Microprocessors & Interfacing

Interrupts (I)

Lecturer : Dr. Annie Guo

Lecture Overview

- Introduction to Interrupts
  - Interrupt system specifications
  - Multiple sources of interrupts
  - Interrupt priorities
- Interrupts in AVR
  - Interrupt vector table
  - Interrupt service routines
  - System reset
    - Watchdog reset

CPU Interacts with I/O

Two approaches:

- Polling
  - Software queries I/O devices
  - No hardware needed
  - Not efficient
    - CPU may waste processor cycles to query a device even if it does not need any service.
- Interrupts
  - I/O devices generate signals to request services from CPU Need special hardware to implement interrupt services
  - Efficient
    - A signal is generated only if the I/O device needs services from CPU.

Interrupt Systems

- An interrupt system implements interrupt services
- It basically performs three tasks:
  - Recognize interrupt events
  - Respond to the interrupts
  - Resume normal programmed task

Recognize Interrupt Events

- Interrupt events
  - Associated with interrupt signals:
    - In different forms, including signal levels and edges.
  - Can be multiple and synchronous
    - Namely, there may be many sources to generate an interrupts; a number of interrupts can be generated at the same time.
- Approaches are required to
  - Identify an interrupt event among multiple sources
  - Determine which interrupts to serve if there are multiple simultaneous interrupts

Respond to Interrupts

- Handling interrupt
  - Wait for the current instruction to finish.
  - Acknowledge the interrupting device.
  - Branch to the correct ***interrupt service routine*** (interrupt handler) to service interrupting device.

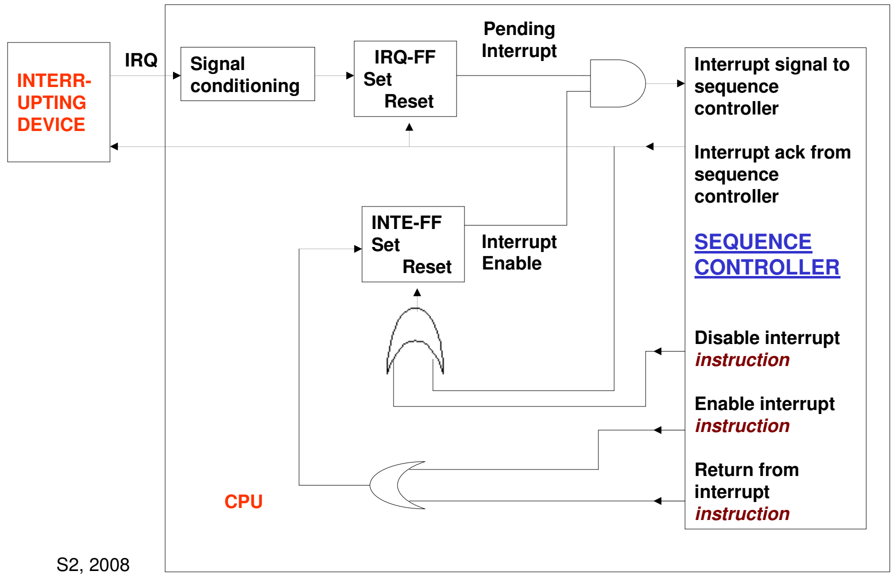
Resume Normal Task

- Return to the interrupted program at the point it was interrupted.

Interrupt Process Control

- Interrupts can be enabled or disabled
- Can be controlled in two ways:
  - Software control
    - Allow programmers to enable and disable selected/all interrupts.
  - Hardware control
    - Disable further interrupts while an interrupt is being serviced

Interrupt Recognition and Acknowledgement Hardware


INTERR-UPTING DEVICE
IRQ
Signal conditioning
IRQ-FF Set Reset
Pending Interrupt
Interrupt signal to sequence controller
Interrupt ack from sequence controller
INTE-FF Set Reset
Interrupt Enable
SEQUENCE CONTROLLER
Disable interrupt instruction
Enable interrupt instruction
Return from interrupt instruction
CPU

Interrupt Recognition and Ack.

