CprE 288 – Introduction to Embedded Systems
Exam 2 Review

Instructor:
Dr. Phillip Jones
Announcements
EXAM
Exam Topics

Programming ATMeg128 I/O modules and functions

- USART
- ADC
- Input capture (Timer/Counter)
  - In the Normal mode
- Output compare (Timer/Counter)
  - In the PWM, Normal, and CTC modes

On each subject, be familiar with

- Application background, working principles, and related concepts
- Programming interface
- Writing C functions for common purposes
- Typical application scenarios
Exam Questions

Some common question styles

• Short questions
  – Conceptual
  – Analysis
  – Calculation

• Programming: for a given application
  – Initialize an I/O module
  – Access I/O data
  – Interrupt programming

And others
Exam Questions: Data Sheet, Read it & ask questions

• Flavors of some potential Exam 2 questions
  – Program configuration registers to meet given requirements
    • UART, ADC, Input Capture, Output Compare, Timers, Interrupts
    • There is a section for each device mentioned above in the data sheet
  – Based on a given configuration, answer questions about how a program will behave
    • E.g. How long will something take to occur?
    • E.g. How many times a second will something occur?
  – Explain why a given configuration is incorrect for implementing a specified behavior
  – Assuming a given configuration, write a short program to implement a specific behavior
  – ADC calculation problems

http://class.ece.iastate.edu/cpre288
How to prepare

• Review the lecture slides
  – Read datasheet when needed

• Review/redo homework
  – Proficiency and efficiency are importance

• Ask questions
  – Office hour visits
  – Appointments
  – Emails
USART
USART: Serial Communication

- **USART = Universal Synchronous & Asynchronous Serial Receiver & Transmitter**
  - We only studied the Asynchronous part (UART)

- **Serial communication:** Data is transmitted bit by bit at the physical layer of network
  - Can transmit over long link distances
  - Uses *start* and *stop* to sandwich data bits
  - *parity bit* can be used for error detection

http://class.ece.iastate.edu/cpre288
Important concepts

- **Baud rate**: Number of symbols transmitted per second from the transmitter to the receiver
  - It’s also the rate of symbol changes to the transmission media
- **Frame format**: The format of a single data packet
  - USART transmits one data packet per request
  - One data packet contains a single data character, plus start bit, stop bit(s), and optional parity bit
Frame Format

Start bit: logic low, 1 bit
Data bits: 5, 6, 7, 8, or 9 bits
Parity bit: Optional 1 bit, Odd, Even or none
Stop bit: logic high, 1 bit or 2 bits

*Both sides of communication should use the same frame format and baud rate*
Baud Rate

• Baud rate: 1 $\text{baud} = 1 \text{ symbol per second}$
  – In USART hardware, the clock rate used to TX/RX data bits
• Baud rate is not the actual data rate
  – Baud rate includes overhead of start/stop/parity bits
• Baud rate = $f / (16 \times (\text{UBRR}n + 1))$, w/o double speed
  $f / (8 \times (\text{UBRR}n + 1))$, with double speed
  – $f$ is the system clock (default 16MHz)
  – $\text{UBRR}n + 1$ is a factor of clock prescalar
  – 8/16 is another factor of clock prescalar
• $\text{UBRR}n = f / (16 \times \text{Baud\_rate}) - 1$, w/o double speed
  $f / (8 \times \text{Baud\_rate}) - 1$, with double speed
## Example UBRR Settings

<table>
<thead>
<tr>
<th>Baud Rate (bps)</th>
<th>( f_{\text{osc}} = 16.0000 \text{ MHz} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \text{U2X = 0} )</td>
</tr>
<tr>
<td></td>
<td>UBRR</td>
</tr>
<tr>
<td>2400</td>
<td>416</td>
</tr>
<tr>
<td>4800</td>
<td>207</td>
</tr>
<tr>
<td>9600</td>
<td>103</td>
</tr>
<tr>
<td>14.4k</td>
<td>68</td>
</tr>
<tr>
<td>19.2k</td>
<td>51</td>
</tr>
<tr>
<td>28.8k</td>
<td>34</td>
</tr>
<tr>
<td>38.4k</td>
<td>25</td>
</tr>
<tr>
<td>57.6k</td>
<td>16</td>
</tr>
</tbody>
</table>

