Design with Microprocessors

Year III Computer Science
1-st Semester

Lecture 5a: Timing events with Arduino

Timing events with Arduino

Delay functions

- **delay**(unsigned long ms) - Pauses the program for the amount of time (in miliseconds) specified as parameter.
- **delayMicroseconds**(unsigned int us) – Pauses the program for the amount of time (in microseconds) specified as parameter

Example1: http://arduino.cc/en/Reference/Delay

```
int ledPin = 13;                  // LED connected to digital pin 13
void setup()
{
  pinMode(ledPin, OUTPUT);   // sets the digital pin as output
}

void loop()
{
  digitalWrite(ledPin, HIGH);    // sets the LED on
  delay(500);                    // waits for half second
  digitalWrite(ledPin, LOW);     // sets the LED off
  delay(500);                    // waits for half second
}
```

Timing events with Arduino

Delay functions

- **delay**(unsigned long ms) - Pauses the program for the amount of time (in miliseconds) specified as parameter.

http://arduino.cc/en/Reference/Delay

The use of delay() in a sketch has significant drawbacks:

•No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function ⇒ brings most other activity to a halt.

•For alternative approaches to controlling timing see the millis() function

•Avoid the use of delay() for timing of events longer than 10's of milliseconds unless the Arduino sketch is very simple.

Certain things **do go on** while the delay() function is controlling the ATmega chip however, because the delay function does not disable interrupts:

•Serial communication that appears at the RX pin is recorded

•PWM (analogWrite) values and pin states are maintained,

•interrupts will work as they should.

Timing events with Arduino

Time reading functions

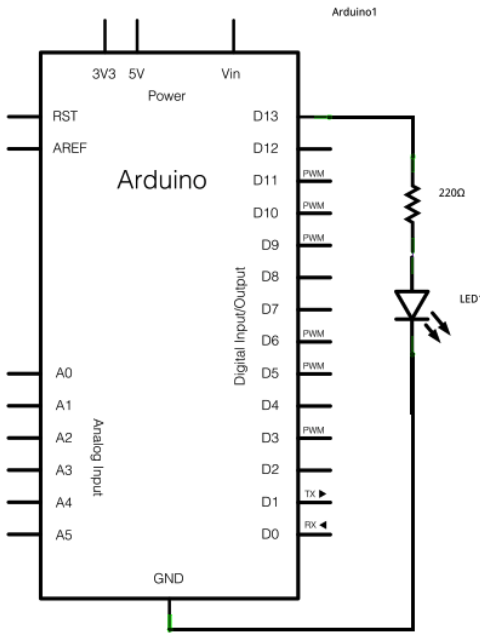
- unsigned long **millis**() – returns the no. of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.
- unsigned long **micros**() – returns the no. of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes. On 16 MHz Arduino boards (e.g. Uno, Mega), this function has a resolution of 4 us (i.e. the value returned is always a multiple of 4 us).

Example 2: http://arduino.cc/en/Tutorial/BlinkWithoutDelay - How to blink the LED without using delay().

- Keeps track of the last time the Arduino turned the LED on or off.
- Each time through loop(), it checks if a long enough interval has passed.
- If it has, it toggles the LED on or off.

Other code can run at the same time without being interrupted by the LED code !

Arduino UNO/MEGA have an on the board LED attached to pin 13, so no hardware is needed for this example !



Timing events with Arduino

Example 2: Blink without Delay - http://arduino.cc/en/Tutorial/BlinkWithoutDelay

```
// constants won't change. Used here to set pin numbers:
const int ledPin =  13;      // the number of the LED pin

// Variables will change:
int ledState = LOW;             // ledState used to set the LED (2 bytes: http://arduino.cc/en/Reference/Int)
long previousMillis = 0;        // will store last time LED was updated (4 bytes: http://arduino.cc/en/Reference/Long)

long interval = 1000;           // interval at which to blink (milliseconds)

void setup() {
   pinMode(ledPin, OUTPUT);    // set the digital pin as output:
}

void loop() // here is where you'd put code that needs to be running all the time.
{
  unsigned long currentMillis = millis();  // current time

   // check if it's time to blink the LED: (current_time – last_time) > the interval at which you want to blink the LED
  if(currentMillis - previousMillis > interval)
    {
        previousMillis = currentMillis;     // save the last time you blinked the LED

        if (ledState == LOW)     // toggle LED status
            ledState = HIGH;
        else
            ledState = LOW;

         // set the LED with the ledState of the variable:
         digitalWrite(ledPin, ledState);
     }
}
```