- An Interrupt Request (IRQ) may occur at any time.
  - It may have rising or falling edges or high or low levels.
  - Frequently it is an active-low signal
    - multiple devices are wire-ORed together.
- Signal Conditioning Circuit detects these different types of signals.
- Interrupt Request Flip-Flop (IRQ-FF) records the interrupt request until it is acknowledged.
  - When IRQ-FF is set, it generates a pending interrupt signal that goes towards the Sequence Controller.
  - IRQ-FF is reset when CPU acknowledges the interrupt with INTA signal.

Interrupt Recognition and Ack. (cont.)

- Interrupts can be enabled and disabled by software instructions, which is supported by the hardware Interrupt Enable Flip-Flop (INTE-FF).
- When the INTE-FF is set, all interrupts are enabled and the pending interrupt is allowed through the AND gate to the sequence controller.
- The INTE-FF is reset in the following cases.
  - CPU acknowledges the interrupt.
  - CPU is reset.
  - the Disable Interrupt Instruction is executed.

Interrupt Recognition and Ack. (cont.)

- An interrupt acknowledge signal is generated by the CPU when the current instruction has finished execution and CPU has detected the IRQ.
  - This resets the IRQ-FF and INTE-FF and signals the interrupting device that CPU is ready to execute the interrupting device routine.
- At the end of the interrupt service routine, CPU executes a return-from-interrupt instruction.
  - Part of this instruction’s job is to set the INTE-FF to re-enable interrupts.
- Nested interrupts can happen If the INTE-FF is set during an interrupt service routine
  - An interrupt can therefore interrupt interrupting interrupts.

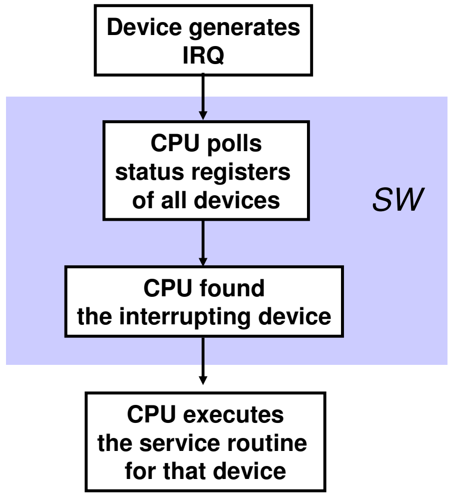
Multiple Sources of Interrupts

- To handle multiple sources of interrupts, the interrupt system must
  - Identify which device has generated the IRQ.
    - Using polling approach
    - Using vectoring approach
  - Resolve simultaneous interrupt requests
    - using prioritization schemes.

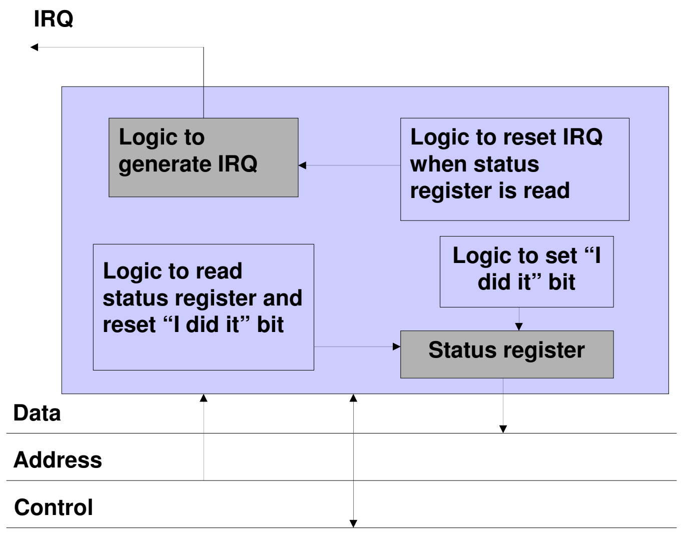
Polled Interrupts

- Software, instead of hardware, is responsible for finding the interrupting source.
  - The device must have logic to generate the IRQ signal and to set an "I did it" bit in a status register that is read by CPU.
  - The bit is reset after the register has been read.

