Counter & Timer

Klaus-Peter Zauner

ELEC1201: Programming
Sequencing and Timing...

... at the very beginning of the development that led to computers.
... very important in process control.
... very wide field of applications for microcontroller.

⇒ Significant amount of on-chip hardware, many options
⇒ This looks quite complicated on first sight
Sequencing and Timing... 

... at the very beginning of the development that led to computers. 
... very important in process control. 
... very wide field of applications for microcontroller.

⇒ Significant amount of on-chip hardware, many options
⇒ This looks quite complicated on first sight
Sample Applications for Timers/Counters

- Controlling Motor Speed (PWM)
  - With semiconductors fast ON-OFF switching is much more efficient than resistive (linear) regulation.

- Sequencing events
  - laundry machine, drum machine

- Counting pulses: incremental position encoder

- Waking CPU from sleep → sensor networks

- Timing external Events
Schedule for Week 8

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Location</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mo, 16th Nov</td>
<td>Now</td>
<td>2a/2065</td>
<td>Timer Lecture</td>
</tr>
<tr>
<td></td>
<td>14:00</td>
<td>58/1067</td>
<td>Intro to C8 &amp; Open Clinic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>before Thus</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thus, 19th Nov</td>
<td>Group</td>
<td>Zepler Labs</td>
<td>Prepare C8</td>
</tr>
<tr>
<td>Fri, 20st Nov</td>
<td>15:00</td>
<td>29/1101</td>
<td>C8 Feedback &amp; Open Clinic</td>
</tr>
</tbody>
</table>

Timers are a complex subject—you are advised to **take a look at the timer section of the data sheet today** to get a feel for how much time it will take to prepare for C8.
Resources

- ATmega644P Datasheet (pp.94–160)
- Application Notes:
  - AVR130: Setup and Use the AVR Timers
  - AVR131: Using the AVR’s High-speed PWM
- AVR Libc
  - #include <avr/io.h>
- Il Matto Quick Reference Card
Resources

- ATmega644P Datasheet (pp.94–160)
- Application Notes:
  - AVR130: Setup and Use the AVR Timers
  - AVR131: Using the AVR’s High-speed PWM

- AVR Libc
  - `#include <avr/io.h>`

- Il Matto Quick Reference Card
Resources

- ATmega644P Datasheet (pp.94–160)
- Application Notes:
  - AVR130: Setup and Use the AVR Timers
  - AVR131: Using the AVR’s High-speed PWM
- AVR Libc
  - #include <avr/io.h>
- Il Matto Quick Reference Card
Resources

- ATmega644P Datasheet (pp.94–160)
- Application Notes:
  - AVR130: Setup and Use the AVR Timers
  - AVR131: Using the AVR’s High-speed PWM
- AVR Libc
  - #include <avr/io.h>
- Il Matto Quick Reference Card
Resources

- ATmega644P Datasheet (pp.94–160)
- Application Notes:
  - AVR130: Setup and Use the AVR Timers
  - AVR131: Using the AVR’s High-speed PWM
- AVR Libc
  - `#include <avr/io.h>`
- Il Matto Quick Reference Card
Counter

++

Register
Counter

++

Register
Counter

++

---

Register

read
write
Timer = Counter + Clock
Timer = Counter + Clock
Timer = Counter + Clock

Clock

Register

++

read
write
overflow

Action

Comparator

Register
Actions initiated by AVR Timers

- Set a flag
- Cause an interrupt
- Change state of an output pin
Actions initiated by AVR Timers

- Set a flag
- Cause an interrupt
- Change state of an output pin
Actions initiated by AVR Timers

- Set a flag
- Cause an interrupt
- Change state of an output pin
Timers on the ATmega644P

All timers have/can act as:

- Phase correct PWM
- Frequency Generator
- Two Output-Compare Units
- Clear on Compare Match
- Interrupt on Overflow
- Interrupt on Compare Match
## Key Differences among the Timers

<table>
<thead>
<tr>
<th>Feature</th>
<th>Timer 0</th>
<th>Timer 1</th>
<th>Timer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range [bit]</td>
<td>8</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>External Event Counter</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Pulse Freq. Modulation</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>External Clock</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
### Key Differences among the Timers

<table>
<thead>
<tr>
<th>Feature</th>
<th>Timer 0</th>
<th>Timer 1</th>
<th>Timer 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range [bit]</td>
<td>8</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>External Event Counter</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Pulse Freq. Modulation</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>External Clock</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Timer 0 Overview

Control-, flag-, and mask-registers not shown
Setting Up

- Select a Clock Source → starts timer

With Interrupts:

- Setup Interrupt Service → interrupts.h
- Enable Timer Interrupt
- Enable Interrupts Globally

With Output Pin:

- Enable Pin as Output → Data Direction Reg.
Setting Up

▶ Select a Clock Source  →  starts timer

With Interrupts:

▶ Setup Interrupt Service  →  interrupts.h
▶ Enable Timer Interrupt
▶ Enable Interrupts Globally

With Output Pin:

▶ Enable Pin as Output  →  Data Direction Reg.
Setting Up

- Select a Clock Source — starts timer

With Interrupts:

- Setup Interrupt Service — interrupts.h
- Enable Timer Interrupt
- Enable Interrupts Globally

With Output Pin:

- Enable Pin as Output — Data Direction Reg.
Setting Up

▶ Select a Clock Source → starts timer

With Interrupts:

▶ Setup Interrupt Service → interrupts.h
▶ Enable Timer Interrupt
▶ Enable Interrupts Globally

With Output Pin:

▶ Enable Pin as Output → Data Direction Reg.
Setting Up

Select a Clock Source

With Interrupts:

- Setup Interrupt Service
- Enable Timer Interrupt
- Enable Interrupts Globally

With Output Pin:

- Enable Pin as Output
Clock Source

- System Clock
  - possibly too fast
- Scaled down System Clock
- External Event $\rightarrow T_0, T_1$
- Asynchronous Ext. Clock $\rightarrow T_2$
Clocking with the System Clock

\[ TOV_{f_{CPU}} = \frac{f_{CPU}/P_{VAL}}{2^{Range}} \]

Overflow interrupts per second at \( f_{CPU} = 12 \) MHz:

<table>
<thead>
<tr>
<th>Prescaler ((P_{VAL}))</th>
<th>8 bit Range</th>
<th>16 bit Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46875.00</td>
<td>183.11</td>
</tr>
<tr>
<td>8</td>
<td>5859.38</td>
<td>22.89</td>
</tr>
<tr>
<td>32</td>
<td>1464.84</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>732.42</td>
<td>2.86</td>
</tr>
<tr>
<td>128</td>
<td>366.21</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>183.11</td>
<td>0.72</td>
</tr>
<tr>
<td>1024</td>
<td>45.78</td>
<td>0.18</td>
</tr>
</tbody>
</table>

A prescaler of 32 and of 128 is only available for Timer 2.
Timer Modes

Defined by:

- **Compare Output Mode**
- **Wave Form Generation Mode**

Compare Output Modes:

- Timer output pin not used
- Compare match → Set pin high
- Compare match → Set pin low
- Compare match → Toggle pin
Timer Modes

Defined by:

- Compare Output Mode
- Wave Form Generation Mode

Compare Output Modes:

- Timer output pin not used
- Compare match $\rightarrow$ Set pin high
- Compare match $\rightarrow$ Set pin low
- Compare match $\rightarrow$ Toggle pin
Wave Form Generation Mode

Modes for Timer 1:

- **Normal mode**
  - Counts up and overflows

- **Clear Timer on Compare Match (CTC)**
  - Resets at specified limit

- **Fast Pulse Width Modulation Mode (PWM)**
  - Counts up, changes state at set point

- **Phase Correct PWM Mode**
  - Counts up/down, changes state at set point

- **Phase and Frequency Correct PWM Mode**
  - Updates limits at minimum point
Wave Form Generation Mode

Modes for Timer 1:

▶ Normal mode
  ▶ Counts up and overflows

▶ Clear Timer on Compare Match (CTC)
  ▶ Resets at specified limit

▶ Fast Pulse Width Modulation Mode (PWM)
  ▶ counts up, changes state at set point

▶ Phase Correct PWM Mode
  ▶ counts up/down, changes state at set point

▶ Phase and Frequency Correct PWM Mode
  ▶ Updates limits at minimum point
Wave Form Generation Mode

Modes for Timer 1:

- Normal mode
  - Counts up and overflows
- Clear Timer on Compare Match (CTC)
  - Resets at specified limit
- Fast Pulse Width Modulation Mode (PWM)
  - Counts up, changes state at set point
- Phase Correct PWM Mode
  - Counts up/down, changes state at set point
- Phase and Frequency Correct PWM Mode
  - Updates limits at minimum point
Fast PWM Mode

OCR\textsubscript{nx}/TOP Update and TOV\textsubscript{n} Interrupt Flag Set and OC\textsubscript{nA} Interrupt Flag Set or ICF\textsubscript{n} Interrupt Flag Set (Interrupt on TOP)

