Design with Microprocessors

Lecture 5

Year 3 CS
Academic year 2023/2024
1st Semester

Lecturer: Radu Dănescu

Timing on Arduino

- **Delay functions**
  - **delay**(unsigned long ms) – delay for a specified number of milliseconds
  - **delayMicroseconds**(unsigned int us) – delay for a specified number of microseconds
- **Functions for reading time**
  - unsigned long **millis**() – returns the time, in milliseconds, since the program started. Overflows in about 50 days.
  - unsigned long **micros**() – returns the time, in microseconds, since the current program was started. Overflows in about 70 minutes. For a 16 MHz clock microcontroller, this function has a 4 us resolution.

Timing on Arduino

- **Example: timing without delay()**

```
const int ledPin =  13;                              // pin with a LED

int ledState = LOW;                                  // led state, initially off
long previousMillis = 0;                             // variable holding the time of the last state update

long interval = 1000;                                // blink interval, in ms

void setup() {
  pinMode(ledPin, OUTPUT);                           // setting up the LED pin for output
}

void loop()
{
 unsigned long currentMillis = millis();             // read the current time

 if(currentMillis - previousMillis > interval) {     // if the elapsed time is larger than the preset interval
   previousMillis = currentMillis;                   // update the previous time

   if (ledState == LOW)                              // switch the LED state
     ledState = HIGH;
   else
     ledState = LOW;

   digitalWrite(ledPin, ledState);                   // write the LED state to output
 }
}
```

Source: http://arduino.cc/en/Tutorial/BlinkWithoutDelay

Timing on Arduino

- **Example: timing without delay() – for multitasking !**
  - **Two leds, blinking every 1 second, with 0.5 seconds delay between them**

```
long sequenceDelay = 500;              // offset between the two actions
long flashDelay = 1000;

boolean LED13state = false;            // states of the two LEDs, on pins 13 and 12
boolean LED12state = false;            // initially they are both off
long waitUntil13 = 0;                  // first LED will be lit immediately
long waitUntil12 = sequenceDelay;      // the second one after 0.5 sec
void setup() {
  pinMode(13, OUTPUT);                 // set up the pins
  pinMode(12, OUTPUT);
}
void loop() {
  digitalWrite(13, LED13state);        // every iteration, write the update pin state
  digitalWrite(12, LED12state);

  if (millis() >= waitUntil13) {       // check time for the first LED
    LED13state = !(LED13state);        // if 1 second has passed, switch the state
    waitUntil13  += flashDelay;        // new target time, +1 second
  }

 if (millis() >= waitUntil12) {        // check time for the second LED
    LED12state = !(LED12state);        // if 1 second has passed, switch the state
    waitUntil12 += flashDelay;         // new target time, +1 second
  }
}
```

Source: http://www.baldengineer.com/blog/2011/01/06/millis-tutorial/

Timing on Arduino

- **Using the Timer library for synching / timing**
- **http://playground.arduino.cc/Code/Timer**
- **Class methods:**
- **int every(long period, callback)**: runs the ‘callback’ function every ‘period’ miliseconds. Returns the ID of the programmed event.
- **int every(long period, callback, int repeatCount)**: runs the ‘callback’ function every ‘period’ milliseconds, for ‘repeatCount’ times.
- **int after(long duration, callback)**: runs the ‘callback’ function once, after ‘duration’ miliseconds.
- **int oscillate(int pin, long period, int startingValue)**: signal generation. Changes the state of ‘pin’ every ‘period’ milliseconds. The initial pin state is specified by ‘startingValue’, HIGH or LOW.
- **int oscillate(int pin, long period, int startingValue, int repeatCount)**: changes the state of ‘pin’, every ‘period’ milliseconds, for ‘repeatCount’ times.
- **int pulse(int pin, long period, int startingValue)**: changes the state of ‘pin’ once, after ‘period’ miliseconds. The initial value is specified by ‘startingValue’.
- **int stop(int id)**: all the above functions return an identifier of the programmed event. This ID can be used by the *stop* function to stop the event. At most 10 events can be active at once.
- **int update()**: this function must be called in the loop function, to update the state of the Timer object.