Polled Interrupt Execution Flow


Device generates IRQ
CPU polls status registers of all devices
CPU found the interrupting device
CPU executes the service routine for that device
SW

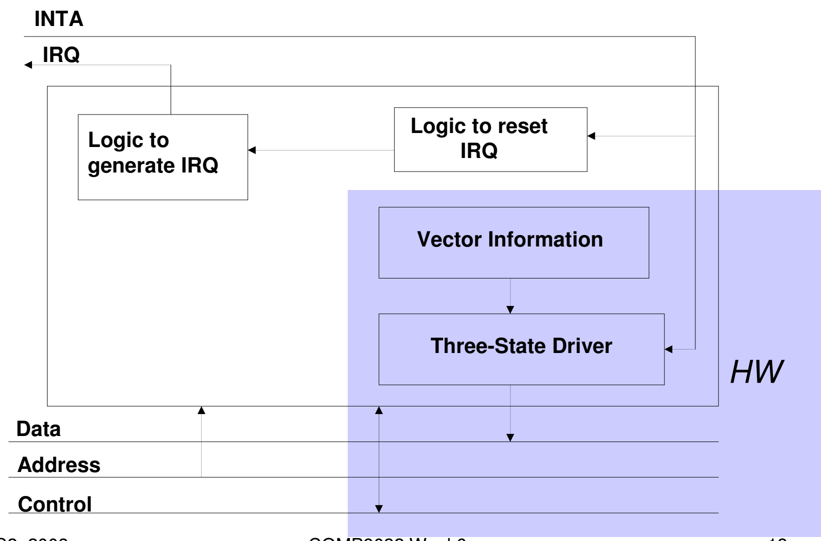
Polled Interrupt Logic


IRQ
Logic to generate IRQ
Logic to reset IRQ when status register is read
Logic to read status register and reset "I did it" bit
Logic to set "I did it" bit
Status register
Data
Address
Control

Vectored Interrupts

- CPU's response to IRQ is to assert INTA.
- The interrupting device uses INTA to place information that identifies itself, called vector, onto the data bus for CPU to read.
- CPU uses the vector to execute the interrupt service routine.

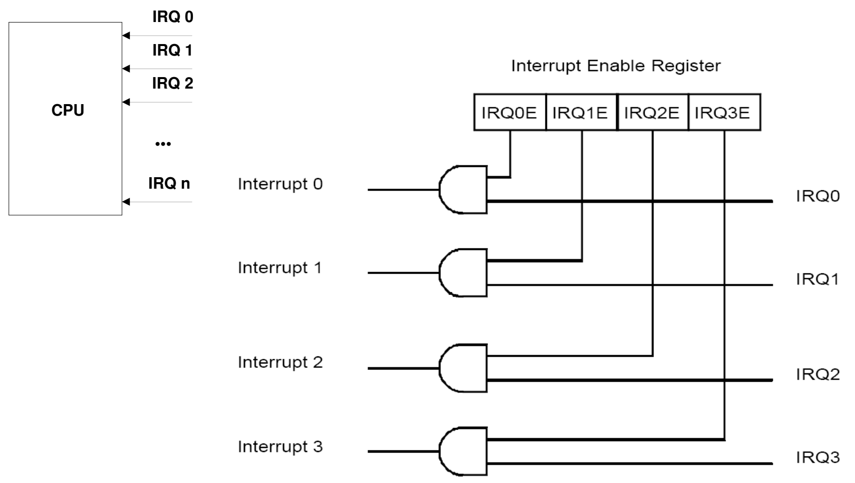
Vectored Interrupting Device Hardware



Multiple Interrupt Masking

- CPU has multiple IRQ input pins.
- Masking enables some interrupts and disables other interrupts
- CPU designers reserve specific memory locations for a vector associated with each IRQ line.
- Individual disable/enable bit is assigned to each interrupting source.

