Microprocessors & Interfacing

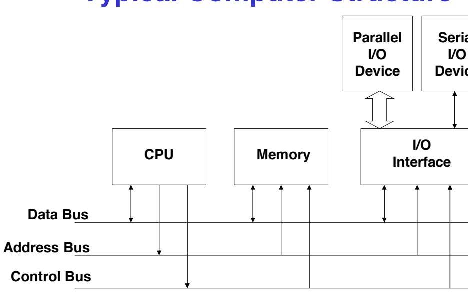
Parallel Input/Output

Lecturer : Annie Guo

Lecture Overview

- I/O Addressing
  - Memory mapped I/O
  - Separate I/O
- Parallel Input/Output
  - AVR examples

Typical Computer Structure

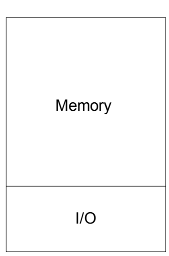


I/O Addressing

- If the same address bus is used for both memory and I/O, how does hardware distinguish between memory reads/writes and I/O reads/writes?
  - Two approaches:
    - Memory-mapped I/O
    - Separate I/O
  - Both adopted in AVR

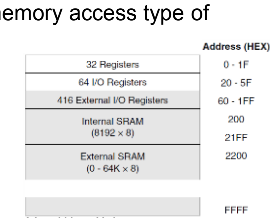
Memory Mapped I/O

- The entire memory address space contains a section for I/O registers.



AVR Memory Mapped I/O

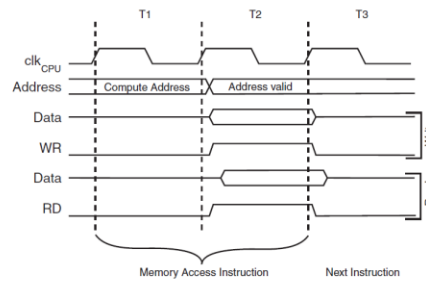
- In AVR, 64+ I/O registers are mapped into memory space \$0020 ~ \$01FF
  - with 2-byte address
- With such memory addresses, the access to the I/Os uses memory access type of instructions
  - E.g. *st* and *ld*



Memory Mapped I/O (cont.)

- Advantages:
  - Simple CPU design
  - No special instructions for I/O accesses
  - Scalable
- Disadvantages:
  - I/O devices reduce the amount of memory space available for application programs.
  - The address decoder needs to decode the full address bus to avoid conflict with memory addresses.

On-Chip Memory Access Cycles

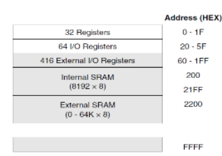


Separate I/O

- Two separate spaces for memory and I/O.
  - Less expensive address decoders than those needed for memory-mapped I/O
- Special I/O instructions are required.

Separate I/O (cont.)

- In AVR, the first 64 I/O registers can be addressed with separate addresses \$00 ~ \$3F
  - 1 byte address
- With such separate addresses, the access to the I/Os uses I/O specific instructions.
  - E.g. *in* and *out*



I/O Synchronization

- CPU is typically much faster than I/O devices.
- Therefore, synchronization between CPU and I/O devices is required.
- Two synchronization approaches:
  - Software synchronization
  - Hardware synchronization

Software Synchronization

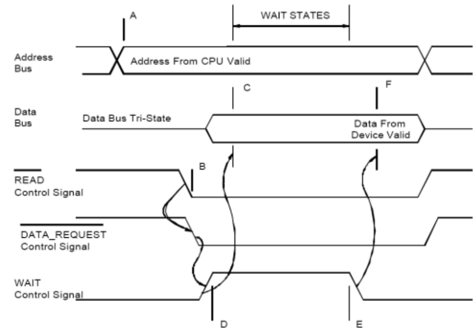
Two software synchronization methods:

- Real-time synchronization
  - Uses a software delay to match CPU to the timing requirements of the I/O device.
    - The timing requirement must be known
    - Sensitive to CPU clock frequency
    - Consumes CPU time.
- Polling I/O
  - A status register, with a DATA_READY bit, is added to the device. The software keeps reading the status register until the DATA_READY bit is set.
    - Not sensitive to CPU clock frequency
    - Still consumes CPU time, but CPU can do other tasks at the same time.
- Examples will be given later

Handshaking I/O

- A hardware oriented synchronization approach with control signal READY or WAIT.
  - For an input device, when CPU is asking for input data, the input device will assert WAIT if the input data is NOT available. When the input data is available, it will de-assert WAIT.
  - For an output device, when CPU is sending output data via the data bus, the output device will assert WAIT if it is not ready to take the data. When it is ready, it will de-assert WAIT.
  - While WAIT is asserted, CPU must wait until this control signal is de-asserted.

