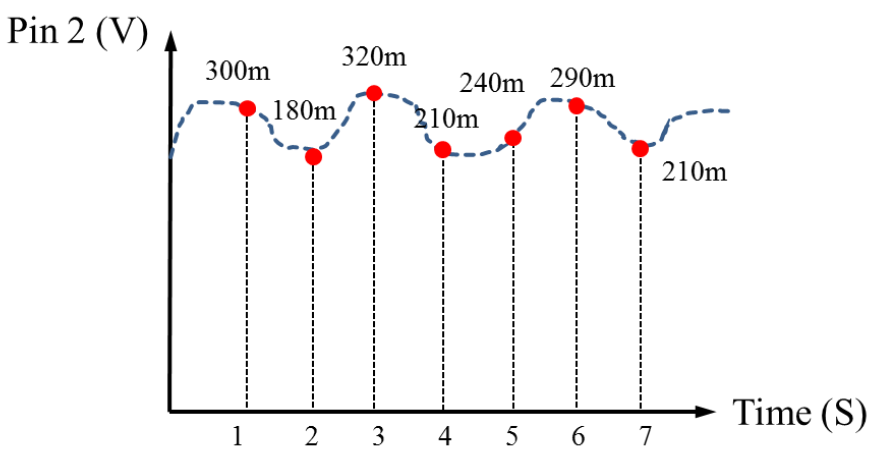


Figure 8. recording the temperature in analog form every second.

At every second, the ADC reads the output voltage from the temperature sensor and converts sampled analog voltage value to binary. Since the Launchpad's max voltage is 3.3V and it uses 12 bits for digital conversion, the binary representation of the output voltage can be calculated as:

For example V(pin 2) at 2s is 180mV, so ADC data is (4096 x 180mV) / 3300mV = 223.

To calculate the temperature, I used the ADC resolution formula which is given as, 3.3*(ADC reading)/4096. To display rounded number, we can use %10000. So the result will be:

```
// 3.3volts*(data)/4096

result=((33*1000*data)/4096)%100000;//rounding off to maximum 5 digits
```

In order to display with special characters such as "/n" and "/r" and space, we used ASCII codes for these characters. 10, 13 and 32 are the ASCII code for "/n" and "/r" and space respectively. Also in order to convert the result to ASCII code, we need to add "0x30" to the beginning of its array as such:

```
Temp_s[0]=0x30;  // ASCII for '0'

        Temp_s[3]='.';

         // Now Temp_s looks like this (? represents an indeterminate value:

  // 0 ? ? . ? ? ? ? ? ? ? ? ?
```

```
   // Now we're filling Temp_s backwards from the 5th index,
   // with the respective digits of `result`. The `continue` skips over
   // index 3 so we don't overwrite the '.' we just put there.
          for(i=0;i<6;i++)
                    {
                    if(i==2){continue;}
                              Temp_s[5-i]=(result%10)+0x30; // 0x30 is just ASCII '0'.
                              result=result/10;
                     // Let's say that result was 12345. Now Temp_s looks like this:
   // 1 2 3 . 4 5 ? ? ? ? ? ? ?
                    }
                    Temp_s[i]=32;
Temp_s[i+1]=32;Temp_s[i+2]='*';Temp_s[i+3]=32;Temp_s[i+4]='C';Temp_s[i+5]=13;Temp_s[i+6]=10;


                      // In the end, Temp_s looks like this:
   // 1 2 3 . 4 5 [space] [space] * [space] C \r \n
}
```

In the code, C function ADCdata = read_ temperature ();

4.4 Configuring ADC module for temperature reading

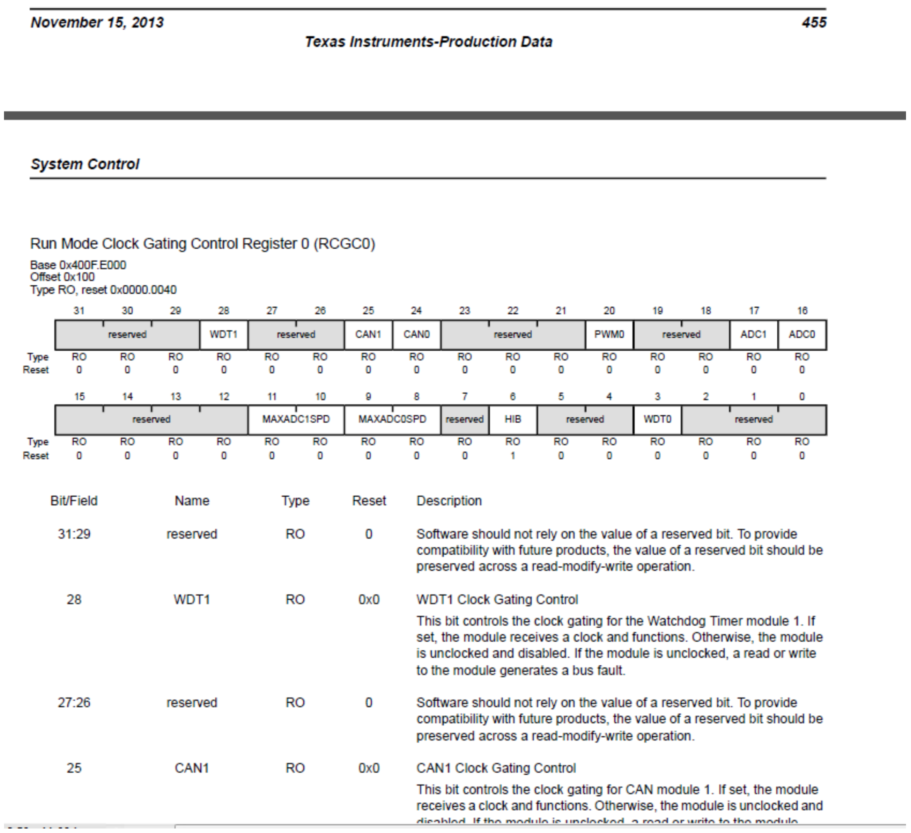
The ADC initialization is as follows:

1- First disabling the ADC while initializing; since I chose SS3 so it should be cleared.

```
ADC0_ACTSS_R &= ~0x0008;       // 9) disable sample sequencer 3
```

2- Clock: Setting the ADC0 clock by setting the bit 16 in SysCTL_RCG0_R to 1

```
SYSCTL_RCGC0_R |= 0x00010000;   // 6) activate ADC0  bit 16 to set clk on
```



November 15, 2013 455
Texas Instruments-Production Data

System Control

Run Mode Clock Gating Control Register 0 (RCGC0)

Base 0x400F.E000
Offset 0x100
Type RO, reset 0x0000.0040

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved			WDT1	reserved		CAN1	CAN0	reserved			PWM0	reserved		ADC1	ADC0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved				MAXADC1SPD		MAXADC0SPD		reserved	HIB	reserved		WDT0	reserved		
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