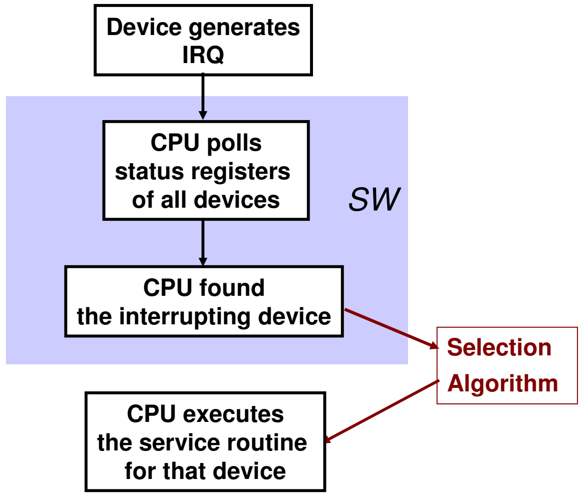
Multiple Interrupt Masking Circuit



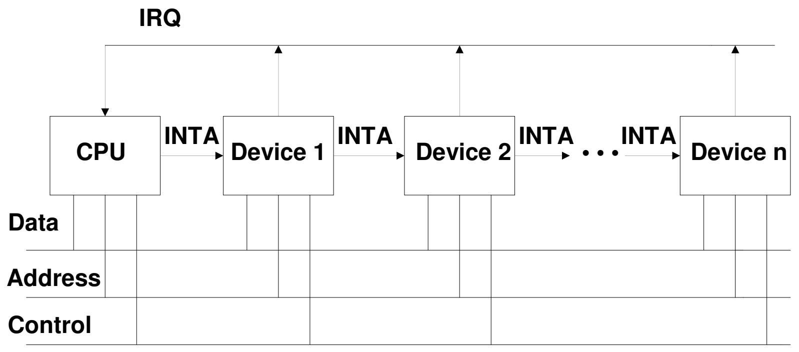
Interrupt Priorities

- When multiple interrupts occur at the same time, which one will be serviced first?
- Two resolution approaches:
  - Software resolution
    - Polling software determines which interrupting source is serviced first.
  - Hardware resolution
    - Daisy chain.
    - Others

Software Resolution


Device generates IRQ
CPU polls status registers of all devices
SW
CPU found the interrupting device
Selection Algorithm
CPU executes the service routine for that device

Daisy Chain Priority Resolution



Daisy Chain Priority Resolution (cont.)

- CPU asserts INTA that is passed down the chain from device to device. The higher-priority device is closer to CPU.
- When the INTA reaches a device that generated the IRQ, that device puts its vector on the data bus and does not pass along the INTA. So lower-priority devices do NOT receive the INTA.

Transferring Control to Interrupt Service Routine

- Hardware needs to save the return address.
  - Most processors save the return address on the stack.
- Hardware may also save some registers such as program status register.
  - AVR does not save any register. It is programmers' responsibility to save program status register and conflict registers.
- The delay from the time the IRQ is generated by the interrupting device to the time the Interrupt Service Routine (ISR) starts to execute is called *interrupt latency*.

Interrupt Service Routine

- A sequence of code to be executed when the corresponding interrupt is responded by CPU.
- Interrupt service routine is a special subroutine, therefore can be constructed with three parts:
  - Prologue:
    - Code for saving conflict registers on the stack.
  - Body:
    - Code for doing the required task.
  - Epilogue:
    - Code for restoring all saved registers from the stack.
    - The last instruction is the return-from-interrupt instruction.

Software Interrupt

- Software interrupt is the interrupt generated by software without a hardware-generated-IRQ.
- Software interrupt is typically used to implement system calls in OS.
- Some processors have a special machine instruction to generate software interrupt.
  - SWI in ARM.
- AVR does NOT provide a software interrupt instruction.
  - Programmers can use External Interrupts to implement software interrupts.

Exceptions

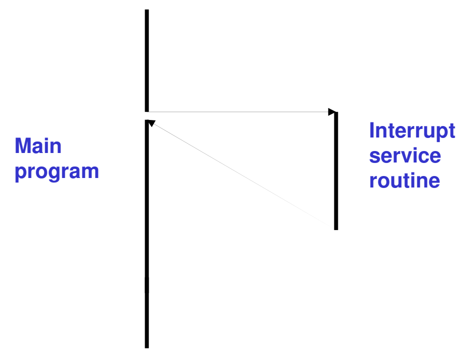
- Abnormalities that occur during the normal operation of the processor.
  - Examples are internal bus error, memory access error and attempts to execute illegal instructions.
- Some processors handle exceptions in the same way as interrupts.
  - AVR does not handle exceptions.

