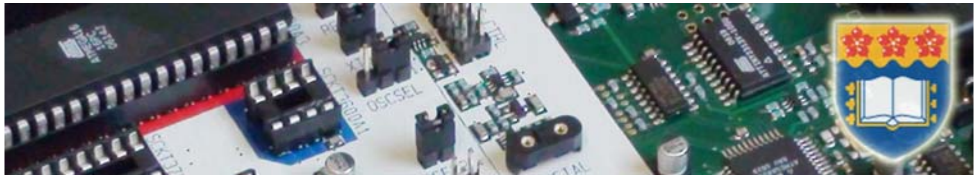


ECTE333
Lecture 9 - Timers

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

ECTE333 Spring 2011 — Schedule

Week	Lecture (2h)	Tutorial (1h)	Lab (2h)
1	L7: C programming for the ATMEL AVR		
2		Tutorial 7	Lab 7
3	L8: Serial communications		
4		Tutorial 8	Lab 8
5	L9: Timers		
6		Tutorial 9	Lab 9
7	L10: Pulse width modulator		
8		Tutorial 10	Lab 10
9	L11: Analogue-to-digital converter		
10		Tutorial 11	Lab 11
11	L12: Case studies		
12			Lab 12
13	L13: Revision lecture		
	Final exam (25%), Practical exam (20%), Labs (5%)		

Lecture 9's sequence

9.1 Interrupt programming in C for ATmega16

9.2 Timers in ATmega16

9.3 Timer applications

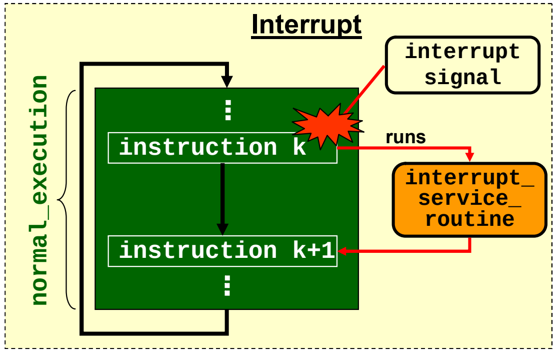
9.1 Interrupt programming in C for ATmega16

- 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

Polling

```
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

our focus

  - 1 analogue comparator interrupt
  - 1 SPI interrupt
  - 1 TWI interrupt
  - 2 memory interrupts

ATmega16 interrupt subsystem: Complete list

Table 9.1: Interrupts in ATmega16.

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

ATmega16 interrupt subsystem: Complete list

For Table 9.1,

- Vector No
  - An interrupt with a lower 'Vector No' will have a higher priority.
  - E.g., INT0 has a higher priority then 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 in C, five steps are required.
  1. Include header file <avr\interrupt.h>.
  2. Use C macro ISR() to declare the interrupt handler and update IVT.
  3. Enable the specific interrupt.
  4. Configure details about 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 declare the handler for a given interrupt.
- It basic syntax is given as

```
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

```
ISR(USART_RXC_vect){
  PORTB = UDR;  // put the received character in Port B
}
```