U2X for double speed
USART Programming Interface

**UCSRnA, UCSRnB, UCSRnC**: Control and Status Registers
- Three 8-bit registers for control and status checking
- \( n \) is either 0 or 1, e.g. **UCSR0A** is for USART0
- There are two USART units, USART0 and USART1; USART1 used for communication with iRobot Create

**UBRRnH and UBRRnL**: Baud Rate Registers
- Two 8-bit registers used together as 16-bit register

**UDRn**: 8-bit Register for reading and writing data
**UCSRnA: Mostly a Status Register**

- **Bit 7 – RXCn:** USART Receive Complete
  This flag bit is set when there are unread data in the receive buffer and cleared when the receive buffer is empty
- **Bit 6 – TXCn:** USART Transmit Complete
  This flag bit is set when the entire frame in the Transmit Shift Register has been shifted out
- **Bit 5 – UDREn:** USART Transmit Data Register Empty
- **Bit 4 – FEEn:** Frame Error
- **Bit 3 – DORn:** Data OverRun
- **Bit 2 – UPEn:** Parity Error
- **Bit 1 – U2Xn:** Double the USART Transmission Speed
- **Bit 0 – MPCMn:** Multi-Processor Communication Mode
**UCSRnB**: Mostly a Control Register

- **RXCIE, TXCIE, UDRIE**: Receive, Transmit, UDR interrupt enable
- **RXEN, TXEN**: Receive, Transmit enable
- **UCSZn2**: To decide number of “data” bits in the frame (see also UCSZn1 and UCSZn0)
- **RXB8n and TXB8n**: Used in 9 data bit frame setting. Not to be used in this course
**UCSRnC**: A Control Register

<table>
<thead>
<tr>
<th>-</th>
<th>UMSEL</th>
<th>UPM1</th>
<th>UPM0</th>
<th>USB</th>
<th>UCSZ1</th>
<th>UCSZ0</th>
<th>UCPOL</th>
</tr>
</thead>
</table>

- **UMSEL**: Asynchronous or Synchronous
  - 0 for Async, 1 for Sync; always use 0 only in this course
- **UPM1-0**: Parity mode – 00 (disabled), 10 (even), 11 (odd)
- **USB**: stop bit select: 0 (1 stop bit), 1 (2 stop bits)
- **UCSZ2-0**: Number of **data** bits in a frame
  - 000 (5-bits), 001 (6), 010 (7), 011 (8), 111 (9)
- **UCPOL**: Invert the polarity (invert logic low and high)
  - Use 0 in this course
// Template of initialization
void serial_init(void)
{
    // Set baud rate to ?
    UBRR0H = ________________;
    UBRR0L = ________________;

    // U2X = ?
    UCSR0A = ________________;

    // What parity mode? 1 or 2 stop bits? UCSZ1/USSZ0 = ?
    UCSR0C = ________________;

    // Enable TX/RX? Enable any intr? UCSZ2 = ?
    UCSR0B = ________________;
}
Transmitting

// Transmit a data character
void serial_putc(char data) {
    // Wait for empty transmit buffer
    while (! (UCSR0A & (1<<UDRE)))
        {}  

    // Send the data
    UDR0 = data;
}

Note: UDRE and the other bit-field names can be used in programming
// Receive a data character
char serial_getc() {
    // Wait for the receive complete
    while (! (UCSR0A & (1<<RXC)))
        ()

    // Read and clear data from the receive buffer
    return UDR0;
}
Interrupts

- Vector names for interrupts
  - USART0_RX_vect (Receive complete)
  - USART0_TX_vect (Transmit complete)
  - USART0_UDRE_vect (Data register empty)

// The serial receive interrupt
ISR (USART0_RX_vect) {
    char received_byte = UDR0;
}

Most pins have Alternative Functions: USART, ADC, input capture, output compare, and others

**USART0 uses port E**
- PE2: External Clock (PE – Port E)
- PE1: Transmit Pin
- PE0: Receive Pin

**USART1 uses port D**
- PD5: External Clock
- PD3: Transmit Pin
- PD2: Receive Pin
ADC
Sensor and ADC

Temperature Sensor

Temperature vs. Voltage (Sensor Specification)

T_{\text{min}} = 0 \degree C

T_{\text{max}} = 200 \degree C

Sensor Input (T)

Sensor_V_{\text{min}} = 0

A/D_V_{\text{min}} = 0 V

A/D_V_{\text{max}} = 3.3 V

V_{\text{max}} = 3.3 V

A/D Input (V)

