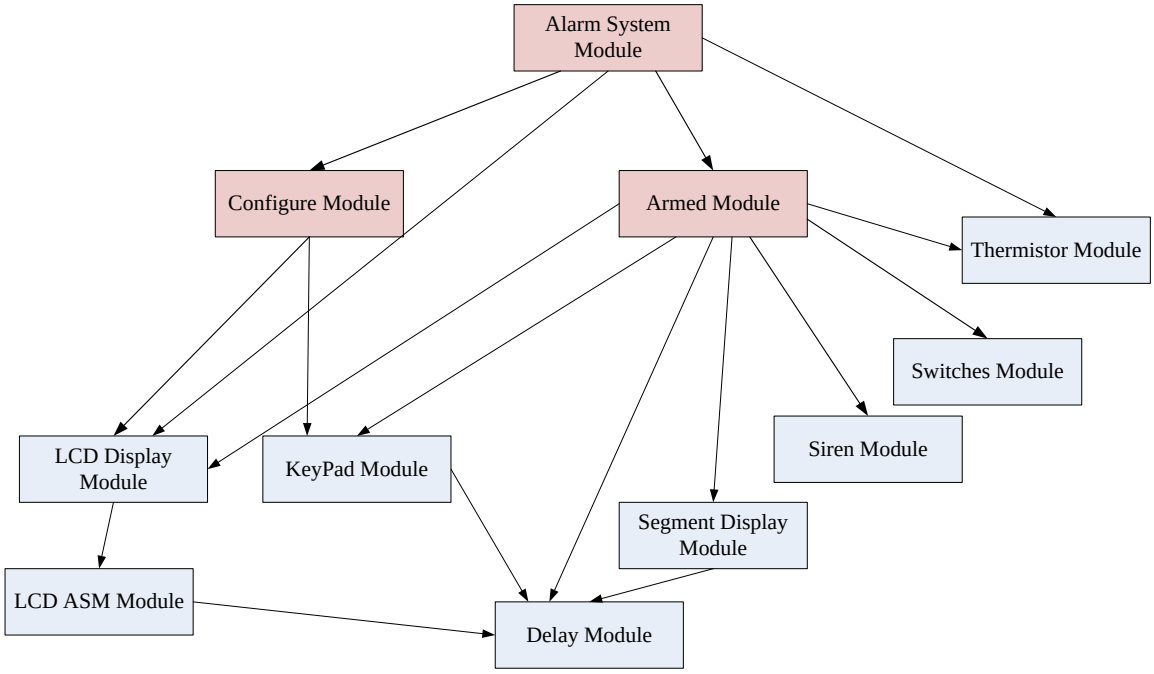
**Lab5 – Using the A/D Converter**
**Design Document**

The main objective of the lab is to use the ATD to monitor temperature. A thermistor on the Dragon-12 card serves as a transducer to translate temperature to an electrical signal. A new module, the Thermistor module provides the means to update regularly a variable that reflects the temperature. This module will be integrated into the Alarm System project to emulate a smoke detector function that turns on the alarm when fire conditions exist. The code will be developed using modular design starting with modules developed during lab 4.

**Modules**




Update to SegDisplay – manipulate only lower 4 bits of PortP

**Test Project**: Contains the main code that obtains a Celsius value from the thermistor module (see Thermistor module below), and writes the value on the 7-segment displays. In addition, an LEDs is lit when the temperature is above 26 $^{o}$C. Note that the LEDs are managed by this module. Pin 4 of Port P is used to control the LED (recall that pins 0 to 3 are used to control which 7-segment display is active).

**Segment Display Module:** This module consists of the module developed in Lab 4 with 2 additional routines to manage decimal points on the displayThe subroutines required in this lab consist of:

- `turnOnDP(int dNum)`; This function turns on the decimal point of the display identified by dNum. Note that adding a new character to display should not affect the decimal point.

- `turnOffDP(int dNum)`; This function turns off the decimal point of the display identified by dNum. Note that adding a new character to display should not affect the decimal point.

**Thermistor Module:** This module provides two entry point functions:

- `void initThermistor(void)`: This functions initializes the ATD peripheral module to read the temperature every 100 ms and update a conversion variable (16-bit memory location) with the 10-bit conversion value obtained. Use one of the Timer channels to generate an interrupt every 100 ms; the corresponding ISR should initiate a conversion in the ATD which should be configured for a single conversion on pin PAD05. At the end of the conversion, an interrupt should be generated and the corresponding ISR will update the variable. HINT: counting 100 ms with 1.333 tick time.
- `int getTemp(void)`: This function reads the 10-bit conversion value found in the conversion variables and translate it to an integer value between (000 and 700, i.e. 00.0 $^{o}$C and 70.0 $^{o}$C). The converted value is returned.