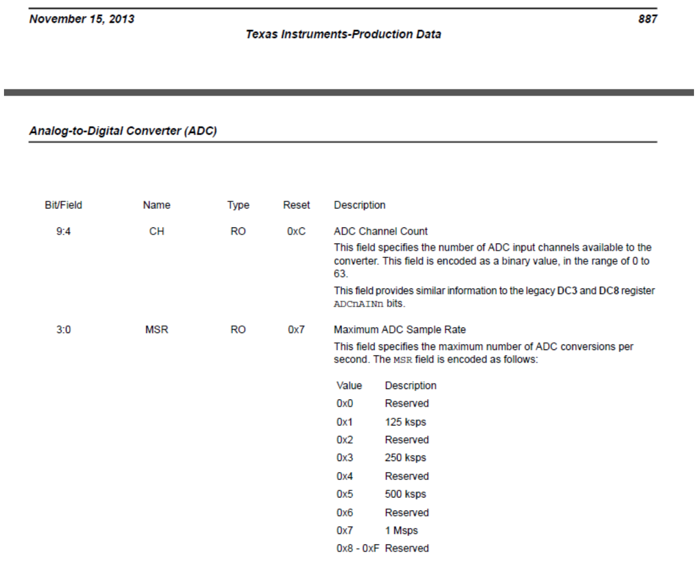
Bit/Field	Name	Type	Reset	Description
31:29	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
28	WDT1	RO	0x0	WDT1 Clock Gating Control This bit controls the clock gating for the Watchdog Timer module 1. If set, the module receives a clock and functions. Otherwise, the module is unclocked and disabled. If the module is unclocked, a read or write to the module generates a bus fault.
27:26	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
25	CAN1	RO	0x0	CAN1 Clock Gating Control This bit controls the clock gating for CAN module 1. If set, the module receives a clock and functions. Otherwise, the module is unclocked and disabled. If the module is unclocked, a read or write to the module

Figure 9. Datasheet page. 456 [6]

3- Sampling rate: Setting sampling rate

Bits 8 and 9 are to set ADC sampling rate. If we want to choose 125k bit/second. Bits 8 and 9 should be cleared.

SYSCTL_RCGC0_R &= ~0x00000300;   // 7) configure for 125K bit/s 8 and 9 specify the maximum sampling rate



November 15, 2013 887

Texas Instruments-Production Data

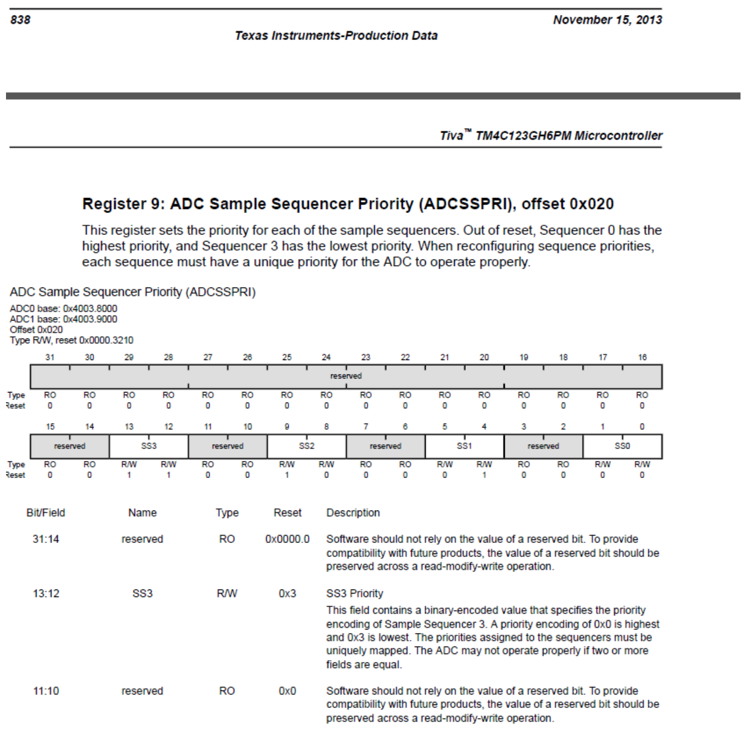
Analog-to-Digital Converter (ADC)

Bit/Field	Name	Type	Reset	Description
9:4	CH	RO	0xC	ADC Channel Count This field specifies the number of ADC input channels available to the converter. This field is encoded as a binary value, in the range of 0 to 63. This field provides similar information to the legacy **DC3** and **DC8** register `ADCnAINn` bits.
3:0	MSR	RO	0x7	Maximum ADC Sample Rate This field specifies the maximum number of ADC conversions per second. The `MSR` field is encoded as follows:

Value	Description
0x0	Reserved
0x1	125 ksps
0x2	Reserved
0x3	250 ksps
0x4	Reserved
0x5	500 ksps
0x6	Reserved
0x7	1 Msps
0x8 - 0xF	Reserved

Figure 10. Datasheet page. 888

To Set priority of SS3: bits 12 and 13 in ADC0_SSPRI_R are set to 0;



838 November 15, 2013

Texas Instruments-Production Data

Tiva™ TM4C123GH6PM Microcontroller

Register 9: ADC Sample Sequencer Priority (ADCSSPRI), offset 0x020

This register sets the priority for each of the sample sequencers. Out of reset, Sequencer 0 has the highest priority, and Sequencer 3 has the lowest priority. When reconfiguring sequence priorities, each sequence must have a unique priority for the ADC to operate properly.

ADC Sample Sequencer Priority (ADCSSPRI)
ADC0 base: 0x4003.8000
ADC1 base: 0x4003.9000
Offset 0x020
Type R/W, reset 0x0000.3210

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved		SS3		reserved		SS2		reserved		SS1		reserved		SS0	
Type	RO	RO	R/W	R/W	RO	RO	R/W	R/W	RO	RO	R/W	R/W	RO	RO	R/W	R/W
Reset	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:14	reserved	RO	0x0000.0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
13:12	SS3	R/W	0x3	SS3 Priority This field contains a binary-encoded value that specifies the priority encoding of Sample Sequencer 3. A priority encoding of 0x0 is highest and 0x3 is lowest. The priorities assigned to the sequencers must be uniquely mapped. The ADC may not operate properly if two or more fields are equal.
11:10	reserved	RO	0x0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.