Reset

- Reset is a type of of interrupt in most processors (including AVR).
- Nonmaskable.
- It does not do other interrupt processes, such as saving conflict registers. It initialize the system to some initial state.

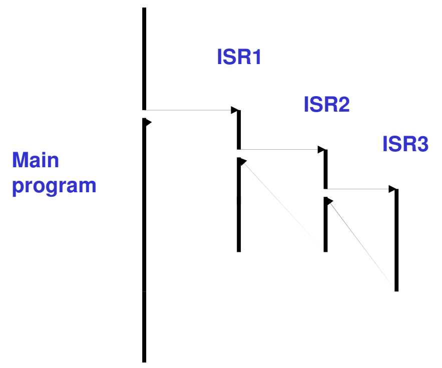
Non-Nested Interrupts

- Interrupt service routines cannot be interrupted by another interrupt.



Nested Interrupts

- Interrupt service routines can be interrupted by another interrupt.



AVR Interrupts

- Basically can be divided into internal and external interrupts
- Each has a separated interrupt vector
- Hardware is used to recognize interrupt
- To enable an interrupt, two control bits must be set
  - the Global Interrupt Enable bit (I bit) in the Status Register
    - Using SEI
  - the enable bit for that interrupt
- To disable all maskable interrupts, reset the I bit in SREG
  - Using CLI instruction
- Priority of interrupts is used to handle multiple simultaneous interrupts

Set Global Interrupt Flag

- Syntax: ***sei***
- Operands: none
- Operation: I 1.
  - Sets the global interrupt flag (I) in SREG. The instruction following SEI will be executed before any pending interrupts.
- Words: 1
- Cycles: 1
- Example:

```
sei     ; set global interrupt enable
sleep   ; enter sleep state, waiting for an interrupt
```

Clear Global Interrupt Flag

- Syntax: ***cli***
- Operands: none
- Operation: I 0
  - Clears the Global interrupt flag in SREG. Interrupts will be immediately disabled.
- Words: 1
- Cycles: 1
- Example:

```
in r18, SREG          ; store SREG value
cli                   ; disable interrupts
; do something very important here
out SREG, r18         ; restore SREG value
```

Interrupt Response Time

- The interrupt execution response for all the enabled AVR interrupts is basically four clock cycles minimum.
  - For saving the Program Counter (1 clock cycle)
  - For jumping to the interrupt routine (3 clock cycles)

Interrupt Vectors

- Each interrupt has a 4-byte (2-word) interrupt vector, containing an instruction to be executed after MCU has accepted the interrupt.
- The lowest addresses in the program memory space are by default defined as the section for Interrupt Vectors.
- The priority of an interrupt is based on the position of its vector in the program memory
  - The lower the address the higher is the priority level.
- RESET has the highest priority

Interrupt Vectors in Mega64

Vector No.	Program Address(2)	Source	Interrupt Definition
1	0x0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	INT2	External Interrupt Request 2
5	0x0008	INT3	External Interrupt Request 3
6	0x000A	INT4	External Interrupt Request 4
7	0x000C	INT5	External Interrupt Request 5
8	0x000E	INT6	External Interrupt Request 6
9	0x0010	INT7	External Interrupt Request 7
10	0x0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	0x0014	TIMER2 OVF	Timer/Counter2 Overflow
12	0x0016	TIMER1 CAPT	Timer/Counter1 Capture Event

Interrupt Vectors in Mega64 (cont.)

13	0x0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	0x001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	0x001C	TIMER1 OVF	Timer/Counter1 Overflow
16	0x001E	TIMER0 COMP	Timer/Counter0 Compare Match
17	0x0020	TIMER0 OVF	Timer/Counter0 Overflow
18	0x0022	SPI, STC	SPI Serial Transfer Complete
19	0x0024	USART0, RX	USART0, Rx Complete
20	0x0026	USART0, UDRE	USART0 Data Register Empty
21	0x0028	USART0, TX	USART0, Tx Complete
22	0x002A	ADC	ADC Conversion Complete
23	0x002C	EE READY	EEPROM Ready
24	0x002E	ANALOG COMP	Analog Comparator

Interrupt Vectors in Mega64 (cont.)

