2.15 Digital Port Input/Output

Hardware and software aspects of digital input/output

EE302 – Real time and embedded systems

1

Overview

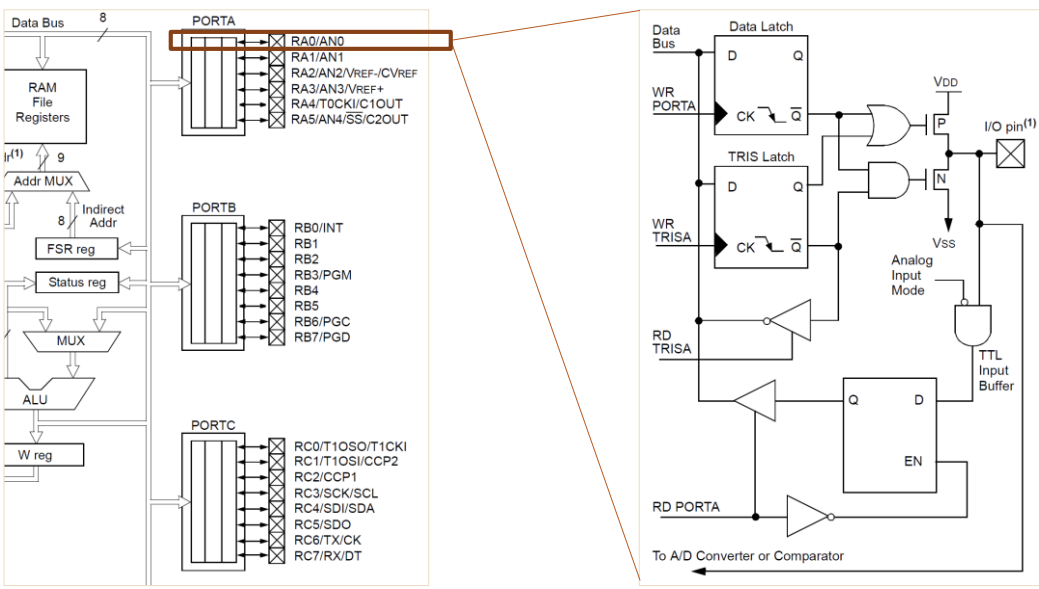
- Aims
  - Study I/O synchronization for digital ports
- Learning outcomes – you should be able to…
  - Show how to handle various electrical issues, e.g. driving a high current device from a current limited port
  - Explain switch debounce and show how to handle it in hardware or software
  - Write pseudocode/code to poll digital I/O ports and take action (1) continuously, (2) only on change, (3) only after a delay, etc.

2

Basic digital input/output

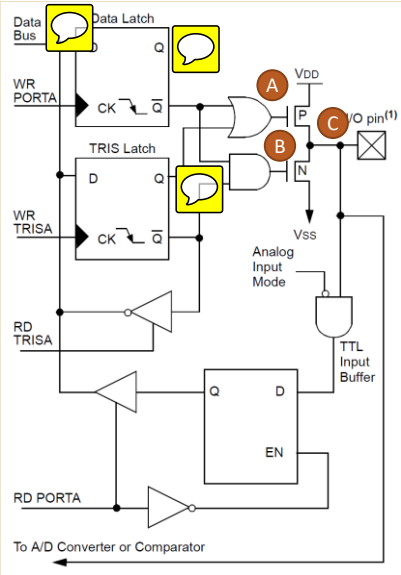
- Basic digital I/O devices include
  - Switch (input)
  - Simple keypad (e.g. 3 x 4 phone keypad)
  - LED (output)
  - 7 segment display (output)
  - Speaker/beeper (output)
  - Etc.
- All the devices above are <u>not bus aware</u>, so they may only be connected to CPU via a digital input or output port as appropriate

Architecture of a “simple” I/O port



Figures reproduced from PIC16F87x data sheet

Architecture of “simple” I/O port contd.



