COMP2121: Microprocessors and Interfacing

Interrupts

http://www.cse.unsw.edu.au/~cs2121

Lecturer: Hui Wu

Term 2, 2019

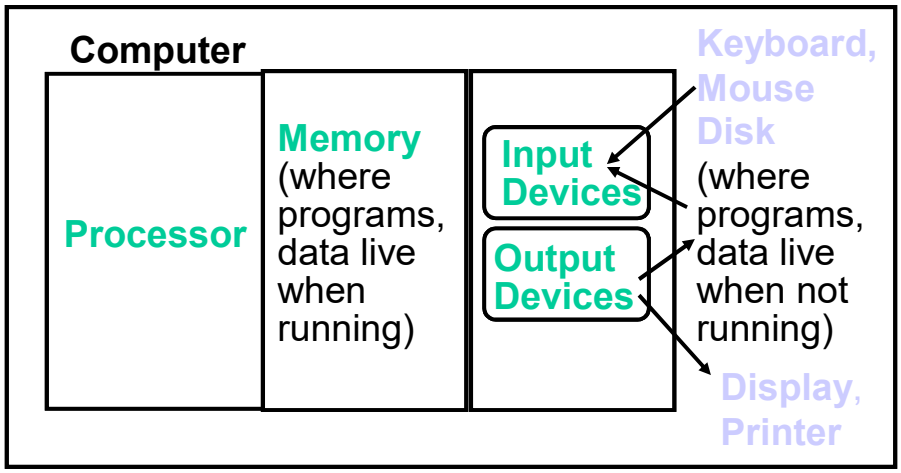
1

Overview

- Interrupt System Specifications
- Multiple Sources of Interrupts
- Interrupt Priorities
- Polling
- AVR Interrupts
- Interrupt Vector Table
- System Reset
- Watchdog Timer
- Timer/Counter0
- External Interrupts
- Interrupt Service Routines

2

Major Components of any Computer



3

How CPU Interacts with I/O?

Two Choices:

- Interrupts.
  - ❑ I/O devices generate signals to request services from CPU .
  - ❑ Need special hardware to implement interrupts.
  - ❑ Efficient.
    - ❖ A signal is generated only if the I/O device needs services from CPU.
- 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.

4

Interrupt System Specifications (1/2)

1. Allow for asynchronous events to occur and be recognized.
2. Wait for the current instruction to finish before taking care of any interrupt.
3. Branch to the correct interrupt handler, also called interrupt service routine, to service the interrupting device.
4. Return to the interrupted program at the point it was interrupted.
5. Allow for a variety of interrupting signals, including levels and edges.
6. Signal the interrupting device with an acknowledge signal when the interrupt has been recognized.

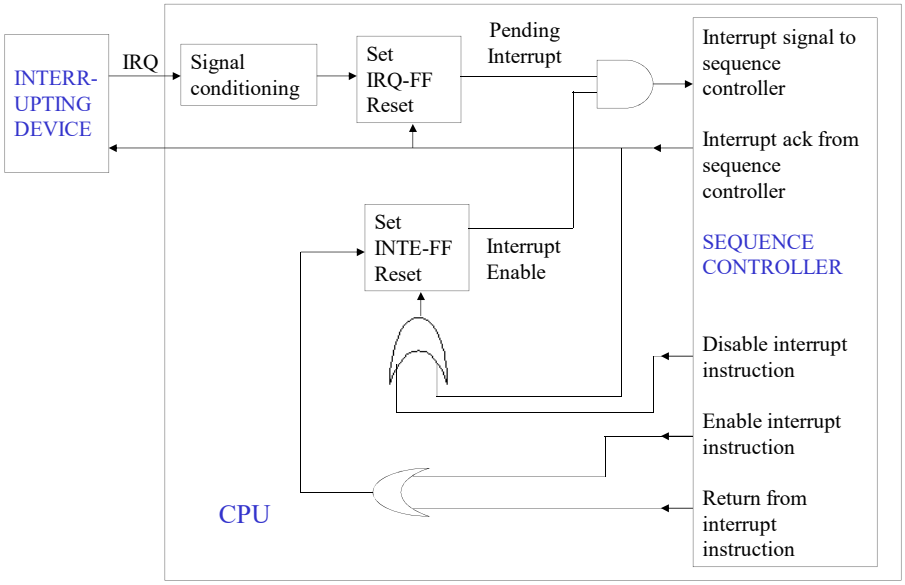
5

Interrupt System Specifications (2/2)

7. Allow programmers to selectively enable and disable all interrupts.
8. Allow programmers to enable and disable selected interrupts.
9. Disable further interrupts while the first is being serviced
10. Deal with multiple sources of interrupts.
11. Deal with multiple, simultaneous interrupts.

6

Interrupt Recognition and Ack (1/3)



7

Interrupt Recognition and Ack (2/4)

- An Interrupt Request (IRQ) may occur at any time.
  - ❑ It may have rising or falling edges or high or low levels.
  - ❑ Frequently it is a active-low signal and multiple devices are wire-ORed together.
- Signal Conditioning Circuit detects these different types of signals.
- Interrupt Request Flip-Flop (IRQ-FF) remembers that an interrupt request has been generated 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.

8

Interrupt Recognition and Ack (3/4)

- The programmer has control over interrupting process by enabling and disabling interrupts with explicit instructions
  - ❑ The hardware that allows this is Interrupt Enable Flip-Flop (INTE-FF).
  - ❑ When the INTE-FF is set, all interrupts are enabled and the pending interrupt 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.

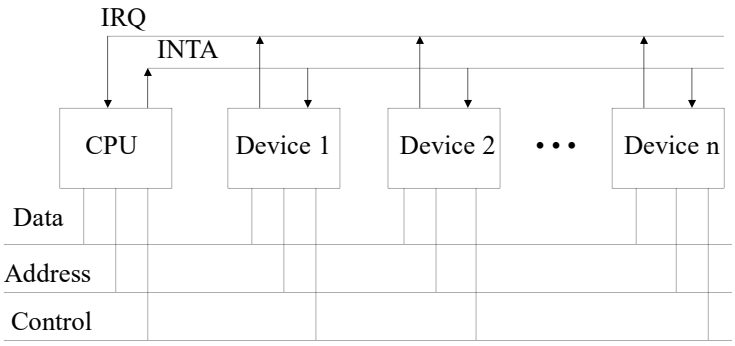
9

Interrupt Recognition and Ack (4/4)

- 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 reenable interrupts.
  - ❑ If the IRQ-FF is set during an interrupt service routine a pending interrupt will be recognized by the sequence controller immediately after the INTE-FF is set. This allows nested interrupts i.e. interrupts interrupting interrupts.