Timing events with Arduino

Example 3: Arduino Multitasking – 2 LEDs, each switched every 1s with a 0.5s delay between (http://www.baldengineer.com/blog/2011/01/06/millis-tutorial/)

```
long sequenceDelay = 500;                // seed / offset for the 2-nd LED toggle (defazaj temporal !!!)
long flashDelay = 1000;                  // LED Flash period

boolean LED13state = false;               // LEDs on pin 13 and 12 (initially both OFF)
boolean LED12state = false;
long waitUntil13 = 0;                    // First LED lit-on immediately
long waitUntil12 = sequenceDelay;        // 2-nd LED with a 0.5 sec delay (offset added to the absolute time)
void setup() {
   pinMode(13, OUTPUT);
   pinMode(12, OUTPUT);
}
void loop() {
   digitalWrite(13, LED13state);         // each iteration of loop() will set the IO pins,
   digitalWrite(12, LED12state);

// checking to see if enough time has elapsed
   if (millis() >= waitUntil13) {        // check the time for the 1-st LED
      LED13state = !(LED13state);        //  if time elapsed toggle the status of the 1-st LED
      waitUntil13  += flashDelay;        // next toggle time = current time + 1000 ms
   }

// keep in mind, waitUntil12 was already seeded with a value of 500 ms
   if (millis() >= waitUntil12) {        // check the time for the 2-nd LED
      LED12state = !(LED12state);        //  if time elapsed toggle the status of the 2-nd LED
      waitUntil12 += flashDelay;         // next toggle time = current time + 1000 ms
   }
}
```

Timing events with Arduino

TIMER library for synchronization and timing
http://playground.arduino.cc/Code/Timer

Metods:

•**int every(long *period*, *callback*)**: runs function '*callback*' at regular time intervals (*period* [ms])

•**int every(long period, callback, int repeatCount)**: runs function '*callback*' at regular time intervals ('*period'* [ms]) for a limited no of times: '*repeatCount*'

•**int after(long duration, callback)**: runs function '*callback*' after a time interval ('*duration'* [ms])

•**int oscillate(int pin, long period, int startingValue)**: signal generation – changes the status of 'pin' after each 'period' [ms]. Initial pin state in 'startingValue' (HIGH or LOW).

•**int oscillate(int pin, long period, int startingValue, int repeatCount)**: changes the status of 'pin' after each 'period' [ms] for a no. of 'repeatCount' times.

•**int pulse(int pin, long period, int startingValue)**: changes the status of 'pin' once, after 'period' [ms]. Initial pin state in 'startingValue' (HIGH or LOW).

•**int stop(int id)**: All the functions above are returning an 'id' for the programmed event. Use this function to stop the event (id). Max events / timer = 10.

•**int update()**: must be called in the main loop to update the status of the Timer object !

Timing events with Arduino

Example 4: generating a long pulse without blocking the system
http://www.doctormonk.com/2012/01/arduino-timer-library.html

```
// Classic approach (delay)

void setup()
{
  pinMode(13, OUTPUT);
  digitalWrite(pin, HIGH);
  delay(10 * 60 * 1000);
  digitalWrite(pin, LOW);
}

void loop()
{
}
```

The disadvantage of the delay approach is that nothing else can go on while the 'delay' is happening. You cannot update a display, or check for key presses for example.

```
// Timer based approach

#include "Timer.h"

Timer t;    // declare the Timer object
int pin = 13;

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

void loop()
{
  t.update();  // update timer object
  // the update function call duration is us

  // insert other processing here: i.e display,
  // sensor input, actuators control etc.
}
```