Figures reproduced from PIC16F87x data sheet

To write PORTA, TRISA must be 0 and WR PORTA enabled.
To read PORTA, TRISA must be 1 and RD PORTA enabled.

To write TRISA, WR TRISA must be enabled.
To read TRISA, RD TRISA must be enabled.

Pmos: input=0 => “on”, input=1 => “off”
Nmos: input=0 => “off”, input=1 => “on”

Example - writing a 1 to the port pin:
Assume TRISA has already been set to 0 so TRIS Latch Q=0, ~Q=1.
Write 1 on the data bus and enable WR PORTA, so the Data Latch ~Q=0.
That means point A is 0 => Pmos is “on”.
Point B is also 0 => Nmos “off”.
Therefore point C is disconnected from VSS and connected to VDD = High = logic 1.

Contd.

- Input port comprises
  - Buffer: isolate input latch from input current source
  - Input latch: to briefly hold (*) the current value on the latch output (connected to the data bus) so that the CPU can read it.
- Output port comprises
  - Output latch: briefly enabled (*) so that CPU can set output level via the data bus; then disabled so that output is held stable at that value (while data bus reused for next instruction)
- Bidirectional port
  - Contains additional latch and logic to enable just one of the input or output directions

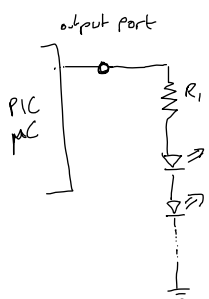
* NOTE: When a latch is enabled values at the latch inputs appear at the outputs almost immediately. When disabled, the latch output is disconnected from the latch input and held steady – changes to the latch input do not appear at the output.

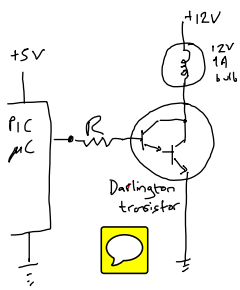
“Digital” I/O port electrical considerations

- Current limits
  - PIC Microcontroller limits current sourced or sunk on any one I/O pin to 25 mA; total current sourced/sunk by any port is 200 mA
  - This affects fan out/fan in for connection to TTL logic gates or similar.
  - Generally requires addition of a current limiting resistor when driving low impedance output (like an LED)

7

Current limiting





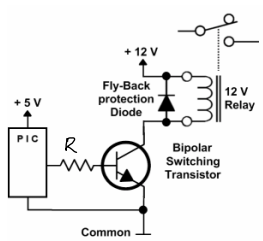


Figure reproduced from The Electronics Club

Current flows when output port is at logic high voltage. How should the value of R1 be chosen?
(A LED typically requires about 25mA current and may have about 2V voltage drop.)

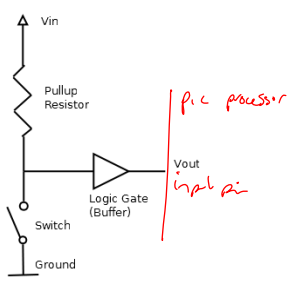
If a high current load must be driven, then a relay and/or buffer transistor is needed.
NOTE: resistor, R, still needed to limit port current from the microcontroller.

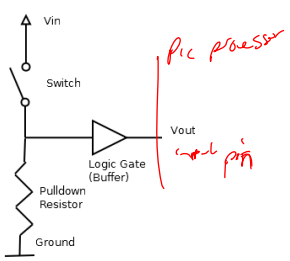
8

“Digital” I/O port electrical considerations

- How are “floating” pins handled?
  - Particularly relevant to switch inputs which are open circuit when not pressed
  - We have to ensure that pins are never truly “disconnected” even when a switch is open or a button is not pressed, but that they are tied to a predictable voltage in these situations
- To address this we use Pull-up or pull-down resistors
  - When associated with push buttons, these circuits determine the voltage when the push button switch is open and when it is closed (i.e. is it active high or active low?)

Pull up, pull down





Question: assuming the use of push-button switches, which of these configurations is for an active-low button?

