ECTE333
Lecture 9 - Timers

School of Electrical, Computer and Telecommunications Engineering
University of Wollongong
Australia

ECTE333’s schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Lecture (2h)</th>
<th>Tutorial (1h)</th>
<th>Lab (2h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>L7: C programming for the ATMEL AVR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Tutorial 7</td>
<td>Lab 7</td>
</tr>
<tr>
<td>3</td>
<td>L8: Serial communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Tutorial 8</td>
<td>Lab 8</td>
</tr>
<tr>
<td>5</td>
<td>L9: Timers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Tutorial 9</td>
<td>Lab 9</td>
</tr>
<tr>
<td>7</td>
<td>L10: Pulse width modulator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Tutorial 10</td>
<td>Lab 10</td>
</tr>
<tr>
<td>9</td>
<td>L11: Analogue-to-digital converter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Tutorial 11</td>
<td>Lab 11</td>
</tr>
<tr>
<td>11</td>
<td>L12: Revision lecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>Lab 12</td>
</tr>
<tr>
<td>13</td>
<td>L13: Self-study guide (no lecture)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Final exam (25%), Practical exam (20%), Labs (5%)

9.1 Interrupt programming in C for ATmega16

- In Semester 1, we learnt
  - the interrupt-driven approach and the ATmega8515,
  - writing an interrupt-driven program in the assembly language.

- In this lecture, we will learn
  - the interrupt subsystem in the ATmega16,
  - writing an interrupt-driven program in C.

- Compared to polling, interrupt is a more efficient approach for the CPU to handle peripheral devices, e.g.
  - serial port, external switches, timers, PWM, and ADC.
Polling versus Interrupt

while (1){
    get_device_status;
    if (service_required){
        service_routine;
    }
    normal_execution;
}

Using polling, the CPU must continually check the device’s status.

Using interrupt:
- A device will send an interrupt signal when needed.
- In response, the CPU will perform an interrupt service routine, and then resume its normal execution.

Interrupt execution sequence

1. A device issues an interrupt
2. CPU finishes the current instruction
3. CPU acknowledges the interrupt
4. CPU saves its states and PC onto stack
5. CPU loads the address of ISR onto PC
6. CPU executes the ISR
7. CPU retrieves its states and PC from stack
8. Normal execution resumes

ATmega16 interrupt subsystem

The ATmega16 has 21 interrupts:
- 1 reset interrupt
- 3 external interrupts
- 8 timer interrupts
- 3 serial port interrupts
- 1 ADC interrupt
- 1 analogue comparator interrupt
- 1 SPI interrupt
- 1 TWI interrupt
- 2 memory interrupts