Timing on Arduino

- **Example: generating a long pulse, without blocking the program**

```
#include "Timer.h"

Timer t;                                    // declaration of the Timer object
int pin = 13;

void setup()
{
  pinMode(pin, OUTPUT);
  t.pulse(pin, 10 * 60 * 1000, HIGH);       // 10 minutes pulse, initial value HIGH
}

void loop()
{
  t.update();                               // required for the timer to work
                                            // cost of calling this function is several microseconds

  // the rest of the cycle is free for other tasks
}
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

Timing on Arduino

- **Example: calling a function periodically, while generating a periodic signal.**

```
#include "Timer.h"

Timer t;                                // declaration of the Timer object
int pin = 13;                           // the oscillating pin

void setup()
{
  Serial.begin(9600);                   // initialization of the Serial interface
  pinMode(pin, OUTPUT);                 // set up the pin for output
  t.oscillate(pin, 100, LOW);           // set up the period of the oscillating signal (100 ms)
  t.every(1000, takeReading);           // call the takeReading function every 1000 ms
}

void loop()
{
  t.update();                           // required for the timer to operate
}

void takeReading()                      // the function called every second
{
  Serial.println(analogRead(0));        // read an analog pin and send its value by Serial
}
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

Timing on Arduino

- **Example: stopping a process**

```
#include "Timer.h"

Timer t;

int ledEvent;                                   // event ID

void setup()
{
  Serial.begin(9600);                           // set up the Serial interface
  int tickEvent = t.every(2000, doSomething);   // call doSomething every 2 seconds
  Serial.print("2 second tick started id=");    // write through Serial that we have started …
  Serial.println(tickEvent);                    // an event with this ID

  pinMode(13, OUTPUT);
  ledEvent = t.oscillate(13, 50, HIGH);         // start a 50 ms period oscillating event
  Serial.print("LED event started id=");        // write through Serial that we have started …
  Serial.println(ledEvent);                     // another event, with another ID

  int afterEvent = t.after(10000, doAfter);     // program the function doAfter to execute after 10s
  Serial.print("After event started id=");      // write again what we have started
  Serial.println(afterEvent);                   // and its ID
  }
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

Timing on Arduino

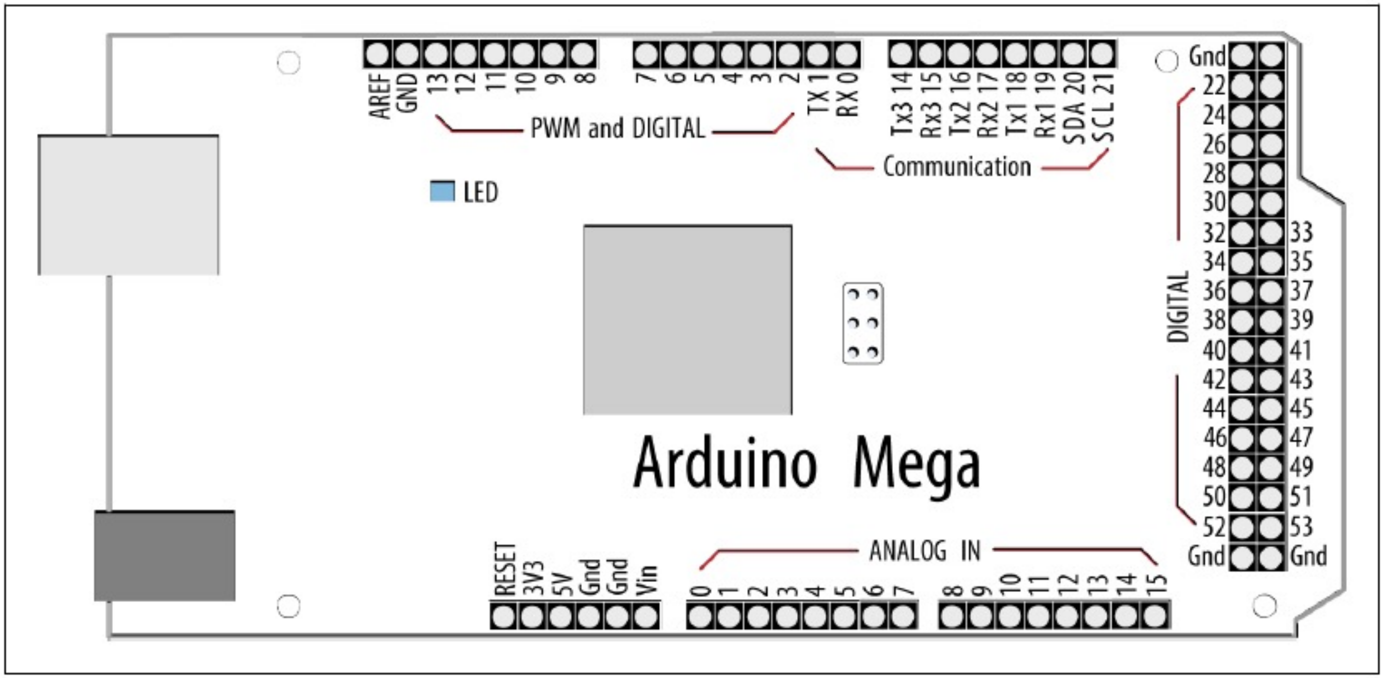
- **Example: stopping a process (continued)**