TCNT\textsubscript{n}

OC\textsubscript{nx}

OC\textsubscript{nx}

Period

1 2 3 4 5 6 7 8
PWM for Digital to Analogue conversion

- Square wave with variable length of pulses

\[
V_{AV} = \frac{t_H \cdot V_H + t_L \cdot V_L}{t_H + t_L}
\]

\[
= \frac{OCR_x \cdot V_H + (MaxVal - OCR_x) \cdot V_L}{MaxVal}
\]

Where \( t_H \) is the duty cycle and \( V_H \) is the voltage of the high state, \( t_L \) and \( V_L \) respectively for the low state.
Phase and Frequency Correct PWM Mode

- OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)
- OCRnx/TOP Update and TOVn Interrupt Flag Set (Interrupt on Bottom)

TCNTn

OCnx

OCnx

Period

1 2 3 4

(COMnx1:0 = 2)

(COMnx1:0 = 3)
C8
C8: Playing Music with a Synthesizer

Different time scales:

- Timing of Notes
- Tone Frequency Generation
- Fast PWM for analogue output
  - above audible frequency
C8: Structure

3.1 Produce a Square Wave
   ▶ Only one timer needed
   ▶ Simple circuit: two resistors

3.2 Play a Tonal Scale → loop over array

3.3 Play a Tune → frequencies from string
   ▶ One Timer, first simple circuit

3.4 Volume Control → PWM

4. Extra work: Stereo Panning
C8: Preparation

1. Understand Timer Allocation
2. Work out Timer Configuration
   - Ready for coding it in C
3. Note Pin Connections
### Timer Choice and Configuration

#### Tone Oscillator

If one considers the requirements for frequencies (section 1.2), the different capabilities of the timers available on the ATMEGA644P (see the datasheet [3, pp. 93–160]³), and the clock frequency of the Il Matto Board (see the Il Matto Quick Reference Card[4]), one will arrive at the conclusion that Timer 1 is most appropriate as tone oscillator.

1. Briefly explain why Timer 1 is a good choice.
C8 Prep: Allocate Timer for Tone Osc.

ATmega 644P Timers:
▶ 1 × 16 bit
▶ 2 × 8 bit

8-bit resolution is not enough to cover the audio range with a resolution that would match musical notes

```c
#include <et_scale.h>

uint16_t et_scale[] = {
  62,  /* 61.74 Hz */
  65,  /* 65.41 Hz */
  69,  /* 69.30 Hz */
  73,  /* 73.42 Hz */
... 
  3951,  /* 3951.07 Hz */
  4186,  /* 4186.01 Hz */
  4435,  /* 4434.92 Hz */
};  /* et_scale */
```

⇒ Frequency range ≈ 5 kHz
⇒ Resolution < 10 Hz
C8 Prep: Allocate Timer for Tone Osc.

ATmega 644P Timers:

- 1 × 16 bit
- 2 × 8 bit

8-bit resolution is not enough to cover the audio range with a resolution that would match musical notes.

```c
uint16_t et_scale[] = {
    62,  /* 61.74 Hz */
    65,  /* 65.41 Hz */
    69,  /* 69.30 Hz */
    73,  /* 73.42 Hz */
    ...,
    3951,  /* 3951.07 Hz */
    4186,  /* 4186.01 Hz */
    4435,  /* 4434.92 Hz */
}; /* et_scale */
```

⇒ Frequency range ≈ 5 kHz
⇒ Resolution < 10 Hz
ATmega 644P Timers:

- **1 × 16 bit**
- **2 × 8 bit**

8-bit resolution is not enough to cover the audio range with a resolution that would match musical notes.

```c
uint16_t et_scale[] = {
  62,    /* 61.74 Hz */
  65,    /* 65.41 Hz */
  69,    /* 69.30 Hz */
  73,    /* 73.42 Hz */
  ...,   /* et_scale */
  3951,  /* 3951.07 Hz */
  4186,  /* 4186.01 Hz */
  4435,  /* 4434.92 Hz */
}; /* et_scale */
```

⇒ Frequency range ≈ 5 kHz
⇒ Resolution < 10 Hz
2. Determine from the ATMEGA644P datasheet[3] how to **configure Timer 1** to work in **Phase and Frequency Correct PWM Mode** and **toggle Compare Match Output A** when the counter matches the corresponding **output compare register**. Draw in your logbook each register that needs to be set, labelled with its proper name\(^4\) and the bit-setting required to configure the timer.