Figure 11. Datasheet page. 839

Set trigger event to software: `ADC0_EMUX_R &= ~0xF000;        // 10) seq3 is software trigger`

For SS3, EM3 are bits 15 to 12 in ADC0_SSMUX3_R set to 0:

```
ADC0_SSMUX3_R &= ~0x000F;      // 11) clear SS3 field
```

Set channel 1 (Ain1) in ADC0_SSMUX3_R to 1, :

```
ADC0_SSMUX3_R += 1;            //   set channel Ain1 (PE2)
```

Set flag on sampling:

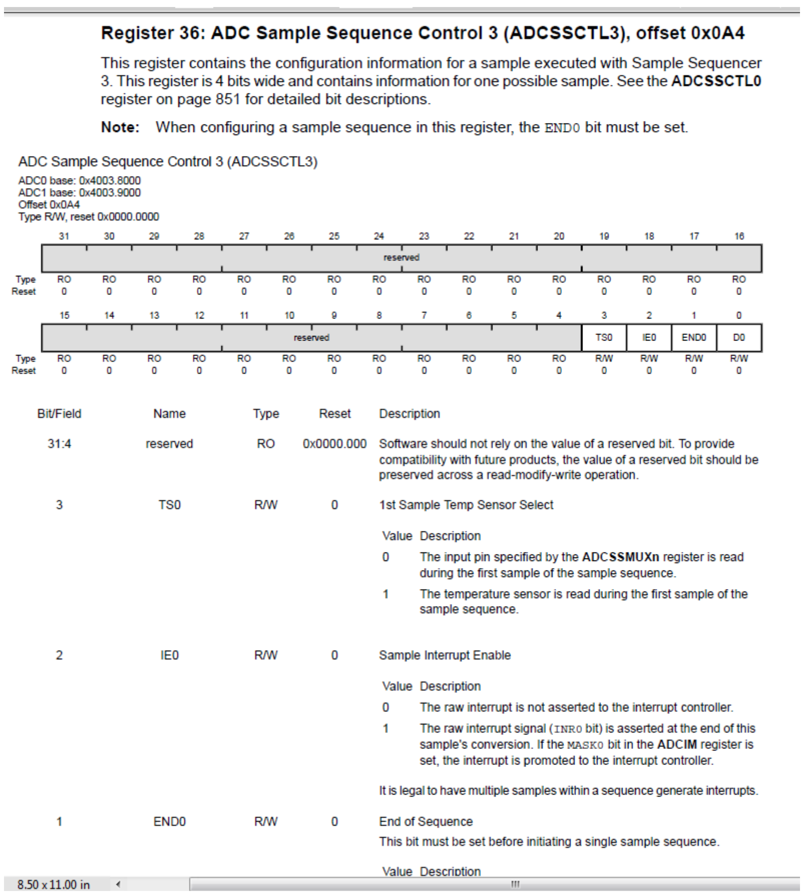
```
ADC0_SSCTL3_R = 0x0006;        // 12) no TS0 D0, yes IE0 END0
```

TS0 should be cleared because the input pin is specified by ADCSSMUX3 to Ain1.

IE0 should be set because raw interrupt is accreted at the end of sample.

END0 should be set because this is end of sequence.

D0 should be cleared because it analog inputs are not differentially sampled.



Register 36: ADC Sample Sequence Control 3 (ADCSSCTL3), offset 0x0A4

This register contains the configuration information for a sample executed with Sample Sequencer 3. This register is 4 bits wide and contains information for one possible sample. See the **ADCSSCTL0** register on page 851 for detailed bit descriptions.

Note: When configuring a sample sequence in this register, the END0 bit must be set.

ADC Sample Sequence Control 3 (ADCSSCTL3)
ADC0 base: 0x4003.8000
ADC1 base: 0x4003.9000
Offset 0x0A4
Type R/W, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved												TS0	IE0	END0	D0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:4	reserved	RO	0x0000.000	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
3	TS0	R/W	0	1st Sample Temp Sensor Select Value Description 0 The input pin specified by the **ADCSSMUXn** register is read during the first sample of the sample sequence. 1 The temperature sensor is read during the first sample of the sample sequence.
2	IE0	R/W	0	Sample Interrupt Enable Value Description 0 The raw interrupt is not asserted to the interrupt controller. 1 The raw interrupt signal (INR0 bit) is asserted at the end of this sample's conversion. If the MASK0 bit in the **ADCIM** register is set, the interrupt is promoted to the interrupt controller. It is legal to have multiple samples within a sequence generate interrupts.
1	END0	R/W	0	End of Sequence This bit must be set before initiating a single sample sequence. Value Description

8.50 x 11.00 in

Figure 12. Datasheet page. 874

Enabling ADC by, setting ADC0_ACTSS_R

4.4.1 Read temperature function in ADC

First we need to initialize the SS3 buy setting bit 4 in ADCPSSI register to 1.

```
ADC0_PSSI_R = 0x0008;          // 1) initiate SS3
```

Since an interrupt is triggered by ADC, we need to check when the SS3 raw interrupt status bit in ADCRIS register is occurred (is 1).

So when its 0 nothing will happen.

```
while((ADC0_RIS_R&0x08)==0){};   // 2) wait for conversion done
```

After a sample sequence completes execution, the result data can be retrieved from the ADC

Sample Sequence Result FIFO (ADCSSFIFO3) register.

Bits from 0 to 11 in ADCSSFIFO3 can be read by:

```
result = ADC0_SSFIFO3_R&0xFFF;   // 3) read result
```

The ADCRIS register has been acknowledged by setting bit 4 is ADCRIS register.

```
ADC0_ISC_R = 0x0008;           // 4) acknowledge completion
```

4.5 Displaying the temperature in PuTTY

UART peripherals typically have several configurable parameters required to support different standards. There are five parameters which must be configured correctly to establish a basic serial connection:

- Baud rate: Baud rate is the number of symbols or modulations per second. Basically, the baud rate indicates how many times the lines can change state (high or low) per second. Since each symbol represents one bit, the bit rate equals the baud rate. For example, if the baud rate is 11520, there are 11520symbols sent per second and therefore the bit rate is 11520bits per second (bps).
- Number of data bits: The number of data bits transmitted is typically between 5 and 8, with 7 and 8 being the most common since an ASCII character is 7 bits for the standard set and 8 bits for the extended.
- Parity: The parity can be even, odd, mark or space. The UART peripheral calculates the number of 1s present in the transmission. If the parity is configured to even and the number of 1's is even then the parity bit is set zero. If the number of 1s is odd, the parity bit is set to a 1 to make the count even. If the parity is configured to odd, and the number of 1s is odd, then parity bit is set to 0. Otherwise it is set to 1 to make the count odd. Mark and space parity mean that the parity bit will either be one or zero respectively for every transmission.
- Stop bits: The number of stop bits is most commonly configurable to either one or two. On some devices, half bits are supported as well, for example 1.5 stop bits. The number of stop bits determines how much of a break is required between concurrent transmissions.

- Endianess: Some UART peripherals offer the option to send the data in either LSB (least significant bit) or MSB (most significant bit). Serial communication of ASCII characters is almost always LSB.

All of these parameters must be set to the same configuration on both devices for successful communication. The following image is an example of a UART transmission.

To check if the FIFO is empty or full, different flags are used; to send data, buffer TXFF flag which is called the FIFO full flag is used. So the UARTFR register and UARTRXFF register are repeatedly getting checked. And as long as they are **0**, which means that there's no data to send, this is used in busy wait method.