**Updates to design of Alarm System for Monitoring Temperature**

The alarm system modules Alarm and Armed have been modified as follows to deal with monitoring temperature:

1) Temperature is monitored continually, even when system is disarmed. When temperature exceeds a preset value (27 to make the lab practical) the alarm siren will be turned on by calling the `triggerAlarm()` function found in the Armed module (it is necessary to turn off the alarm by entering a valid code). There are two places that the alarm can be triggered by a high temperature: when the system is disarmed (i.e. in the main function of the Alarm module awaiting a command) and when the system is armed (after entering a valid code to arm the system and after the 10 second delay). All other times (during delays or during configuration) the alarm will not be sounded if the temperature rises. There is some logic since during those times a person is in the home interacting with the alarm console. For the purpose of the lab and this course, this assumption keeps the alarm code simpler.
2) The temperature is continually displayed on the 7-segment display (and the temperature LED lit) by calling the functions provided in the Segment diplay module. Only during countdowns when arming and disarming the alarm system, will the display not show temperature. To simplify the software, an interrupt service routine is responsible for reading the termperature from the Thermistor module and updating the display with its current value using the functions available in the Segment display module. A variable `displayTempFlag` controls whether the temperature should appear on the 7-segment displays: set to `TRUE` (1) to display the temperature and `FALSE` (0) to stop displaying the temperature.

Timer Channel 0: Delay Module
Timer Channel 1: Segment Display Module
Timer Channel 2: Thermistor Module
Timer Channel 5: Siren Module
Timer Channel 6: Alarm Module

**Main Module**
**Lab5Test**

This module consists of the main program that calls two subroutines. It also contains an interrupt service routine the services the abort signal. When such a signal is received, any countdown is aborted. The variable "abort" is set when the interrupt is received.

Main program: Main program that loops infinitely. It first calls the Thermistor module to obtain a current temperature (getTemp), displays the temperature on the 7-segment displays, then lights the RGB LED if the temperature is high or low.

**Subroutine**: initMain

The main module initializes the Timer with a clock tick of CLOCK_TICK (2.6666 micro-seconds). The channels are configured by the initialization routines of modules that use the timer. Call initDisp and initTherm to initialize the modules. Then it initializes PORTP (bits 4 and 5) as an output port to use pins 4 and 5 for that controls the RGB LED.

**Thermistor Module**

Routine that controls the ATD process for reading the temperature. Two functions are provided:

InitTherm – to initialize the ATD module. Initializes the ATD as follows:

- Setup interrupt vector of ATD to atdISR
- Enable (delay of 20 microseconds), fast clear, enable interrupt:
  ATDCTL2 ← %1100 0010.
- 10-bit resolution, maximum hold (16 cycles), ADT clock 500 kHz (prescaler of 48 – 10111):
  ATDCTL4 ← %11110111
- Single conversion, no FIFO:
  ATDCTL3 ← %0000 1000
- Single pin, pin 5
- Enable interrupt on conversion.

Configure Timer Channel 1 for 100 ms interrupt (2.6666 microsec TCNT tick)

- Setup interrupt vector of channel TC1 to thermISR
- Set IOS 1 in TIOS to 1
- Set C01 to 1 in TIE
- Add TEMP_TIMEOUT (37500) to TCNT and store in TC0

**getTemp** –

***8-bit Conversion value:***

Converts 8-bit value at currentTemp to a hex value of the current temperature.

$$V_{OUT} = T_C T_A = (10.0 T_A) mV$$

where

- $V_{OUT}$ is the LM45 output voltage applied to the PAD05,
- $T_C$ equals 10 $^oC$/mV (coefficient temperature),
- $T_A$ is the ambient temperature.