- **Timer 1**
- **Phase and Freq. correct PWM**
- **Output A**
- **Toggle Output**
- **Match the Output Compare Register**
2. Determine from the ATMEGA644P datasheet[3] how to **configure Timer 1** to work in **Phase and Frequency Correct PWM Mode** and **toggle Compare Match Output A** when the counter matches the corresponding **output compare register**. Draw in your logbook each register that needs to be set, labelled with its proper name\(^4\) and the bit-setting required to configure the timer.

- **Timer 1**
- **Phase and Freq. correct PWM**
- **Output A**
- **Toggle Output**
- **Match the Output Compare Register**
2. Determine from the ATMEGA644P datasheet[3] how to **configure Timer 1** to work in **Phase and Frequency Correct PWM Mode** and toggle Compare Match Output A when the counter matches the corresponding output compare register. Draw in your logbook each register that needs to be set, labelled with its proper name⁴ and the bit-setting required to configure the timer.

---

**ATmega164PA/324PA/644PA/1284P**

15. **16-bit Timer/Counter1** with PWM

**Phase and Frequency Correct PWM Mode**

The *phase and frequency correct Pulse Width Modulation*, or phase and frequency correct PWM mode (WGMn3:0 = 8 or 9) provides a high resolution phase and frequency correct PWM waveform generation option. The phase and frequency correct PWM mode is, like the phase correct PWM mode, based on a dual-slope operation. The counter counts repeatedly from BOTTOM (0x0000) to TOP and then from TOP to BOTTOM. In non-inverting Compare Output mode, the Output Compare (OCnx) is cleared on the compare match between TCNTn and OCRnx while...
2. Determine from the ATMEGA644P datasheet[3] how to configure Timer 1 to work in Phase and Frequency Correct PWM Mode and toggle Compare Match Output A when the counter matches the corresponding output compare register. Draw in your logbook each register that needs to be set, labelled with its proper name[4] and the bit-setting required to configure the timer.

Using the ICRn Register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCRnA as TOP is clearly a better choice due to its double buffer feature.
Prep: Configure Timer 1 as Tone Osc.

Using the ICRn Register for defining TOP works well when using fixed TOP values. By using ICRn, the OCRnA Register is free to be used for generating a PWM output on OCnA. However, if the base PWM frequency is actively changed by changing the TOP value, using the OCRnA as TOP is clearly a better choice due to its double buffer feature.

TCNTn when the counter decrements. The PWM frequency for the output when using phase and frequency correct PWM can be calculated by the following equation:

\[ f_{OCnxPFCPWM} = \frac{f_{\text{clk}_I/O}}{2 \cdot N \cdot \text{TOP}} \]

The N variable represents the prescaler divider (1, 8, 64, 256, or 1024).

The extreme values for the OCRnx Register represents special cases when generating a PWM waveform output in the phase correct PWM mode. If the OCRnx is set equal to BOTTOM the output will be continuously low and if set equal to TOP the output will be set to high for non-inverted PWM mode. For inverted PWM the output will have the opposite logic values. If OCR1A is used to define the TOP value (WGM13:0 = 9) and COM1A1:0 = 1, the OC1A output will toggle with a 50% duty cycle.
15.11 Register Description

15.11.1 TCCR1A – Timer/Counter1 Control Register A

<table>
<thead>
<tr>
<th>Bit (0x80)</th>
<th>COM1A1</th>
<th>COM1A0</th>
<th>COM1B1</th>
<th>COM1B0</th>
<th>–</th>
<th>–</th>
<th>WGM11</th>
<th>WGM10</th>
<th>TCCR1A</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>6</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>5</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>
<td>0</td>
</tr>
<tr>
<td>4</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>
<td>0</td>
</tr>
<tr>
<td>3</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>
<td>0</td>
</tr>
<tr>
<td>2</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>
<td>0</td>
</tr>
<tr>
<td>1</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>
<td>0</td>
</tr>
<tr>
<td>0</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>
<td>0</td>
</tr>
</tbody>
</table>

Table 15-4. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

<table>
<thead>
<tr>
<th>COMnA1/COMnB1</th>
<th>COMnA0/COMnB0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OCnA/OCnB disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WGMn3:0 = 9 or 11: <strong>Toggle OCnA on Compare Match</strong>, OCnB disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OCnA/OCnB on Compare Match when upcounting. Set OCnA/OCnB on Compare Match when downcounting.</td>
</tr>
</tbody>
</table>
C8-Prep: Timer Control Registers