10

Multiple Sources of Interrupts



- Determine which of the multiple devices has generated the IRQ to be able to execute its interrupt service routine.
  - ❑ Two approaches: Polled interrupts and vectored interrupts.
- Resolve simultaneous requests from interrupts with a prioritization scheme.

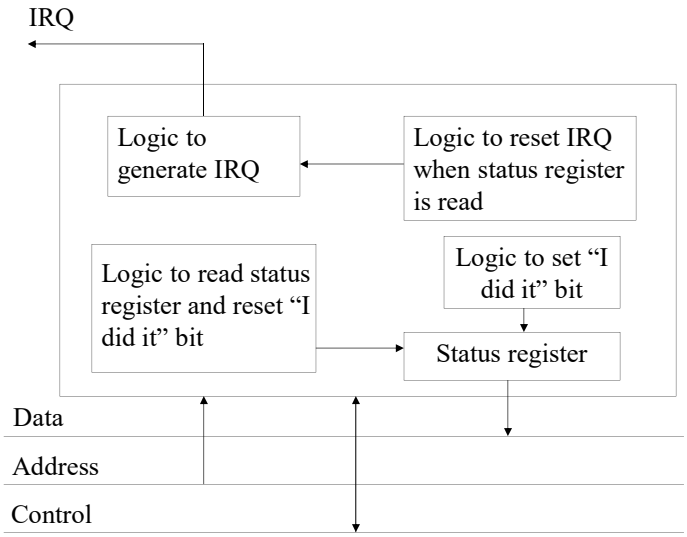
11

Polled Interrupts

- Software, instead of hardware, is responsible for determining the interrupting device.
  - ❑ 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.
- IRQ signals the sequence controller to start executing an interrupt service routine that first polls the device then branches to the correct service routine.

12

Polled Interrupt Logic



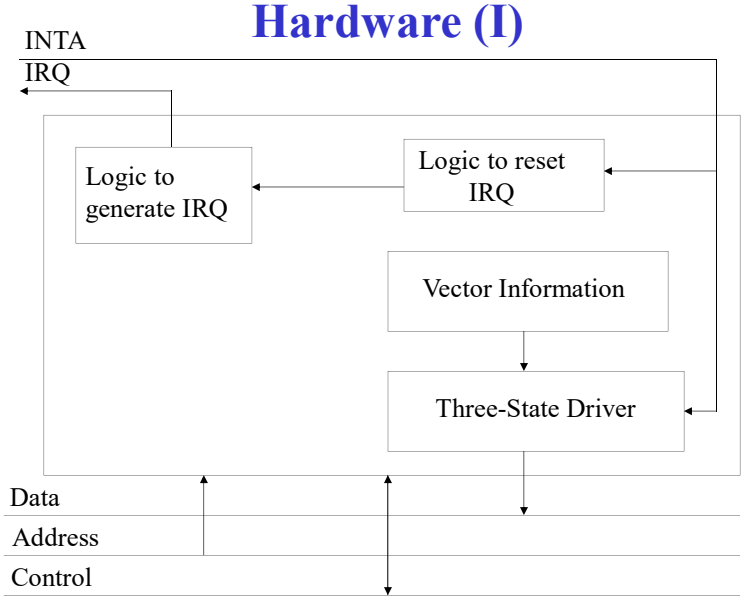
13

Vectored Interrupts (I)

- 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.
  - ❑ An vector is the address of an interrupt service routine.
- CPU uses the vector to execute the interrupt service routine.

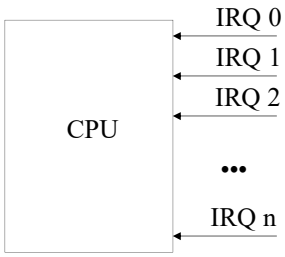
14

Vectored Interrupting Device Hardware (I)



15

Vector Interrupts (II)



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

16

Interrupt Priorities

- When multiple interrupts occurs 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.
    - ❖ Separate IRQ lines.
    - ❖ Hierarchical prioritization.
    - ❖ Nonmaskable interrupts.

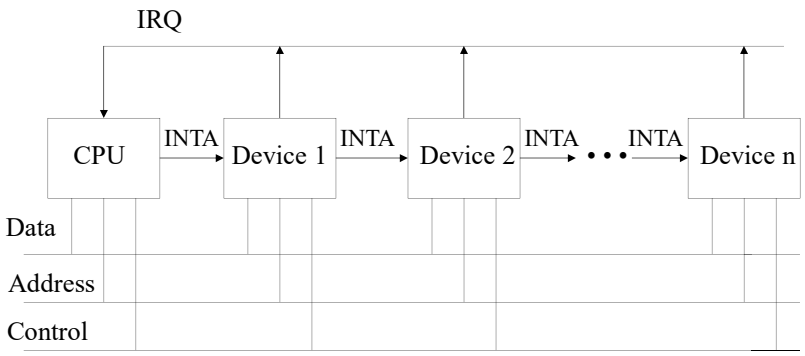
17

Daisy Chain Priority Resolution (1/2)

- 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 the device that generated the IRQ, that device puts its vector on the data bus and not passing along the INTA. So lower-priority devices do NOT receive the INTA.

18

Daisy Chain Priority Resolution (2/2)



19

Hardware Priority Resolution

- 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 priory interrupts are allowed while lower ones are masked.
- Nonmaskable Interrupts.
  - ❑ Cannot be disabled.
  - ❑ Used for important events such as power failure.

20

Transferring Control to Interrupt Handler

- Hardware needs to save the return address.
  - Most processors save the return on the stack.
  - ARM uses a special register, link register, to store the return address.
- Hardware may also save some registers such as program status register.
  - AVR does not save any register. It is programmer's responsibility to save program status register and conflicting registers.
- The delay from the time the IRQ is generated by the interrupting device to the time the interrupt handler starts to execute is called interrupt latency.

21

Interrupt Handler

- A sequence of code to be executed when the corresponding interrupt is responded by CPU.
- Consists of three parts: Prologue, Body and Epilogue.
- Prologue:
  - Code for saving conflicting 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.
    - reti in AVR.

22

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.
- Most processors provide 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.

23

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.