Read Cycle with Wait States

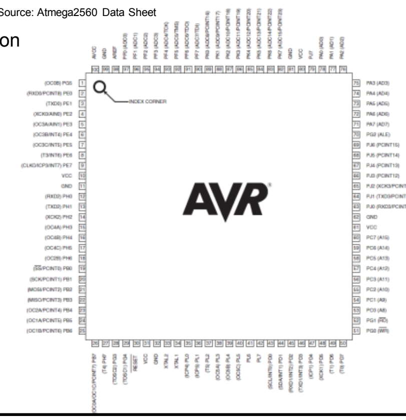


Parallel Input/Output in AVR

- Communication through ports
- Two special instructions designed for parallel input/output operations
  - in
  - out

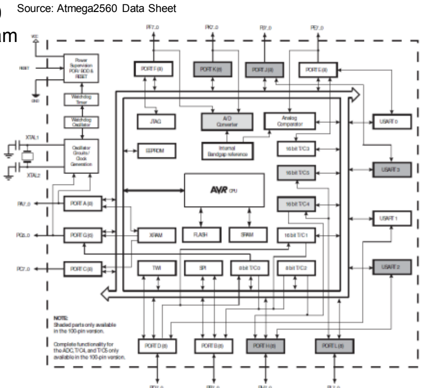
Atmega2560 Pin Configuration

Source: Atmega2560 Data Sheet



Atmega2560 Block Diagram

Source: Atmega2560 Data Sheet



AVR PORTs

- Can be configured to receive or send data
- Include physical pins and related circuitry to enable input/output operations.
- Different AVR microcontroller devices have different port design
  - ATmega2560 has 100 pins, most of them form eleven ports for parallel input/output.
    - Port A to Port G
      - Having separate I/O addresses
        - » using *in* or *out* instructions
    - Port H to Port L
      - Only having memory-mapped addresses
    - Three I/O addresses are allocated for each port. For example, for port *x*
      - PORTx is data register
      - DDRx is data direction register
      - PINx is port input pins

Load I/O Data to Register

- Syntax: *in Rd, A*
- Operands: 0 ≤ d ≤31, 0 ≤ A ≤63
- Operation: Rd ← I/O(A)

- Words: 1
- Cycles: 1
- Example:

```
in r25, 0x03          ; read port B
```

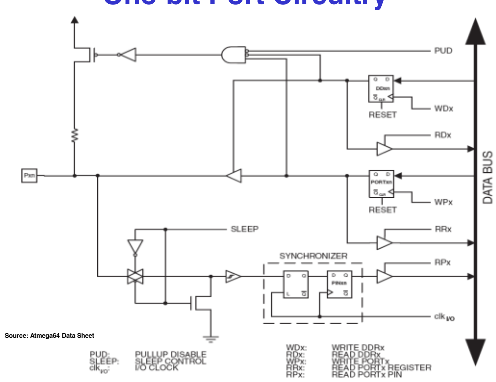
•The names of the I/O ports are given in the device definition file, m2560def.inc.
•0x03 is an I/O register address of port B

Store Register Data to I/O Location

- Syntax: *out A, Rr*
- Operands: 0 ≤ r ≤31, 0 ≤ A ≤63
- Operation: I/O(A) ← Rr
- Words: 1
- Cycles: 1
- Example:

```
out 0x05, r16          ; write to port B
```


One-bit Port Circuitry*



How does it work?

- Each port pin circuit consists of three register bits. E.g. for pin *n* of port *x,* we have
  - DDRxn, PORTxn, and PINxn.
- The DDRxn bit in the DDRx Register selects the direction of this pin.
  - If DDxn is written logic one, the pin is configured as an output pin. If DDxn is written logic zero, the pin is configured as an input pin.