15.11 Register Description

15.11.1 TCCR1A – Timer/Counter1 Control Register A

<table>
<thead>
<tr>
<th>Bit (0x80)</th>
<th>COM1A1</th>
<th>COM1A0</th>
<th>COM1B1</th>
<th>COM1B0</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R/W</th>
<th>R</th>
<th>R</th>
<th>R/W</th>
<th>R/W</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td>2</td>
<td>WGM11</td>
<td>WGM10</td>
<td></td>
<td></td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 15-4. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

<table>
<thead>
<tr>
<th>COMnA1/COMnB1</th>
<th>COMnA0/COMnB0</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Normal port operation, OCnA/OCnB disconnected.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>WGMn3:0 = 9 or 11: <strong>Toggle OCnA on Compare Match</strong>, OCnB disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Clear OCnA/OCnB on Compare Match when upcounting. Set OCnA/OCnB on Compare Match when downcounting.</td>
</tr>
</tbody>
</table>
C8-Prep: Timer Control Register A

Table 15-5. 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</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>0xFFFE</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>
</tbody>
</table>

Bit (0x80)
Read/Write
Initial Value

7            6            5            4            3            2            1            0
COM1A1       COM1A0       COM1B1       COM1B0       -            -            WGM11       WGM10
R/W          R/W          R/W          R/W          R            R            R/W          R/W
0            0            0            0            0            0            0            0

7            0            1            1            1            1            Fast PWM, 10-bit
8            1            0            0            0            0            PWM, Phase and Frequency Correct
9            0            0            0            0            0            PWM, Phase and Frequency Correct
10           0            0            0            0            0            PWM, Phase Correct
11           0            0            0            0            0            PWM, Phase Correct
C8-3.1: Tone Oscillator

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

/*
 | Signal   | Port | Pin |
 |----------+------|-----|
 |----------+------|-----|
 | Tone     | D    | 5   |
 * /

#define TONE_FREQ 262
#define TONE_PRESCALER 8UL

void init_tone(void);
void tone(uint16_t frequency);

int main(void) {
  init_tone();
  for(;;) tone(TONE_FREQ);
}
```

General structure:

1. Load libraries
2. Comment on pin usage
3. Define Constants
   - no “magic numbers”
   - note: unsigned long
4. Declare functions
5. Entry of Program
6. Initialisation
   - called once only
7. Do something
8. Do not terminate
#include <avr/io.h>

/*
 | Signal    | Port | Pin |
 |-----------+------+-----|
 | Tone Osc  | D    | 5    |
 */

#define TONE_FREQ 262
#define TONE_PRESCALER 8UL

void init_tone(void);
void tone(uint16_t frequency);

int main(void) {
    init_tone();
    for(;;) tone(TONE_FREQ);
}

...
C8-3.1: Tone Oscillator

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

/*
<table>
<thead>
<tr>
<th>Signal</th>
<th>Port</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tone Osc</td>
<td>D</td>
<td>5</td>
</tr>
</tbody>
</table>
*/
#define TONE_FREQ 262
#define TONE_PRESCALER 8UL

void init_tone(void);
void tone(uint16_t frequency);

int main(void) {
    init_tone();
    for(;;) tone(TONE_FREQ);
}
```

General structure:
1. Load libraries
2. Comment on pin usage
3. Define Constants
   - no “magic numbers”
   - note: unsigned long
4. Declare functions
5. Entry of Program
6. Initialisation
   - called once only
7. Do something
8. Do not terminate
```c
#include <avr/io.h>

/*
   | Signal   | Port | Pin |
   |----------+------+-----|
   | Tone Osc | D    | 5    |
  */

#define TONE_FREQ 262
#define TONE_PRESCALER 8UL

void init_tone(void);
void tone(uint16_t frequency);

int main(void) {
  init_tone();
  for(;;) tone(TONE_FREQ);
}
```

General structure:
1. Load libraries
2. Comment on pin usage
3. Define Constants
   - no "magic numbers"
   - note: unsigned long
4. Declare functions
5. Entry of Program
6. Initialisation
   - called once only
7. Do something
8. Do not terminate
C8-3.1: Tone Oscillator