Figures reproduced from wikimedia

Digital Port Input/Output

16

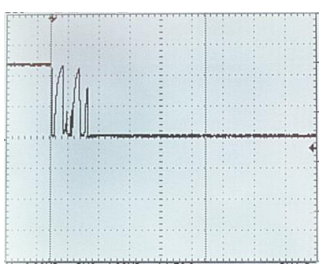
Switch Debounce

20 October 2020

16

Switch debounce in hardware

17



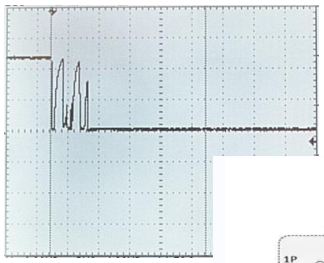
Voltage from a switch that has not been “debounced”

Figure reproduced from Micah Carrick’s blog

20 October 2020

17

Switch debounce in hardware

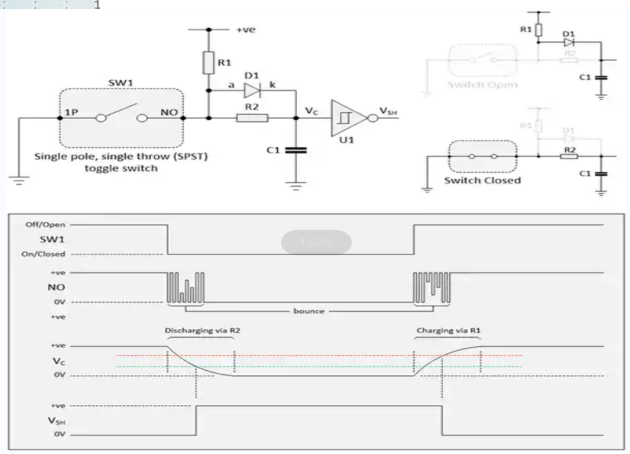


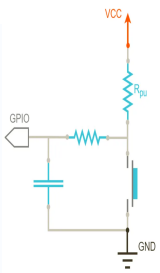
Voltage from a switch
not been "debounced

Figure reproduced from Micah

Hardware debounce solution using

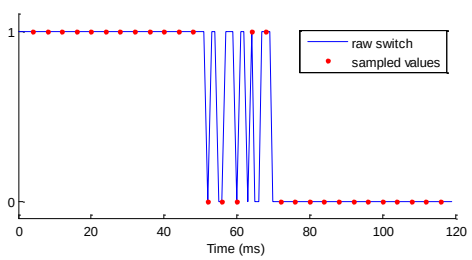
For more details see: A guide to debouncing - http://www.ganssle.com/debouncing.htm





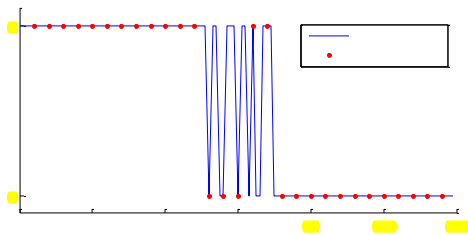
Software debounce using counting

- count based
  - Measure switch value repeatedly
  - value must be same for N polls to be considered debounced/stable



Software debounce using digital filter

- Digital filter based
  - Digital simulation of low pass filter with schmitt trigger
  - Has good EMI filtering properties