How does it work? (cont.)

- When the pin is configured as an input pin, the pull-up resistor can be activated/deactivated.
- To active pull-up resistor for input pin, PORTxn needs to be written logic one.

Sample Code for Output

```
.include "m2560def.inc"

        clr     r16             ; clear r16
        ser     r17             ; set r17
        out     DDRA, r17       ; set Port A for output operation

        out     PORTA, r16      ; write zeros to Port A
        nop                     ; wait (do nothing)
        out     PORTA, r17      ; write ones to Port A
```

Sample Code for Input

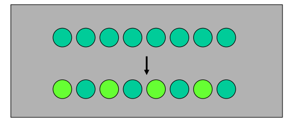
```
.include "m2560def.inc"

        clr     r15
        out     DDRA, r15       ; set Port A for input operation

        in      r25, PINA       ; read Port A
        cpi     r25, 4          ; compare read value with constant
        breq    exit            ; branch if r25=4
        ...
exit:   nop                     ; branch destination (do nothing)
```

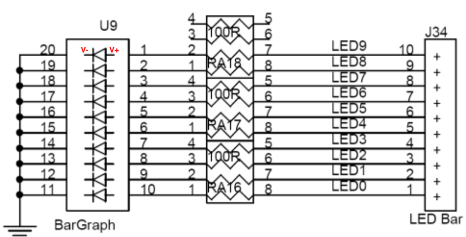

Example 1

- Design a simple control system that can control a set of LEDs to display a fixed pattern.



LED and Its Operation

- For each LED, when its V+ > V-, it will emit light.



Example 1 (solution)

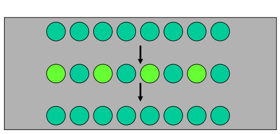
- Design consists of a number of steps:
  - Set a port for the output operation, one pin of the port is connected to one LED
  - Write the pattern value to the port so that it drives the LEDs to display the related pattern.

```
.include "m2560def.inc"
        ser r16
        out DDRB, r16           ; set Port B for output

        ldi r16, 0xAA           ; write the pattern
        out PORTB, r16
end:
        rjmp end
```


Example 2

- Design a simple control system that can control a set of LEDs to display a fixed pattern for *one second and then turn the LEDs off.*



Example 2 (solution)

- Design consists of a number of steps:
  - Set a port for the output operation, one pin of the port is connected to one LED
  - Write the pattern value to the port so that it drives the display of LEDs
  - *Wait for one second*
  - *Write a pattern to set all LEDs off.*

Counting One Second

- Basic idea:
  - Assume the clock cycle period is 1 ms (very very slow, not a real value). Then we can write a program that executes

$$^{1}\!/_{10^{-3}} = 1\times10^{3}$$

    single cycle instructions.

  - Execution of the code will take 1 second if each instruction in the code takes one clock cycle.
- An AVR implementation example is given in the next slide, where the 1 ms clock cycle time is assumed.

Code for One Second Delay (T_{clock}=1ms)

```
.include "m2560def.inc"
.equ loop_count = 124
.def iH = r25
.def iL = r24
.def countH = r17
.def countL = r16
.macro oneSecondDelay
        ldi countL, low(loop_count)        ; 1 cycle
        ldi countH, high(loop_count)
        clr iH                             ; 1
        clr iL
  loop: cp iL, countL                      ; 1
        cpc iH, countH
        brsh done                          ; 1, 2 (if branch)
        adiw iH:iL, 1                      ; 2
        nop
        rjmp loop                          ; 2
  done:
.endmacro
```


Code for Example 2

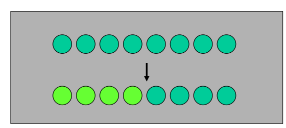
```
.include "m2560def.inc"

        ser r15
        out DDRB, r15            ; set Port B for output

        ldi r15, 0xAA            ; write the pattern
        out PORTB, r15
        oneSecondDelay           ; 1 second delay
        ldi r15, 0x00
        out PORTB, r15           ; turn off the LEDs
end:
        rjmp end
```


Example 3

- Design a simple control system that can control a set of LEDs to display a fixed pattern *specified by the user.*
  - Assume there are switches. Each switch can provide two possible values (switch-on for logic one and switch-off for logic 0)