Table 9.1: Interrupts in ATmega16

<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Program Address</th>
<th>Interrupt vector name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$000</td>
<td>RESET_vect</td>
<td>Reset</td>
</tr>
<tr>
<td>2</td>
<td>$002</td>
<td>INT0_vect</td>
<td>External Interrupt Request 0</td>
</tr>
<tr>
<td>3</td>
<td>$004</td>
<td>INT1_vect</td>
<td>External Interrupt Request 1</td>
</tr>
<tr>
<td>4</td>
<td>$006</td>
<td>TIMER2_COMP_vect</td>
<td>Timer/Counter2 Compare Match</td>
</tr>
<tr>
<td>5</td>
<td>$008</td>
<td>TIMER2_OVF_vect</td>
<td>Timer/Counter2 Overflow</td>
</tr>
<tr>
<td>6</td>
<td>$00A</td>
<td>TIMER1_CAPT_vect</td>
<td>Timer/Counter1 Capture Event</td>
</tr>
<tr>
<td>7</td>
<td>$00C</td>
<td>TIMER1_COMPA_vect</td>
<td>Timer/Counter1 Compare Match A</td>
</tr>
<tr>
<td>8</td>
<td>$00E</td>
<td>TIMER1_COMPB_vect</td>
<td>Timer/Counter1 Compare Match B</td>
</tr>
<tr>
<td>9</td>
<td>$010</td>
<td>TIMER1_OVF_vect</td>
<td>Timer/Counter1 Overflow</td>
</tr>
<tr>
<td>10</td>
<td>$012</td>
<td>TIMER0_OVF_vect</td>
<td>Timer/Counter0 Overflow</td>
</tr>
<tr>
<td>11</td>
<td>$014</td>
<td>SPI_STC_vect</td>
<td>Serial Transfer Complete</td>
</tr>
<tr>
<td>12</td>
<td>$016</td>
<td>USART_RXC_vect</td>
<td>USART, Rx Complete</td>
</tr>
<tr>
<td>13</td>
<td>$018</td>
<td>USART_UDRE_vect</td>
<td>USART Data Register Empty</td>
</tr>
<tr>
<td>14</td>
<td>$01A</td>
<td>USART_TXC_vect</td>
<td>USART, Tx Complete</td>
</tr>
<tr>
<td>15</td>
<td>$01C</td>
<td>ADC_vect</td>
<td>ADC Conversion Complete</td>
</tr>
<tr>
<td>16</td>
<td>$01E</td>
<td>EE_RDY_vect</td>
<td>EEPROM Ready</td>
</tr>
<tr>
<td>17</td>
<td>$020</td>
<td>ANA_COMP_vect</td>
<td>Analog Comparator</td>
</tr>
<tr>
<td>18</td>
<td>$022</td>
<td>TWI_vect</td>
<td>2-wire Serial Interface</td>
</tr>
<tr>
<td>19</td>
<td>$024</td>
<td>INT2_vect</td>
<td>External Interrupt Request 2</td>
</tr>
<tr>
<td>20</td>
<td>$026</td>
<td>TIMER0_COMP_vect</td>
<td>Timer/Counter0 Compare Match</td>
</tr>
<tr>
<td>21</td>
<td>$028</td>
<td>SPM_RDY_vect</td>
<td>Store Program Memory Ready</td>
</tr>
</tbody>
</table>
Table 9.1: Interrupts in ATmega16

- **Vector No**
  - An interrupt with a lower 'Vector No' has a higher priority.
  - E.g., INT0 has a higher priority than INT1 and INT2.

- **Program Address**
  - The fixed memory location for a given interrupt handler.
  - E.g., in response to interrupt INT0, CPU runs instruction at $002.

- **Interrupt Vector Name**
  - This is the interrupt name, to be used with C macro ISR().

Steps to program an interrupt in C

- To program an interrupt, 5 steps are required.
  1. Include header file `<avr/interrupt.h>`.
  2. Use C macro `ISR()` to define the interrupt handler and update IVT.
  3. Enable the specific interrupt.
  4. Configure details of the interrupt by setting relevant registers.
  5. Enable the interrupt subsystem globally using `sei()`.

- Later, we’ll study steps for interrupt programming in C, via 2 examples.
  9.1.1 USART RXD Complete interrupt
  9.1.2 External interrupts

Using C macro ISR()

- The C macro ISR() is used to define the handler for a given interrupt.

- Its syntax is given as
  
  ```c
  ISR(interrupt_vector_name){
      // ... code for interrupt handler here
  }
  ```

  where `interrupt_vector_name` is given in Table 9.1.

- **Example**: To process interrupt 'RXD Complete' and put the received character in Port B, we write
  
  ```c
  ISR(USART_RXC_vect){
      PORTB = UDR; // put the received character in Port B
  }
  ```

