ELEC 3040/3050
Lab 5

Matrix Keypad Interface

Using Parallel I/O

Goals of this lab exercise

- Control a peripheral "device" with the MC9S12C32 microcontroller
- Use parallel I/O ports to control and access a device
- Implement program-controlled and/or interrupt-driven input/output (I/O)
  - Program controlled: software controls timing of data transfers
  - Interrupt-driven: external device triggers data transfer

Ports available on the MC9S12C32/Dragonfly

- PORTT (bits 0-7)
  - General-purpose I/O pins PT7-PT0
    - Access via PTT
    - Pin directions set via DDRT
  - Programmable timer inputs/outputs IOC7-IOC0
    - Enable via timer setup registers
    - Default is general-purpose I/O
  - Bits 0-4 can be connected to the PWM module
    - Select via MODRR (0=timer*, 1 = PWM) *default
  - Programmable internal pull-up/down resistors

Ports available on the MC9S12C32/Dragonfly

- PORT AD (bits 7-0)
  - Analog-to-digital converter inputs AN7-AN0
    - Default function
  - General-purpose digital I/O pins PAD7-PAD0 **
    - Directions set via DDRAD **
    - Access via two I/O registers **
      - PTAD – digital I/O
      - PTIAD – read PAD input pins
    - Programmable pull-up/down resistors

**Register ATDDIEN bit k must be set to 1 to enable digital input buffer k (default is to disable the buffer)

** Must configure both DDRAD and ATDDIEN for digital inputs

Ports available on the MC9S12C32/Dragonfly

- PORT E bits PE0,PE1,PE4,PE7
  - Special functions:
    - PE0 = XIRQ# (nonmaskable interrupt request input)
    - PE1 = IRQ# (interrupt request input)
    - PE4 = ECLK output
    - PE7 = XCLK output
  - General purpose I/O (default operation)
    - Access via PORTE
    - DDRE determines pin directions, EXCEPT:
      - PE0 & PE1 can only be inputs (DDRE bits ignored)
      - PE2-PE7 can be general purpose inputs or outputs

All 8 bits of PORTE available on larger packages

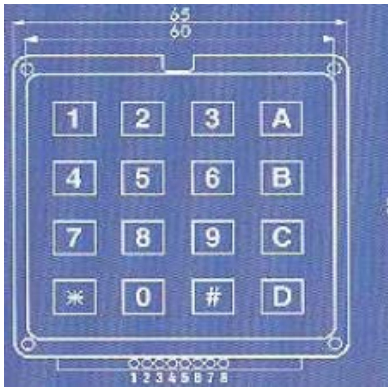
MC9S12C32 pin electronics Pull-up/pull-down control

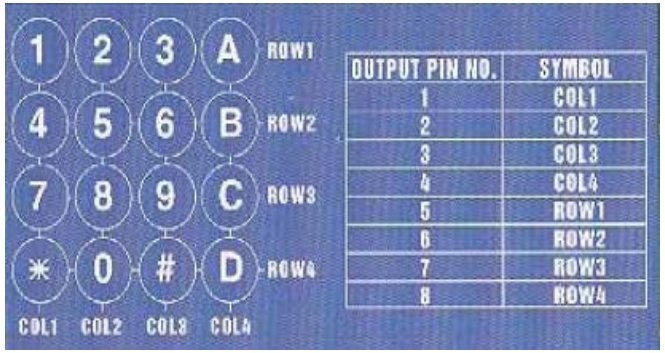
- Normally connect unused input pins to high or low logic level (avoids CMOS latch-up)
- HCS12 provides internal pull-up/down devices for port input pins
  - PUCR – Pull-Up Control Register for ports K, E, B, A
    - 1 bit per port - all pins in port configured the same
    - 0=disable (default on A & B)
    - 1=enable (default on K & E)
  - PERx – Port X pull device enable (x=AD,J,M,P,S,T)
    - 0=disable, 1=enable (see table for defaults)
  - PPSx = Port Polarity Select (x=AD,J,M,P,S,T)
    - 0=pull-up device, 1=pull-down device

Velleman 16-Key Matrix Keypad

- Used on phones, keyless entry systems, etc.
- Comprises a matrix of switches
  - no “active” circuit elements
- Accessed via 8 pins on the back

  (insert carefully into a breadboard)