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

/*
 | Signal   | Port | Pin |
 |----------+------+-----|
 | Tone Osc | D    | 5    |
*/
#define TONE_FREQ 262
#define TONE_PRESCALER 8UL

void init_tone(void);
void tone(uint16_t frequency);

int main(void) {
  init_tone();
  for(;;) tone(TONE_FREQ);
}
```

General structure:
1. Load libraries
2. Comment on pin usage
3. Define Constants
   ▶ no “magic numbers”
   ▶ note: unsigned long
4. Declare functions
5. Entry of Program
6. Initialisation
   ▶ called once only
7. Do something
8. Do not terminate
C8-3.1: Tone Oscillator

1. Load libraries
2. Comment on pin usage
3. Define Constants
   - no “magic numbers”
   - note: unsigned long
4. Declare functions
5. Entry of Program
6. Initialisation
   - called once only
7. Do something
8. Do not terminate
C8-3.1: Tone Oscillator

#include <avr/io.h>

/*
 | Signal   | Port | Pin |
 |----------+------|-----|
 | Tone Osc | D    | 5   |
 */
#define TONE_FREQ 262
#define TONE_PRESCALER 8UL

void init_tone(void);
void tone(uint16_t frequency);

int main(void) {
    init_tone();
    for(;;) tone(TONE_FREQ);
}

General structure:

1. Load libraries
2. Comment on pin usage
3. Define Constants
   ▶ no “magic numbers”
   ▶ note: unsigned long
4. Declare functions
5. Entry of Program
6. Initialisation
   ▶ called once only
7. Do something
8. Do not terminate
#include <avr/io.h>

/*
<table>
<thead>
<tr>
<th>Signal</th>
<th>Port</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tone Osc</td>
<td>D</td>
<td>5</td>
</tr>
</tbody>
</table>
*/

#define TONE_FREQ 262
#define TONE_PRESCALER 8UL

void init_tone(void);
void tone(uint16_t frequency);

int main(void) {
  init_tone();
  for(;;) tone(TONE_FREQ);
}

...
C8-3.1: init_tone()

Timer Initialisation:
1. Enable output driver
2. Setup Timer Register A
3. Setup Timer Register B

To do this we need to know
▶ which timer we want to use
▶ in which waveform mode the timer should operate
▶ how much we want to scale the clock down
▶ which comparator we want to use (A or B)
▶ which comparator output mode we want to use
C8-3.1: init_tone()

Timer Initialisation:
1. Enable output driver
2. Setup Timer Register A
3. Setup Timer Register B

To do this we need to know
- which timer we want to use
- in which wave form mode the timer should operate
- how much we want to scale the clock down
- which comparator we want use (A or B)
- which comparator output mode we want to use
void init_tone(void) {
    DDRD |= _BV(PD5); /* enable output driver for OC1A */
    TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
        | _BV(WGM10); /* frequency (f) correct PWM, */
    TCCR1B = _BV(WGM13) /* varying f with OCR1A */
        | _BV(CS11); /* prescaler set to 8 */
}

- Port/Pin depends on timer and output comparator
- Timer-1 register A is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode
- Timer-1 register B is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
C8-3.1: init_tone() datasheet pp.132-135

```c
void init_tone(void) {
    DDRD |= _BV(PD5); /* enable output driver for OC1A */
    TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
    | _BV(WGM10); /* frequency (f) correct PWM, */
    TCCR1B = _BV(WGM13) /* varying f with OCR1A */
    | _BV(CS11); /* prescaler set to 8 */
}
```

- Port/Pin depends on *timer* and output *comparator*
- Timer-1 register A is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode
- Timer-1 register B is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
C8-3.1: init_tone()  datasheet pp.132-135

```
void init_tone(void){
    DDRD |= _BV(PD5);    /* enable output driver for OC1A */
    TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
        | _BV(WGM10);       /* frequency (f) correct PWM, */
    TCCR1B = _BV(WGM13)  /* varying f with OCR1A */
        | _BV(CS11);        /* prescaler set to 8 */
}
```

- Port/Pin depends on timer and output comparator
- Timer-1 register A is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode
- Timer-1 register B is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
void init_tone(void){
    DDRD |= _BV(PD5); /* enable output driver for OC1A */
    TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
            | _BV(WGM10); /* frequency (f) correct PWM, */
    TCCR1B = _BV(WGM13) /* varying f with OCR1A */
            | _BV(CS11); /* prescaler set to 8 */
}

- Port/Pin depends on *timer* and output *comparator*

- Timer-1 register A is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode

- Timer-1 register B is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
```c
void init_tone(void){
    DDRD |= _BV(PD5); /* enable output driver for OC1A */
    TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
        | _BV(WGM10); /* frequency (f) correct PWM, */
    TCCR1B = _BV(WGM13) /* varying f with OCR1A */
        | _BV(CS11); /* prescaler set to 8 */
}
```

- Port/Pin depends on timer and output comparator
- Timer-1 register A is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode
- Timer-1 register B is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
### C8-3.1: init_tone() datasheet pp.132-135