Learning ATmega16 interrupts

<table>
<thead>
<tr>
<th>Vector No.</th>
<th>Interrupt vector name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RESET_vect</td>
<td>Reset</td>
</tr>
<tr>
<td>2</td>
<td>INT0_vect</td>
<td>External Interrupt Request 0</td>
</tr>
<tr>
<td>3</td>
<td>INT1_vect</td>
<td>External Interrupt Request 1</td>
</tr>
<tr>
<td>4</td>
<td>TIMER2_COMP_vect</td>
<td>Timer/Counter2 Compare Match</td>
</tr>
<tr>
<td>5</td>
<td>TIMER2_OVF_vect</td>
<td>Timer/Counter2 Overflow</td>
</tr>
<tr>
<td>6</td>
<td>TIMER1_CAPT_vect</td>
<td>Timer/Counter1 Capture Event</td>
</tr>
<tr>
<td>7</td>
<td>TIMER1_COMPA_vect</td>
<td>Timer/Counter1 Compare Match A</td>
</tr>
<tr>
<td>8</td>
<td>TIMER1_COMPB_vect</td>
<td>Timer/Counter1 Compare Match B</td>
</tr>
<tr>
<td>9</td>
<td>TIMER1_OVF_vect</td>
<td>Timer/Counter1 Overflow</td>
</tr>
<tr>
<td>10</td>
<td>TIMER0_OVF_vect</td>
<td>Timer/Counter0 Overflow</td>
</tr>
<tr>
<td>11</td>
<td>SPI_STC_vect</td>
<td>Serial Transfer Complete</td>
</tr>
<tr>
<td>12</td>
<td>USART_RXC_vect</td>
<td>USART, Rx Complete</td>
</tr>
<tr>
<td>13</td>
<td>USART_UDRE_vect</td>
<td>USART Data Register Empty</td>
</tr>
<tr>
<td>14</td>
<td>USART_TXC_vect</td>
<td>USART, Tx Complete</td>
</tr>
<tr>
<td>15</td>
<td>ADC_vect</td>
<td>ADC Conversion Complete</td>
</tr>
<tr>
<td>16</td>
<td>EE_RDY_vect</td>
<td>EEPROM Ready</td>
</tr>
<tr>
<td>17</td>
<td>ANA_COMP_vect</td>
<td>Analog Comparator</td>
</tr>
<tr>
<td>18</td>
<td>TWI_vect</td>
<td>2-wire Serial Interface</td>
</tr>
<tr>
<td>19</td>
<td>INT2_vect</td>
<td>External Interrupt Request 2</td>
</tr>
<tr>
<td>20</td>
<td>TIMER0_COMP_vect</td>
<td>Timer/Counter0 Compare Match</td>
</tr>
<tr>
<td>21</td>
<td>SPM_RDY_vect</td>
<td>Store Program Memory Ready</td>
</tr>
</tbody>
</table>

Lecture 9.1.1

Lecture 9.2, 9.3

Lecture 10

Lecture 9.1.2

Lecture 11
9.1.1 Serial RXD interrupt

Write a C interrupt-driven program to use the serial port of ATmega16 at baud rate 1200, no parity, 1 stop bit, 8 data bits, clock speed 1MHz. Whenever a character is received, it should be sent to Port B.

- The serial port on ATmega16 can trigger an RXD interrupt whenever a character is received [Lecture 8].
- We enable this interrupt by setting a flag in a serial port register.
- We then write the interrupt handler, to be run whenever the interrupt is triggered.

Serial RXD interrupt: serial_int.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>

void USART_init(void){
    // Normal speed, disable multi-proc
    UCSRA = 0b00000000;

    // Enable Tx and Rx pins, enable RX interrupt
    UCSRB = 0b10011000;

    // Asynchronous mode, no parity, 1 stop bit, 8 data bits
    UCSRC = 0b10000110;