```
void loop()
{
  t.update();                                   // update the timer
}

void doSomething()                              // the every 2 seconds function
{
  Serial.print("2 second tick: millis()=");     // send the current number of milliseconds
  Serial.println(millis());                     // via the Serial interface
}


void doAfter()                                  // the after 10 secunds, once call function
{
  Serial.println("stop the led event");
  t.stop(ledEvent);                             // stop the LED oscillation event
  t.oscillate(13, 500, HIGH, 5);                // and start another oscillation, 500 ms period
}                                               // only 5 times
```

http://www.doctormonk.com/2012/01/arduino-timer-library.html

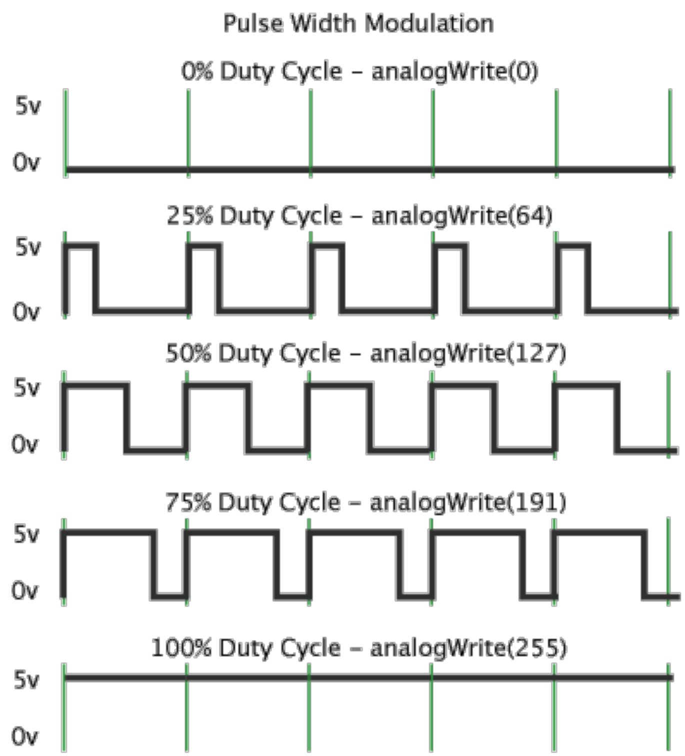
Signal generation