25	0x0030(3)	TIMER1 COMPC	Timer/Countre1 Compare Match C
26	0x0032(3)	TIMER3 CAPT	Timer/Counter3 Capture Event
27	0x0034(3)	TIMER3 COMPA	Timer/Counter3 Compare Match A
28	0x0036(3)	TIMER3 COMPB	Timer/Counter3 Compare Match B
29	0x0038(3)	TIMER3 COMPC	Timer/Counter3 Compare Match C
30	0x003A(3)	TIMER3 OVF	Timer/Counter3 Overflow
31	0x003C(3)	USART1, RX	USART1, Rx Complete
32	0x003E(3)	USART1, UDRE	USART1 Data Register Empty
33	0x0040(3)	USART1, TX	USART1, Tx Complete
34	0x0042(3)	TWI	Two-wire Serial Interface
35	0x0044(3)	SPM READY	Store Program Memory Ready

Interrupt Process

- When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled.
- The user software can set the I-bit to allow nested interrupts
- The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed.
- When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served.
  - Reset interrupt is an exception

Initialization of Interrupt Vector Table in Mega64

- Typically an interrupt vector contains a branch instruction (JMP or RJMP) that branches to the first instruction of the interrupt service routine.
- Or simply RETI (return-from-interrupt) if you don't handle this interrupt.

Example of IVT Initialization in Mega64

```
.include "m64def.inc"
.cseg
.org 0x0000
rjmp RESET      ; Jump to the start of Reset interrupt service routine
                ; Relative jump is used assuming RESET is not far
jmp IRQ0        ; Long jump is used assuming IRQ0 is very far away
reti             ; Return to the break point (No handling for this interrupt).
…
RESET:          ; The interrupt service routine for RESET starts here.
…
IRQ0:           ; The interrupt service routine for IRQ0 starts here.
```

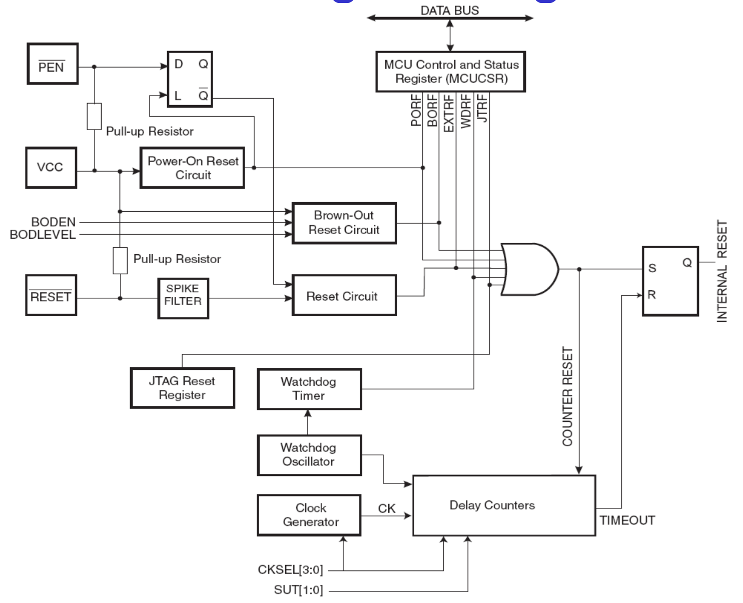
RESET in Mega64

- The ATmega64 has five sources of reset:
  - Power-on Reset.
    - The MCU is reset when the supply voltage is below the Power-on Reset threshold (VPOT).
  - External Reset.
    - The MCU is reset when a low level is present on the RESET pin for longer than the minimum pulse length.
  - Watchdog Reset.
    - The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled.