Timing events with Arduino

Example 5: Usage of 2 timer events - One to flash a LED (oscillating signal) and another that reads analog input A0 and displays the result in the Serial Monitor.
http://www.doctormonk.com/2012/01/arduino-timer-library.html

```
#include "Timer.h"

Timer t;                          // declare the Timer object
int pin = 13;                     //  LED pin (flash event)

void setup()
{
  Serial.begin(9600);             // init. serial communication
  pinMode(pin, OUTPUT);           // configure pin
  t.oscillate(pin, 100, LOW);     // init. oscillating signal (100 ms)
  t.every(1000, takeReading);     // at every 1000 ms call function takeReading()
}

void loop()
{
  t.update();                     // update timer object – compulsory for timer functioning
}

void takeReading()                // function called every 1000 ms (1 Hz)
{
  Serial.println(analogRead(0));  // send on the serial link the value red from the analogue pin 0
  // analogRead(pin) -  returns  a digital value (0 to 1023) obtained by converting
  // the input voltage (0 .. 5V) using the ADC
}
```

Timing events with Arduino

Example 6: stopping an event

Write to the serial monitor every 2 seconds(*tickEvent*), flash the LED(*ledEvent*) fast and after 10 seconds(afterEvent), stops the LED flashing fast, and flash it 5 times slowly.
http://www.doctormonk.com/2012/01/arduino-timer-library.html

```
#include "Timer.h"

Timer t;

int tickEvent, ledEvent, afterEvent, ledEventNew;          // events IDs

void setup()
{
  Serial.begin(9600);                                      // init serial comm.
  tickEvent = t.every(2000, doSomething);                  // call doSomething every 2 sec.
  Serial.print("2 second tick started id=");               // write the ID(tickEvent) on serial interface
  Serial.println(tickEvent);

  pinMode(13, OUTPUT);
  ledEvent = t.oscillate(13, 50, HIGH);                    // start  ledEvent (LED flashing)  - 20 Hz toggle
  Serial.print("LED event started id=");                   // write the ID(ledEvent) on serial interface
  Serial.println(ledEvent);

  afterEvent = t.after(10000, doAfter);                    // schedule doAfter execution, after 10 sec.
  Serial.print("After event started id=");                 // write the ID(ledEvent) on serial interface
  Serial.println(afterEvent);
}
```

Timing events with Arduino

Example 6: stopping an event - continued

Write to the serial monitor every 2 seconds(*tickEvent*), flash the LED(*ledEvent*) and after 5 seconds(afterEvent), stop the LED flashing fast, and flash it 5 times slowly.
http://www.doctormonk.com/2012/01/arduino-timer-library.html

