Overview

- AVR Microcontrollers
- I/O Ports

What is a Microcontroller?

- A microprocessor is a device which integrates a number of the components of a microprocessor system onto a single microchip.
- Microcontrollers are used in embedded systems.

  - An embedded system is a combination of hardware and software to perform a specific function. It is a part of a larger system which works in a reactive and time-constrained environment.
  - Typical examples of embedded systems include:
    - Consumer electronics (cellular phones, personal digital assistants, interactive game boxes, cameras, …)
    - Consumer products (washers, microwave ovens, …)
    - Automobiles (anti-lock braking, engine control, …)
    - Industrial process controllers & avionics/defence applications.

Major Components in a Microcontroller

- A microcontroller includes the following major components:
  - CPU core
  - Memory (Both RAM and ROM)
  - I/O devices
    - The types and the number of I/O devices varies with microcontrollers. Typical I/O devices include
      - Parallel I/O (Buses)
      - Serial I/O (UART, SPI)
      - ADC (Analog-to-Digital Converter)
      - Timers

- I/O Ports
AVR Microcontrollers

- Harvard architecture
  - Separate memories and buses for program and data.
    - Program memory (FLASH) and data memory (SRAM) can be accessed at the same time.
- 8-bit microcontrollers.
  - Registers are 8-bits long.
  - The register file contains 32 x 8-bit general purpose working registers with a single clock cycle access time.
- A family of microcontrollers.
  - Vary in memory sizes and the types and the number of I/O devices.
  - Mega 64 is used in our AVR development board.

AVR Pipelining

- Two stages: fetch and execute
  - Parallel fetches and executions are enabled by the Harvard Architecture and the fast-access register file.

AVR Pipelining (Cont.)

- All ALU operations take a single clock cycle.
What is I/O?

• I/O is Input or Output (Input/Output). It can be:
  - A number of digital bits formed into a number of digital inputs or outputs called a **port**. These are usually eight bits wide and thus referred to as a BYTE wide port. i.e. byte wide input port, byte wide output port.
  - A digital I/O port can be implemented by a number of D type flip-flops.
  - A serial line from the microprocessor (Transmit or TX) and a serial line to the microprocessor (Receive or RX) allowing serial data in the form of a bit stream to be transmitted or received via a two wire interface.
  - Other I/O devices such as Analogue-to-Digital Converters (ADC) and Digital-to-Analogue Converters (DAC), Timer modules, Interrupt controllers etc.

Internal Structure of an Input Port

• The following diagram shows the structure of 4-bit input port. The input port allows outside world inputs to be stored in the data latches so they can be read by the microprocessor via the data bus.

![Internal Structure of an Input Port Diagram]

Tri-state Gates

• The data bus connections must be via tri-state buffers so that the input port is only connected to the data bus when the input port is selected. This is achieved by connecting a chip select signal to the enable input signal line. Note that the tri-state enable is active low.

![Tri-state Gate Diagram]

Internal Structure of an Output Port

• An input port can be realised by a number of D type flip-flops.
• The following diagram shows a 4-bit output port. The inputs are connected to data bus and the outputs are connected to any output interface.
AVR I/O Ports (Cont.)

- All AVR ports have true Read-Modify-Write functionality when used as general digital I/O ports.
  - The direction of one port pin can be changed without unintentionally changing the direction of any other pin with the SBI and CBI instructions.
- Three I/O memory address locations are allocated for each port, one each for the Data Register – PORTx, Data Direction Register – DDRx, and the Port Input Pins – PINx.
  - x is one of A, B, C, D, E and F.
  - The Port Input Pins I/O location is read only, while the Data Register and the Data Direction Register are read/write.

AVR I/O Ports (Cont.)

- Each port pin consists of three register bits: DDxn, PORTxn, and PINxn.
  - The DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.
  - The DDxn bit in the DDRx Register selects the direction of this pin.
    - If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.
  - If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).
AVR I/O Ports (Cont.)