In order to display anything in the PuTTY, the text is out in a string (i.e. character array). For example, to display 28.76°C in PC, it needs to be put in an array as { ‘2’, ‘8’ , ‘.’ , ‘7’ , ‘6’ , ‘°’ , ‘C’ }.

The data communication between the Launchpad and PC is via UART feature. UART has a transmit buffers, into which the data bits can traverse to PC. In order to communicate with PuTTY, we need to configure the Baud rate (speed), serial line. These values have to match with Launchpad configuration.

The baud rate for the Launchpad is 115200 bit/second. In order to calculate how often PC has to check for new data transmission, we can calculate the second/bit by:

1 / 115200 = 8.68 Microsecond / bit. So bits are transmitted every 8.68 micorsecond.

Sending data:

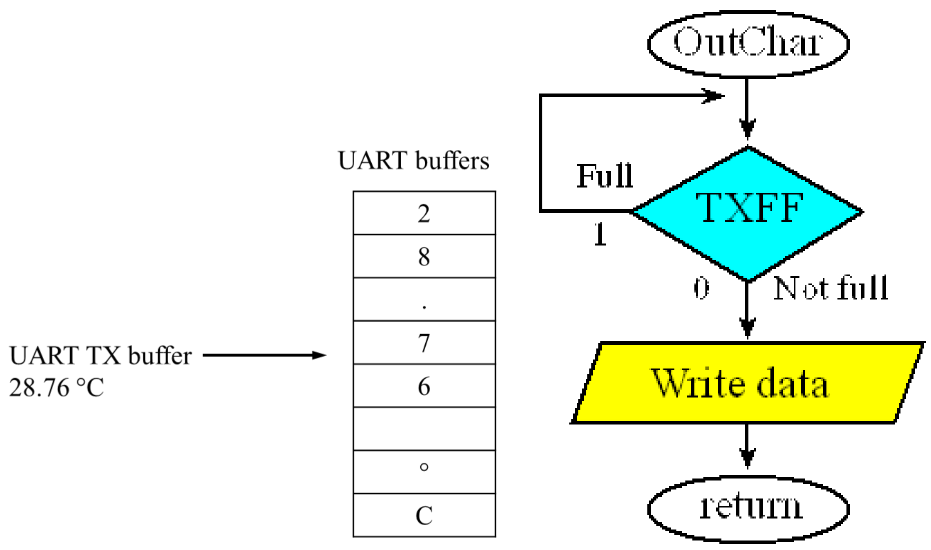


Figure 13. UART TX buffer sample

Pin connection

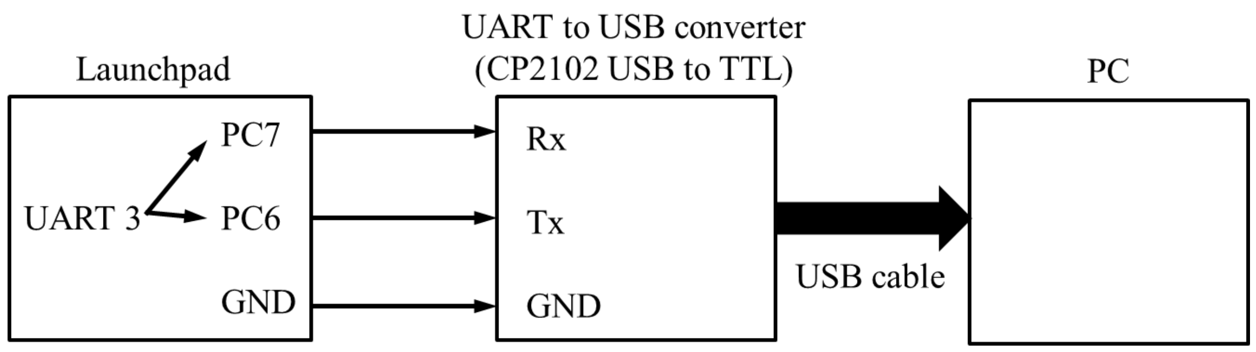
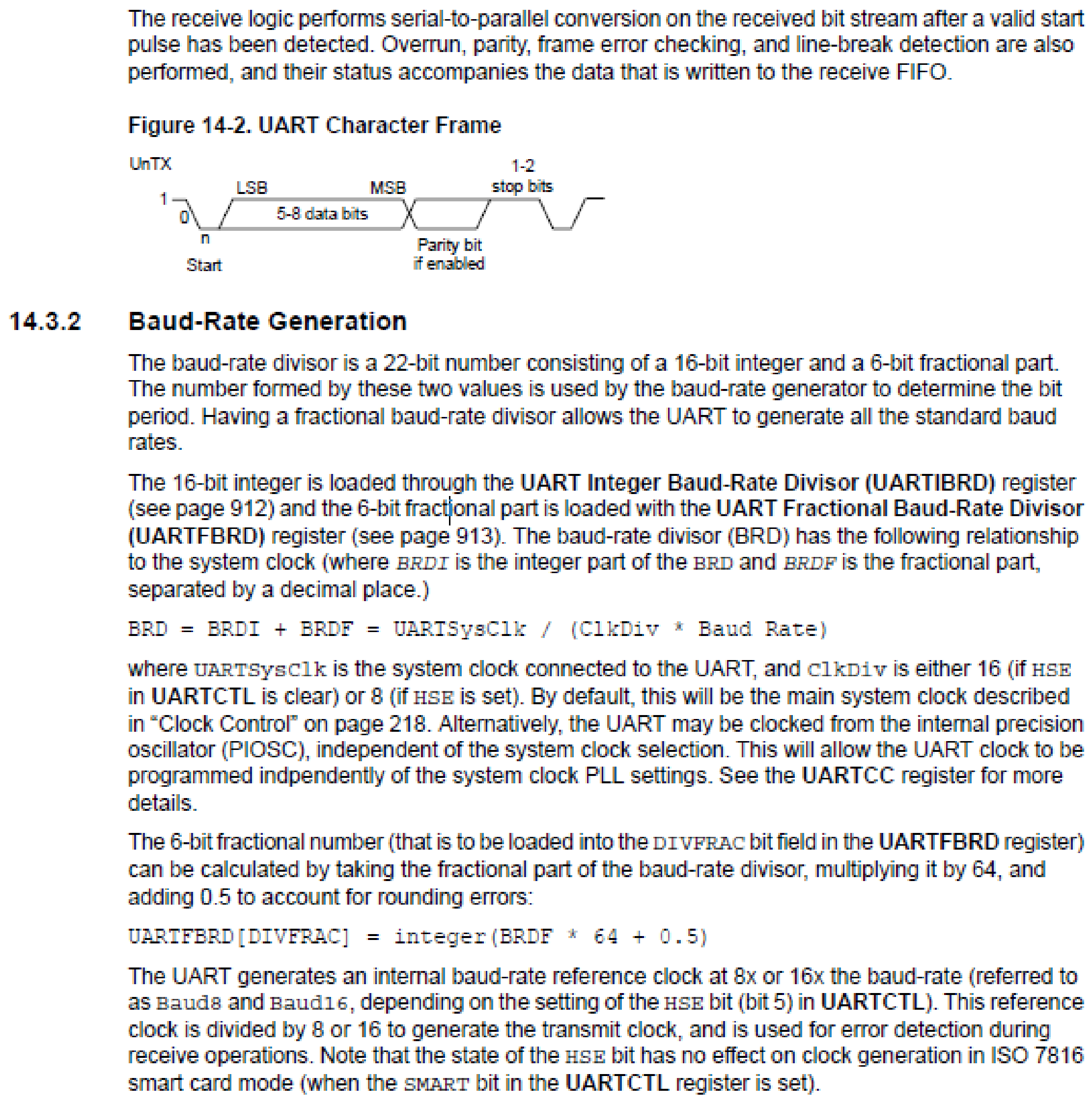


