CMPE423 Embedded System Design

ADC and FSM Application

Objective:

This application demonstrates design of a PIC18FX52 system to read the water level by an ADC in the range 0..5V as an 8-bit integer variable, and, to manipulate a pump and an alarm output designing a Finite-State-Machine.

ADC Configuration, Reading, and UART settings

PORTA and PORTE of PIC18FX52 devices are analog input pins.

Digital/Analog/Reference settings of the i/o pins.
A: Analog, D:Digital.

Microcontroller Pins										ADCON1 [3..0] PCFG
both PIC18F252 and PIC18F252					only PIC18F452					
RA0	RA1	RA2	RA3	RA5	RE0	RE1	RE2	Vdd	Vss	
D	D	D	D	D	D	D	D			**011x**
A	D	D	D	D	D	D	D	V_{R+}	V_{R-}	**1110**
A	A	D	A	D	D	D	D	V_{R+}	V_{R-}	**0100**
A	A	A	A	A	D	D	D	V_{R+}	V_{R-}	**0010**
A	A	A	A	A	A	D	D	V_{R+}	V_{R-}	**1001**
A	A	A	A	A	A	A	A	V_{R+}	V_{R-}	**0000**
A	A	D	V_{R+}	D	D	D	D		V_{R-}	**0101**
A	A	A	V_{R+}	A	D	D	D		V_{R-}	**0011**
A	A	A	V_{R+}	A	A	D	D		V_{R-}	**1010**
A	A	A	V_{R+}	A	A	A	A		V_{R-}	**0001**
A	D	V_{R-}	V_{R+}	D	D	D	D			**1111**
A	A	V_{R-}	V_{R+}	D	D	D	D			**1101**
A	A	V_{R-}	V_{R+}	A	D	D	D			**1100**
A	A	V_{R-}	V_{R+}	A	A	D	D			**1011**
A	A	V_{R-}	V_{R+}	A	A	A	A			**1000**

ADC related configuration registers.

SFR	bit	Explanation
ADCON0	7,6	ADCS1, ADCS0 f_{AD} setting. 00: $f_{AD}= f_{osc}/2$; 01: $f_{AD}= f_{osc}/4$; 10: $f_{AD}= f_{osc}/16$; 11: $f_{ADC}= f_{AD.RC}$=450kHz (internal RC)
	5,4,3	0,..,7 selects RA0,RA1,RA2,RA3,RA5,RE0,RE1,RE2
	2	GO_DONE : Set to begin conversion.
	1	don't care
	0	ADON : 1=ADC is powered up
ADCON1	7	ADFM: ADC result format, 0: MSB justified, ADRESH;ADRESL = xxxxxxxx xx000000 1: LSB justified, ADRESH;ADRESL = 000000xx xxxxxxxx
	6	ADCS2 , divides f_{AD} by 2. $f_{ADC}= f_{AD}$ / (ADCS2+1)
	5,4	don't care
	3,2,1,0	PCFG: Port configuration bits
ADRESH	7,..,0	8-bit ADC result register.
ADRESL	7,,0	Lower byte of 16-bit ADC result register.

f_{ADC}= AD clock freq. For correct conversion 160 kHz < f_{ADC} < 625kHz;

$T_{ADC} = 1/F_{ADC}$ Conversion takes $T_{AD} = 12\ T_{ADC}$, = almost 25μs

8-bit: $N=(V_{in} - V_{Ref-})*255/(V_{Ref+} - V_{Ref-})$; 10-bit: $N=(V_{in} - V_{Ref-})*1023/(V_{Ref+} - V_{Ref-})$

Depending on the required number of analog inputs, we shall pick the most suitable configuration from the ADC-io-pin configuration table, and set PCFG bits of ADCON1 accordingly. The ADC unit has an independent clock, which is generated either by an internal RC oscillator or by dividing the oscillator clock. The ADC clock frequency is set by three bits

$f_{ADC}=fosc / 2^{\{ADCS1, ADCS0, ADCS2\}_{bin}+1}$.