A/D: Analog Input vs. Digital Output
(M = 2^n-1 steps (or bins): D_{\text{max}} = V_{\text{max}})

A/D Output (D)

D = 0

D_{\text{max}} = 1023

10-bit
Sensor and ADC

Temperature Sensor

100 C

Sensor output

1.65V

A/D input

Digital output

25

Sensor_Vmin = 0

Vmax = 3.3V

Sensor_Vmin = 0

Vmax = 3.3V

A/D: Analog Input vs. Digital Output
(M = 2^n-1 steps (or bins): Dmax = Vmax)

A/D_Vmin = 0 V

A/D_Vmax = 3.3V

A/D: Analog Input (V)

Digital Output (D)

Slope = Resolution

0.0032 V/bit

Slope = Sensitive

60.61 C/V

Slope = Resolution

0.0032 V/bit

Slope = Resolution

0.0032 V/bit

Digital Output (D)

D = 0

Dmax = 1023

T_min = 0 C

T_max = 200 C

Temperature vs. Voltage
(Sensor Specification)

60.61 C/V

0.0032 V/bit

10-bit
Sampling and Conversion

Mapping between Analog and Digital

\[ V_{\text{max}} = 7.5V \]

<table>
<thead>
<tr>
<th>Analog Input (V)</th>
<th>Digital Output (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0V</td>
<td>1111</td>
</tr>
<tr>
<td>6.5V</td>
<td>1110</td>
</tr>
<tr>
<td>6.0V</td>
<td>1101</td>
</tr>
<tr>
<td>5.5V</td>
<td>1100</td>
</tr>
<tr>
<td>5.0V</td>
<td>1011</td>
</tr>
<tr>
<td>4.5V</td>
<td>1010</td>
</tr>
<tr>
<td>4.0V</td>
<td>1001</td>
</tr>
<tr>
<td>3.5V</td>
<td>1000</td>
</tr>
<tr>
<td>3.0V</td>
<td>0111</td>
</tr>
<tr>
<td>2.5V</td>
<td>0110</td>
</tr>
<tr>
<td>2.0V</td>
<td>0101</td>
</tr>
<tr>
<td>1.5V</td>
<td>0100</td>
</tr>
<tr>
<td>1.0V</td>
<td>0011</td>
</tr>
<tr>
<td>0.5V</td>
<td>0010</td>
</tr>
<tr>
<td>0V</td>
<td>0001</td>
</tr>
</tbody>
</table>

proportionality

\[ V_{\text{max}} = 7.5V \]

0V

\[ V_{\text{min}} = 0V \]

\[ D_{\text{max}} = 15 (1111) \]

Digital sampling of an analog signal

Digital generation of an analog signal

\[ V_{\text{max}} = 7.5V \]

\[ V_{\text{min}} = 0V \]

\[ D_{\text{max}} = 15 (1111) \]

\[ D = 0 (0000) \]

\[ 2.0V \]

4-bit

\[ \text{analog to digital} \]

\[ \text{digital to analog} \]

*Embedded Systems Design: A Unified Hardware/Software Introduction*, (c) 2000 Vahid/Givargis
Formula for Conversion

Assume linear sensor, ADC is always linear

\[ \frac{a}{A_{\text{max}}} \frac{A_{\text{min}}}{A_{\text{min}}} = \frac{\nu}{V_{\text{max}}} \frac{V_{\text{min}}}{V_{\text{min}}} = \frac{d}{M} \]

– Sensor converts analog signal to electrical signal (voltage)
– ADC converts an electrical signal (voltage) to a digital number
### Constructing the ADC (Successive Approximation)

### It’s built upon a DAC

- DAC: Digital to Analog Converter
- SAR: Successive Approximation Register

**Analog Input**

- \( V_{\text{max}} = 16 \text{V} \)
- \( V_{\text{min}} = 0 \text{V} \)
- \( a = 9.5 \text{V} \)

**Let \( M = 2^n \)**

<table>
<thead>
<tr>
<th>Step</th>
<th>Range</th>
<th>Mid (digital)</th>
<th>Mid (voltage)</th>
<th>Is a &gt;= Guess (voltage)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0bxxxx</td>
<td>0b1000</td>
<td>8 Volts</td>
<td>Yes</td>
</tr>
<tr>
<td>1</td>
<td>0b1xxx</td>
<td>0b1100</td>
<td>12 Volts</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>0b10xx</td>
<td>0b1010</td>
<td>10 Volts</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>0b100x</td>
<td>0b1001</td>
<td>9 Volts</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>0b1001</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ATMega128 ADC