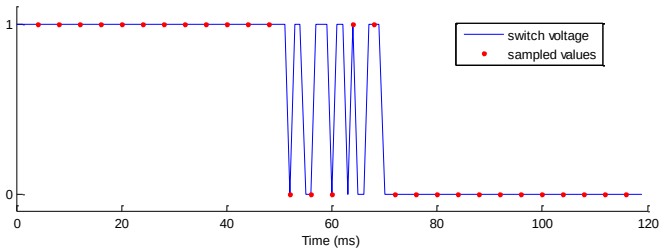


Switch (button) debounce in software


```
// Simple edge trigger algorithm (there are many possible variations)
// Verify signal is stable using a counter
// DEBOUNCE_TICKS should be chosen to give enough time for bounces
// to disappear (often 10-20ms but depends on button mechanics)
// checkSwitch would be called once per superloop
global gDebouncedSwitchState = OPEN // default value at start of app

// get debounced state of switch and indicate whether it has changed
checkSwitch()
  static debounceCounter = DEBOUNCE_TICKS // initial state
  debouncedSwitchChanged = FALSE // default unless change verified
  rawSwitchState = read switch input pin from IO port

  if rawSwitchState == gDebouncedSwitchState // switch state stable?
    debounceCounter = DEBOUNCE_TICKS // prepare to debounce next change
  else // switch state changed, so wait for it to become stable
    decrement debounceCounter
    if debounceCounter is 0 // debouncing all done?
      gDebouncedSwitchState = rawSwitch // accept the change
      debounceCounter = DEBOUNCE_TICKS // prepare to debounce next change
      debounceSwitchChanged = TRUE

  return debounceSwitchChanged
```

Software debounce – example walk through

rawSwitchState
(red dots)



gDebouncedSwitchState

debounceCounter

debouncedSwitchChanged

23

Digital Port Input/Output

Common/useful polling patterns

24

General port I/O polling pseudocode


```
// General app structure
Main…

setup()
  configure port direction
  configure internal pull up resistors

loop()
    pollInputPort()
    pollOutputPort()
    …
    delay SUPERLOOP_TICK

pollInputPort()
  // see next slides

pollOutputPort()
  // see next slides
```


Polling design patterns

- What follows are some very common polling problems with some typical solutions on following slides
- Timing or repeating problems occur when the rate at which your superloop repeats and I/O function is called doesn't match the rate or delay or repetitions that is suitable for the I/O operation you want to do
- Typical patterns…
  1. Poll to detect the instantaneous input state. Do something continuously (every superloop) while the input is in a particular state (e.g. while a button is pressed)
  2. Poll to detect specific input changes. Take action once only when the specific change has occurred (e.g. edge triggered only when a button changes from not-pressed to pressed)
  3. Poll with a period of N times the superloop period

Pattern 1: poll to detect instantaneous state, act continuously


```
// General app structure
global gSwitchClosed = FALSE

Main…

loop()
    pollSwitch()
    updateLED()
    delay SUPERLOOP_TICK

setup()
  setup button and LED…
```

```
pollSwitch()
  // active low button (assuming
  // hardware debounce)
  if BUTTON_PIN is LOW
    set gSwitchClosed = TRUE
  else
    set gSwitchClosed = FALSE

updateLED()
  if gSwitchClosed is TRUE
    set LED_PIN = HIGH // i.e. on
  else
    set LED_PIN = LOW // i.e. off
```

Note that the updateLED code sets the LED value on every single superloop (i.e. continuously based on the instantaneous state of the switch).

Both the button (input) and LED (output) are polled once per superloop

Pattern 2: poll to detect specific changes, act once only when specific change has occurred


```
// General app structure
global gSwitchJustClosed = FALSE

Main…

loop()
    pollSwitch()
    updateLED()
    delay SUPERLOOP_TICK

setup()
  setup button and LED…
```

```
pollSwitch()
  static prevButtonVal = HIGH

  // detect HIGH->LOW transition
  // on active low button (assuming
  // hardware debounce)
  if BUTTON_PIN is LOW
        AND prevButtonVal is HIGH
    set gSwitchJustClosed = TRUE
  else
    set gSwitchJustClosed = FALSE

  // remember prev for next time
  set prevButtonVal = BUTTON_PIN

updateLED()
  if gSwitchJustClosed is TRUE
    toggle LED_PIN
```

Note that the updateLED code only toggles the LED once on the button HIGH to LOW transition. In all other cases (including while the button stays LOW) the LED is untouched.
This pattern requires a static variable to detect the previous input state so as to detect transitions.