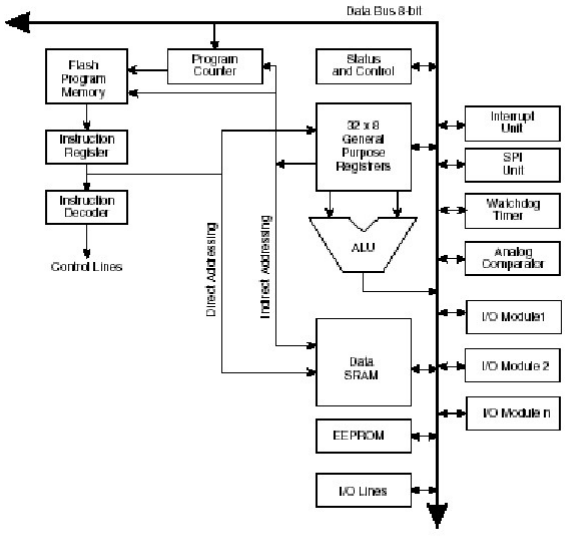
24

Reset

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

25

AVR MCU Architecture



26

Interrupts in AVR Microcontrollers (1/3)

- The number of interrupts varies with specific AVR devices.
- Two types of interrupts: Internal interrupts and external interrupts.
  - Internal interrupts: Generated by on-chip I/O devices.
  - External interrupts: Generated by external I/O devices.
- For most internal interrupts, they don't have an individual enable/disable bit.
  - Program cannot enable/disable these interrupts.
- External interrupts have an individual enable/disable bit.
  - Program can enable/disable these interrupts.
  - An external interrupt can be rising edge-triggered, or falling edge-triggered or low level-triggered).
    - Special I/O registers (External Interrupt Control Registers EICRA and EICRB in ATmega2560) to specify how each external interrupt is triggered.

27

Interrupts in AVR (2/3)

- There is a global interrupt enable/disable bit, the I-bit, in Program Status Register SREG.
  - Setting the I-bit will enable all interrupts except those with individual enable/disable bit. Those interrupts are enabled only if both I and their own enable/disable bit are set.
  - The I-bit is cleared when an interrupt occurs and is set by the instruction reti.
  - Programmers can use sei and cli to set and clear the I-bit.
  - If the I-bit is enabled in the interrupt service routine, nested interrupts are allowed.
- SREG is not automatically saved by hardware when entering an interrupt service routine.
  - An interrupt service routine needs to save it and other conflict registers on the stack at the beginning and restore them at the end.

28

Interrupts in AVR (3/3)

- Reset is handled as a nonmaskable interrupt.
- Each interrupt has a 4-byte interrupt vector, containing an instruction to be executed after MCU has accepted the interrupt.
- Each interrupt vector has a vector number, an integer from 1 to n, the maximum number of interrupts.
- The priority of each interrupt is determined by its vector number.
  - The lower the vector number, the higher priority.
- All interrupt vectors, called Interrupt Vector Table, are stored in a contiguous section in flash memory.
  - Starts from address 0 by default.
  - Can be relocated.

29

Interrupt Vectors in ATmega2560 (1/4)

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B

30

Interrupt Vectors in ATmega2560 (2/4)

16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator

31

Interrupt Vectors in ATmega2560 (3/4)

30	$003A	ADC	ADC Conversion Complete
31	$003C	EE READY	EEPROM Ready
32	$003E	TIMER3 CAPT	Timer/Counter3 Capture Event
33	$0040	TIMER3 COMPA	Timer/Counter3 Compare Match A
34	$0042	TIMER3 COMPB	Timer/Counter3 Compare Match B
35	$0044	TIMER3 COMPC	Timer/Counter3 Compare Match C
36	$0046	TIMER3 OVF	Timer/Counter3 Overflow
37	$0048	USART1 RX	USART1 Rx Complete
38	$004A	USART1 UDRE	USART1 Data Register Empty
39	$004C	USART1 TX	USART1 Tx Complete
40	$004E	TWI	2-wire Serial Interface
41	$0050	SPM READY	Store Program Memory Ready
42	$0052[3]	TIMER4 CAPT	Timer/Counter4 Capture Event
43	$0054	TIMER4 COMPA	Timer/Counter4 Compare Match A
44	$0056	TIMER4 COMPB	Timer/Counter4 Compare Match B
45	$0058	TIMER4 COMPC	Timer/Counter4 Compare Match C

32

Interrupt Vectors in ATmega2560 (4/4)

46	$005A	TIMER4 OVF	Timer/Counter4 Overflow
47	$005C(3)	TIMER5 CAPT	Timer/Counter5 Capture Event
48	$005E	TIMER5 COMPA	Timer/Counter5 Compare Match A
49	$0060	TIMER5 COMPB	Timer/Counter5 Compare Match B
50	$0062	TIMER5 COMPC	Timer/Counter5 Compare Match C
51	$0064	TIMER5 OVF	Timer/Counter5 Overflow
52	$0066(3)	USART2 RX	USART2 Rx Complete
53	$0068(3)	USART2 UDRE	USART2 Data Register Empty
54	$006A(3)	USART2 TX	USART2 Tx Complete
55	$006C(3)	USART3 RX	USART3 Rx Complete
56	$006E(3))	USART3 UDRE	USART3 Data Register Empty
57	$0070(3)	USART3 TX	USART3 Tx Complete

33

Interrupt Vectors in ATmega2560 (2/3)

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
25	0x0030(3)	TIMER1 COMPC	Timer/Countre1 Compare Match C

34

Interrupt Vector Initialization in ATmega2560 (1/5)

• Typically an interrupt vector contains a branch instruction (jmp) that branches to the first instruction of the interrupt handler, or reti (return from interrupt), indicating that this interrupt is not handled.

35

Interrupt Vector Initialization in ATmega2560 (2/5)

```
Address   Labels   Code                    Comments
0x0000             jmp     RESET           ; Reset Handler
0x0002             jmp     INT0            ; IRQ0 Handler
0x0004             jmp     INT1            ; IRQ1 Handler
0x0006             jmp     INT2            ; IRQ2 Handler
0x0008             jmp     INT3            ; IRQ3 Handler
0x000A             jmp     INT4            ; IRQ4 Handler
0x000C             jmp     INT5            ; IRQ5 Handler
0x000E             jmp     INT6            ; IRQ6 Handler
0x0010             jmp     INT7            ; IRQ7 Handler
0x0012             jmp     PCINT0          ; PCINT0 Handler
0x0014             jmp     PCINT1          ; PCINT1 Handler
0x0016             jmp     PCINT2          ; PCINT2 Handler
0X0018             jmp     WDT             ; Watchdog Timeout Handler
0x001A             jmp     TIM2_COMPA      ; Timer2 CompareA Handler
0x001C             jmp     TIM2_COMPB      ; Timer2 CompareB Handler
```