- 10-bit ADC conversion
- Eight input channels through a MUX
- Analog input on one of ADC0-ADC7 pins
- Up to 15K samples per second
- 0 – Vcc or 0 – 2.56V ADC input voltage range

Which pins on the ATMega128 are used for the ADC?
- Alternative I/O Functions (See next slides)
Programming Interface Registers

Three registers

**ADCSRA**: ADC control and status register A

**ADMUX**: ADC input selection

**ADCW**: ADC result register, 16-bit
**Program Interface: ADCSRA**

**ADEN**: ADC Enable
- Write 1 to this bit to enable ADC

**ADSC**: ADC Start Conversion
- Write 1 to start ADC conversion. It turns to 0 after conversion is done

**ADFR**: ADC Free Running Select
- 1: continues sampling, 0: one shot

**ADIF**: ADC Interrupt Flag
- 1: interrupt raised, 0: otherwise

**ADIE**: ADC Interrupt Enable

**ADPS2:0**: ADC Prescaler Select Bits
- 000: 2, 001: 2, 010: 4, 011: 8,
- 100: 16, 101: 32, 110: 64, 111: 128
Template Functions

• At the end, you will be able to write ADC functions like the follows:

```c
void ADC_init()
{
    // Reference voltage, alignment, channel ADMUX = _____;

    // Enable, running mode, interrupt,
    // and clock select
    ADCSRA = __________;
}
```
Coding Example

Get a reading from a given ADC channel

```c
unsigned ADC_read(char channel)
{
    ADMUX |= (channel & 0x1F); // Set the ADC multiplexer channel
    ADCSRA |= _BV(ADSC); // Start the ADC conversion
    while (ADCSRA & _BV(ADCS))
        {} // Wait for conversion to complete
    return ADCW; // Return the ADC result
}
```
INPUT CAPTURE AND OUTPUT COMPARE
Input capture and output compare work with digital waveforms

IC: Recognize waveforms by capturing the time of events

OC: Generate waveforms by setting the time of events

ATMega128 has two multi-purpose 16-bit timer/counter units: Timer/Counter 1 and Timer/Counter 3
- One input capture unit
- Three independent output compare units
Output Compare in Timer/Counter

TCNTx: Timer/Counter
ICRx: Input Capture Reg
OCRxY: Output Compare Reg
OCxY: Output Compare Pin

ATmega128 16-bit timer/counter
Registers/Values in Timer/Counter

Timer/Counter 1: TCNT1, ICR1, OCR1A, OCR1B, OCR1C
Timer/Counter 3: TCNT3, ICR3, OCR3A, OCR3B, OCR3C

• **TCNTn**: Timer/Counter register
  – Increment from 0 to TOP and then reset to 0, repeating

• **ICRn**: Input capture register
  – Capture the time of incoming events

• **OCRn**: Output compare register
  – Set the time of outgoing events
  – One OCR per channel

• **TOP**: choice of fixed values, ICRn, or OCRnA
WGM: Waveform Generation Modes

Sixteen WGMs, three categories:

• **Normal** Mode
  – Generate an event when TCNT\(_n\) = OCR\(_n\)
  – TOP is 16-bit fixed (0xFFFF)

• **CTC** Modes: Like Normal, but can adjust TOP value
  – Generate an event when TCNT\(_n\) = OCR\(_n\)
  – TOP is in ICR\(_n\) or OCR\(_nA\)

• **PWM** Modes:
  – Generate an event when TCNT\(_n\) = OCR\(_n\) or TCNT\(_n\) = TOP
  – TOP is either 8-, 9-, 10-bit fixed or in ICR\(_n\) or OCR\(_nA\)
### Table 61. Waveform Generation Mode Bit Description