- **Pulse Width Modulation (PWM) on Arduino:** some of the Arduino pins support PWM, achieved using the internal timers.
- **Arduino Mega:** pins 2-13 support PWM
- **Fixed frequency:** approximately 500 Hz



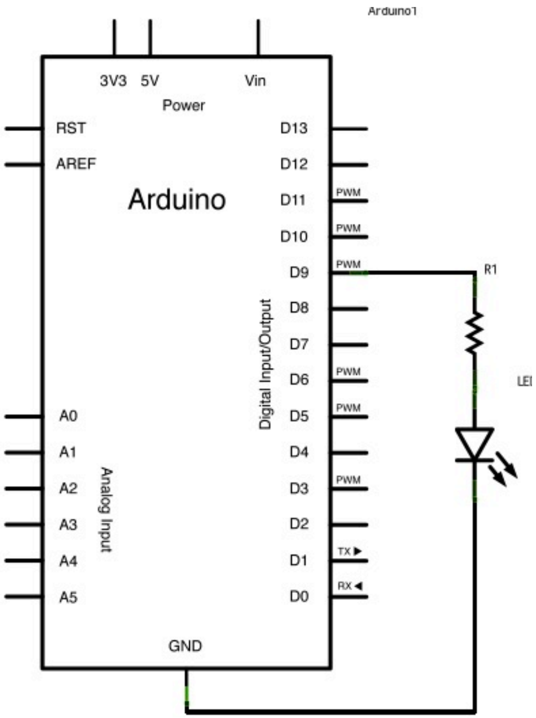
Signal generation

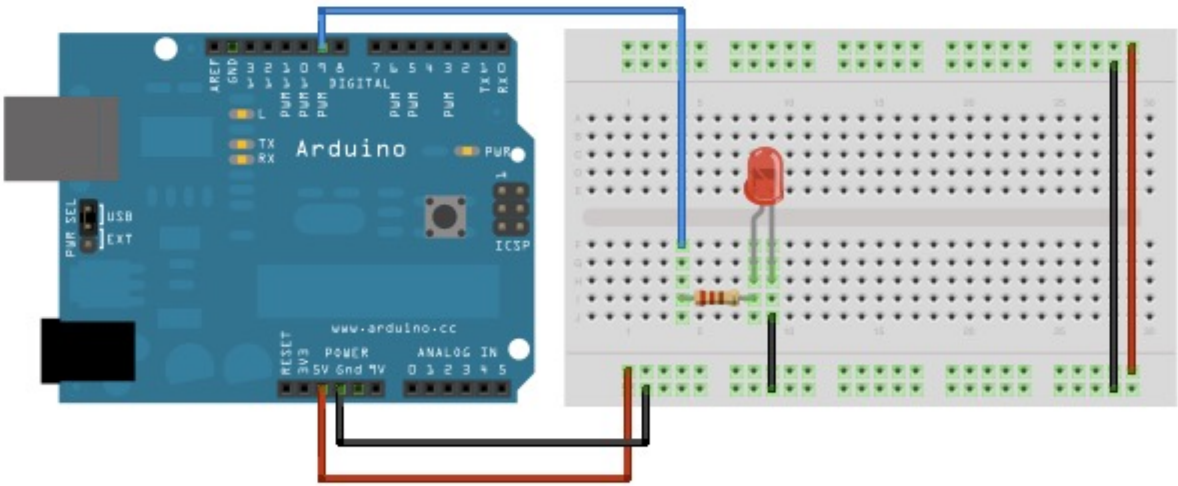
- Calling the function **analogWrite (pin, value)** causes a PWM signal to be generated on the ‘pin’, having a duty cycle specified by ‘value’.
- ‘value’ can be between 0 and 255, corresponding to duty cycles from 0% to 100%
- The pin used for PWM generation must be set up as output.



Signal generation

- **Example:** fade-in, fade-out, using an external led
- **Source:** http://arduino.cc/en/Tutorial/Fade





Signal generation

- **Example:** fade-in, fade-out, using an external led
- **Source:** http://arduino.cc/en/Tutorial/Fade