36

Interrupt Vector Initialization in ATmega2560 (3/5)

```
0x001E    jmp    TIM2_OVF        ; Timer2 Overflow Handler
0x0020    jmp    TIM1_CAPT       ; Timer1 Capture Handler
0x0022    jmp    TIM1_COMPA      ; Timer1 CompareA Handler
0x0024    jmp    TIM1_COMPB      ; Timer1 CompareB Handler
0x0026    jmp    TIM1_COMPC      ; Timer1 CompareC Handler
0x0028    jmp    TIM1_OVF        ; Timer1 Overflow Handler
0x002A    jmp    TIM0_COMPA      ; Timer0 CompareA Handler
0x002C    jmp    TIM0_COMPB      ; Timer0 CompareB Handler
0x002E    jmp    TIM0_OVF        ; Timer0 Overflow Handler
0x0030    jmp    SPI_STC         ; SPI Transfer Complete Handler
0x0032    jmp    USART0_RXC      ; USART0 RX Complete Handler
0x0034    jmp    USART0_UDRE     ; USART0,UDR Empty Handler
0x0036    jmp    USART0_TXC      ; USART0 TX Complete Handler
0x0038    jmp    ANA_COMP        ; Analog Comparator Handler
0x003A    jmp    ADC             ; ADC Conversion Complete Handler
0x003C    jmp    EE_RDY          ; EEPROM Ready Handler
0x003E    jmp    TIM3_CAPT       ; Timer3 Capture Handler
```

37

Interrupt Vector Initialization in ATmega2560 (4/5)

```
0x0040    jmp    TIM3_COMPA      ; Timer3 CompareA Handler
0x0042    jmp    TIM3_COMPB      ; Timer3 CompareB Handler
0x0044    jmp    TIM3_COMPC      ; Timer3 CompareC Handler
0x0046    jmp    TIM3_OVF        ; Timer3 Overflow Handler
0x0048    jmp    USART1_RXC      ; USART1 RX Complete Handler
0x004A    jmp    USART1_UDRE     ; USART1,UDR Empty Handler
0x004C    jmp    USART1_TXC      ; USART1 TX Complete Handler
0x004E    jmp    TWI             ; 2-wire Serial Handler
0x0050    jmp    SPM_RDY         ; SPM Ready Handler
0x0052    jmp    TIM4_CAPT       ; Timer4 Capture Handler
0x0054    jmp    TIM4_COMPA      ; Timer4 CompareA Handler
0x0056    jmp    TIM4_COMPB      ; Timer4 CompareB Handler
0x0058    jmp    TIM4_COMPC      ; Timer4 CompareC Handler
0x005A    jmp    TIM4_OVF        ; Timer4 Overflow Handler
0x005C    jmp    TIM5_CAPT       ; Timer5 Capture Handler
0x005E    jmp    TIM5_COMPA      ; Timer5 CompareA Handler
```

38

Interrupt Vector Initialization in ATmega2560 (5/5)

```
0x0060          jmp     TIM5_COMPB      ; Timer5 CompareB Handler
0x0062          jmp     TIM5_COMPC      ; Timer5 CompareC Handler
0x0064          jmp     TIM5_OVF        ; Timer5 Overflow Handler
0x0066          jmp     USART2_RXC      ; USART2 RX Complete Handler
0x0068          jmp     USART2_UDRE     ; USART2,UDR Empty Handler
0x006A          jmp     USART2_TXC      ; USART2 TX Complete Handler
0x006C          jmp     USART3_RXC      ; USART3 RX Complete Handler
0x006E          jmp     USART3_UDRE     ; USART3,UDR Empty Handler
0x0070          jmp     USART3_TXC      ; USART3 TX Complete Handler
;
0x0072  RESET:  ldi     r16, high(RAMEND)  ; Main program start
0x0073          out     SPH,r16            ; Set Stack Pointer to top of RAM
0x0074          ldi     r16, low(RAMEND)
0x0075          out     SPL,r16
0x0076          sei                        ; Enable interrupts
0x0077          <instr> xxx
  ...     ...     ...     ...
```

39

An Example (1/2)

```
.include "m2560def.inc"
.cseg
.org 0x0000   ; Reset vector is at address 0x0000
jmp RESET     ; Jump to the start of Reset interrupt handler
.org INT0addr ; INT0addr is the address of INT0 defined in m2560def.inc
jmp IRQ0      ; Jump to the start of the interrupt handler of IRQ0
.org INT1addr ; INT1addr is the address of INT1 defined in m2560def.inc
reti          ; Return to the breakpoint where INT1 occurred
…
.org 0x0072   ; Next instruction start at 0x0072
IRQ0: push r0 ; Save r0
      push r1 ; Save r1
      …       ; Body of the interrupt handler of INT0
      pop r1  ; Restore r1
      pop r0  ; Restore r0
      reti    ; Return to the breakpoint where INT0 occurred
```

40

An Example (2/2)

```
RESET: ldi r16, high(RAMEND)   ; Interrupt handler for RESET
out SPH, r16                   ; Set the stack pointer SP to the top of data memory
ldi r16, low(RAMEND)
out SPL,r16
sei                            ; Enable interrupts
main:                          ; Main program starts here
…
loopforever: rjmp loopforever  ; Infinite loop
```

41

RESET in ATmega2560 (1/2)

A RESET interrupt is used to restore the microcontroller to an initial state.

ATmega2560 microcontroller 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.

42

RESET in ATmega2560 (2/2)

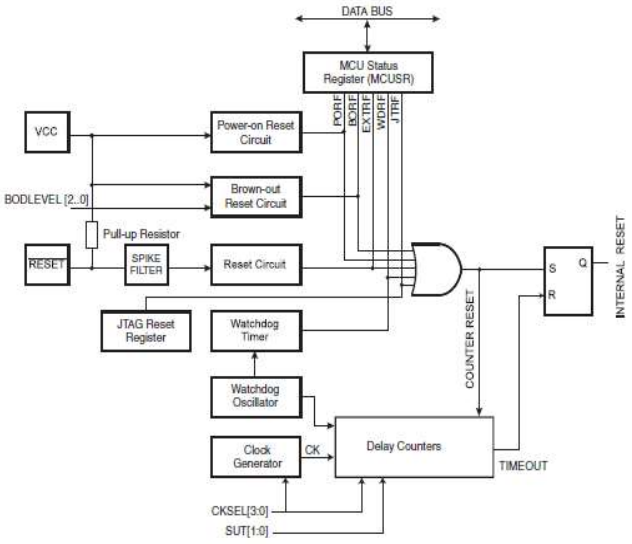
- Brown-out Reset
  - ❑ The MCU is reset when the supply voltage VCC is below the Brown-out Reset threshold (VBOT) and the Brown-out Detector is enabled.
- JTAG AVR Reset
  - ❑ The MCU is reset as long as there is a logic one in the Reset Register, one of the scan chains of the JTAG system.