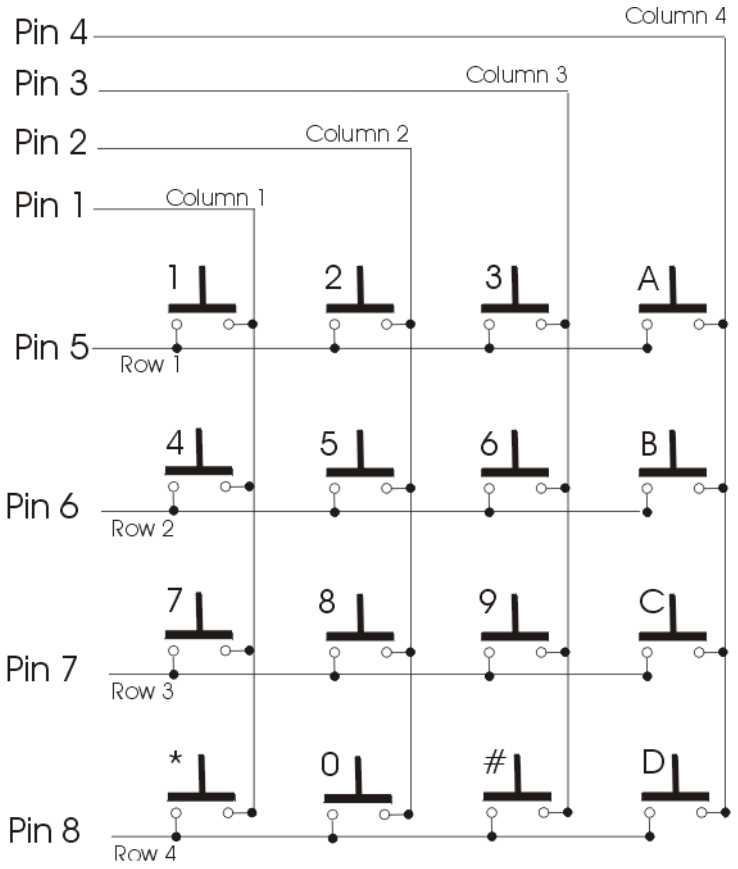


Pins: 1-2-3-4-5-6-7-8

Matrix keypad circuit diagram

- 16 keys/contact pairs: in 4 rows & 4 columns
- One key at each row-column intersection
- Spring normally holds key away from contacts
- Pressing a key connects the contacts

(“short circuits” row-to-column)



“Scanning” the keypad

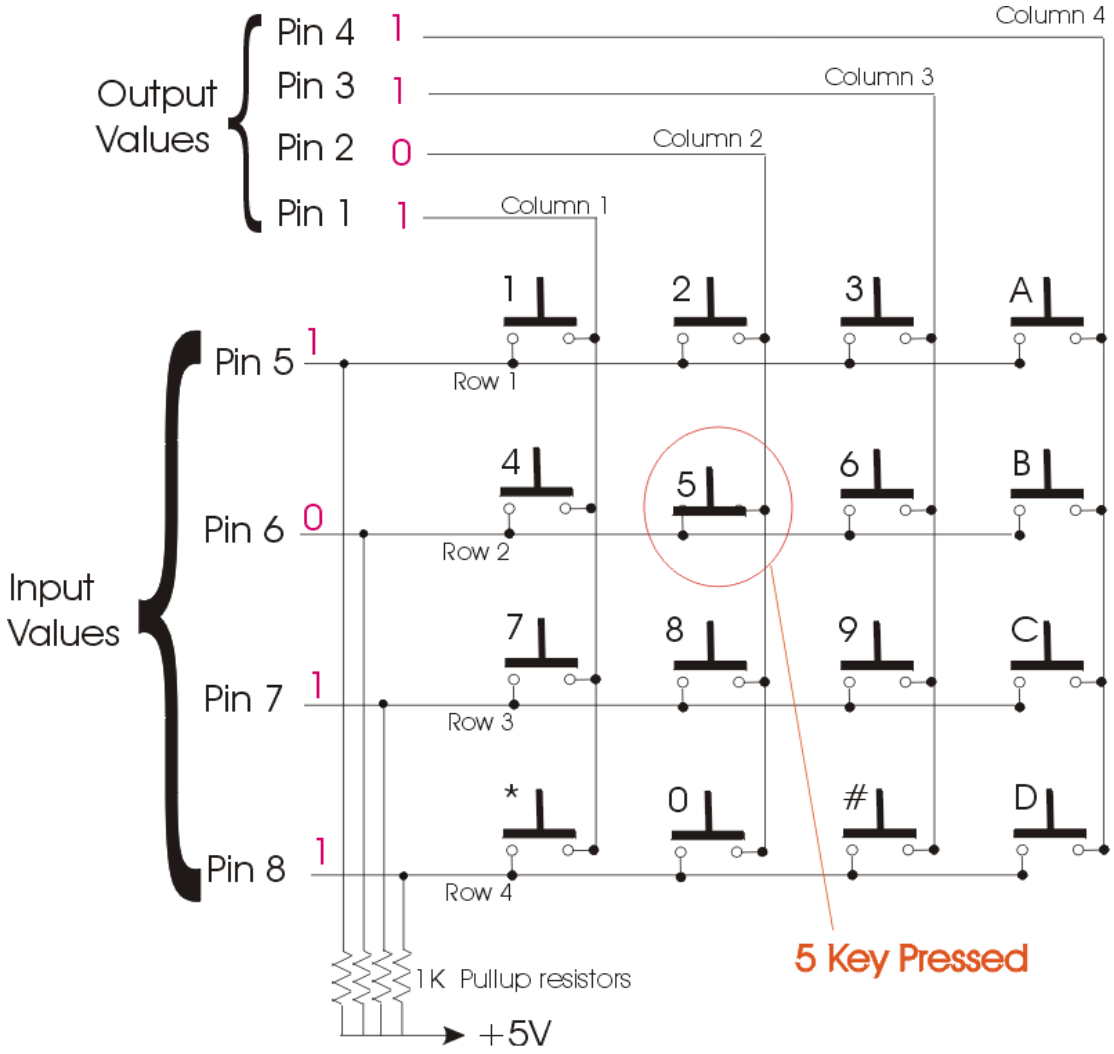
- Drive column wires with output port
- Read states of row wires via input port
  - Connect each row to +5 V with a pull-up resistor
    - Use microcontroller port internal pull-up if available
  - If no row-column shorts ➔ all row lines pulled high
  - Row line K can only be low if it is shorted to a column line N that is being driven low