    // Baud rate 1200bps, assuming 1MHz clock
    UBRRL = 0x33; UBRRHI = 0x00;
}

ISR(USART_RXC_vect){ // Handler for RXD interrupt
    PORTB = UDR; // Received character is displayed on port B
}

int main(void) {
    USART_init(); // initialise USART
    sei(); // enable interrupt subsystem globally
    DDRB = 0xFF; // set port B for output
    while (1) {} // infinite loop
    return 0;
}
```

Serial RXD interrupt: Testing

- To test the serial RXD interrupt example:
  - Connect RXD pin (pin D.0) to RXD pin of RS232 Spare.
  - Connect TXD pin (pin D.1) to TXD pin of RS232 Spare.
  - Connect Port B to LED connector.
  - Compile, download program.
  - Connect RS232 Spare Connector to Serial Port of PC.
  - Configure and run HyperTerminal and use it to send characters.

- Video demo: [avr]/ecte333/serial_int.mp4
  ```
  avr = http://www.elec.uow.edu.au/avr
  ```
#include <avr/io.h>

void USART_init(void){
    // Normal speed, disable multi-proc
    UCSRA = 0b00000000;
    // Enable Tx and Rx, disable interrupts
    UCSRB = 0b00111000;
    // Asynchronous mode, no parity, 1 stop bit, 8 data bits
    UCSRC = 0b10000110;
    // Baud rate 1200bps, assuming 1MHz clock
    UBRRL = 0x33; UBRRR = 0x00;
}

int main(void) {
    USART_init(); // initialise USART
    DDRB = 0xFF; // set port B for output
    while(1) {
        // Poll until RXC flag = 1
        while ((UCSRA & (1<<RXC)) == 0x00){}
        PORTB = UDR; // received character is displayed on port B
    }
    return 0;
}

## 9.1.2 External interrupts

External interrupts on ATmega16 and ATmega8515 are similar.

- Key references on ATmega16 external interrupts: ATmega16 user manual, ‘External Interrupts’ section.

- Three external interrupts can be triggered.
  - INT0 on pin D.2,
  - INT1 on pin D.3,
  - INT2 on pin B.2.

- Key steps in using external interrupts:
  - enable the interrupt,
  - specify what types of event will trigger the interrupt.

### External interrupts: Enabling

To enable an external interrupt, set a flag in General Interrupt Control Register (GICR).

<table>
<thead>
<tr>
<th></th>
<th>INT1</th>
<th>INT0</th>
<th>INT2</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>IVSEL</th>
<th>IVSEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Example:** To enable INT1 on pin D.3, we can write

```c
    GICR = (1 << INT1);
```

Note that INT1 and GICR names are already defined in `<avr/io.h>`.

```c
#define INT1 7
```
External interrupts: Specifying events

- Specify the events that trigger an external interrupt by using
  - MCU Control Register (INT0 and INT1), or
  - MCU Control and Status Register (INT2).

- For INT2:
  - MCU Control Register and Status Register (INT2).

- For INT0 and INT1:
  - MCU Control Register

<table>
<thead>
<tr>
<th>JTD</th>
<th>ISC2</th>
<th>-</th>
<th>JTRF</th>
<th>WDRF</th>
<th>BORF</th>
<th>EXTRF</th>
<th>PORF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For INT2:

MCUCSR

- 0: falling edge triggers an interrupt INT2
- 1: rising edge triggers an interrupt INT2

External interrupts: Example

Write a C program to toggle port B whenever a switch on the STK500 board is pressed. The program should use an external interrupt.

- Let's use interrupt INT1. This interrupt is triggered on pin D.3.

- To enable interrupt INT1:
  
  ```c
  GICR = (1 << INT1);
  ```

- To specify that INT1 is triggered on any change in pin D.3:
  
  ```c
  MCUCR = (1 << ISC10);
  ```

- Then, we write interrupt handler and enable interrupt subsystem.