```c
1 void init_tone(void){
2     DDRD |= _BV(PD5); /* enable output driver for OC1A */
3     TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
4         | _BV(WGM10); /* frequency (f) correct PWM, */
5     TCCR1B = _BV(WGM13) /* varying f with OCR1A */
6         | _BV(CS11); /* prescaler set to 8 */
7 }
```

- **Port/Pin** depends on *timer* and output *comparator*

- **Timer-1 register A** is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode

- **Timer-1 register B** is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
void init_tone(void){
    DDRD |= _BV(PD5); /* enable output driver for OC1A */
    TCCR1A = _BV(COM1A0) /* toggle OC1A on match */
    | _BV(WGM10); /* frequency (f) correct PWM, */
    TCCR1B = _BV(WGM13) /* varying f with OCR1A */
    | _BV(CS11); /* prescaler set to 8 */
}

- Port/Pin depends on timer and output comparator
- Timer-1 register A is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode
- Timer-1 register B is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
void init_tone(void){
    DDRD |= _BV(PD5); /* enable output driver for OC1A */
    TCCR1A = _BV(COM1A0); /* toggle OC1A on match */
    | _BV(WGM10); /* frequency (f) correct PWM, */
    TCCR1B = _BV(WGM13) /* varying f with OCR1A */
    | _BV(CS11); /* prescaler set to 8 */
}

- Port/Pin depends on timer and output comparator
- Timer-1 register A is overwritten:
  - Use output A: toggle on match with TOP register
  - Frequency correct PWM mode
- Timer-1 register B is overwritten:
  - Use OCR1A as TOP register
  - Set the prescaler to 8
Set output frequency:

1. Which register?
2. What to write to it?
3. Desired frequency \( \rightarrow \) register value

- depends on timer, wave-form mode, and comparator
- we need to make sure we use unsigned integers
- For mode 9 (PWM, phase and frequency correct):

\[
f = \frac{f_{CPU}}{2 \cdot N \cdot TOP}
\]
Set output frequency:

1. Which register?
2. What to write to it?
3. Desired frequency $\rightarrow$ register value

- depends on timer, wave-form mode, and comparator
- we need to make sure we use unsigned integers
- For mode 9 (PWM, phase and frequency correct):

$$f = \frac{f_{CPU}}{2 \cdot N \cdot TOP}$$
Set output frequency:

1. Which register?
2. What to write to it?
3. Desired frequency $\rightarrow$ register value

- depends on timer, wave-form mode, and comparator
- we need to make sure we use unsigned integers
- For mode 9 (PWM, phase and frequency correct):

$$f = \frac{f_{CPU}}{2 \cdot N \cdot TOP}$$
Set output frequency:

1. Which register?
2. What to write to it?
3. Desired frequency $\rightarrow$ register value

- depends on timer, wave-form mode, and comparator
- we need to make sure we use unsigned integers
- For mode 9 (PWM, phase and frequency correct):

\[
  f = \frac{f_{CPU}}{2 \cdot N \cdot TOP}
\]
### C8-3.1: tone() datasheet pp.129-130

Set output frequency:

1. Which register?
2. What to write to it?
3. Desired frequency $\rightarrow$ register value

- depends on timer, wave-form mode, and comparator
- we need to make sure we use unsigned integers
- For mode 9 (PWM, phase and frequency correct):

\[
f = \frac{f_{CPU}}{2 \cdot N \cdot TOP}\]
Set output frequency:

1. Which register?
2. What to write to it?
3. Desired frequency $\rightarrow$ register value

- depends on timer, wave-form mode, and comparator
- we need to make sure we use unsigned integers
- For mode 9 (PWM, phase and frequency correct):

$$f = \frac{f_{CPU}}{2 \cdot N \cdot TOP}$$
Set output frequency:

1. Which register?
2. What to write to it?
3. Desired frequency $\rightarrow$ register value

- depends on timer, wave-form mode, and comparator
- we need to make sure we use unsigned integers
- For mode 9 (PWM, phase and frequency correct):
\[
f = \frac{f_{CPU}}{2 \cdot N \cdot TOP}
\]
where here $TOP = OCR1A$
Phase and Frequency Correct PWM Mode

- OCnA Interrupt Flag Set or ICFn Interrupt Flag Set (Interrupt on TOP)
- OCRnx/TOP Update and TOVn Interrupt Flag Set (Interrupt on Bottom)

Diagram labels:
- TCNTn
- OCnx
- Period

- (COMnx1:0 = 2)
- (COMnx1:0 = 3)
C8-3.1: tone()