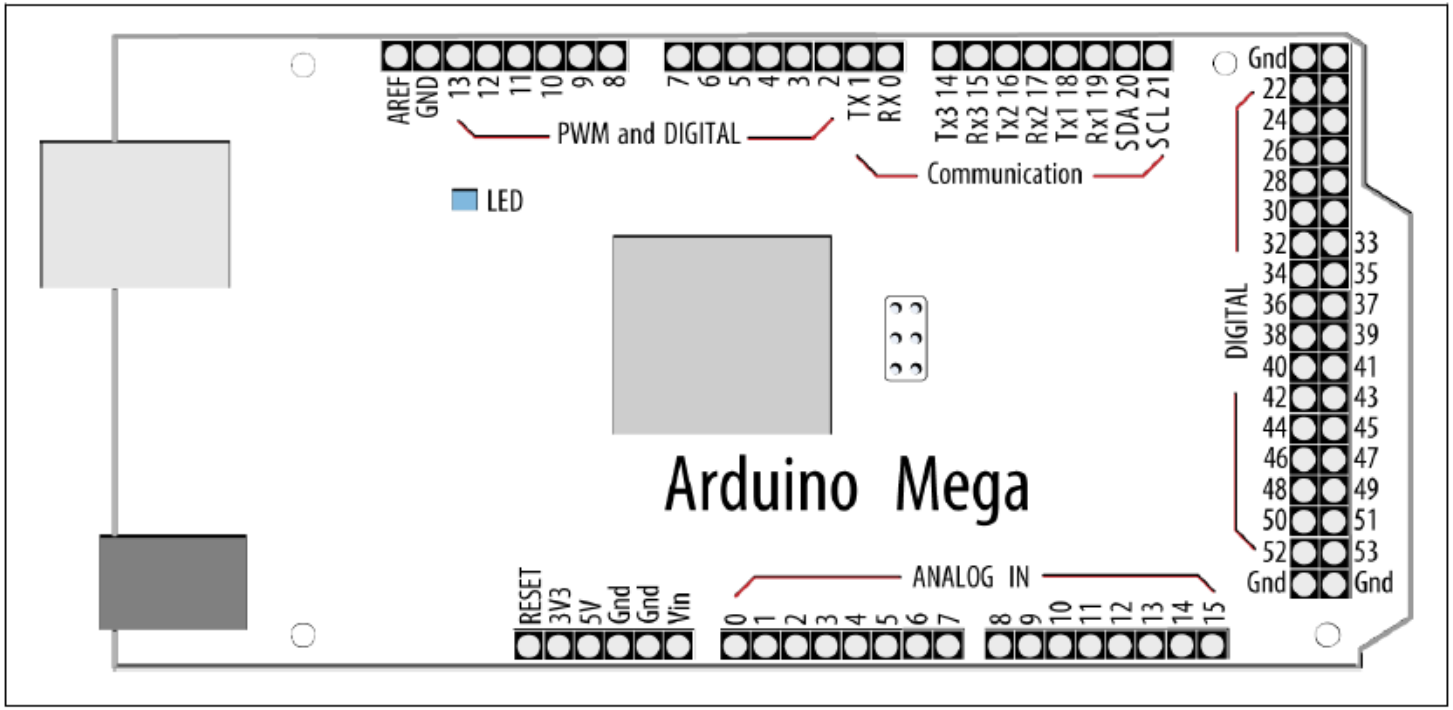
```
void loop()
{
  t.update();                                  // update timer object – compulsory
}

void doSomething()                             // Called every 2 sec.
{
  Serial.print("2 second tick: millis()=");     // send current time [ms]
  Serial.println(millis());                    // on the serial interface
}

void doAfter()                                 // called after 10 sec. / once
{
  Serial.println("stop the led event");
  t.stop(ledEvent);                            // Stops the initial 20 Hz oscillation of the LED
 ledEventNew = t.oscillate(13, 500, HIGH, 5);  // Starts a new oscillation (2 Hz toggle / 5 times)
 Serial.print(“New LED event started id=");    // write the ID(ledEventNew) on serial interface
 Serial.println(ledEventNew);
}
```

Q: ledEvent == ledEventNew ???

Signal generation (PWM)

Pulse Width Modulation (PWM) – for Arduino boards some digital pins provide the function (implemented internally using timers):

- Arduino Uno: 3, 5, 6, 10, 11 (~ symbol)
- Arduino Mega: 2 .. 13
- PWM frequency is fixed: ≈ 500 Hz