```
int led = 9;
int brightness = 0;                    // current state of the LED, initially off
int fadeAmount = 5;                    // increment of the LED state


void setup()  {
  pinMode(led, OUTPUT);                // pin 9, output
}


void loop()  {

  analogWrite(led, brightness);        // set the fill factor of the PWM signal

  brightness = brightness + fadeAmount; // change the current fill factor by the increment

  if (brightness == 0 || brightness == 255) { // at the ends of the interval, change the sign of the increment
    fadeAmount = -fadeAmount ;
  }

  delay(30);                    // small delay
}
```

Signal generation

- The **tone ()** function causes the generation of pulses of variable frequency and 50% duty cycle:
- **tone(pin, frequency)** – causes a signal of given ‘frequency’ on the specified ‘pin’, for an unlimited length of time.
- **tone(pin, frequency, duration)** – causes a signal of given ‘frequency’ on the specified ‘pin’, for ‘duration’ milliseconds.

- The **noTone(pin)** function stops signal generation for ‘pin’.
- Only one pin can generate a tone at a given time. If we want to generate another tone, on another pin, first we must call **noTone()** to stop tone generation for the active pin.
- On some Arduino boards, tone generation may interfere with PWM generation capabilities.

Signal generation

- **Example: playing a song by notes.**

```
const int speakerPin = 9;  // Connect speaker to pin 9

char noteNames[] =     {'C','D','E','F','G','a','b'}; // Name of the notes
unsigned int frequencies[] = {262,294,330,349,392,440,494};  // The associated frequencies
const byte noteCount = sizeof(noteNames); // Number of notes in the table

// The song score, space means pause
char score[] = "CCGGaaGFFEEDDC GGFFEEDGGFFEED CCGGaaGFFEEDDC ";
const byte scoreLen = sizeof(score); // Number of notes in the song

void setup()
{
}

void loop()
{
  for (int i = 0; i < scoreLen; i++) // Scan the score
  {
    int duration = 333;    // Play each note for 0.33 seconds
    playNote(score[i], duration); // Call the note playing function (next slide)
  }

  delay(4000);  // Long pause before starting the song again
}
```

Signal generation

- **Example: playing a song by notes (continued).**

```
void playNote(char note, int duration)
{
  // Scan the list of notes
  for (int i = 0; i < noteCount; i++)
  {
    // Find the note to play in the list
    if (noteNames[i] == note)  // If found…
      tone(speakerPin, frequencies[i], duration);  // Generate the tone with the corresponding frequency
  }
  // The delay is executed when the note is found, but also when it is not found
  delay(duration);
}
```

Advanced use of timers

- Sometimes more flexibility is required when using timers.
- **Two options:** use of a dedicated library, or configuring the timers directly using the AVR registers
- The **Timer1 library**: http://playground.arduino.cc/Code/Timer1
  - Provides functions for using the 16 bit timer Timer 1.
  - Not all signal generating pins of Timer 1 can be used by Arduino Mega. If more pins are needed, **Timer 3** can be used: http://playground.arduino.cc/uploads/Code/TimerThree.zip, with the same functions.
- Most important methods of the Timer1 class:
- **initialize(period)** – initializing the timer with ‘period’ microseconds. The period is the interval in which the timer performs a complete counting sequence.
- **setPeriod(period)** – changes the period of an already initialized timer.
- **pwm(pin, duty, period)** – generates a PWM signal on ‘pin’, using the specified duty cycle, having values between 0 and 1023, and with an (optional) period of microseconds. For ‘pin’ you can specify only the values connected to the output compare outputs of the timer (Timer 1 is connected to pins 9 and 10, Timer 3 is connected to pins 2, 3 and 5 – Arduino Mega).
- **attachInterrupt(function, period)** – attaches a ‘function’ to be called every time the timer finishes a counting sequence, or at intervals specified by the optional parameter ‘period’.

Advanced use of timers

- **detachInterrupt()** – de-activates the interrupt and detaches the ISR function specified by attachInterrupt().
- **disablePwm(pin)** – de-activates the PWM signal generation on the specified pin.
- **read()** – returns the time since the last counter overflow, in microseconds.

Relationship between periods, resolution and the prescaler (for 16 MHz boards):

Prescaler	**Time between counter increments**	**Maximum period**
1	0.0625 uS	8.192 mS
8	0.5 uS	65.536 mS
64	4 uS	524.288 mS
256	16 uS	2097.152 mS
1024	64 uS	8388.608 mS

Advanced use of timers

- **Example – using the Timer 1 library**
- Starts generating a PWM signal on pin 9, with a 50% duty cycle, and activates an interrupt that changes the state of pin 10 every half second.