For each RESET, there is a flag (bit) in MCU Control Register MCUCSR.

- These bits are used to determine the source of the RESET interrupt.

43

RESET Logic in ATmega2560



44

MCU Status Register in ATmega2560 (1/3)

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

Bit 4 – JTRF: JTAG Reset Flag

This bit is set if a reset is being caused by a logic one in the JTAG Reset Register selected by the JTAG instruction AVR_RESET. This bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

Bit 3 – WDRF: Watchdog Reset Flag

This bit is set if a Watchdog Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

45

MCU Status Register in ATmega2560 (2/3)

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

Bit 2 – BORF: Brown-out Reset Flag

This bit is set if a Brown-out Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

Bit 1 – EXTRF: External Reset Flag

This bit is set if an External Reset occurs. The bit is reset by a Power-on Reset, or by writing a logic zero to the flag.

46

MCU Status Register in ATmega2560 (3/3)

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	–	–	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUSR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

Bit 0 – PORF: Power-on Reset Flag This bit is set if a Power-on Reset occurs. The bit is reset only by writing a logic zero to the flag. To make use of the Reset Flags to identify a reset condition, the user should read and then

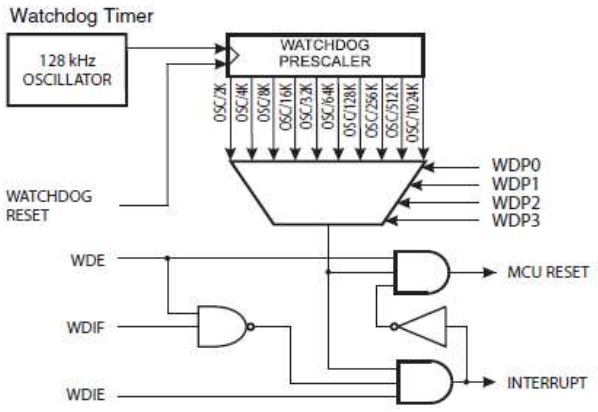
47

Watchdog Timer in ATmega2560

- Used to detect software crash.
- Can be enabled or disabled by properly updating WDCE bit and WDE bit in Watchdog Timer Control Register WDTCR.
- 10 different periods determined by WDP3, WDP2, WDP1 and WDP0 bits in WDTCR.
- If enabled, it generates a Watchdog Reset interrupt when its period expires.
- So program needs to reset it before its period expires by executing instruction WDR.
- 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.

48

Watchdog Timer Logic



49

Watchdog Timer Period

Table 12-2. Watchdog Timer Prescale Select

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

50

Watchdog Timer Control Register (1/4)

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Bit 7 - WDIF: Watchdog Interrupt Flag

This bit is set when a time-out occurs in the Watchdog Timer and the Watchdog Timer is configured for interrupt. WDIF is cleared by hardware when executing the corresponding interrupt handling vector. Alternatively, WDIF is cleared by writing a logic zero to the flag. When the I-bit in SREG and WDIE are set, the Watchdog Time-out Interrupt is executed.

51

Watchdog Timer Control Register (2/4)

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Bit 6 - WDIE: Watchdog Interrupt Enable

When this bit is written to one and the I-bit in the Status Register is set, the Watchdog Interrupt is enabled. If WDE is cleared in combination with this setting, the Watchdog Timer is in Interrupt Mode, and the corresponding interrupt is executed if time-out in the Watchdog Timer occurs. If WDE is set, the Watchdog Timer is in Interrupt and System Reset Mode. The first time-out in the Watchdog Timer will set WDIF. Executing the corresponding interrupt vector will clear WDIE and WDIF automatically by hardware (the Watchdog goes to System Reset Mode).

52

Watchdog Timer Control Register (3/4)

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Bit 4 - WDCE: Watchdog Change Enable

This bit is used in timed sequences for changing WDE and prescaler bits. To clear the WDE bit, and/or change the prescaler bits, WDCE must be set. Once written to one, hardware will clear WDCE after four clock cycles.

Bit 3 - WDE: Watchdog System Reset Enable

WDE is overridden by WDRF in MCUSR. This means that WDE is always set when WDRF is set. To clear WDE, WDRF must be cleared first.

53

Watchdog Timer Control Register (4/4)

Bit	7	6	5	4	3	2	1	0	
(0x60)	WDIF	WDIE	WDP3	WDCE	WDE	WDP2	WDP1	WDP0	WDTCSR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	0	0	0	

Bit 5, 2:0 - WDP3:0: Watchdog Timer Prescaler 3, 2, 1 and 0

The WDP3:0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is running.

54

Timer Interrupt (1/2)

Timer interrupt has many applications:

- Used to schedule (real-time) tasks (threads)
  - ❑ Round-Robin scheduling
    - ❖ All tasks take turn to execute for some fixed period.
  - ❑ Real-time scheduling
    - ❖ Some tasks must be started at a particular time and finished by a deadline.
    - ❖ Some tasks must be periodically executed.
- Used to implement a clock
  - ❑ How much time has passed since the system started?

55

Timer Interrupt (2/2)

- Used to synchronize tasks.
  - ❑ Task A can be started only if a certain amount of time has passed since the completion of task B.
- Can be coupled with a wave-form generator to support Pulse-Width Modulation (PWM).
  - ❑ Details to be covered in the lecture about Analog Input and Output.

56

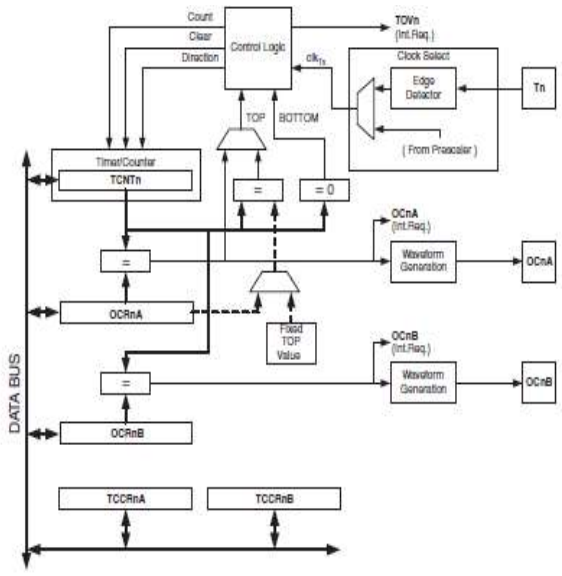
Timer0 in ATmega2560

The 8-bit timer has the following features:

- Glitch-free, Phase Correct Pulse Width Modulators (PWM).
- 10-bit Clock Prescaler.
- The counter can count up or count down depending on the specific operation mode.
- One Overflow Interrupt source (TOV0) and two Compare Match Interrupt sources (OC0A and OC0B).
  - ❑ The timer can generate a Timer0 Overflow Interrupt TOV0 when the counter TCNT0 overflows.
  - ❑ The timer can generate two Timer/Counter0 Output Match Interrupts: Timer/Counter Compare Match A interrupt and Timer/Counter Compare Match B interrupt when TCNT0 contains the same value as in Output Compare Registers (OCR0A and OCR0B)
  - ❑ TOV0 and two Timer/Counter0 Output Match Interrupts can be individually enabled/disabled.