Figure 14. Pin connections for UART3 to PC

C function: PC_UART3_OutString(temp); example: temp = { ‘2’, ‘8’ , ‘.’ , ‘7’ , ‘6’ , ‘°’ , ‘C’ }.

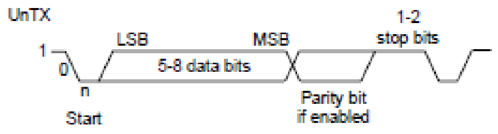
To initialize the UART3, following steps were taken:

1- activate UART3 Clock
2- activate port D Clock
3- In order to configure the baud rate, we need to disable UART3 first. This has been done by clearing UART3CTL register to UART_CTL_UARTEN
4- The baud rate for these devices is set to 115200, to set the IBRD and FBRD registers, baud rate generator formula from datasheet is used.



The receive logic performs serial-to-parallel conversion on the received bit stream after a valid start pulse has been detected. Overrun, parity, frame error checking, and line-break detection are also performed, and their status accompanies the data that is written to the receive FIFO.

Figure 14-2. UART Character Frame



14.3.2 Baud-Rate Generation

The baud-rate divisor is a 22-bit number consisting of a 16-bit integer and a 6-bit fractional part. The number formed by these two values is used by the baud-rate generator to determine the bit period. Having a fractional baud-rate divisor allows the UART to generate all the standard baud rates.

The 16-bit integer is loaded through the **UART Integer Baud-Rate Divisor (UARTIBRD)** register (see page 912) and the 6-bit fractional part is loaded with the **UART Fractional Baud-Rate Divisor (UARTFBRD)** register (see page 913). The baud-rate divisor (BRD) has the following relationship to the system clock (where *BRDI* is the integer part of the BRD and *BRDF* is the fractional part, separated by a decimal place.)

```
BRD = BRDI + BRDF = UARTSysClk / (ClkDiv * Baud Rate)
```

where `UARTSysClk` is the system clock connected to the UART, and `ClkDiv` is either 16 (if `HSE` in **UARTCTL** is clear) or 8 (if `HSE` is set). By default, this will be the main system clock described in "Clock Control" on page 218. Alternatively, the UART may be clocked from the internal precision oscillator (PIOSC), independent of the system clock selection. This will allow the UART clock to be programmed indpendently of the system clock PLL settings. See the **UARTCC** register for more details.

The 6-bit fractional number (that is to be loaded into the `DIVFRAC` bit field in the **UARTFBRD** register) can be calculated by taking the fractional part of the baud-rate divisor, multiplying it by 64, and adding 0.5 to account for rounding errors:

```
UARTFBRD[DIVFRAC] = integer(BRDF * 64 + 0.5)
```

The UART generates an internal baud-rate reference clock at 8x or 16x the baud-rate (referred to as `Baud8` and `Baud16`, depending on the setting of the `HSE` bit (bit 5) in **UARTCTL**). This reference clock is divided by 8 or 16 to generate the transmit clock, and is used for error detection during receive operations. Note that the state of the `HSE` bit has no effect on clock generation in ISO 7816 smart card mode (when the `SMART` bit in the **UARTCTL** register is set).

Figure 15. Datasheet page 894

5- The word length is set to 8bits without parity bit, by setting LCRH_WLEN_8 and LCRH_FEN registers.
6- PC6 and 7 are configured as UART Tx and Rx ports
7- To set their alternate function, we need to set their respective bits in PMC6 and 7.
8- Disable analog function on PC6 and 7

```
void PC_UART3_Init(void){

// Using PC5,PC4

  SYSCTL_RCGCUART_R |= 0x08; // activate UART3
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOC; // activate port D

  UART3_CTL_R &= ~UART_CTL_UARTEN;      // disable UAR/

            /*

            * BAUD RATE calculation

            */

  UART3_IBRD_R = 8;                   // IBRD = int(16,000,000 / (16 * 115200)) = int(8.68)

  UART3_FBRD_R = 44;                   // FBRD = round(0.68 * 64) = 44

                                // 8 bit word length (no parity bits, one stop bit, FIFOs)

  UART3_LCRH_R = (UART_LCRH_WLEN_8|UART_LCRH_FEN);

  UART3_CTL_R |= UART_CTL_UARTEN;       // enable UART

  GPIO_PORTC_AFSEL_R |= 0xC0;;          // enable alt funct on PC7,PC6

  GPIO_PORTC_DEN_R |= 0xC0;            // enable digital I/O on PC7,PC6

                                // configure PC7,PC6 as UART3

  GPIO_PORTC_PCTL_R = (GPIO_PORTC_PCTL_R&0x00FFFFFF)+0x11000000;//11 is required to alternate
function of PC7,6 as UART3 ( from data sheet)

  GPIO_PORTC_AMSEL_R &= ~0xC0;          // disable analog functionality on PC7,PC6

}
```