```
#include "TimerOne.h"

void setup()
{
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);              // init timer 1, with a 0.5 seconds period
  Timer1.pwm(9, 512);                     // pin 9 PWM, 50% duty cycle
  Timer1.attachInterrupt(callback);       // attack the callback() function as interrupt handler
}                                         // interrupt triggered when the timer saturates

void callback()
{
  digitalWrite(10, digitalRead(10) ^ 1);  // change the state of pin 10
}

void loop()
{
  // the main loop is completely free for other tasks
}
```

Advanced use of timers

- **Using the configuration registers**
- **Example:** the setPeriod function of the Timer1 library

```
#define RESOLUTION 65536    // A 16 bit timer's maximum value

void TimerOne::setPeriod(long microseconds)
{
long cycles = (F_CPU / 2000000) * microseconds;             // PWM phase correct, counts twice for a period, thus division by 2 mil

// check if the number of cycles fits in the maximum value of the counter
if(cycles < RESOLUTION)              clockSelectBits = _BV(CS10);              // no prescaling
else if((cycles >>= 3) < RESOLUTION) clockSelectBits = _BV(CS11);              // prescaling by 8
else if((cycles >>= 3) < RESOLUTION) clockSelectBits = _BV(CS11) | _BV(CS10);  // prescaling by 64
else if((cycles >>= 2) < RESOLUTION) clockSelectBits = _BV(CS12);              // prescaling by 256
else if((cycles >>= 2) < RESOLUTION) clockSelectBits = _BV(CS12) | _BV(CS10);  // prescaling by 1024
else        cycles = RESOLUTION - 1, clockSelectBits = _BV(CS12) | _BV(CS10);  // impossible, use maximum prescaling and
                                                                               //number of cycles

  oldSREG = SREG;                                          // save the state register
  cli();                                                   // deactivate the interrupt system
  ICR1 = pwmPeriod = cycles;                               // configure register ICR1
  SREG = oldSREG;                                          // restore SREG
  TCCR1B &= ~(_BV(CS10) | _BV(CS11) | _BV(CS12));   // clear clock select bits for Timer 1
  TCCR1B |= clockSelectBits;                               // configure clock select bits as computed above

}
```

Advanced use of timers

- **Using the configuration registers**
- **The involved registers (from: http://www.atmel.com/images/doc2545.pdf):**

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

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.

The 16 bit register ICR1 is divided into ICR1H and ICR1L. This register is used for specifying the TOP value of the counter. After reaching TOP, the counter counts backwards to 0, to end the period.

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
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	

Advanced use of timers

- **Configuration modes of 16 bit timers**

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

Exercises

- **Assuming that we don't have the functions delay() and millis(), implement them using timers.**

- **Using a 2 digit 7 segment display, display any given number using timers.**

- **Write a program for lighting control. Each of the 24 hours of a day will be defined (using a LUT) as day, evening, morning or night. Depending on the hour type, the light will be either turned off, medium on, or full on.**

- **Assume that we have an external Digital to Analog Converter (DAC) connected to PORTA, which instantaneously converts the received 8-bit number into an analog voltage, 0 V for the value 0, and 3.3 V for the value 255. Write the Arduino program to generate a sine function having the mean value 1.5 V, the amplitude of 1.5 V, and the frequency 50 Hz.**