Frank Yee (109374525), Liting Zhang (110036278)

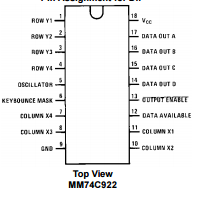
Laboratory 10: HVAC System I

Section: 01

Bench: 07

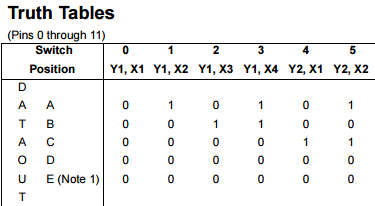
Theory of Operation (Hardware + Software)

The hardware introduced in this lab was the MM74C922 16-Key Encoder. The integrated circuit can then scan upto 16 keys in a logical key matrix. The IC MM74C922 16-Key Encoder then interacts with the ATmega16A micro-controller , by “telling” it which keys or pushbuttons were pressed. The pin assignment for MM74C922 16-Key Encoder can be seen below:

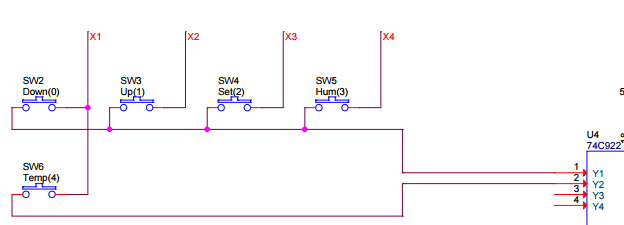


It is important to note that because output enable is active low, it is necessary to ground the pin to enable outputs. Key bounce-mask is present to debounce the buttons, therefore the debouching circuitry use for the flip-flop’s clock can be scrapped, and software debouncing is not needed. Data available manages the clock of the flip-flop.

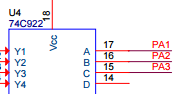
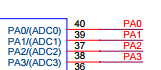
In this lab we only use **FIVE (5)** keys: T (Temperature-keycode 4)**,** H (Humidity-keycode 3), S (Set-keycode 2)**,** U (Up-keycode 1)**,** D (Down-keycode 0)**.** With these designated key-codes, proper wiring is necessary to align the keys with the corresponding truth-tables (from datasheet):



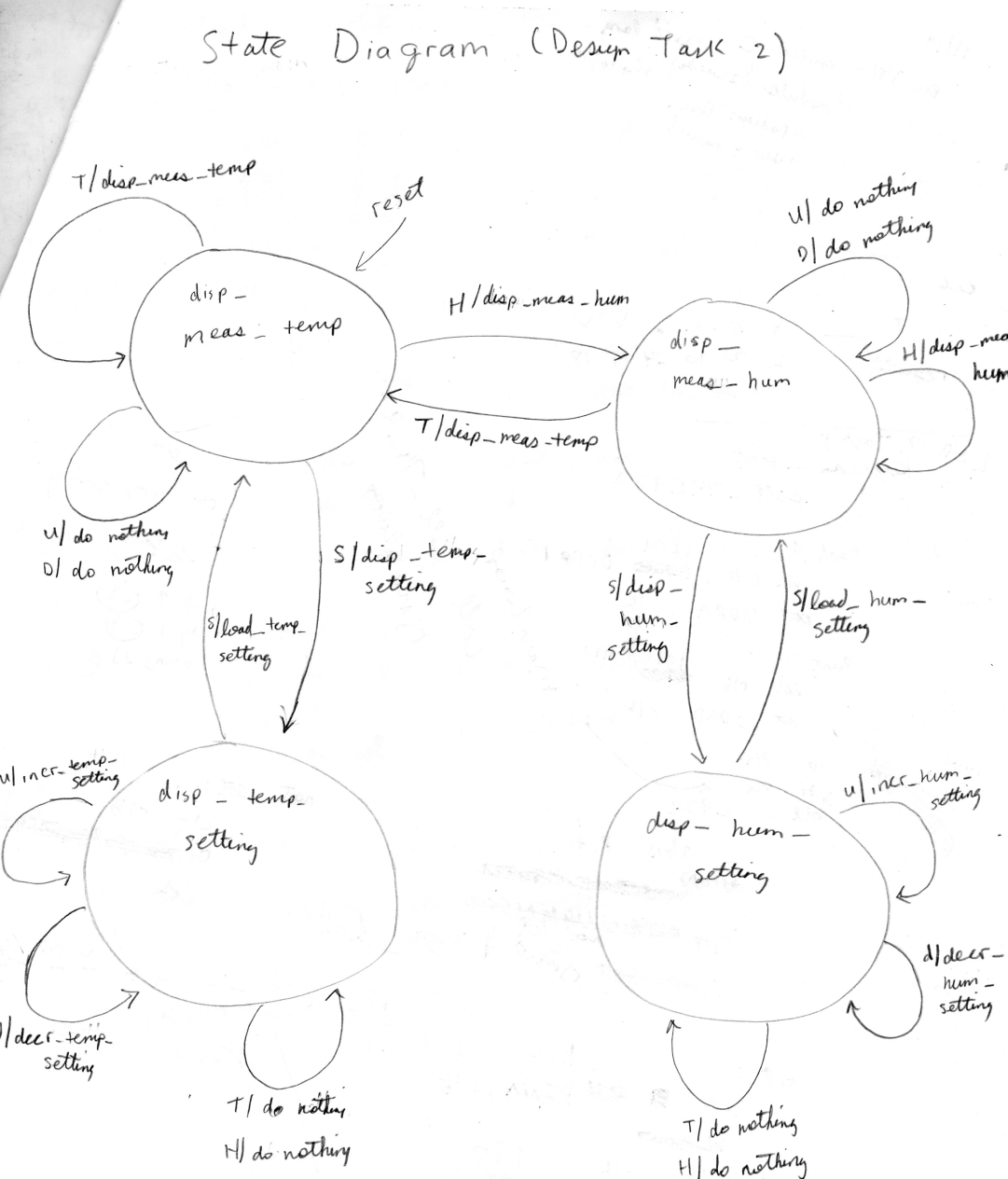
Following the designated key codes and corresponding truth-table it results in the following schematic only Y1 and Y2 are used:



Based on the truth-table, the only DATA OUT bits being modified are the least significant 3. (DATAOUT D and E are all 0s for the first 5 keys). Therefore from the MM74C922, only the 3 least significant bits (C, B, A) are connected to PA3 through PA1 respectively. As seen from the schematics that follow, the outputs of the MM74C922 are read by port A of the ATmega16A. (PA0 is still connected to the same sensor from previous lab).



With the hardware properly wired and configured. The software must be written in a format to follow set key sequence commands. To design the finite state machine, see the diagram below:



The sequence list associated with the lab tasks are as follows:

* T: Displays the measured temperature
* TS: Displays the temperature setpoint
* TSU...U: Displays the temperature setpoint and increase the respectiv setpoint by 1 degree C for each press of U.
* TSU...US: Displays the temperature setpoint and increase the respective setpoint by 1 degree C for each press of U. And when S is pressed again, change the temperature setpoint to the new value and returns to T: Displays the measured temperature.
* TSD...D: Displays the temperature setpoint and decrease setpoint by 1 degree C for each press of D.
* TSD...DS: Displays the temperature setpoint and decrease setpoint by 1 degree C for each press of D. And when S is pressed again, change setpoint to the new value and returns to T: Displays the measured temperature.
* H: Displays the measured humidity
* HS: Displays the humidity setpoint
* HSU...U: Displays the humidity setpoint and increase the respective setpoint by 1 %RH for each press of U.
* TSU...US: Displays the humidity setpoint and increase the respective setpoint by 1 %RH for each press of U. And when S is pressed again, change setpoint to the new value and returns to H: Displays the measured humidity .
* HSD...D: Displays the humidity setpoint and decrease the respective setpoint by 1 %RH for each press of D.
* HSD...DS: Displays the humidity setpoint and decrease the respective setpoint by 1 %RH for each press of D. And when S is pressed again, change setpoint to the new value and returns to H: Displays the measured humidity.

As an example, in the actual software the finite state table is implemented as:

state\_table:

s0: .dw i2, s1, task1

.dw i3, s1, task1

.dw eol, s0, task2

s1: .dw i1, s0, task0

.dw i3, s0, task0

.dw eol, s1, task2

Where s0: and s1: are the current states, i0-i3 and eol are defined as input symbols. These input symbols are based on the keys read by the microchip in the poll\_74C922 subroutine. The poll\_74C922 subroutine scans the keys and determines which buttons were pressed. The input symbol eol, stands for end of list. It is very critical to include said input symbol (eol), because it prevents the system from being left hanging, should some key sequence was not explicitly defined the program reaches the end of the list, also works as an otherwise functionality.

The next column after input symbols, are the next state address. It directs the software to the next corresponding state in the state table, s0: -> s1: (etc). The last column contains the task subroutine addresses. It directs your software to the next subroutine to be performed, in other words task.

It is important to note that the software finite state machine utilizes pointers to direct the next state and corresponding task. Therefore initialization of the stack pointer is crucial to the finite state machine’s functionality, otherwise without address the system does not know how to proceed or process the keys scanned.

Below is the code for the finite state machine, detailed with comments explain the functionality of the code and how it interacts with the state-table to operate as desired.

fsm: ;load Z with a byte pointer to the sub-table corresponding to the present state

lds r17, pstate

mov ZL, r17 ;load Z pointer with pstate address \* 2

add ZL, ZL ;since Z will be used as a byte pointer with the lpm instr.

lds r17, pstate+1

mov ZH, r17

adc ZH, ZH

search: ;search subtable rows for input symbol match

lpm r18, Z ;get symbol from state table

cp r18, r16 ;compare table entry with input symbol

breq match

check\_eol: ;check input symbol against eol

cpi r18, eol ;compare low byte of table entry with eol

breq match

nomatch:

adiw ZL, $06 ;adjust Z to point to next row of state table

rjmp search ;continue searching

;a match on input value to row input value has been found the next word in this row is the next state address, the word following that is the task subroutine's address

match: ;make present state equal to next state value in row this accomplishes the stat transition

adiw ZL, $02 ;point to low byte of state address

lpm r17, Z+ ;copy next state addr. from table to preseent stat

sts pstate, r17

lpm r17, Z+

sts pstate+1, r17

;execute the subroutine that accomplihes the task associated

;with the transition

lpm r20, Z+ ;get subroutine address from state table

lpm r21, Z ;and put it in Z pointer

mov ZL, r20

mov ZH, r21

icall ;Z pointer is now used as a word pointer

ret

For further detailed or code specifics concerning the software explanation, check the comments found in program .lss files submitted.