Learning ATmega16 interrupts

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

Lecture 9.1

Lecture 9.2, 9.3

Lecture 10

Lecture 9.1

Lecture 11

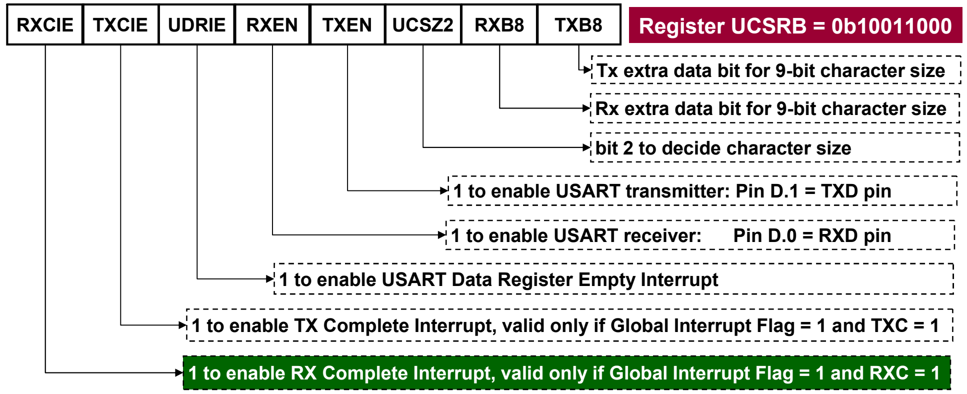
Optional

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 need to write the interrupt handler, to be run whenever the interrupt is triggered.

Serial RXD interrupt: Enabling



- For any interrupt, the ATmega16 manual can be searched to learn how to enable the interrupt.
- E.g., for serial RXD interrupt, we look at 'USART' section.

Serial RXD interrupt: serial_int.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; UBRRH = 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 link: [avr]/ecte333/serial_int.mp4

  avr = http://www.elec.uow.edu.au/avr

Serial RXD ─ Polling approach

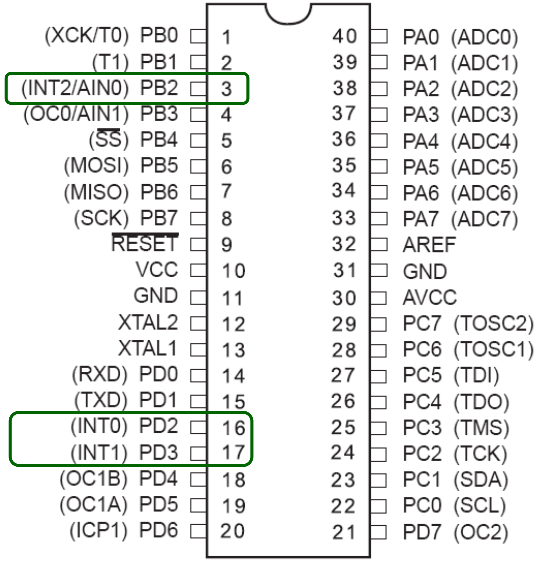
- For comparison, the program below uses polling for the same effect.

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

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

   // Enable Tx and Rx, disable interrupts
   UCSRB = 0b00011000;

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

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

int main(void) {
   USART_init(); // initialise USART
   DDRB = 0xFF;  // set port B for output
   while (1) {   // infinite loop
        // 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 ─ Relevant pins



ATmega16 chip

External interrupts: Enabling

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

	INT1	INT0	INT2	-	-	-	IVSEL	IVSEL
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W
Initial value	0	0	0	0	0	0	0	0

- 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>.

External interrupts: Specifying events

- To specify the type of events that triggers an external interrupt, set **MCU Control Register** or **MCU Control and Status Register**.

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	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

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	
	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR

ISC2:
0: falling edge generates an interrupt INT2
1: rising edge generates an interrupt INT2

External interrupts: Example

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

- Let us use interrupt INT1. This interrupt is triggered on pin D.3.
- To enable interrupt INT1

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

- To specify that INT1 is triggered on any change in pin D.3

```
MCUCR = (1<<ISC10);
```

- We then write interrupt handler and enable interrupt subsystem globally as usual.