Scan algorithm

1. Drive one column line N low, and the other columns high
2. Read the states of the row lines
   - If row K is low, it is shorted to column N
   - If row K is high, *either*:
     - K is not shorted to any column line & remains pulled high
     - *or*, K is shorted to a column line that is being driven high
3. Repeat steps 1 & 2, but with a different column line driven low (with others high)

- A key press is detected when a row line is detected in the low state
  - Key position is intersection of that row and the column being driven low

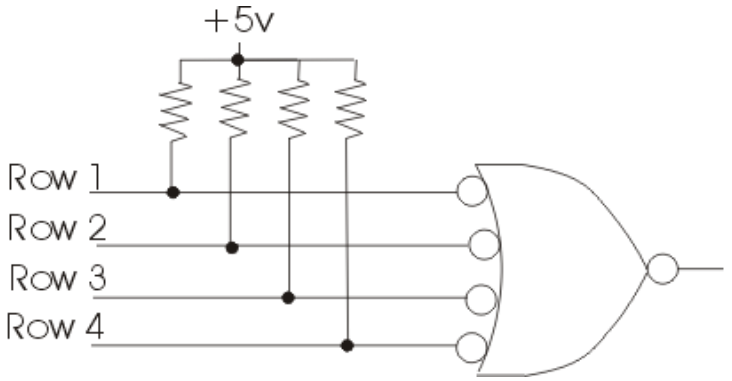
Example


Output Values
Pin 4 1
Pin 3 1
Pin 2 0
Pin 1 1
Column 4
Column 3
Column 2
Column 1
Input Values
Pin 5 1
Pin 6 0
Pin 7 1
Pin 8 1
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D
Row 1
Row 2
Row 3
Row 4
1K Pullup resistors
+5V
5 Key Pressed

Interrupting keypad - hardware

- Generate interrupt signal when a key pressed
  - Logical OR of active-low row lines
  - All column lines driven low
  - AND gate:

$$ABCD = \overline{\overline{A} + \overline{B} + \overline{C} + \overline{D}}$$



- Use the above to trigger IRQ# signal
  - Configure IRQ# as edge-triggered
    - Program IRQE bit in register INTCR
    - Captures the signal (sets a flip-flop) until seen by the CPU
    - Flip-flop clears when CPU responds

Hardware design

- Insert keypad in breadboard, with microcontroller internal pull-up resistors enabled on row lines (or use external pull ups)
- Connect I/O port pins to "devices" as follows:

Parallel Port Pins	Connected Devices
Port AD, bits 3-0	LEDs
Port T, bits 7-4	Keypad columns 4-1 (outputs)
Port T, bits 3-0	Keypad rows 4-1 (inputs)
Port E, bit 1	IRQ# (if interrupts used)

Use logic analyzer/oscilloscope to help debug connections

Software design

- Review how to read/write I/O ports in C
- Review how to set/clear/test bits in C
- Test program is to be a continuous loop, incrementing a one-digit decimal count once per second
  - If a pressed key detected (by software or interrupt) – display the key number on the 7-segment display digit for 10 seconds
  - Continue counting while the key number is displayed
  - Resume displaying the count after the 10 seconds has elapsed

  Notes:
  - After reading inputs – mask all but the row bits
  - Consider a scan loop rather than 16 "if" statements for detecting keys

Interrupt-driven setup

- Add HW to interrupt the CPU when a key is pressed (other I/O ports should be the same)
  - Use IRQ# interrupt input pin (Port E, bit 1)

- SW should include a "main program" and the keypad ISR
  - The "main" program should do all initialization and then enter an endless counting loop
    - All column lines should be low until an interrupt occurs
  - The ISR should be similar to the program-controlled keypad scanning routine, but executed if and only if the CPU is interrupted by a key press.

Debug suggestions

- A switch can be connected to IRQ# instead of the keypad to manually trigger interrupts to test the ISR
- The ISR can write a unique pattern to LEDs to indicate that it was entered
- Global variables can be modified by the ISR to indicate that the ISR was executed
- An "unimplemented ISR" error, detected by the debugger, indicates that there was no interrupt vector in the vector table
- If time permits, experiment with both edge-triggered and level-sensitive IRQ#