Using 8-bit conversion with $V_{HREF}$ = 5V, and $V_{LREF}$ = 0V,

$$T_b = 255\frac{V_{out}}{5000} = \frac{255}{5000}V_{out}$$

$$V_{out} = \frac{5000}{255}T_b$$

Where $T_b$ is the binary value representing the temperature. By substituting for $V_{out}$ in the temperature equation, the ambient temperature $T_A$ is related to the binary value $T_b$ as follows:

$$V_{OUT} = 10T_A$$

$$\frac{5000}{255}T_b = 10T_A$$

$$T_A = \frac{5000}{2550}T_b \approx 2T_b$$

The function will take the value found in currentTemp double it, and return the hex value.

***10-bit Conversion value:***

Converts 10-bit value at currentTemp to a hex value of the current temperature.

$$V_{OUT} = T_C T_A = (10.0 T_A) mV$$

where

- $V_{OUT}$ is the LM45 output voltage applied to the PAD05,
- $T_C$ equals 10.0 mV/$^o$C (coefficient temperature),
- $T_A$ is the ambient temperature.

Using 10-bit conversion with $V_{HREF}$ = 5V, and $V_{LREF}$ = 0V,

$$T_b = 1023 \frac{V_{out}}{5000} = \frac{1023}{5000} V_{out}$$

$$V_{out} = \frac{5000}{1023} T_b$$

Where $T_b$ is the binary value representing the temperature. By substituting for $V_{out}$ in the temperature equation, the ambient temperature $T_A$ is related to the binary value $T_b$ as follows::

$$V_{OUT} = 10 T_A$$

$$\frac{5000}{1023} T_b = 10 T_A$$

$$T_A = \frac{5000}{10230} T_b = 0.488 T_b$$

Given we are dealing with integers, and wish to display 10ths of degrees, multiply the expression by 10, such that the resulting integers gives the numbers of 10$^{th}$`s of degrees:

$$T_A = (0.488 T_b) 10 = 4.88 T_b$$

The above formula can be implemented using arithmetic with real numbers or with integers only if 4.88 is rounded to 5. This would introduce a 2.5% error. Testing has shown the rounding error: 24.8 degrees is shown as 25.5 degrees . The function will take the value found in the global variable `temperature`, multiply by 10, divide by 4, and then subtract 205. The returned results represents 10$^{th}$`s of degrees which can be displayed accordingly.

1 bit = 5000mV/1023 = 4.9 mV

Coefficient = 10 C/mV

4.9 mV / 10 mV/C = 4.9/10 = 0.5 C

**thermISR**: Starts a conversion sequence.

Writing into ATDCTL5 starts a conversion sequence. Configures ATDCTL5 for left justification, unsigned, no scan, single pin, on pin 5 as follows:

- 0000 0101

Also turns off the Timer interrupt for TC1 by adding TEMP_TIMEOUT to TC1.

**atdISR**: Turns off interrupt and stores conversion value into `currentTemp`.

Reading the result register ATDR0 turns of the interrupt (fast clear), and store the value read from ATDR0 into `currentTemp`.

**Segment Display** Entry points:

- initDisp – initialises port **B** to work with the 7-segment displays, i.e. all outputs and initialises PORTE to all outputs and Port P to outputs for selecting displays to activate. Also configures Timer channel 0 to invoke an interrupt every 50 ms to run dispISR.
- dispHexNum – sets up the hex temperature received (translates the code) for display on the 7-segment displays.

Local Subroutines/Interrupt service routines

- setCharDisplay – sets the given character (translates the code) for display on specified 7-segment display.
- getCode – translates an ASCII character to the code for the 7-segment display.
- dispISR – displays characters on the 7-segment displays using stored codes, changing displays at each interrupt.
- clearDisp – clears the display (writes 0 to the dispCodes).

Use the 4 bytes to define characters to appear on 7-segment displays. Note that in the case of the second byte, the decimal point may be encoded.

**Subroutine**: initDisp

Using the DDRB and DDRP registers, to set all pins as output pins. Also configure TC0 for generating interrupts every 50 ms.