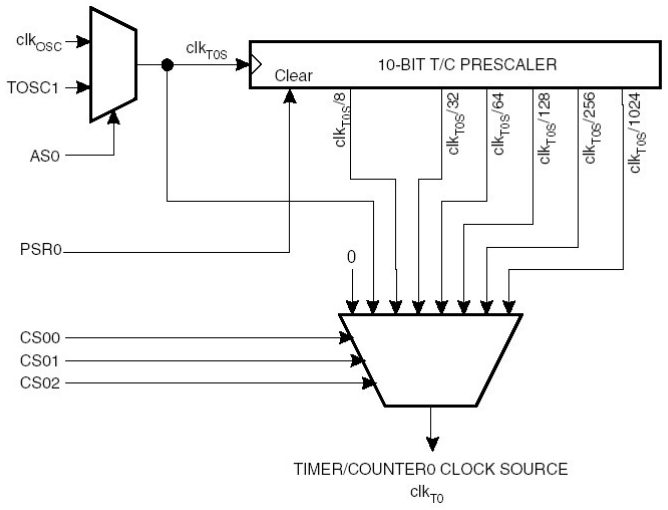
57

Timer0 in ATmega2560–Block Diagram



58

Prescaler for Timer0



59

Timer0 Operation Modes (1/2)

- Timer0 can operate in different operation modes.
- An operation mode specifies the behaviour of the timer and the Output Compare pins.
  - The behaviour of Output Compare pins and PWM modes will be covered in the lecture about Analog Input and Output.
- A mode is defined by the combination of the Waveform Generation mode (WGM2:0 in TCCR0A and TCCR0B registers) and Compare Output mode (COM0x1:0 in TCCR0A and TCCR0B registers) (x=A or B) bits. The COM0x1:0 bits control whether the output should be set, cleared, or toggled at a Compare Match.

60

Timer0 Operation Modes (2/2)

- The simplest operation mode is the normal mode (WGM02:0 = 000 in TCCR0A and TCCR0B registers), which is typically used to generate an interrupt, either a Timer Overflow Interrupt, or a Compare Match Interrupt.
- In the normal mode, the counting direction is always up (incrementing). The counter simply overruns when it passes its maximum 8-bit value 0xFF and then restarts from 0x00.
  - ❑ We focus on the normal mode now, and will learn the other modes in the lecture about Analog Input and Output.

61

Timer0 Registers (1/8)

I/O registers for Timer0:

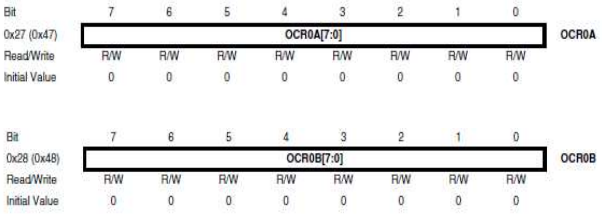
- Timer/Counter Register TCNT0.
  - ❑ Contains the current timer/counter value.
  - ❑ The value is increased or decreased by one every clock cycle based on the specific operation mode.

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

62

Timer0 Registers (2/8)

- Output Compare Registers OCR0A and OCR0B.
  - ❑ Each contains an 8-bit value set by the application program.
  - ❑ Timer0 continuously compares the value of OCR0A (OCR0B) with the counter value of TCNT0.
  - ❑ When the two values of TCNT0 and OCR0A (OCR0B) are equal, the timer may generate a Timer/Counter Compare Match A interrupt (Timer/Counter Compare Match B interrupt).



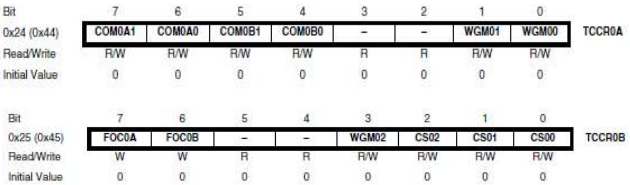
Bit	7	6	5	4	3	2	1	0	
0x27 (0x47)	OCR0A[7:0]								OCR0A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x28 (0x48)	OCR0B[7:0]								OCR0B
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

63

Timer0 Registers (3/8)

- Timer/Counter Control Registers TCCR0A and TCCR0B.
  - ❑ Contains control bits.
  - ❑ Normal mode when WGM02:0=000.
  - ❑ By default, WGM02:0=000.



Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

64

Timer0 Registers (4/8)

- Bits CS02:0 select the clock for Timer0.
- No prescaling when CS02=001.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge

65

Timer0 Registers (5/8)

- Timer/Counter Interrupt Mask Register TIMSK0.
  - ❑ Contains interrupt enable/disable bits.
  - ❑ Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable bit.
    - ❑ When the OCIE0B bit is written to one, and the I-bit in the Status Register SREG is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed when a Compare Match in Timer/Counter occurs.

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

66

Timer0 Registers (6/8)

- Timer/Counter Interrupt Mask Register TIMSK0.
  - ❑ Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable bit.
    - ❑ When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs.
  - ❑ Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable bit.
    - ❑ When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs.

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

67

Timer0 Registers (7/8)

- Timer/Counter Interrupt Flag Register TIFR0.
  - ❑ Contains interrupt flags.
  - ❑ Bit 2 – OCF0B: Timer/Counter 0 Output Compare B Match Flag.
    - ❑ The OCF0B bit is set when a Compare Match occurs between the Timer/Counter and the data in OCR0B – Output Compare Register0 B. OCF0B is cleared by hardware when executing the corresponding interrupt handler. Alternatively, OCF0B is cleared by writing a logic zero to the flag. When the I-bit in SREG, OCIE0B (Timer/Counter Compare B Match Interrupt Enable), and OCF0B are set, the Timer/Counter Compare Match Interrupt is executed.