```c
#include <avr/io.h>
#include <avr/interrupt.h>

ISR(INT1_vect){ // handler for INT1 interrupt
    PORTB = (~PORTB); // toggle port B
}

int main(void) {
    GICR = (1 << INT1); // enable interrupt INT1
    MCUCR = (1<<ISC10); // triggered on any change to INT1 pin (D.3)
    sei(); // enable interrupt subsystem globally
    DDRB = 0x00; // set port B for output
    PORTB = 0b10101010; // initial value
    while (1) { // infinite main loop
        return 0;
    }
}
```
External interrupts: Testing ext_int.c

- To test the external interrupt example:
  - Connect INT1 pin (pin D.3) to switch SW7 on STK500 board.
  - Connect GRD pin of Port D to GRD pin of SW connector.
  - Connect Port B to LED connector.
  - Compile, download program.
  - Press switch SW7.

- Video demo: [avr/ecte333/ext_int.mp4]

Lecture 9’s sequence

9.1 Interrupt programming in C for ATmega16

9.2 Timers in ATmega16

9.3 Timer applications

9.2 Timers in ATmega16

- Many computer applications require accurate timing.

- Examples include:
  - recording the time when an event occurs,
  - calculating the time difference between events,
  - performing tasks at specific or periodic times,
  - creating accurate time delays,
  - generating waveforms of certain shape, period, or duty cycle.

- Lectures 9 and 10 focus on using timers to perform time-related tasks.

Timer terminology

- **Input Capture**:
  - Input signal is connected to a pin, called input capture, of the timer.
  - When a preset event (rising edge, falling edge, change) occurs on this pin, the current timer value is automatically stored in a register.

- **Output Compare**:
  - A timer typically has a pin, called output compare.
  - When the timer reaches a preset value, the output compare pin can be automatically changed to logic 0 or logic 1.
Overview of Timers in ATmega16

- ATmega16 has three timers: Timer 0, Timer 1 and Timer 2.
- Each timer is associated with a counter and a clock signal.
- The counter is incremented by 1 in every clock cycle of the timer.
- The clock signal of a timer can come from
  - the internal system clock or
  - an external clock source.

When the internal system clock is used, a prescaler can be applied to make the timer count at a slower rate.

**Example:**
- Consider a system clock of 1Mhz (i.e. 1μs per cycle).
- Suppose that a timer prescaler of 64 is used.
- Then, timer will increment every 64μs.

---

### Overview of Timers in ATmega16

<table>
<thead>
<tr>
<th></th>
<th>Timer 0</th>
<th>Timer 1</th>
<th>Timer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overall</strong></td>
<td>- 8-bit counter</td>
<td>- 16-bit counter</td>
<td>- 8-bit counter</td>
</tr>
<tr>
<td></td>
<td>- 10-bit prescaler</td>
<td>- 10-bit prescaler</td>
<td>- 10-bit prescaler</td>
</tr>
<tr>
<td><strong>Functions</strong></td>
<td>- PWM</td>
<td>- PWM</td>
<td>- PWM</td>
</tr>
<tr>
<td></td>
<td>- Frequency generation</td>
<td>- Frequency generation</td>
<td>- Frequency generation</td>
</tr>
<tr>
<td></td>
<td>- Event counter</td>
<td>- Event counter</td>
<td>- Event counter</td>
</tr>
<tr>
<td></td>
<td>- Output compare</td>
<td>- Output compare channels: 2</td>
<td>- Output compare</td>
</tr>
<tr>
<td></td>
<td>- Input capture</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Operation modes</strong></td>
<td>- Normal mode</td>
<td>- Normal mode</td>
<td>- Normal mode</td>
</tr>
<tr>
<td></td>
<td>- Clear timer on compare match</td>
<td>- Clear timer on compare match</td>
<td>- Clear timer on compare match</td>
</tr>
<tr>
<td></td>
<td>- Fast PWM</td>
<td>- Fast PWM</td>
<td>- Fast PWM</td>
</tr>
<tr>
<td></td>
<td>- Phase correct PWM</td>
<td>- Phase correct PWM</td>
<td>- Phase correct PWM</td>
</tr>
</tbody>
</table>

- Timer 1 has the most capability among the three timers.

---

### Study plan

- **In Lecture 9, we focus on**
  - operations of Timer 1,
  - using Timer 1 overflow interrupt,
  - using Timer 1 input capture interrupt,
  - measuring time, creating time delay,
  - measuring period/duty cycle of a signal,
  - information required for Lab 9.
- **In Lecture 10, we will learn**
  - using Timer 1 output compare interrupt,
  - generating PWM signals,
  - information required for Lab 10.
Timer 1: An overview

- 16-bit counter.
- 10-bit prescaler: 8, 64, 256, and 1024
- can trigger a timer overflow interrupt when counter reaches MAX.
- can trigger an input capture interrupt when an event occurs on the input capture pin.
  - timer value is stored automatically in a register.
  - input capture pin for Timer 1 is ICP1 (D.6).
- can trigger an output compare match interrupt when timer reaches a preset value.
  - There are two independent output compare channels A and B.

Timer 1: Block diagram