```c
void tone(uint16_t frequency)
{
    /* We rely on the compiler to substitute the constant part of the expression. 1/2 for symmetric PWM & 1/2 for toggle output */
    OCR1A = (uint16_t) (F_CPU/(2*2*TONE_PRESCALER)/frequency);
}
```

- Write to the register selected with the mode (TOP→OCR1x) and comparator (A)
- Cast to `unsigned` 16-bit
- Defined according to compiler option
- Additional factor because output is set to toggle
- This constant is declared `unsigned long`
void tone(uint16_t frequency)
{
    /* We rely on the compiler to substitute the constant part of the expression.
       1/2 for symmetric PWM & 1/2 for toggle output */
    OCR1A = (uint16_t) (F_CPU/(2*2*TONE_PRESCALER)/frequency);
}

- Write to the register selected with the mode (TOP→OCR1x) and comparator (A)
- Cast to unsigned 16-bit
  - Defined according to compiler option
  - Additional factor because output is set to toggle
  - This constant is declared unsigned long
void tone(uint16_t frequency)
{
    /* We rely on the compiler to substitute the constant part of the expression.
       1/2 for symmetric PWM & 1/2 for toggle output */
    OCR1A = (uint16_t) (F_CPU/(2*2*TONE_PRESCALER)/frequency);
}

- Write to the register selected with the mode (TOP→OCR1x) and comparator (A)
- Cast to unsigned 16-bit
- Defined according to compiler option
  - Additional factor because output is set to toggle
  - This constant is declared unsigned long
C8-3.1: tone()

```c
void tone(uint16_t frequency)
{
/* We rely on the compiler to substitute the constant
   part of the expression.
   1/2 for symmetric PWM & 1/2 for toggle output */
OCR1A = (uint16_t) (F_CPU/(2*2*TONE_PRESCALER)/frequency);
}
```

- Write to the register selected with the mode (TOP→OCR1x) and comparator (A)
- Cast to \textit{unsigned} 16-bit
- Defined according to compiler option
- Additional factor because output is set to toggle
- This constant is declared \textit{unsigned long}
C8-3.1: tone()

```c
void tone(uint16_t frequency)
{
    /* We rely on the compiler to substitute the constant part of the expression.
       1/2 for symmetric PWM & 1/2 for toggle output */
    OCR1A = (uint16_t) (F_CPU/(2*2*TONE_PRESCALER)/frequency);
}
```

- Write to the register selected with the mode (TOP→OCR1x) and comparator (A)
- Cast to *unsigned* 16-bit
- Defined according to compiler option
- Additional factor because output is set to toggle
- This constant is declared *unsigned long*
C8 Prep: Allocate Timer for PWM

ATmega 644P Timers:
- 1 × 16 bit
- 2 × 8 bit

Only 8-bit timers left, which one to choose?

Fast PWM
- Prescaler = 1
- Fast PWM mode

We want two output channels for stereo panning, but one of the outputs of timer 0 is also used for the USB interface (see IlMatto circuit diagram). ⇒ better select Timer 2
C8 Prep: Allocate Timer for PWM

ATmega 644P Timers:

- 1 × 16 bit
- 2 × 8 bit

Only 8-bit timers left, which one to choose?

Fast PWM
- Prescaler = 1
- Fast PWM mode

We want two output channels for stereo panning, but one of the outputs of timer 0 is also used for the USB interface (see IlMatto circuit diagram). ⇒ better select Timer 2
C8 Prep: Allocate Timer for PWM

ATmega 644P Timers:
▶ 1 × 16 bit
▶ 2 × 8 bit

Only 8-bit timers left, which one to choose?

Fast PWM
▶ Prescaler = 1
▶ Fast PWM mode

We want two output channels for stereo panning, but one of the outputs of timer 0 is also used for the USB interface (see IlMatto circuit diagram). ⇒ better select Timer 2
Picture Credits

Astronomic Clock at Lund.
Available through:

Punch cards used by Jacquard loom: Museum of Science and Industry in Manchester, Photo by George H. Williams, 2004.
Available through:

C8 Titlepage: Wolfgang Amadeus Mozart score for Idomeneo, 1781.
Available through http://staatsbibliothek-berlin.de/, sourced from:
http://commons.wikimedia.org/wiki/File:Idomeneo_score.jpg

ATmega644P Datasheet. Available through
https://secure.ecs.soton.ac.uk/notes/elec1201/doc8152.pdf