<table>
<thead>
<tr>
<th>Mode</th>
<th>WGMn3</th>
<th>WGMn2 (CTCn)</th>
<th>WGMn1 (PWMn1)</th>
<th>WGMn0 (PWMn0)</th>
<th>Timer/Counter Mode of Operation&lt;sup&gt;(1)&lt;/sup&gt;</th>
<th>TOP</th>
<th>Update of OCRnx at</th>
<th>TOVn Flag Set on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Normal</td>
<td>0xFFFF</td>
<td>Immediate</td>
<td>MAX</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase Correct, 8-bit</td>
<td>0x00FF</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>PWM, Phase Correct, 9-bit</td>
<td>0x01FF</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>PWM, Phase Correct, 10-bit</td>
<td>0x03FF</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CTC</td>
<td>OCRnA</td>
<td>Immediate</td>
<td>MAX</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Fast PWM, 8-bit</td>
<td>0x00FF</td>
<td>BOTTOM</td>
<td>TOP</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Fast PWM, 9-bit</td>
<td>0x01FF</td>
<td>BOTTOM</td>
<td>TOP</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Fast PWM, 10-bit</td>
<td>0x03FF</td>
<td>BOTTOM</td>
<td>TOP</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PWM, Phase and Frequency Correct</td>
<td>ICRn</td>
<td>BOTTOM</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase and Frequency Correct</td>
<td>OCRnA</td>
<td>BOTTOM</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>PWM, Phase Correct</td>
<td>ICRn</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>PWM, Phase Correct</td>
<td>OCRnA</td>
<td>TOP</td>
<td>BOTTOM</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CTC</td>
<td>ICRn</td>
<td>Immediate</td>
<td>MAX</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(Reserved)</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Fast PWM</td>
<td>ICRn</td>
<td>BOTTOM</td>
<td>TOP</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Fast PWM</td>
<td>OCRnA</td>
<td>BOTTOM</td>
<td>TOP</td>
</tr>
</tbody>
</table>

**Note:**
1. The CTCn and PWMn1:0 bit definition names are obsolete. Use the WGMn2:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.
ATmega128 16-bit Timer/Counter

Port Connection:

• Timer/Counter 1
  – IC: Port D pin 4
  – OC Channels A, B, C: Port B pins 5, 6, 7

• Timer/Counter 3
  – IC: Port E pin 7
  – Channels A, B, C: Port E pins 3, 4, 5

Note: Those pins have to be configured as input pin for IC and output pin for OC
### Programming Interface: Output Compare

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCCR&lt;sub&gt;nA&lt;/sub&gt;</td>
<td>Control Register A</td>
</tr>
<tr>
<td>TCCR&lt;sub&gt;nB&lt;/sub&gt;</td>
<td>Control Register B</td>
</tr>
<tr>
<td>TCCR&lt;sub&gt;nC&lt;/sub&gt;</td>
<td>Control Register C</td>
</tr>
<tr>
<td>TCNT&lt;sub&gt;n&lt;/sub&gt;</td>
<td>Timer/Counter Register</td>
</tr>
<tr>
<td>ICR&lt;sub&gt;n&lt;/sub&gt;</td>
<td>Input Capture Register</td>
</tr>
<tr>
<td>OCR&lt;sub&gt;nA&lt;/sub&gt;, OCR&lt;sub&gt;nB&lt;/sub&gt;, OCR&lt;sub&gt;nC&lt;/sub&gt;</td>
<td>Output Compare Registers</td>
</tr>
<tr>
<td>TIMSK</td>
<td>Timer/Counter Interrupt Mask</td>
</tr>
<tr>
<td>ETIMSK</td>
<td>Extended Timer/Counter Interrupt Mask</td>
</tr>
<tr>
<td>TIFR</td>
<td>Timer/Counter Interrupt Flag Register</td>
</tr>
<tr>
<td>ETIFR</td>
<td>Extended Timer/Counter Interrupt Flag Reg.</td>
</tr>
</tbody>
</table>
Inside those TCCRs:

**COM 1:0** (A): Compare Output Mode

**WGM 3:0** (A, B): Waveform Generator Mode

**ICNC** (B): Input Capture Noise Canceller

**ICES** (B): Input Capture Edge Select

**CS 2:0** (B): Clock Select

**FOC 2:0** (B): Force Output Compare
**COM: Compare Output Mode**

We don’t care COM bits at this moment – set them to zero in lab 7

**WGM: Waveform Generator Mode**

To select Timer/Counter function. Four bits in total (WGM33 and WGM32 in TCCR3B)

To use Input Capture:

\[
WGM33 = 0, \ WGM32 = 0, \ WGM31 = 0, \ WGM30 = 0
\]
ICNC3: Input Capture Noise Canceller, requires four-cycle duration for an event; use it in lab 7