Timer 1 – Relevant pins

- (XCK/T0) PB0
- (T1) PB1
- (INT2/AIN0) PB2
- (OC0/A/AIN1) PB3
- (SS) PB4
- (MOSI) PB5
- (MISO) PB6
- (SCK) PB7
- RESET
- VCC
- GND
- XTAL2
- XTAL1
- (RXD) PD0
- (TXD) PD1
- (INT0) PD2
- (INT1) PD3
- (OC1B) PD4
- (OC1A) PD5
- (ICP1) PD6

used in this lecture

Timer 1 – Five groups of registers

1) Timer/Counter 1
- TCNT1
  - 16-bit register that stores the current value of the timer.

2) Timer/Counter 1 Control Registers
- TCCR1A and TCCR1B
  - To configure the operations of Timer 1.

3) Input Capture Register
- ICR1
  - to store timer value when an event occurs on input capture pin.

4) Interrupt registers
- TIMSK to enable timer interrupts
- TIFR to monitor status of timer interrupts.

5) Output Compare Registers
- OCR1A, OCR1B
  - To store the preset values for output compare.

will be covered in Lecture 10.
Timer 1 — Five groups of registers

We now study the important registers for Timer 1.

9.2.1 Timer/Counter 1 Control Register A (TCCR1A)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM1A1</td>
<td>COM1A0</td>
<td>COM1B1</td>
<td>COM1B0</td>
<td>FOC1A</td>
<td>FOC1B</td>
<td>WGM11</td>
<td>WGM10</td>
</tr>
</tbody>
</table>

- Specify Waveform Generation Mode used with WGM13, WGM12
- WGM13..WGM10 = 0000 for normal mode.
- 1 to force output compare on channel B:
- Output compare mode for channel B:
- Output compare mode for channel A:
- 1 to force output compare on channel A:

9.2.2 Timer/Counter 1 Control Register B (TCCR1B)

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICNC1</td>
<td>ICES1</td>
<td>-</td>
<td>WGM13</td>
<td>WGM12</td>
<td>CS12</td>
<td>CS11</td>
<td>CS10</td>
</tr>
</tbody>
</table>

- Select Waveform Generation Mode, used with WGM11, WGM10
- WGM13..WGM10 = 0000 for normal mode.
- Input capture edge select: 1 will select rising edge, 0 will select falling edge
- 1 to activate input capture noise canceller

Clock select

<table>
<thead>
<tr>
<th>CS12</th>
<th>CS11</th>
<th>CS10</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source (timer stopped)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>CLK\textsubscript{i/o}/1 (no prescaling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CLK\textsubscript{i/o}/8 (from prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>CLK\textsubscript{i/o}/64 (from prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CLK\textsubscript{i/o}/256 (from prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>CLK\textsubscript{i/o}/1024 (from prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on T1 pin. Clock on falling edge.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on T1 pin. Clock on rising edge.</td>
</tr>
</tbody>
</table>

- For ATmega16, the default internal clock is \text{clk}_{\text{i/o}} = 1\text{MHz}.
- Timer 1 can use the internal or external clock.
- If using the internal clock, we can set Timer 1 to run 8, 64, 256, or 1024 times slower than the internal clock.
9.2.3 Timer/Counter Interrupt Mask Register (TIMSK)

- OCIE2
- TOIE2
- TICIE1
- OCIE1A
- OCIE1B
- TOIE1
- OCIE0
- TOIE0

For Timer 0:
- Timer 1 Overflow Interrupt Enable
- Timer 1 Output Compare B Match Interrupt Enable: 1 to enable
- Timer 1 Output Compare A Match Interrupt Enable: 1 to enable
- Timer 1 Input Capture Interrupt Enable: 1 to enable

For Timer 2:

9.2.4 Timer/Counter Interrupt Flag Register (TIFR)

- OCF2
- TOV2
- ICF1
- OCF1A
- OCF1B
- TOV1
- OCF0
- TOV0

For Timer 0:
- Timer 1 Overflow Interrupt Flag: set to 1 when overflow
- Timer 1 Output Compare B Match Flag: set to 1 when match
- Timer 1 Output Compare A Match Flag: set to 1 when match
- Timer 1 Input Capture Flag: set to 1 when capture event occurs

For Timer 2:

- This register has flags that indicate when a timer interrupt occurs.

Lecture 9’s sequence

9.1 Interrupt programming in C for ATmega16

9.2 Timers in ATmega16

9.3 Timer applications

- In this section, we consider three applications of Timer 1.

9.3.1 Creating an accurate delay using timer overflow interrupt.

9.3.2 Measuring elapsed time between two events.

9.3.3 Measuring the period of a square signal using input capture interrupt.
9.3.1 Creating an accurate delay

Write a C program for ATmega16 to toggle PORTB every 2 seconds. It should use Timer 1 overflow interrupt to create delays of 2s each.

**Analysis (video)**
- Internal system clock: 1MHz.
- With no prescaler, Timer 1 increments every 1 μs.
- Timer 1 is 16-bit counter, so it will overflow every 216 μs.
- For a 2s delay, we need Timer 1 to overflow for 2s/216 μs = 31 times.

**Coding**
- Write code to enable & intercept Timer 1 overflow interrupt.
- Use interrupt handler to count the number of overflows.
- When the number of overflows is 31, toggle port B.

