Interrupts (Filling in the rest!)

Lecturer: Sri Parameswaran
Notes by: Annie Guo
Interrupt Recognition and Acknowledgement Hardware

INTERRUPTING DEVICE

Signal conditioning

IRQ-FF
Set
Reset

Pending Interrupt

Interrupt signal to sequence controller

Interrupt ack from sequence controller

SEQUENCE CONTROLLER

CPU

Disable interrupt instruction

Enable interrupt instruction

Return from interrupt instruction
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.
      - Recall open-collector gates

- 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.
  - 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 Interrupts Execution Flow

1. Device generates IRQ.
2. CPU polls status registers of all devices.
3. CPU found the interrupting device.
4. CPU executes the service routine for that device.

Flow

SW

Device generates IRQ

CPU polls status registers of all devices

CPU found the interrupting device

CPU executes the service routine for that device
Polled Interrupt Logic

- 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 the vector, onto the data bus for CPU to read.
- CPU uses the vector to execute the interrupt service routine.
Vectored Interrupting Device
Hardware

Logic to generate IRQ

Logic to reset IRQ

Vector Information

Three-State Driver

INTA

IRQ

Data

Address

Control

HW
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

CPU

Interrupt Enable Register

IRQ0E IRQ1E IRQ2E IRQ3E

Interrupt 0
IRQ0

Interrupt 1
IRQ1

Interrupt 2
IRQ2

Interrupt 3
IRQ3
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

CPU found the interrupting device

CPU executes the service routine for that device

Selection Algorithm
Daisy Chain Priority Resolution

CPU → Device 1 → Device 2 → ... → Device n

- IRQ
- INTA
- Data
- Address
- Control
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.
Other Priority Resolutions

- Separate IRQ Lines.
  - Each IRQ line is assigned a fixed priority. For example, IRQ0 has higher priority than IRQ1 and IRQ1 has higher priority than IRQ2 and so on.

- Hierarchical Prioritization.
  - Higher priority interrupts are allowed while lower ones are masked.

- Nonmaskable Interrupts.
  - Cannot be disabled.
  - Used for important events such as power failure.
Non-Nested Interrupts

- Interrupt service routines cannot be interrupted by another interrupt.
Nested Interrupts

- Interrupt service routines can be interrupted by another interrupt.
RESET in Mega2560

- The ATmega2560 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 Mega2560 (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 Mega2560

Diagram showing the reset logic flow of a Mega2560 device, including connections and components such as VCC, Power-On Reset Circuit, Brown-Out Reset Circuit, MCU Control and Status Register (MCUCSR), Watchdog Timer, Watchdog Oscillator, and Delay Counters.
Atmega2560 Pin Configuration

Source: Atmega2560 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 (122 kHz at Vcc=5V)
- It can be controlled to produce different time intervals
  - 8 different periods determined by WDP2, WDP1 and WDP0 bits in WDTCSR.
- Can be enabled or disabled by properly updating WDCE bit and WDE bit in Watchdog Timer Control Register WDTCSR.
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: Atmega2560 Data Sheet
Watchdog Timer Control Register

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

Source: Atmega2560 Data Sheet
WDTCSR Bit Definition

- Bit 7 – WDIF - Watchdog interrupt flag
- Bit 6 – WDIE - Watchdog interrupt enable
- Bit 4
  - WDCE - Watchdog change enable
    - Should be set before any changes to be made
- Bit 3
  - WDE - Watchdog enable
    - Set to enable watchdog; clear to disable the watchdog
- Bits 5,2-0
  - Prescaler
    - Named WDP3, WDP2, WDP1, WPD0
      - Determine the watchdog time reset intervals
Watchdog Timer Prescale

<table>
<thead>
<tr>
<th>WDP3</th>
<th>WDP2</th>
<th>WDP1</th>
<th>WDP0</th>
<th>Number of WDT Oscillator Cycles</th>
<th>Typical Time-out at $V_{CC} = 5.0V$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2K (2048) cycles</td>
<td>16ms</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>4K (4096) cycles</td>
<td>32ms</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8K (8192) cycles</td>
<td>64ms</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>16K (16384) cycles</td>
<td>0.125s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>32K (32768) cycles</td>
<td>0.25s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>64K (65536) cycles</td>
<td>0.5s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>128K (131072) cycles</td>
<td>1.0s</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>256K (262144) cycles</td>
<td>2.0s</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>512K (524288) cycles</td>
<td>4.0s</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1024K (1048576) cycles</td>
<td>8.0s</td>
</tr>
</tbody>
</table>

Source: Atmega64 Data Sheet
Examples

- Enable watchdog

```
; Write logical one to WDE
ldi r16, (1<<WDE)
sts WDTCSR, r16
```
Examples

- Disable watchdog
  - Refer to the data sheet

```assembly
; Write logical one to WDCE and WDE
ldi r16, (1<<WDCE)|(1<<WDE)
sts WDTCSR, r16

; Turn off WDT
ldi r16, (0<<WDE)
sts WDTCSR, r16
```
Examples

- Select a prescale
  - Refer to the data sheet

```assembly
; Write logical one to WDCE and WDE
ldi r16, (1<<WDCE)|(1<<WDE)
sts WDTCSR, r16

; set time-out as 1 second
ldi r16, (1<<WDP2)|(1<<WDP1)
sts WDTCSR, r16
```
Watchdog Reset

- Syntax:  \textit{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 "m2560def.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 "hello, world"
Reading Material

- Chapter 8: Interrupts and Real-Time Events. Microcontrollers and Microcomputers by Fredrick M. Cady.
- Mega2560 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?

```assembly
; Write logical one to WDCE and WDE
ldi r16, (1<<WDCE)|(1<<WDE)
sts WDTCSR, r16

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

; enable watchdog
ldi r16, (1<<WDE)
sts WDTCSR, 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)?