External interrupts: ext_int.c

```
#include <avr/io.h>
#include <avr/interrupt.h>                                   1

ISR(INT1_vect){          // handler for INT1 interrupt
   PORTB = (~PORTB);     // toggle port B                    2
}

int main(void) {
   GICR = (1<< INT1);  // enable interrupt INT1              3
   MCUCR = (1<<ISC10); // triggered on any change to INT1 pin (D.3)  4
   sei();              // enable interrupt subsystem globally  5
   DDRB = 0xFF;        // set port B for output
   PORTB = 0b10101010; // initial value
   while (1) {;}       // infinite 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 link: [avr]/ecte333/ext_int.mp4

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 time instants,
  - creating accurate time delays,
  - generating waveforms of certain shape, period or duty cycle.
- Lecture 9 and Lecture 10 focus on the use of timers to perform the above time-related processing.

Timer terminology

- Input Capture:
  - An input signal is connected to a pin, designated as input capture pin, of the timer.
  - When a preset event (rising edge, falling edge, change) occurs on the input capture pin, the current value of the timer is stored in a register.
- Output Compare:
  - A timer usually has a pin designated as output compare pin.
  - 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 period of the timer's clock signal.
- The clock signal of a timer can come from
  - the internal system clock or
  - an external clock source.

Overview of Timers in ATmega16

- When the internal system clock is used, a prescaler can be used make the timer count at a slower rate.
- Example:
  - Suppose the system clock rate = 1Mhz (1μs per cycle).
  - Suppose a timer prescaler of 64 is used.
  - Then, timer will increment every 64μs.

Overview of Timers in ATmega16

	Timer 0	Timer 1	Timer 2
Overall	- 8-bit counter - 10-bit prescaler	- 16-bit counter - 10-bit prescaler	- 8-bit counter - 10-bit prescaler
Functions	- PWM - Frequency generation - Event counter - Output compare	- PWM - Frequency generation - Event counter - Output compare: 2 channels - Input capture	- PWM - Frequency generation - Event counter - Output compare
Operation modes	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	- Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM

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

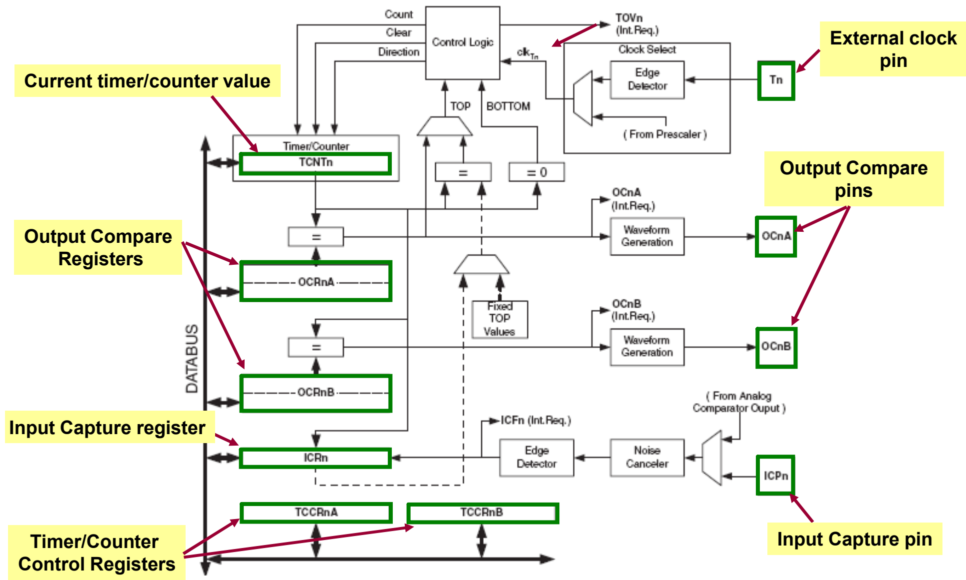
Study plan

- 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

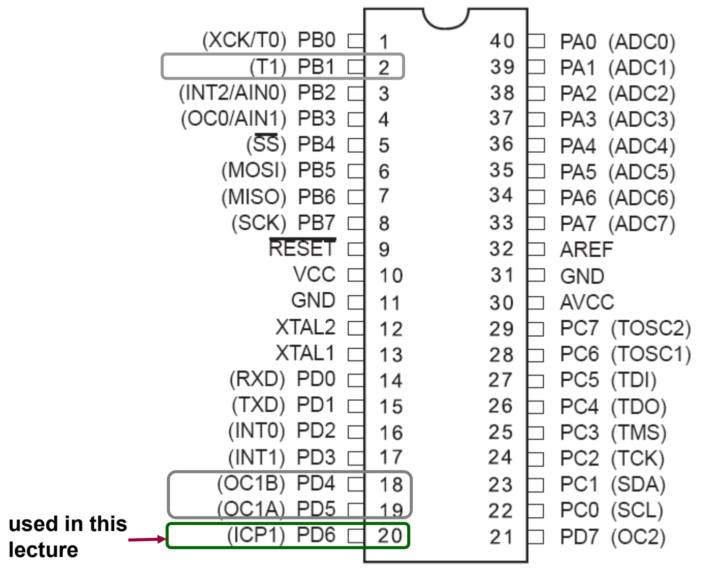
- 16-bit counter.
- 10-bit prescaler: 8, 64, 256, 1024
- can trigger a timer overflow interrupt when counter reaches MAX.
- can trigger an input capture interrupt when an event occurs on the input capture pin.
  - timer value is stored automatically in a register.
  - input capture pin for Timer 1 is 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



Not shown here: TIMSK and TIFR registers

Timer 1 — Relevant pins



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.

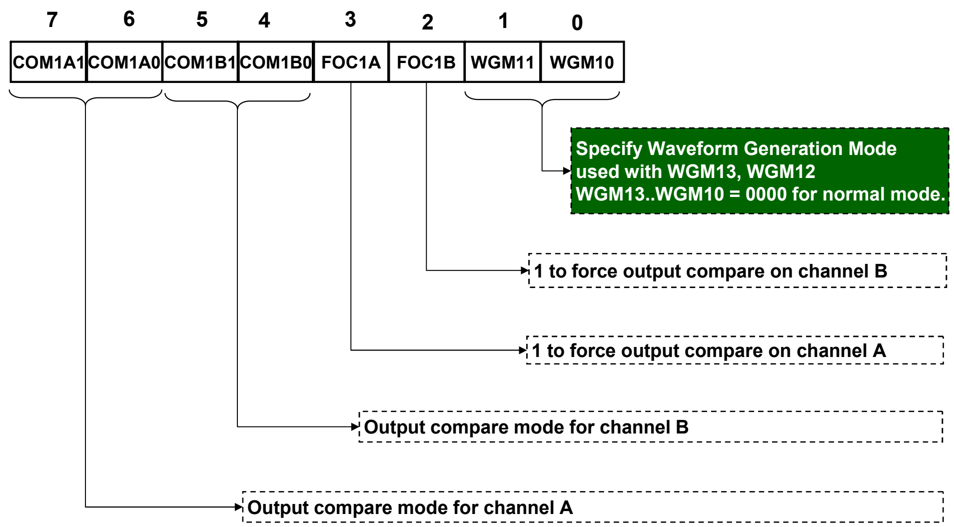
5) Output Compare Registers
- OCR1A, OCR1B
- To store the preset values for output compare.

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)



9.2.2 Timer/Counter 1 Control Register B (TCCR1B)

7	6	5	4	3	2	1	0
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

clock select (next slide)

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

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (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 T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

- For ATmega16, the internal clock is set by default at $clk_{I/O}$ = 1MHz.
- Timer 1 can run using the internal or external clock.
- If using the internal clock, we can set Timer 1 to run at a speed that is 8, 64, 256 or 1024 times slower than the internal clock.

9.2.3 Timer/Counter Interrupt Mask Register (TIMSK)

7	6	5	4	3	2	1	0
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)

7	6	5	4	3	2	1	0
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.

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**
  - Internal system clock: 1MHz.
  - With no prescaler, Timer 1 will increment every 1 µs.
  - Because Timer 1 is 16-bit counter, it will overflow every 2^{16} µs.
  - To have a 2s delay, we need Timer 1 to overflow for $2s/2^{16}$ µs = 31 times.
- **Coding**
  - Write code to enable & intercept Timer 1 overflow interrupt.
  - Use interrupt handler to count the number of overflows.
  - When number of overflows = 31, toggle port B.

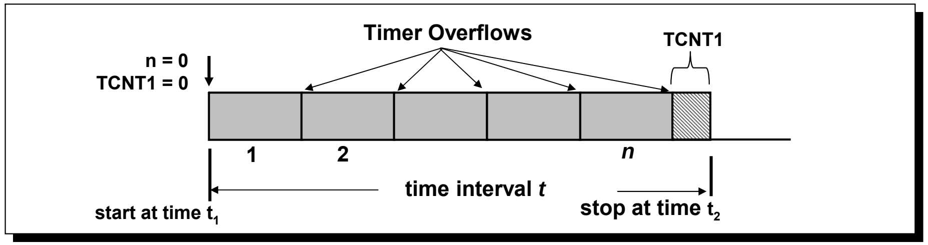
Creating an accurate delay: timer_delay.c