```c
#include <avr/io.h>
#include <avr/interrupt.h>

volatile int overflow_count;

ISR(TIMER1_OVF_vect){ // handler for Timer1 overflow interrupt
  overflow_count++; // increment overflow count
  if (overflow_count >= 31){ // when 2s has passed
    overflow_count = 0; // start new count
    PORTB = ~PORTB; // toggle port B
  }
}

int main(void) {
  DDRB  = 0xFF; // set port B for output
 PORTB = 0x00; // initial value of PORTB
  overflow_count = 0; // initialise overflow count
  TCCR1A = 0b00000000; // normal mode
  TCCR1B = 0b00000001; // no prescaler, internal clock
  TIMSK  = 0b00000100; // enable Timer 1 overflow interrupt
  sei(); // enable interrupt subsystem globally
  while (1){} // infinite loop
  return 0;
}
```

9.3.2 Measuring elapsed time

To measure time using Timer 1, we must keep track of both
- the number of times that Timer 1 has overflowed: n
- the current counter value: TCNT1

If we reset n and TCNT1 at the beginning of the interval, then the time elapse is (assuming no prescaler, 1MHz clock)

\[ t = n \times 65536 + TCNT1 \quad (\mu s) \]

Use Timer 1 to measure the execution time of some custom C code.

**Approach:**
- Clear Timer 1 when the code starts.
- Record Timer 1 when the code finishes.
- Also, use Timer 1 Overflow Interrupt to keep track of how many times it has overflowed.
Measuring elapsed time: measure_time.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

volatile uint32_t n;
ISR(TIMER1_OVF_vect){ // handler for Timer1 overflow interrupt
    n++; // increment overflow count
}

int main(void) {
    int i, j;
    uint32_t elapse_time; // uint32_t is unsigned 32-bit integer data type
    TCCR1A = 0b00000000; // normal mode
    TCCR1B = 0b00000001; // no prescaler, internal clock
    TIMSK  = 0b00000100; // enable Timer 1 overflow interrupt
    n = 0; // reset n
    TCNT1 = 0; // reset Timer 1
    sei(); // enable interrupt subsystem globally
    // ----- start code -------------
    for (i = 0; i < 100; i++)
        for (j = 0; j < 1000; j++)
            ;
    // ----- end code ---------------
    elapse_time = n * 65536 + (uint32_t) TCNT1;
    cli(); // disable interrupt subsystem globally
    return 0;
}
```