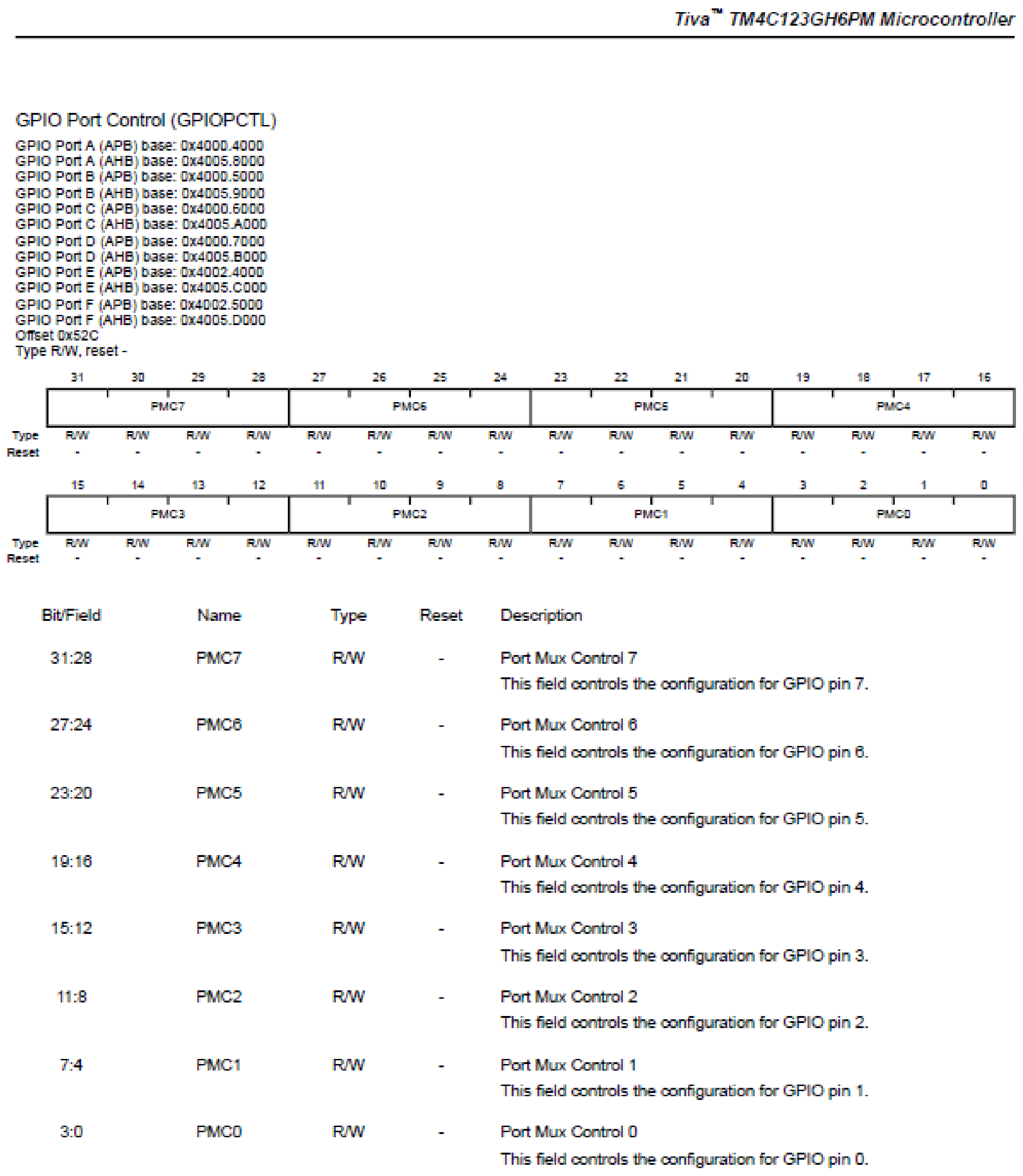
PC_UART3_OutChar function keeps checking if the UART3FR flag register and UARTFR TXFF register are full or not, if they are full (!=0), it means it is not ready to write the data to buffer. PC_UART3_OutString function uses a buffer to takes the transmitting characters from PC_UART3_OutChar function one by one.

```
void PC_UART3_OutChar(unsigned char data){

  while((UART3_FR_R&UART_FR_TXFF) != 0);

  UART3_DR_R = data;

}

void PC_UART3_OutString(unsigned char buffer[]){
```

```
unsigned char i=0;

while(buffer[i]!=0){

        //data=convert_to_ascii();

        PC_UART3_OutChar(buffer[i]);

        i++;

}
```

4.6 Displaying the temperature on Nokie 5110 LCD

Its uses SPI / SSI protocol, The Nokia5110 module needs to send 504 bytes to the LCD to create a new image on the screen. Since the screen is updated 30 times per second, 15120 bytes/sec will flow from the Nokia5110 module to its hardware. PortA is used for the LCD pin out connections. In the SSI initialization, first the SSI0 and port A clocks are activated, then PA6 and 7 are used for the SSI0Fss and CLK pins. Since the other pins in port A are used for SSI pins, I enabled the ALT function register for them and since they will be communicating in binary format, I enabled the digital enable register for them. PCTL register ALT functions is activated for PA2:5 for SSI0 by setting PMC2,3 and 5. And finally the analog function is disabled for pC6 and 7.



Tiva™ TM4C123GH6PM Microcontroller

GPIO Port Control (GPIOPCTL)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x52C
Type R/W, reset -

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	PMC7				PMC6				PMC5				PMC4			
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	PMC3				PMC2				PMC1				PMC0			
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Bit/Field	Name	Type	Reset	Description
31:28	PMC7	R/W	-	Port Mux Control 7 This field controls the configuration for GPIO pin 7.
27:24	PMC6	R/W	-	Port Mux Control 6 This field controls the configuration for GPIO pin 6.
23:20	PMC5	R/W	-	Port Mux Control 5 This field controls the configuration for GPIO pin 5.
19:16	PMC4	R/W	-	Port Mux Control 4 This field controls the configuration for GPIO pin 4.
15:12	PMC3	R/W	-	Port Mux Control 3 This field controls the configuration for GPIO pin 3.
11:8	PMC2	R/W	-	Port Mux Control 2 This field controls the configuration for GPIO pin 2.
7:4	PMC1	R/W	-	Port Mux Control 1 This field controls the configuration for GPIO pin 1.
3:0	PMC0	R/W	-	Port Mux Control 0 This field controls the configuration for GPIO pin 0.

Figure 16. Datasheet page 687

4.7 Pin connection

When D/C is ‘1’, the Data in will be the data that will be displayed on screen. When D/C is ‘0’, the Data in is the command for the LCD, like clear screen and set the curser to the desired x and y positions. The next character should be written after the previous one, which is set by Nokia5110_SetCursor. To print a string of characters to display, Nokia5110_OutString (String) function is used.

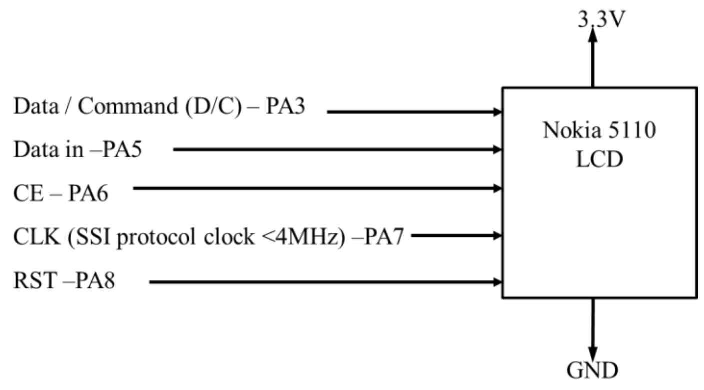

3.3V
Data / Command (D/C) – PA3
Data in –PA5
CE – PA6
CLK (SSI protocol clock <4MHz) –PA7
RST –PA8
Nokia 5110 LCD
GND