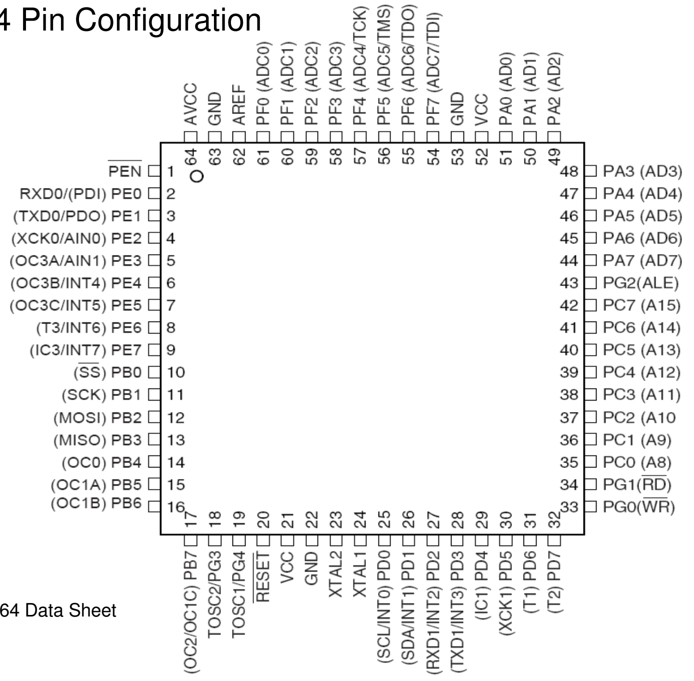
RESET in Mega64 (Cont.)

  - Brown-out Reset.
    - The MCU is reset when the supply voltage VCC is below the Brown-out Reset threshold (VBOT) and the Brown-out Detector is enabled.
  - JTAG AVR Reset.
    - The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system.
- For each reset, there is a flag (bit) in MCU Control and State Register MCUCSR.
  - These bits are used to determine the source of the RESET interrupt.

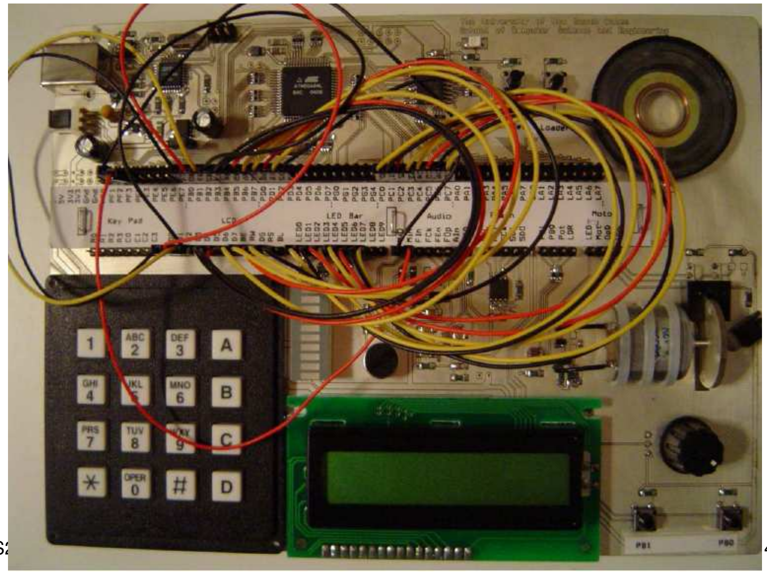
RESET Logic in Mega64



Atmega64 Pin Configuration



Source: Atmega64 Data Sheet



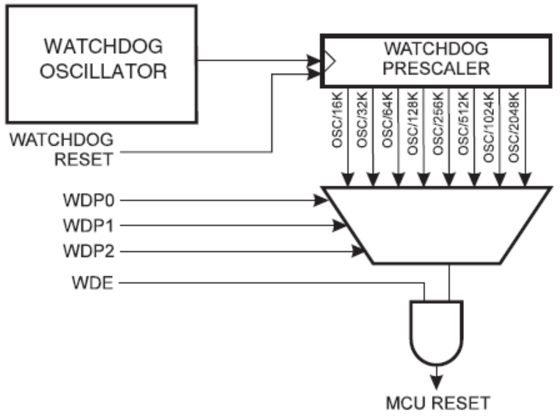
Watchdog Timer

- A peripheral I/O device on the microcontroller.
- It is really a counter that is clocked from a separate On-chip Oscillator (1 Mhz at Vcc=5V)
- It can be controlled to produce different time intervals
  - 8 different periods determined by WDP2, WDP1 and WDP0 bits in WDTCR.
- Can be enabled or disabled by properly updating WDCE bit and WDE bit in Watchdog Timer Control Register WDTCR.