Self test questions


```
global gState = 0
const BITMASK =                        // TODO: choose suitable value to select bit 3

// assume the following functions are called from the superloop…

// Q1. set global gState to 1 continuously (i.e. every time this function is called)
// when PORTB, bit 3 is 0 and to 0 otherwise. (Hint: use the bitmask)
pollInputActContinuously()



// Q2. set global gState to 1 only when PORTB, bit 3 transitions from
// 1 to 0 and set it to 0 otherwise
pollInputActOnChange()
```


C coding self test question


```
// Q. How would you implement the following line of pseudocode in C
//    using a bitmask
// if PORTB, bits 4 and 6 matches binary x0x1 xxxx




// Q. how would you implement the following lines of pseudocode in C
// static prevValue = STARTUP_VALUE
// if PORTB, bit 6 differs from prevValue
// …
// set prevValue bit 6 from inputPort
```

Pattern 3: poll on every Nth superloop only


```
// General app structure

Main…

loop()
    …
    slowPoll()
    delay SUPERLOOP_TICK

setup()
  setup button and LED…
```

```
slowPoll()
  static periodCounter = N

  // only do the real work every
  // Nth time this function is called
  decrement periodCounter
  if (periodCounter is 0)
     readFromDevice() // does the real work
     reset periodCounter = N


readFromDevice()
  …
```

Note the real work of polling the device is done in readFromDevice but this function is only called every Nth time that slowPoll is called.

Self test software questions

In the following

- *assume the superloop runs (and calls the corresponding polling function) once every 5 ms*
- *Make sure the superloop runs every 5 ms – i.e. do not place a delay in your polling function which would upset the superloop timing*

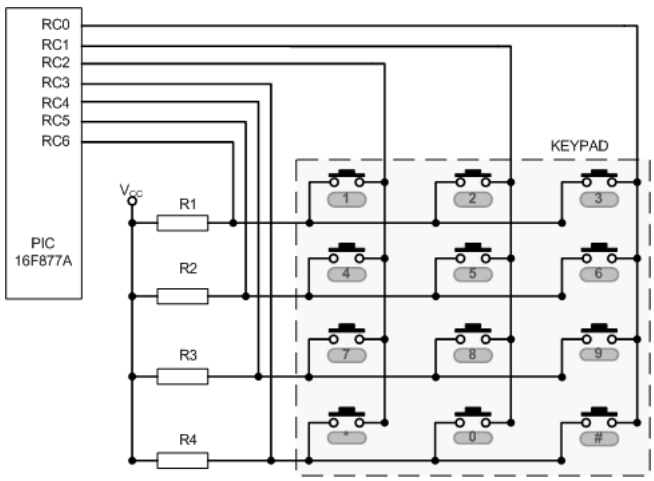
Q. show how set a global variable true only when an active low switch transitions from open to closed assuming hardware debounce

Q. show how, whenever a global variable becomes true, to light an LED for 50 ms and then set the variable false again

33

34

Exam 2012-2013 Q3



The 4x3 keypad is organised as 4 rows and 3 columns of switches. Port C is configured such that RC0 to RC2 are outputs connected to the keypad columns and RC3 to RC6 are inputs connected to the keypad rows.

contd

1. *If RC2 (column 0) is driven low, other columns are high, and no key is pressed, what values would you expect to read in RC3 to RC6? Explain your answer.*
2. *If RC2 is driven low, other columns are high, and the '4' key is pressed, what values would you expect to read in RC3 to RC6? Explain your answer.*
3. *Explain the purpose of resistors R1 to R4. What is the significance of the resistor values?*

contd

Write the pseudocode for a scanKeys function which detects the row and column of the key pressed (if any).

To detect whether a key on a particular column has been pressed, the MCU drives only that column low. With the column driven low, the MCU reads the value of the 4 rows to determine whether a key was pressed or not. By repeating the procedure for all 3 columns any single key press can be detected.

(Note: the identity, e.g. ASCII code, of the detected key could be looked up in a simple 2 dimensional array, using the row and column returned from the scanKeys function as indexes into the array. This is outside the question scope)