```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 a time interval 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 x 65536 + TCNT1 (µs)



Measuring elapsed time

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.

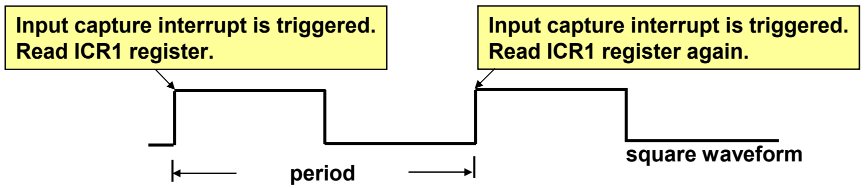
Measuring elapsed time: measure_time.c

```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;

   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 signal.

- **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.



Measuring period of a square signal

- **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.

```c
TCCR1A = 0b00000000;
TCCR1B = 0b00000001;
```

  - **Enable input capture interrupt**:

```c
TIMSK = 0b00100000;
```

measure_period.c

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

uint16_t period;

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) {
   DDRB  = 0xFF;         // set port B for output

   TCCR1A = 0b00000000; // normal mode
   TCCR1B = 0b11000001; // no prescaler, rising edge, noise canceller
   TIMSK  = 0b00100000; // enable Timer 1 input capture interrupt
   sei();               // enable interrupt subsystem globally
   while (1){;}         // infinite loop
   return 0;
}
```


Testing measure_period.c

- 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 ware and observe output on LEDs.
- Video demo link: [avr]/ecte333/measure_period.mp4

Extending measure_period.c

- The code given here assumes that there is no timer overflow between two rising edges of the square signal.
- In Lab 9, you are required to extend the code so that a correct period is measured for low-frequency signals.
- It is necessary to intercept timer overflow (as in Examples 9.3.1 and 9.3.2).
- The measure period should also be sent through the serial port to PC for testing.

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 next activities?
  - Tutorial 9: 'Timers'.
  - 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_task2.mp4

Lecture 9 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]. Manual
- S. F. Barrett and D. J. Pack, Atmel AVR Microcontroller Primer: Programming and Interfacing, 2008, Morgan & Claypool Publishers, [Chapter 5: Timing Subsystem].

© University of Wollongong, 2011.

Lecture 9 references

- M. Mazidi, J. Mazidi, R. McKinlay, "The 8051 microcontroller and embedded systems using assembly and C," 2nd ed., Pearson Prentice Hall, 2006, [Chapters 9].
- M. Mazidi and J. Mazidi, "The 8086 IBM PC and compatible computers," 4th ed., Pearson Prentice Hall, 2003, [Chapters 13].
- P. Spasov, "Microcontroller technology the 68HC11," 3rd ed., Prentice Hall, 1999, [Chapters 11].
- H. Huang, "MC68HC12 an introduction: software and hardware interfacing," Thomson Delmar Learning, 2003, [Chapter 8].

© University of Wollongong, 2011.