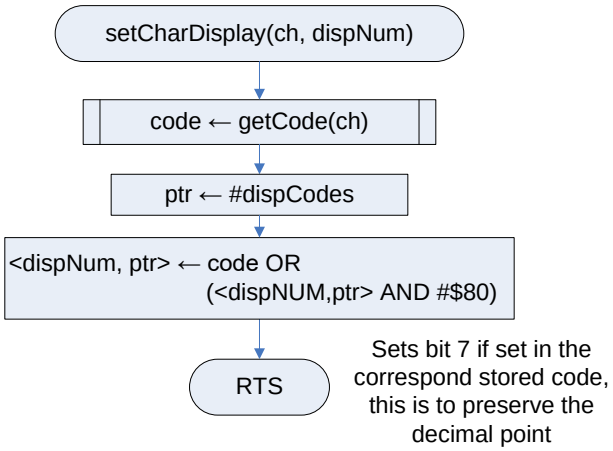
- Setup interrupt vector of channel TC1 to thermISR
- Set IOS 0 in TIOS to 1
- Set C0I to 1 in TIE
- Add DISP_TIMEOUT (18750) to TCNT and store in TC0

**Subroutine**: clearDisp(ch, dispNum)
Simply writes zeros into the 4 byes occupied by dispCodes.

**Subroutine**: dispHexNum(hexNum)
The parameter hexNum is an 8 bit hex value. Displays the the number as three decimal digits on the displays. Takes each hex value, converts to ASCII value and calls setCharDisp to setup the characters.

**Subroutine**: setCharDisplay(ch, dispNum)

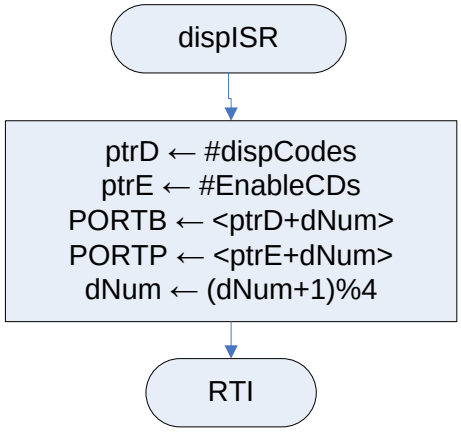
Codes the character given by “ch” for the display with identifier “dispNum” (0 to 3, where 0 is the leftmost display).




**Subroutine**: dispISR

Displays the codes in the code display table (contains four character codes) on the 4 displays. This is accomplished switching display when an interrupt occurs. Use `dNum` to indicate the next display to activate. To enable the displays the following table is used:

```
EnableCDs  db 0x%11111110   ; Display 0
           db 0x%11111101   ; Display 1
           db 0x%11111011   ; Display 2
           db 0x%11110111   ; Display 3
```

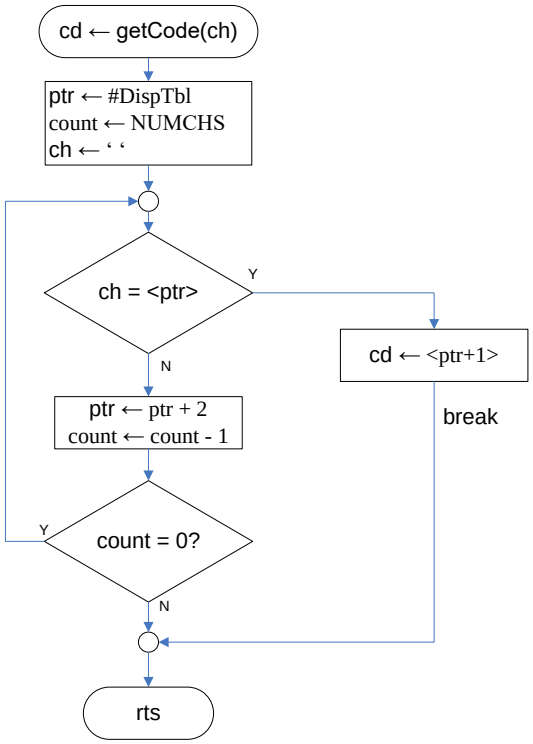


**Subroutine**: cd ← getCode(ch)

This subroutine uses a table to convert ASCII codes to the code for storing into Port B to display corresponding number on display.

The DISPTBL contains the following data for translating ASCII codes to the display codes:

```
                    gfedcba
dispTbl   dc.b '0',%00111111
          dc.b '1',%00000110
          dc.b '2',%01011011
          dc.b '3',%01001111
          dc.b '4',%01100110
          dc.b '5',%01101101
          dc.b '6',%01111101
          dc.b '7',%00000111
          dc.b '8',%01111111
          dc.b '9',%01101111
          dc.b '*',%01000110
          dc.b '#',%01110000
          dc.b 'A',%01110111
          dc.b 'B',%01111100
          dc.b 'C',%00111001
          dc.b 'D',%01011110
          dc.b ' ',%00000000   ; space
```



Circuit Diagram :