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

68

Timer0 Registers (8/8)

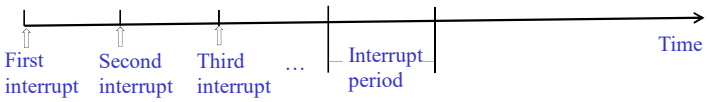
- Timer/Counter Interrupt Flag Register TIFR0.
  - ❑ Bit 1 – OCF0A: Timer/Counter 0 Output Compare A Match Flag.
    - ❑ The OCF0A bit is set when a Compare Match occurs between the Timer/Counter0 and the data in OCR0A – Output Compare Register0. OCF0A is cleared by hardware when executing the corresponding interrupt handler. Alternatively, OCF0A is cleared by writing a logic zero to the flag. When the I-bit in SREG, OCIE0A (Timer/Counter0 Compare Match Interrupt Enable), and OCF0A are set, the Timer/Counter0 Compare Match Interrupt is executed.

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

69

An Interrupt Handler Example (1/9)

- We design a program using AVR assembly language with the following behavior:
  - ❑ Initially, 8 LEDs are set on.
  - ❑ Every second, toggle the 8 LEDs, that is, if an LED is on, turn it off, and if it is off, turn it on.
- Key question: How do we know if one second has passed?
  - ❑ Answer: use Timer0 Overflow Interrupt.
  - ❑ Need to know the interrupt period of Timer0 Overflow Interrupt.



70

An Interrupt Handler Example (2/9)

- Assume the ATmega2560 microcontroller runs at a clock frequency of 16M Hz, and the Timer0 prescalar value is set to 8.
  - The interrupt period = 256*8/16 = 128 us, i.e., every 128 us, Timer0 generates a Timer0 Overflow Interrupt.
  - How many Timer0 Overflow Interrupts are generated each second?
    - 1,000,000/128 =7812.5 interrupts ≈ 7812 interrupts
- We use PortC to drive 8 LEDs.

71

An Interrupt Handler Example (3/9)

```
.include "m2560def.inc"
.equ PATTERN = 0b11110000  ; define a pattern for 8 LEDs
.def  temp = r16
.def  leds = r17   ; r17 stores a LED pattern
; The macro clears a word (2 bytes) in the data memory
; The parameter @0 is the memory address for that word
.macro clear
ldi YL, low(@0)     ; load the memory address to Y pointer
ldi YH, high(@0)
 clr temp           ; set temp to 0
 st Y+, temp        ; clear the two bytes at @0 in SRAM
 st Y, temp
.endmacro
```

72

An Interrupt Handler Example (4/9)

```
.dseg
SecondCounter: .byte 2     ; two-byte counter for counting seconds.
TempCounter: .byte 2       ; temporary counter used to determine if one second has passed
.cseg
.org 0x0000
 jmp RESET
 jmp DEFAULT     ; no handling for IRQ0.
 jmp DEFAULT     ; no handling for IRQ1.
.org OVF0addr      ; OVF0addr is the address of Timer0 Overflow Interrupt Vector
 jmp Timer0OVF   ; jump to the interrupt handler for Timer0 overflow.
 …
 jmp DEFAULT     ; default service for all other interrupts.
 DEFAULT: reti     ; no interrupt handling
```

73

An Interrupt Handler Example (5/9)

```
RESET: ldi temp, high(RAMEND)  ; initialize the stack pointer SP
       out SPH, temp
       ldi temp, low(RAMEND)
       out SPL, temp
       ser temp       ; set Port C as output
       out DDRC, temp
       rjmp main     ; jump to main program
```

74

An Interrupt Handler Example (6/9)

```
Timer0OVF:    ; interrupt subroutine to Timer0
        in temp, SREG
        push temp      ; prologue starts
        push YH        ; save all conflicting registers in the prologue
        push YL
        push r25
        push r24       ; prologue ends
        ; Load the value of the temporary counter
        lds r24, TempCounter
        lds r25, TempCounter+1
        adiw r25:r24, 1      ; increase the temporary counter by one
```

75

An Interrupt Handler Example (7/9)

```
        cpi r24, low(7812)        ; check if (r25:r24) = 7812
        ldi temp, high(7812)      ; 7812 = 106/128
        cpc r25, temp
        brne NotSecond
        com leds                  ; one second has passed, and toggle LEDs now
        out PORTC, leds
        clear TempCounter         ; reset the temporary counter
        ; Load the value of the second counter
        lds r24, SecondCounter
        lds r25, SecondCounter+1
        adiw r25:r24, 1           ; increase the second counter by one
```

76

An Interrupt Handler Example (8/9)

```
            sts SecondCounter, r24
            sts SecondCounter+1, r25
            rjmp EndIF
NotSecond:  ; store the new value of the temporary counter
            sts TempCounter, r24
            sts TempCounter+1, r25
EndIF:      pop r24       ; epilogue starts
            pop r25       ; restore all conflicting registers from the stack
            pop YL
            pop YH
            pop temp
            out SREG, temp
            reti          ; return from the interrupt
```

77

An Interrupt Handler Example (9/9)

```
main:  ldi leds, 0xFF        ; main program starts here
       out PORTC, leds       ; set all LEDs on at the beginning
       ldi leds, PATTERN
       clear TempCounter     ; initialize the temporary counter to 0
       clear SecondCounter   ; initialize the second counter to 0
       ldi temp, 0b00000000
       out TCCR0A, temp
       ldi temp, 0b00000010
       out TCCR0B, temp      ; set prescalar value to 8
       ldi temp, 1<<TOIE0    ; TOIE0 is the bit number of TOIE0 which is 0
       sts TIMSK0, temp      ; enable Timer0 Overflow Interrupt
       sei                   ; enable global interrupt
loop:  rjmp loop             ; loop forever
```

78

External Interrupts

- The external interrupts are triggered by the INT7:0 pins or any of the PCINT23:0 pins.
  - ❑ We use the INT7:0 pins only for external interrupts.
- An external interrupt can be used to generate a software interrupt.
- To enable an external interrupt INTx, the following two bits must be set:
  - ❑ The I bit in SREG, and
  - ❑ The INTx bit in the EIMSK register.
- Each external interrupt can be triggered by a falling or rising edge or a low level.
  - ❑ External Interrupt Control Registers EICRA (for INT3:0) and EICRB (for INT7:4) specify how external interrupts are triggered.

79

Registers for External Interrupts (1/4)

- EICRA – External Interrupt Control Register A
  - ❑ EICRA contains bits for interrupt sense control (INT3:0).