Figure 17. Nokia 5110 connections

To clear screen, Nokia5110_Clear(), is used. The LCD configurations and functions are provides by Dr. Volvano’s online course in EdX website.

```
void Nokia5110_OutString(unsigned char *ptr){
  while(*ptr){
    Nokia5110_OutChar((unsigned char)*ptr);
    ptr = ptr + 1;
  }
}
void Nokia5110_SetCursor(unsigned char newX, unsigned char newY){
  if((newX > 11) || (newY > 5)){        // bad input
    return;                             // do nothing
  }
  // multiply newX by 7 because each character is 7 columns wide
  lcdwrite(COMMAND, 0x80|(newX*7));    // setting bit 7 updates X-position
  lcdwrite(COMMAND, 0x40|newY);        // setting bit 6 updates Y-position
}
void Nokia5110_Clear(void){
  int i;
  for(i=0; i<(MAX_X*MAX_Y/8); i=i+1){
    lcdwrite(DATA, 0x00);
  }
  Nokia5110_SetCursor(0, 0);
}
```

5 GSM module setup

The GSM modem did not operate as expected because it is not compatible with U.S. network carriers; I just put a small review of what are the procedures on configuring a GSM modem with a microcontroller.

In order to send and receive text messages to the Launchpad, I used SIM900A GSM module which is compatible with our Launchpad. Once it powers on, it will look for closest 3G network and once it is connected to 3G network, it is ready to send and receive command text messages to the Launchpad [7].

The programmed commands that the GSM module receives are:

1) ‘A’ : this sets the baud rate of the GSM module to match the Launchpad
2) ‘AT’ : GSM responds with ‘OK’
3) ‘ATEO’ : turn off echo
4) ‘AT+CNMI = 1,2,0,0,0’ : indicates when a message is received
5) ‘AT+CMGD = 1,4’ : delete all messages
6) ‘AT+CMGS = 123456789 (enter)

a. > ALERT (enter) : it sends an alert message to the a programmed phone number 123456789
b. > CTRL + Z (enter)

The GSM module communicates with the Launchpad through UART. This generates an UART interrupt.

6 Configuring the Fan as output

In order to turn on the fan when the temperature is more than a set value; we used a simple GPIO port PE1 initialization for its output. In these functions, if the voltage at PE1 is zero, it will turn off the fan, and when the PE1 voltage is Vcc, it will turn on the fan.

To initialize the PF1 as output to control the fan, these steps are followed.

1- Activate port F clock
2- Add dummy delay to clock
3- Disable analog function on PE1
4- Clear PCTL register on PE1
5- Set DIR register on PE1 bit
6- Clear alt function on PE1
7- Enable pull up resistor on PE1
8- Enable digital bit on PE1

```
void Fan_GPIO_Init(){ volatile unsigned long delay;

  SYSCTL_RCGC2_R |= 0x00000010;     // 1) F clock

  delay = SYSCTL_RCGC2_R;           // delay to allow clock to stabilize

  GPIO_PORTE_AMSEL_R &= ~0x02;       // 2) disable analog function

  GPIO_PORTE_PCTL_R &=~ 0x000000F0;   // 3) GPIO clear bit PCTL

  GPIO_PORTE_DIR_R |= 0x02;         // 4.1) PE1 Output,

  GPIO_PORTE_AFSEL_R &=~ 0x02;       // 5) no alternate function

  GPIO_PORTE_PUR_R |= 0x00;         // 6) enable pullup resistor on PE1
  GPIO_PORTE_DEN_R |= 0x02;        // 7) enable digital pins PE1
}
```

To turn on the fan, PE1 pin is set and to turn off the fan, PE1 pin is cleared. This pin later goes to base of a 2N2222A transistor to amplify the voltage at this pin.

C functions:

```
void Fan_Turn_on(){
        GPIO_PORTE_DATA_R |= 0x00000002;
}
void Fan_Turn_off(){
        GPIO_PORTE_DATA_R &=~ 0x00000002;
}
```

6.1 Pin connections

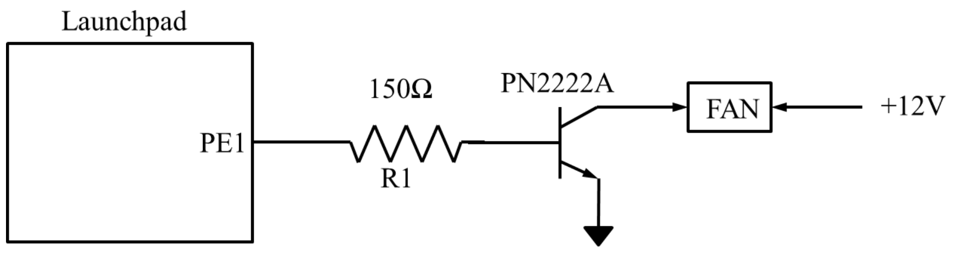


Figure 18. Pin connections for the Fan

6.2 Module initialization

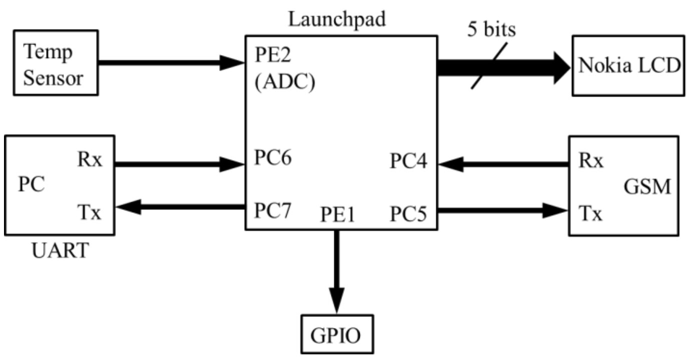


Figure 19. pins of all modules

C functions:

```
Temperature_sensor_ADC0_Init();

timer0A_init();

Nokia5110_Init();

GSM_UART1_Init();

PC_UART3_Init();

Fan_GPIO_Init();
```

7 Main program

The main program contains Convert_data_to_temp(unsigned long data) and the main function in which the Temperature sensor functions initializations, light sensor initializations and sending initial messages to LCD module are declared.

```
int main(void){

        /*

        *  temperature Initializations

        */

        Temperature_sensor_ADC0_Init();
```

```
	timer0A_init();
 Nokia5110_Init();
	Fan_GPIO_Init();
	PC_UART3_Init();
	/*
	*  light Initializations
	*/
	Light_Sensor_GPIO_Init();
	Led_GPIO_Init();
	//delay100();
	/* Sending initial messages to LCD module
	*
	*/
	Nokia5110_SetCursor(0, 0);
	Nokia5110_OutString(TEMP);
	PC_UART3_OutString(TEMP);
```