$160\ kHz < f_{ADC} < 625kHz$ provides conversion at 10-bit resolution. {ADCS1, ADCS0, ADCS2 }={1,1,X} enables the internal RC clock of ADC at f_{ADC}=**450kHz** for all conditions and sets $f_{ADC}= f_{AD.RC}$=450kHz.

Pump and Alarm Problem statement and Design Specifications

A water tank has a level detector with analog output V_L. Detector gives V_L = 0V if tank is empty, and V_L =1V if tank is full. Design a 18F452 circuit to read V_L, to manipulate a pump connected to RB7, and to give alarm if necessary through RB6 by the following procedure.

1. If $V_L \leq 0.1$ volt then stop pumping water,
2. If $V_L > 0.6$ volt then start pumping water,
3. If $V_L > 0.9$ volt then start alarm,
4. If $V_L \leq 0.6$ volt then stop alarm.

Design Preliminaries

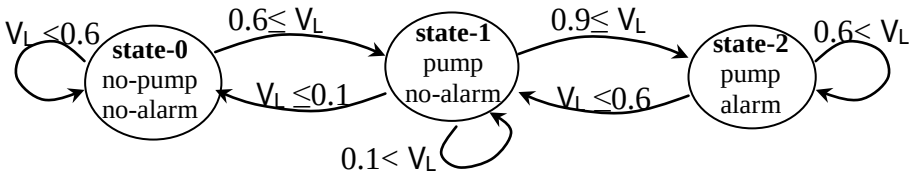
We shall draw the FSM for the pump and alarm outputs depending on V_L. Next we shall find settings of configuration for ADC of V_L to **NVL** from pin RA0. Then, we shall calculate the reading **NVL** of ADC for the values V_L =0.1V, 0.6V, and 0.9V. These values are the bounderies of inequalities in the problem statement, that partitions the range of ADC readings in four intervals. In the mainloop, we shall test **NVL** against these numbers to determine the interval of V_L . This will simplify FSM table by reducing number of inputs. After all these preliminaries we may start to write the complete CC8E program for the described operation.

Develop a solution by an FSM chart:

We need a starting state of FSM to initialize the state before the mainloop.

- **State-0** is the starting state of the FSM, that stops both pump and alarm by making pump=0 and alarm=0. FSM remains at state-0 while $V_L \leq 0.6$. The only exit from state-0 is a transition to state-1 when $V_L > 0.6$.
- **State-1** turns pump on, but keeps no-alarm while $0.1 < V_L \leq 0.9$. State-1 exits to State-0 with $V_L \leq 0.1$ and it exits to state-2 with $V_L > 0.9$.

- **State-2** turns both pump and alarm on while V_L >0.6. It exits to state-2 when $V_L \leq 0.6$.



Reading the analog voltage into 8-bit integer `NVL`:

ADC related configuration registers are **`TRISA`**, **`ADCON0`**, and **`ADCON1`**. We must set **`TRISA.0`**=1 to make RA0 input pin for ADC. **`ADCON0`**= **`0b11.000.0.0.1`** which corresponds to RC oscillator mode for ADC, RA0 selected for conversion, not yet started, don't care, and ADC is powered. **`ADCON1`**=**`0b0.0.00.1111`** which provides 8-bit format, RC-ADC mode, don't care, Analog-RA0 with external reference voltages. In the circuit, RA2 shall be connected to ground, and RA3 shall be connected to a V_{R+} =1Volt reference voltage source. We may also use internal lower reference voltage by choosing PCFG=**`0101`**, and connecting only RA3 to V_{R+} =1 Volt reference voltage source, setting **`ADCON1`**=**`0b0.0.00.0101`**.

Boundaries of VL and NVL

We plan to read ADC-result in 8-bit format with V_{R+}=1V. The expected digital readings are

`NVL1=` ADC-reading for 0.1V=255*0.1/1=25.

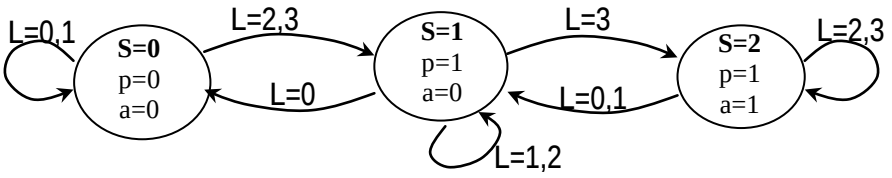
`NVL2=` ADC-reading for 0.6V=255*0.6/1=153,

`NVL3=` ADC-reading for 0.9V=255*0.9/1=229.