Bit	7	6	5	4	3	2	1	0	
(0x69)	ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

80

Registers for External Interrupts (2/4)

- EICRB – External Interrupt Control Register B
  - ❑ EICRB contains bits for interrupt sense control (INT7:4).□

Bit	7	6	5	4	3	2	1	0	
(0x6A)	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ISCn1	ISCn0	Description
0	0	The low level of INTn generates an interrupt request
0	1	Any edge of INTn generates asynchronously an interrupt request
1	0	The falling edge of INTn generates asynchronously an interrupt request
1	1	The rising edge of INTn generates asynchronously an interrupt request

81

Registers for External Interrupts (3/4)

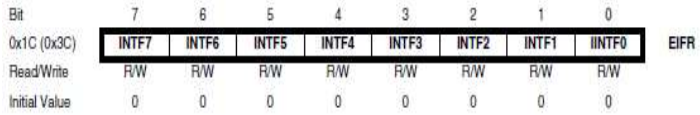
- EIMSK – External Interrupt Mask Register
  - ❑ Bits 7:0 – INT7:0: External Interrupt Request 7 - 0 Enable bits.
  - ❑ When an INT7:0 bit is written to one and the I-bit in the Status Register (SREG) is set to one, the corresponding external pin interrupt is enabled.

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

82

Registers for External Interrupts (4/4)

- EIFR – External Interrupt Flag Register
  - ❑ INTF7:0: External Interrupt Flags 7 – 0.
  - ❑ When an edge or logic change on the INT7:0 pin triggers an interrupt request, INTF7:0 is set to one. If the I-bit in SREG and the corresponding interrupt enable bit, INT7:0 in EIMSK, are set to one, the AVR microcontroller will jump to the interrupt vector. The flag is cleared when the interrupt routine is executed. These flags are always cleared when INT7:0 are configured as level interrupt.



Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	IINTF0	EIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

83

An Example of External Interrupts (1/6)

Write an AVR assembly program to control 8 LEDs using INT0 (External Interrupt 0) and INT1 (External Interrupt 1) as follows:

- If PB0 is pushed, the first 4 LEDs are on and the second 4 LEDs are off
- If PB1 is pushed, the first 4 LEDs are off and the second 4 LEDs are on

Board setting: connect PB0 to PD0 and PB1 to PD1 (Port D)

84

An Example of External Interrupts (2/6)

```
.include "m2560def.inc"
.def temp =r16
.equ HIGH_LEDS = 0b11110000
.equ LOW_LEDS = 0b00001111
.cseg
.org 0x0
jmp RESET       ; interrupt vector for RESET
.org INT0addr   ; INT0addr is the address of EXT_INT0
                ; (External Interrupt 0)
jmp EXT_INT0    ; interrupt vector for External Interrupt 0
.org INT1addr   ; INT1addr is the address of EXT_INT1
                ; (External Interrupt 1)
jmp EXT_INT1    ; interrupt vector for External Interrupt 1
```

85

An Example of External Interrupts (3/6)

```
RESET:
ldi temp, low(RAMEND)  ; initialize stack pointer to point the high
                       ; end of SRAM
out SPL, temp
ldi temp, high(RAMEND)
out SPH, temp
ser temp               ; temp=0b11111111
out DDRC, temp    ; Port C is set to all outputs
clr temp               ; temp=0b00000000
out PORTC, temp
out DDRD, temp    ; Port D is set to all inputs
out PORTD, temp
```

86

An Example of External Interrupts (4/6)

```
ldi temp, (2 << ISC10) | (2 << ISC00)
; The built-in constants ISC10=2 and ISC00=0 are their bit numbers in
; EICRA register
sts EICRA, temp   ; temp=0b00001010, so both interrupts are
                  ; configured as falling edge triggered interrupts
in temp, EIMSK
ori temp, (1<<INT0) | (1<<INT1)  ; INT0=0 & INT1=1
out EIMSK, temp    ; Enable External Interrupts 0 and 1
sei    ; Enable the global interrupt
jmp main
```

87

An Example of External Interrupts (5/6)

```
EXT_INT0:  ; Interrupt handler for External Interrupt 0
push temp
in temp, SREG
push temp
ldi temp, HIGH_LEDS
out PORTC, temp
pop temp
out SREG, temp
pop temp
reti
```

88

An Example of External Interrupts (6/6)

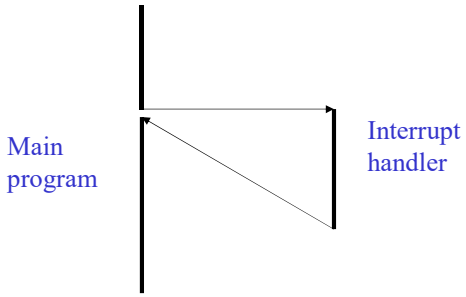
```
EXT_INT1:   ; Interrupt handler for External Interrupt 1
push temp
in temp, SREG
push temp
ldi temp, LOW_LEDS
out PORTC, temp
pop temp
out SREG, temp
pop temp
reti

main:           ; main does nothing but increments a counter
clr temp
loop: inc temp
rjmp loop       ; An infinite loop must be at the end of the interrupt
                ; handler for RESET
```

89

Non-Nested Interrupts

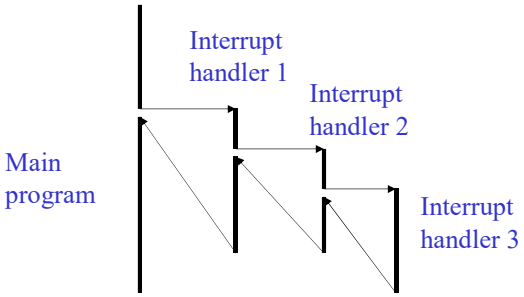
- In AVR microcontrollers, by default, an interrupt handler (service routine) cannot be interrupted by another interrupt.
  - ❑ The global interrupt flag I is cleared when an interrupt occurs, preventing an interrupt service routine from being interrupted by another interrupt.



90

Nested Interrupts

- In AVR microcontrollers, the interrupt handler of an interrupt can be interrupted by another interrupt if the interrupt handler sets the global interrupt flag I in SREG using the instruction sei.



91

Reading Material

1. Chapter 8. Microcontrollers and Microcomputers.
2. Read the following sections in ATmega2560 Data Sheet:
   a. Overview
   b. System Control and Reset.
   c. Watchdog Timer.
   d. Interrupts.
   e. External Interrupts.
   f. 8-bit Time/Counter0

92