Port Pin Configurations

<table>
<thead>
<tr>
<th>DDxn</th>
<th>PORTxn</th>
<th>PUD (in SFIOR)</th>
<th>I/O</th>
<th>Pull-up</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td></td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>Input</td>
<td>Yes</td>
<td>Pull will source current if ext. pulled low.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Input</td>
<td>No</td>
<td>Tri-state (Hi-Z)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output Low (Sink)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Output</td>
<td>No</td>
<td>Output High (Source)</td>
</tr>
</tbody>
</table>

PUD (Pull-UP Disable) is a bit in SFIOR(Special Function IO Register). When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ([DDxn, PORTxn] = 0b01).

---

Reading An Externally Applied Pin Value

- Independent of the setting of Data Direction bit DDxn, the port pin can be read through the PINxn Register bit.
- The PINxn Register bit and the preceding latch constitute a synchronizer. This is needed to avoid metastability if the physical pin changes value near the edge of the internal clock, but it also introduces a delay.
- The maximum and minimum propagation delays are denoted by $t_{pd,max}$ and $t_{pd,min}$ respectively.

---

Reading A Software Assigned Pin Value

Synchronization when Reading a Software Assigned Pin Value
Reading A Software Assigned Pin Value (Cont.)

- Consider the clock period starting shortly after the first falling edge of the system clock. The latch is closed when the clock is low, and goes transparent when the clock is high, as indicated by the shaded region of the “SYNC LATCH” signal. The signal value is latched when the system clock goes low. It is clocked into the PINxn Register at the succeeding positive clock edge. As indicated by the two arrows $t_{pd,max}$ and $t_{pd,min}$, a single signal transition on the pin will be delayed between $\frac{1}{2}$ and $1-\frac{1}{2}$ system clock period depending upon the time of assertion.
- When reading back a software assigned pin value, a $nop$ instruction must be inserted. The $out$ instruction sets the “SYNC LATCH” signal at the positive edge of the clock. In this case, the delay $t_{pd}$ through the synchronizer is one system clock period.

Example 1: Reading a Pin Value

; Define pull-ups and set outputs high
; Define directions for port pins
ldi r16,((1<<PB7)|(1<<PB6)|(1<<PB1)|(1<<PB0)
ldi r17,((1<<DDB3)|(1<<DDB2)|(1<<DDB1)|(1<<DDB0)
out PORTB,r16
out DDRB,r17
; Insert nop for synchronization
nop
; Read port pins
in r16,PINB
...

Example 1: Reading a Pin Value (Cont)

• Example 1 shows how to set Port B pins 0 and 1 high, 2 and 3 low, and define the port pins from 4 to 7 as input with pull-ups assigned to port pins 6 and 7.
• The resulting pin values are read back again, a $nop$ instruction is included to be able to read back the value recently assigned to some of the pins.

Example 2: Controlling LEDs

Consider our AVR development board. Assume that
- Push Button PB0 is connected to PA0 (PINA0), and
- Eight LEDs (LED 0 to LED 7) are connected to PC0 (PINC0) to PC7 (PINC7), respectively.
Whenever PB0 is pushed, the following program turns the LEDs on if they are off; otherwise, it turns the LEDs off.

.include "m64def.inc"
.def temp =r16
.def count=r15
.equ PATTERN1 = 0x00
.equ PATTERN2 = 0xFF
clr count ; Set count to 0
ser temp ; Set temp to 0b11111111
Example 2: Controlling LEDs (Cont.)

out PORTC, temp    ; Write ones to all the LEDs
out DDRC, temp     ; PORTC is all outputs
out PORTA, temp    ; Enable pull-up resistors on PORTA
clr temp
out DDRA, temp     ; PORTA is all inputs
loop:
sbic PINA, 0       ; Skip the next instruction if PB0 is pushed
rjmp loop          ; If not pushed, check PB0 again
cpi count, 0       ; Check the count
breq ledon

Example 2: Controlling LEDs (Cont.)

ldi temp, PATTERN1 ; Turn the LEDs off if they are on
out PORTC, temp
clr count
rjmp loop
ledon:
ldi temp, PATTERN2 ; Turn the LEDs on if they are off.
out PORTC, temp
inc count
rjmp loop

Reading Material

1. Overview, AVR CPU Core, I/O Port in ATmega 64 Data Sheet.
2. Introduction to Pull-Up Resistors.
   http://www.seattlerobotics.org/encoder/mar97/basics.html
3. What is Tri-state Buffer?