9.3.3 Measuring period of a square signal

Use Timer 1 input capture interrupt to measure the period of a square wave.

- **Analysis:**
  - The period of a square wave = the time difference between two consecutive rising edges.
  - Connect the square wave to input capture pin of Timer 1.
  - Configure input capture module to trigger on a rising edge.

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

uint16_t period; // data type: unsigned 16-bit integer
ISR(TIMER1_CAPT_vect){ // handler for Timer1 input capture interrupt
    period = ICR1; // period = value of Timer 1 stored in ICR1
    TCNT1 = 0; // reset Timer 1
    PORTB = ~(period >> 8); // display top 8-bit of period on PORT B
}

int main(void) {
    DDRA = 0xFF; // set port A for output
    TCCR1A = 0b00000000; // normal mode
    TCCR1B = 0b11000001; // no prescaler, rising edge, noise canceller
    TIMSK = 0b00010000; // enable Timer 1 input capture interrupt
    sei(); // enable interrupt subsystem globally
    while (1){} // infinite loop
    return 0;
}
```

Assumption: the input signal has a high frequency, hence timer overflow can be ignored.

**Implementation:**

- Select timer operations: normal, no prescaler, internal clock 1MHz, noise canceller enabled, input capture for rising edges.

```
TCCR1A = 0b00000000;
TCCR1B = 0b11000001;
```

- Enable input capture interrupt:

```
TIMSK = 0b00010000;
```
To test the code for measuring period:

- Connect Input Capture pin (D.6) to square wave generator on WishMaker.
- Connect GRD pin of Port D to GRD pin of WishMaker.
- Connect Port B to LED connector.
- Compile, download program.
- Change frequency of square wave and observe output on LEDs.

**Video demo:** [avr/ecte333/measure_period.mp4](avr/ecte333/measure_period.mp4)

---

This example assumes no timer overflow between two rising edges of the square signal.

In Lab 9, you are required to extend the code to measure the period for low-frequency signals.

It is necessary to intercept timer overflow (see Examples 9.3.1 & 9.3.2).

For testing, the measured period should be sent to PC via serial port.

---

**Lecture 9’s summary**

- **What we learnt in this lecture:**
  - How to write an interrupt-driven program in C for ATmega16.
  - Programming serial and external interrupts in C.
  - Overview of timers in ATmega16.
  - Using Timer1 overflow and input capture interrupts in 3 applications.

- **What are the next activities?**
  - Lab 9: ‘Timers’
    - Complete the online Pre-lab Quiz for Lab 9.
    - Write programs for Tasks 1 and 2 of Lab 9.
    - See video demos of Lab 9: [avr/ecte333/lab09_task1.mp4](avr/ecte333/lab09_task1.mp4) and [avr/ecte333/lab09_task2.mp4](avr/ecte333/lab09_task2.mp4)

---

**Lecture 9’s references**

- J. Pardue, C Programming for Microcontrollers, 2005, SmileyMicros, [Chapter 7: Interrupts...].
- Atmel Corp., 8-bit AVR microcontroller with 16K Bytes In-System Programmable Flash ATmega16/ATmega16L, 2007, [Interrupts], [External Interrupts] and [Timers].


H. Huang, “MC68HC12 an introduction: software and hardware interfacing,” Thomson Delmar Learning, 2003, [Chapter 8].