ICES3: Input Capture Edge Select – Which edge will trigger input capture? 0 for falling edge, 1 for rising edge

WGM32, WGM32: See previous slide
CS3x: Clock Select bits
Table in ATmega128 User Guide, page 137

<table>
<thead>
<tr>
<th>CSn2</th>
<th>CSn1</th>
<th>CSn0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source. (Timer/Counter stopped)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>\text{(\text{clk}_{\text{i/o}}/1)} (No prescaling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>\text{(\text{clk}_{\text{i/o}}/8)} (From prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>\text{(\text{clk}_{\text{i/o}}/64)} (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>\text{(\text{clk}_{\text{i/o}}/256)} (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>\text{(\text{clk}_{\text{i/o}}/1024)} (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on Tn pin. Clock on falling edge</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on Tn pin. Clock on rising edge</td>
</tr>
</tbody>
</table>
**FOC**: Force output compare on channel A, B or C

Write 0s to those bits in lab 7 or don’t write it; Force output compare is not used for Lab7
IC Programming Example

Input capture programming is usually ISR-based

// Timer/Counter 1 input capture ISR
ISR (TIMER1_CAPT_vect)
{
  event_time = ICR1;

  // Time-related processing
  ...
}

http://class.ece.iastate.edu/cpre288
In lab 8, the *suggestion* is to

- Use **WGM 1111**
- Use **OCRnA** to set the **TOP** (pulse interval – 1)
- Use **OCRnB** or **OCRnC** to store (pulse width-1) (depends on whether channel B or C is used)

Calculate `pulse_width` and `pulse_period`, then

```c
OCR3B = pulse_width - 1; // assume chan. B
OCR3A = pulse_period – 1;
```

Other fast PWM may be used
Example: Servo Programming

```c
unsigned pulse_period = ...;  // pulse period in cycles

void timer3_init()
{
    OCR3A = pulse_period-1;    // number of cycles in the interval
    OCR3B = mid_point-1;        // if you want to move servo to the middle
    TCCR3A = __________;        // set COM and WGM (bits 3 and 2)
    TCCR3B = __________;        // set WGM (bits 1 and 0) and CS

    // it’s necessary to set the OC3B (PE4) pin as the output
    DDRE |= _BV(4);            // set Port E pin 4 (OC3B) as output
}
```
void move_servo(unsigned degree)
{
    unsigned pulse_width; // pulse width in cycles

    ... // calculate pulse width in cycles

    OCR3B = pulse_width-1; // set pulse width

    // you need to call wait_ms() here to enforce a delay for the servo to
    /// move to the position
}
WGM (with OCR and TOP) decides the timing of events
COM decides the operation (action) at the OC PIN when an event occurs

Possible Actions:
– No change (maintain the current signal level)
– Set the OC pin (make the signal level high)
– Clear the PC pin (make the signal level low)
– Toggle the OC pin (invert the signal level)

Two COM tables: One for PWM, and one for Normal/CTC
// Use Normal Mode to generate a square waveform of
// 2M timer cycles, using Timer/Counter 1

unsigned count = 0;

ISR (TIMER1_COMPA_vect)
{
    count += M;
    OCR1A = count;
}
To generate a different waveform, what’s the code in the interrupt handler?

• Generate a periodic waveform repeating the following: 100-cycle low, 100 high, 200 low, 200 high, 300 low, 300 high.
/ assume appropriate configuration, and
/ the output is initially high

#define SEGS 6
unsigned count[SEGS] = {100, 100, 200, 200, 300, 300};
int pos = 0;

ISR (TIMER1_COMPA_vect)
{
    OCR1A += count[pos];
    pos = (pos+1) % SEGS;
}
CTC Mode: Square Waveform

Generate a square waveform: Initialize Timer/Counter 3’s OC unit as CTC mode 4 (OCRnA as TOP), 64 prescalar, toggle-on-match, and uses channel A only, with 200-cycle period. No interrupt is needed.

timer_init()
{
    DDRE |= _BV(OC3A);
    OCR3A = (100-1);
    //COM3A=01, WGM=0100, FOC3A=0, CS3=011, OCIE3A=0
    TCCR3A = _BV(COM3A0);
    TCCR3B = _BV(CS31)|_BV(CS30)|_BV(WGM32);
    TCCR3C = 0;
}