In the main loop, the ADC data reads the temperature value and passes the ADCdata to Convert_data_to_temp function to calculate the temperature in Fahrenheit. Light_sensor function is initialized here as well. In the if condition, if the ADCdata is less than a programmed value, the program reports the temperature on LCD and PuTTY every second and keep the fan turned off. But if the temperature gets higher than that value, it will display “ALERT” message on LCD and PuTTY and turns on the fan.

```
while(1){
```

```
if (newTemperatureData)

{

    Convert_data_to_temp(ADCdata);



    //light_sensor();

        if(ADCdata < 970) {//800 means round 70 degrees

                    Nokia5110_Clear();

                    Nokia5110_SetCursor(1, 0);

                    Nokia5110_OutString(Temp_s);

                    PC_UART3_OutString(Temp_s);

                    Fan_Turn_off();

                    //light_sensor();

        }

        else{

                    Nokia5110_Clear();

                    Nokia5110_SetCursor(1, 0);

                    Nokia5110_OutString(ALERT);

                    PC_UART3_OutString(ALERT);

                    Fan_Turn_on();

                    //light_sensor();

                    while (ADCdata < 970){
                        Nokia5110_Clear();

                        Nokia5110_SetCursor(1, 0);

                        Nokia5110_OutString(Temp_s);

                        PC_UART3_OutString(Temp_s);

                        Fan_Turn_off();

                        //light_sensor();

        }

    }  // if(ADCdata < 970)



    }  // if (newTemperatureData)

            }  // while(1)

}
```

The complete connections are shown in figure below.

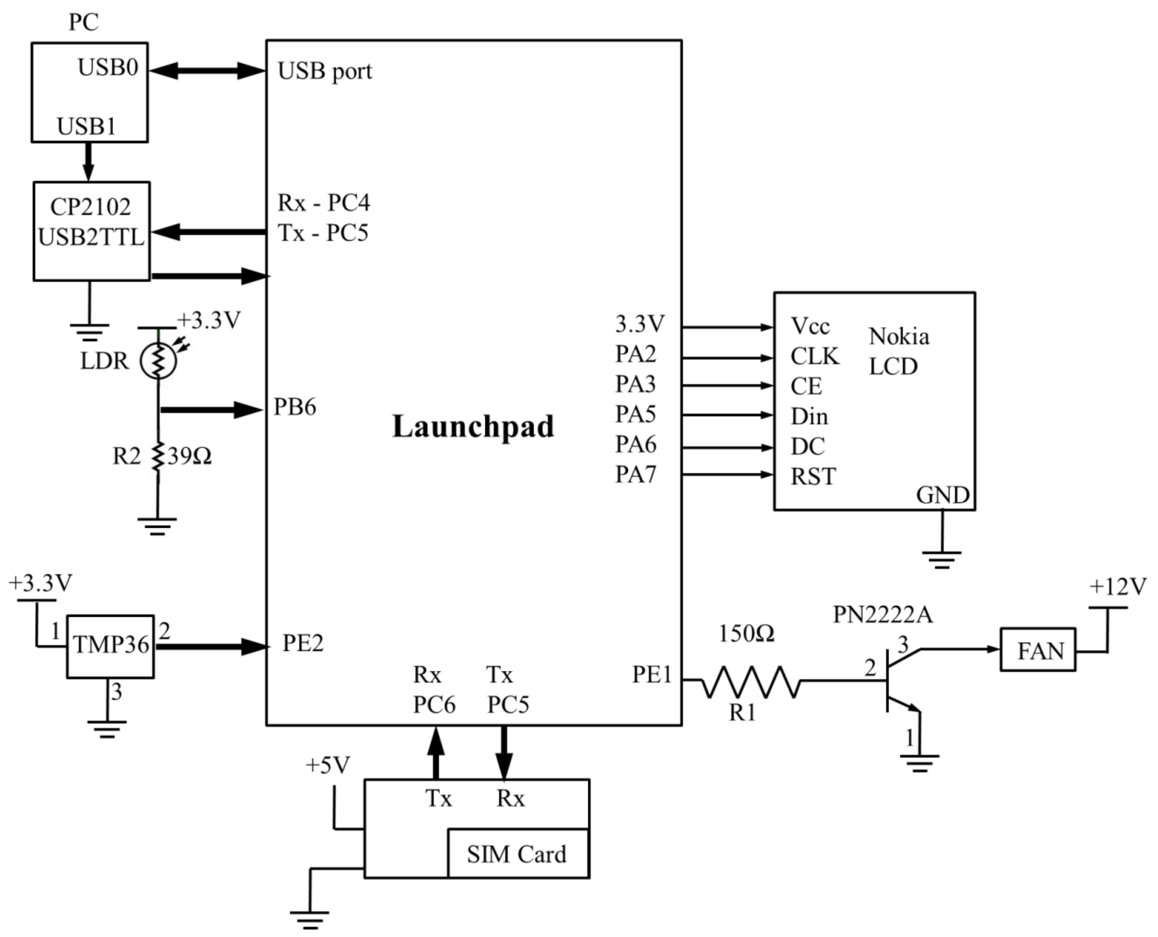


Figure 20. complete connections

8 Conclusion

The home monitoring system containing temperature and light sensors is successfully implemented. The temperature is recorded using ADC and is displayed to the user on two platforms, PC and LCD, every second and if the temperature is higher than programmed value, it will display the ALERT message on both platforms. The light sensor senses the room's light using timer interrupt and if it's dark, it will turn on an LED connected to the launchpad and if the room is bright, it will turn it off. The GSM modem did not operate as expected but majority of the modules are working correctly.

The demos are in my facebook page: https://www.facebook.com/profile.php?id=1329625402

References

[1] T. Perumal, M. N. Sulaiman, and C. Y. Leong, "Internet of Things (IoT) enabled water monitoring system," in *2015 IEEE 4th Global Conference on Consumer Electronics (GCCE)*, 2015, pp. 86-87.

[2] Y. Jie, J. Y. Pei, L. Jun, G. Yun, and X. Wei, "Smart Home System Based on IOT Technologies," in *2013 International Conference on Computational and Information Sciences*, 2013, pp. 1789-1791.

[3] T. Perumal, M. N. Sulaiman, N. Mustapha, A. Shahi, and R. Thinaharan, "Proactive architecture for Internet of Things (IoTs) management in smart homes," in *2014 IEEE 3rd Global Conference on Consumer Electronics (GCCE)*, 2014, pp. 16-17.

[4] B. Boris, Z. Hocenski, and L. Cvitas, "Optimal Approximation Parameters of Temperature Sensor Transfer Characteristic for Implementation in Low Cost Microcontroller Systems," in *2006 IEEE International Symposium on Industrial Electronics*, 2006, pp. 2784-2787.

[5] L. Yanping, Q. Jianbin, and W. Tao, "Realization of heat management system based on low voltage power line carrier," in *2010 8th World Congress on Intelligent Control and Automation*, 2010, pp. 4220-4224.

[6] www.ti.com/lit/gpn/tm4c123gh6pm. (2017). *Tiva™ C Series TM4C123GH6PM Microcontroller Data Sheet (Rev. E)*.

[7] www.propox.com/download/docs/SIM900_AT.pdf. (2012). *AT Commands Set - propox*.