Simplification of FSM using enumerated partitions of NVL

To simplify implementation of FSM, we plan to enumerate the ranges of V_L by **`L=0,1,2,3`** for four intervals of V_L. Consequently **`L=0`** means V_L <=0.1 (equivalently **`NVL<25`**); **`L=1`** means 0.1<= V_L <0.6 (i.e., **`25<=VL<153`**); **`L=2`** means 0.6<= V_L <=0.9 (i.e., **`153<=NVL<=229`**), and **`L=3`** means V_L >0.9 (**`NVL>229`**).

transition condition: **`L`**	0	1	2	3
input range: V_L	$V_L \leq 0.1$	$0.1 < V_L < 0.6$	$0.6 < V_L < 0.9$	$0.9 \leq V_L$
	`VL<26`	**`25<VL<154`**	**`153<VL<231`**	**`230<VL`**



Next, we may prepare the next state table, and the output tables using the input+state index **`i= 4*S+L`**, so that entries 0,1,2,3 keep the next states for

current state-0, while entries 4,5,6,7 keep the next states are for state-1 and entries 8,9,10,11 are for state-2.

We will write the state-transition-table of the FSM in Moore style, where the inputs column and Next-state column contain a list of items instead of a single item, and the outputs are directly produced from the present state. It provides a dense and convenient representation of the next state and output tables. The output specifies both Pump, and Alarm flags which are placed to the port through bit-7 (Pump) and bit-6 (Alarm).

State S	Input L	Next-State S	Output to PORTB 0b.paxxxxxx
0	0, 1, 2, 3	0, 0, 1, 1	0b.00000000
1	0, 1, 2, 3	0, 1, 1, 2	0b.10000000
2	0, 1, 2, 3	1, 1, 2, 2	0b.11000000

We construct the next state table index by `i= 4*S+L`, which fits to the placement of the variables `S` and `L`, and number of values of `L` in the table.

Next state table for `i= 4*S+L`, is

```
NSTT={ 0, 0, 1, 1,   0, 1, 1, 2,   1, 1, 2, 2}
```

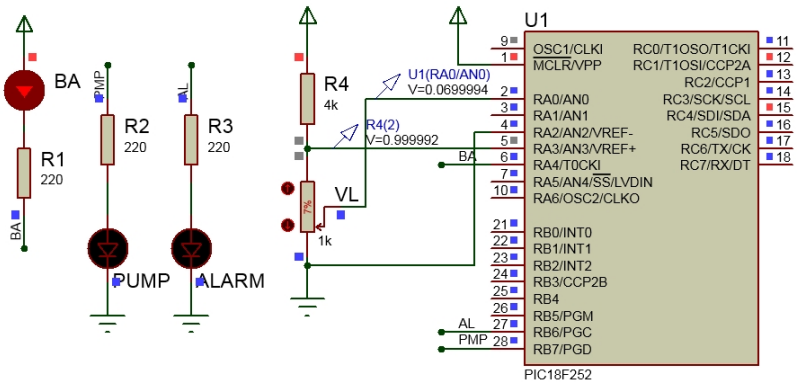
Outputs depend only on state `S`:

```
OUTT[]={ 0b00000000, 0b10000000, 0b11000000 };
```

Circuit Schematics and CC8E coding of the software

Our main program will contain initialization code and mainline loop code. `Initialize()` is planned for initialization, `ADC0()` is planned to put the ADC conversion of RA0 into `NVL`, `FSM()` is planned to calculate `L` from the ranges of `NVL`, to use it to find `i`, the next state index, and to read the next state from the next state table. The outputs are read from an output table.