Example 3 (solution)

- Design
  - Connect the switches to the pins of a port
  - Set the port for input
  - Read the input
  - Set another port for the output operation, each pin of the ports is connected to one LED
  - Write the pattern value provided by the input switches to the port so that it drives the display of LEDs
- Execution
  - Set the switches for a desired input value
  - Start the control system

Code for Example 3

```
.include "m2560def.inc"

        clr r17
        out DDRC, r17            ; set Port C for input
        ser r17
        out PORTC, r17           ; activate the pull up

        in r17, PINC             ; read the pattern set by the user
                                 ; from the switches
        ser r16
        out DDRB, r16            ; set Port B for output

        out PORTB, r17           ; write the input pattern
end:
        rjmp end
```

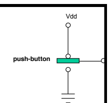
Example 4

- Design a simple control system that can control a set of LEDs to display a pattern specified by the user *during the execution*.

Example 4 (solution)

- One solution is that the processor continues checking if there is an input for read. If there is, then processor reads the input and goes to next task, otherwise the processor is in a waiting state for the input.
  - Using polling to handle dynamic input.

Example 4 (solution)



- Design
  - Set one port for input and connect each pin of the port to one switch
  - Set another port for the output operation, each pin of the ports is connected to one LED
  - Set a pin for input and connect the pin to the push-button,
    - When the button is pressed, it signals "Input Pattern is ready"
  - Poll the pin until "Input Pattern is ready"
  - Read the input pattern
  - Write the pattern to the port so that it drives the display of LEDs
- During execution
  - Set the switches for the input value
  - Push the button
  - The LEDs show the pattern as specified by the user.

Code for Example 4

- Set an extra input bit for signal from user when the input is ready.

```
.include "m2560def.inc"

        cbi DDRD, 7             ; set Port D bit 7 for input
        clr r17
        out DDRC, r17           ; set Port C for input
        ser r17
        out PORTC, r17          ; activate the pull up
        ;ser r17
        out DDRB, r17           ; set Port B for output

waiting:                        ; check if that bit is clear
        sbic PIND, 7            ; if yes skip the next instruction
        rjmp waiting            ; waiting

        in r17, PINC            ; read pattern set by the user
                                ; from the switches
        out PORTB, r17
        rjmp waiting
```


Announcements

- Quiz next week
  - Lecture class
  - 1 hour
  - Cover materials wk1-5
- Lab boards available for loan this week

Reading Materials

- Chapter 9: Computer Buses and Parallel Input and Output. Microcontrollers and Microcomputers by Fredrick M. Cady.
- Mega2560 Data Sheet
  - Ports

Homework

1. Refer to the AVR Instruction Set manual, study the following instructions:

   - Arithmetic and logic instructions
     - ser
   - Data transfer instructions
     - in, out
   - Bit operations
     - sbi, cbi
   - Program control instructions
     - sbic, sbis
   - MCU control instructions
     - nop

Homework

2. Study the following code. What is the function ?

```
.include "m2560def.inc"
.def temp =r16

.equ PATTERN1 = 0x5B
.equ PATTERN2 = 0xAA

        ser temp
        out PORTC, temp          ; Write ones to all the LEDs
        out DDRC, temp           ; PORTC is output
        out PORTA, temp          ; Enable pull-up resistors on PORTA
        clr temp
        out DDRA, temp           ; PORTA is input
switch0:
        sbic PINA, 0             ; Skip the next instruction
                                 ; if switch0 is pushed
        rjmp switch1             ; If not pushed, check the other switch
        ldi temp, PATTERN1       ; Store PATTERN1 to the LEDs
        out PORTC, temp          ; if the switch was pushed
switch1:
        sbic PINA, 1             ; Skip the next instruction
                                 ; if switch 1 is pushed
        rjmp switch0             ; If not pushed, check the other switch
        ldi temp, PATTERN2       ; Store PATTERN2 to the LEDs
        out PORTC, temp          ; if the switch was pushed
        rjmp switch0             ; Now check switch 0 again
```


Homework

3. Refer to section 3 “Introduction to AVR Microprocessor Development Board” in lab3. Study the lab board. Write the assembly code to display pattern 10110111 on the LED bar through each of the following I/O ports:

   (a) port C

   (b) port F

   (c) port L