Watchdog Timer (cont.)

- Often used to detect software crash.
  - If enabled, it generates a Watchdog Reset interrupt when its period expires.
    - When its period expires, Watchdog Reset Flag WDRF in MCU Control Register MCUCSR is set.
      - This flag is used to determine if the watchdog timer has generated a RESET interrupt.
  - Program needs to reset it before its period expires by executing instruction *WDR*.

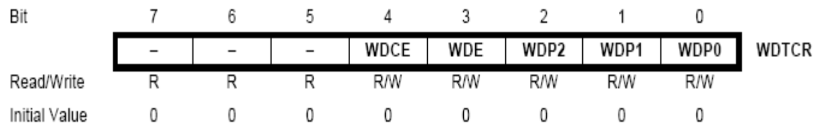
Watchdog Timer Diagram



Source: Atmega64 Data Sheet

Watchdog Timer Control Register

- WDTCR is used to control the scale of the watchdog timer. It is an I/O register in AVR



Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDCE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Source: Atmega64 Data Sheet

WDTCR Bit Definition

- Bits 7-5
  - Not in use. Reserved.
- Bit 4
  - Watchdog change enable
    - Named WDCE
      - Should be set before any changes to be made
- Bit 3
  - Watchdog enable
    - Named WDE
      - Set to enable watchdog; clear to disable the watchdog
- Bits 2-0
  - Prescaler
    - Named WDP2, WDP1, WPD0
      - Determine the watchdog time reset intervals

Watchdog Timer Prescale

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

Source: Atmega64 Data Sheet

Examples

- Enable watchdog

```
; Write logical one to WDE

ldi r16, (1<<WDE)
out WDTCR, r16
```

Examples

- Disable watchdog
  - Refer to the data sheet

```
; Write logical one to WDCE and WDE

ldi r16, (1<<WDCE)|(1<<WDE)
out WDTCR, r16

; Turn off WDT
ldi r16, (0<<WDE)
out WDTCR, r16
```

Examples

- Select a prescale
  - Refer to the data sheet

```
; Write logical one to WDCE and WDE

ldi r16, (1<<WDCE)|(1<<WDE)
out WDTCR, r16

; set time-out as 1 second
ldi r16, (1<<WDP2)|(1<<WDP1)
out WDTCR, r16
```

Watchdog Reset

- Syntax: *wdr*
- Operands: none
- Operation: reset the watchdog timer.
- Words: 1
- Cycles: 1

Example

- The program in the next slide is not robust. May lead to a crash. Why? How to detect the crash?

```
; The program returns the length of a string.

.include "m64def.inc"
.def i=r15          ; store the string length when execution finishes.
.def c=r16          ; store s[i], a string character

.cseg

main:
            ldi ZL, low(s<<1)
            ldi ZH, high(s<<1)
            clr i
            lpm c, z+
loop:
            cpi c, 0
            breq endloop
            inc i
            lpm c, Z+
            rjmp loop
endloop: ...

s: .DB 'h', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd'
```

Reading Material

- Chapter 8: Interrupts and Real-Time Events. Microcontrollers and Microcomputers by Fredrick M. Cady.
- Mega64 Data Sheet.
  - System Control and Reset.
  - Watchdog Timer.
  - Interrupts.

Homework

1. Refer to the AVR Instruction Set manual, study the following instructions:
   - Bit operations
     - sei, cli
     - sbi, cbi
   - MCU control instructions
     - wdr

Homework

1. What is the function of the following code?

```
; Write logical one to WDCE and WDE
        ldi r16, (1<<WDCE)|(1<<WDE)
        out WDTCR, r16

; set time-out as 2.1 second
        ldi r16, (1<<WDP2)|(1<<WDP1)|(1<<WDP0)
        out WDTCR, r16

; enable watchdog
        ldi r16, (1<<WDE)
        out WDTCR, r16

loop:   oneSecondDelay          ; macro for one second delay
        wdr
        rjmp loop
```

Homework

2. How an I/O device signals the microprocessor that it needs service?

Homework

3. Why do you need software to disable interrupts (except for the non-maskable interrupts)?