For this circuit, we plan to use a Blink-Alive LED connected to RA4, and two LEDs, PUMP and ALARM to indicate pump and alarm outputs. We obtain an adjustable voltage in the range 0…1V by a voltage divider with a potentiometer (R4 and VL) supplied with +5V. The reference voltage V_{R+} is obtained from the same voltage divider.



```
1.  //-----------------------------------------------------------------
2.  // Mehmet Bodur (c) 2012 Exercise program for water tank pump and alarm
3.  // RA4 is blinkalive  Fosc=4MHz
4.  // RA0 is analog VL input
5.  // RA2 is ground, and  RA3 is external 1V reference
6.  // RB6 is pump output;   RB7 is alarm output
7.  // FSM has 3 states, one input, two outputs
8.  // Input L=0 if NVL<26; L=1 if 25<NVL<154;
9.  //       L=2 if 153<NVL<231; L=3 if NVL>230
10. // Outputs (P,A) Pump=1 runs pump, Alarm=1 runs alarm.
11. // States St=0 gives P=0,A=0; keeps St=0 with L=0,1;
12. //      exits to S=1 with L=2,3
13. // St=1 gives P=1, A=0; keeps St=1 with L=1; exits to St=0 with L=0;
14. //      exits to St=2 with L=3
15. // St=2 gives P=1, A=1; keeps St=2 with L=2,3; exits to St=1 with L=0,1
16. char St, NVL, L, i; //global variables for state, Vin, intervals and
    index
17. const char NSTT[]={ 0, 0, 1, 1,   0, 1, 1, 2,   1, 1, 2, 2},
18.            OUTT[]={ 0b00000000, 0b10000000, 0b11000000 };
19. // ADC settings for VL, 8-bit, RC clock, RA0, ext.ref.
20. bit ADC_go_done @ ADCON0.2;
21. #define ADCON0_VL 0b11.000.0.0.1
22. #define ADCON1_VL 0b0.0.00.1111
23. // 8-bit ADC for VL corresponding to 0.1V, 0.6V and 0.9V
24. #define  NVL1  25
25. #define  NVL2 153
26. #define  NVL3 230
27. // Timer0 settings for 100ms and for 1s with 4MHz Xtal
28. //  16-bit counts with 32 prescaler 1000000/32=31250 cc for 1s
29. #define  T0CON_100ms 0b10000100
30. #define  TMR0_100ms (65536-3125)
31. #define  T0CON_1s  0b10000100
32. #define  TMR0_1s (65536-31250)
33.
34. void Initialize(void){
35.    TRISA= 0b00001101; // RA0 Analog-inp; RA2,RA3 ref-inp; RA4 LED-out
36.    TRISB= 0;   // RB6 RB7, Pump and Alarm output
37.    TRISC= 0;   // For ADDC test
38.    ADCON0 = ADCON0_VL; ADCON1 = ADCON1_VL;
39.    }
40. void ADC0(void){
41.    ADC_go_done=1; do{}while(ADC_go_done); // wait ADC to be over
42.    }
43. void FSM(void){
44.    char i,ix;
45.    L=0; if(NVL>NVL1) L=1; if(NVL>NVL2) L=2; if(NVL>NVL3) L=3;
46.    i=St*4 + L;
47.    St=NSTT[i];
48.    PORTB=OUTT[St];
49.    }
50. void SetTimer0(char CON, uns16 TN){
51.     T0CON=CON; TMR0H=TN/256;TMR0L=TN%256; TMR0IF=0;
52.     }
53. void BlinkAlive(void){
54.    if(TMR0IF){
55.       if(PORTA.4){ // turn on LED for 100ms
56.                SetTimer0(T0CON_100ms,TMR0_100ms); PORTA.4=0;}
57.             else { // turn off LED for 1s
58.                SetTimer0(T0CON_1s,TMR0_1s); PORTA.4=1;}
59.        }
60.    }
61. void main(void){
62.    Initialize();
63.    St=0;
64.    SetTimer0(T0CON_100ms,TMR0_100ms);
65.    do{   // mainloop
66.      ADC0(); NVL=ADRESH; PORTC=NVL;
67.      FSM();
68.      BlinkAlive();
69.      }while(1);
70.    }
```

The `blinkalive()` procedure is polling the timer0 timeover flag at every pass of mainloop. On timeover condition it sets the timer depending on the status of blink-alive LED, so that the on and off periods of the LED are set to 0.1s and 1s, respectively.