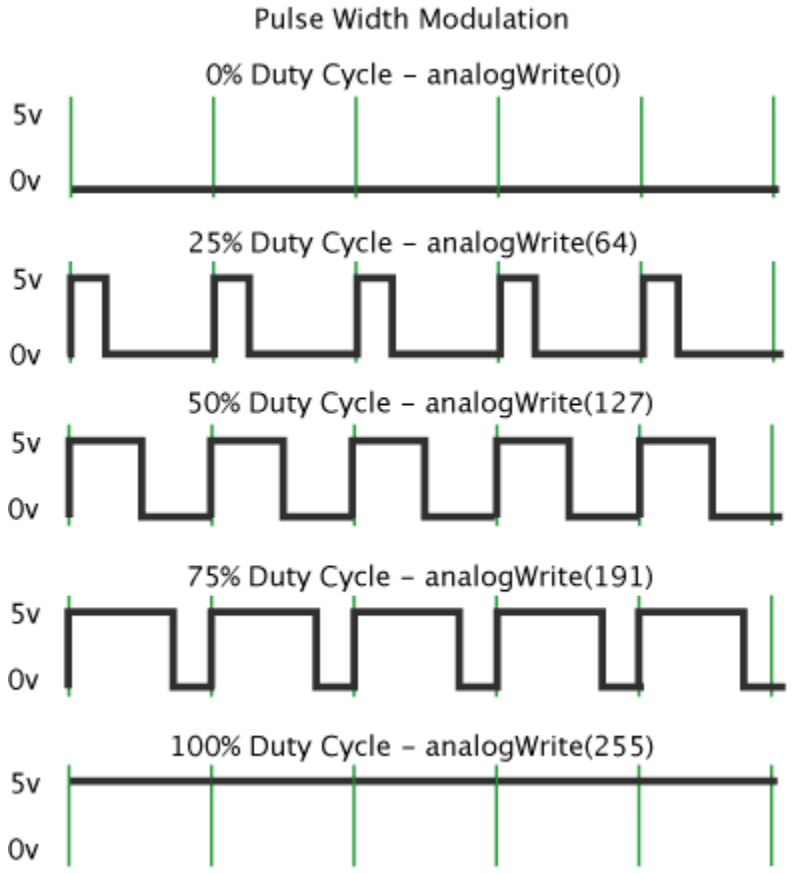
Signal generation (PWM)

- **analogWrite (pin, value)** – generates a PWM signal on ‘pin’, with duty cycle specified in ‘value’
- ‘value’ = 0 .. 255, (duty cycle: D = 0% … 100%)
- ‘pin’ must be configured as output

$$D = \frac{T_{on}}{T_{on} + T_{off}}$$

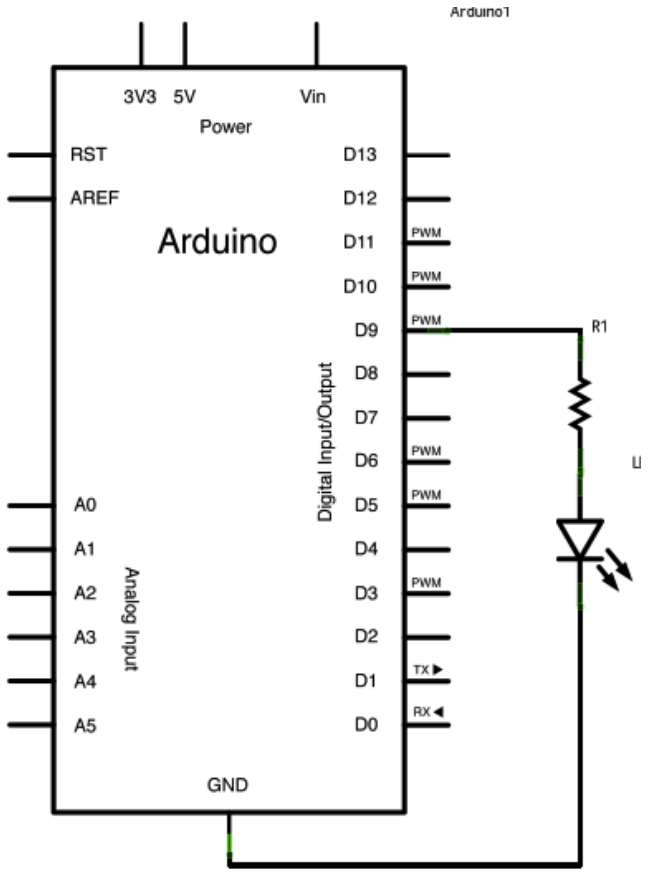
$$D = \frac{value}{255}$$

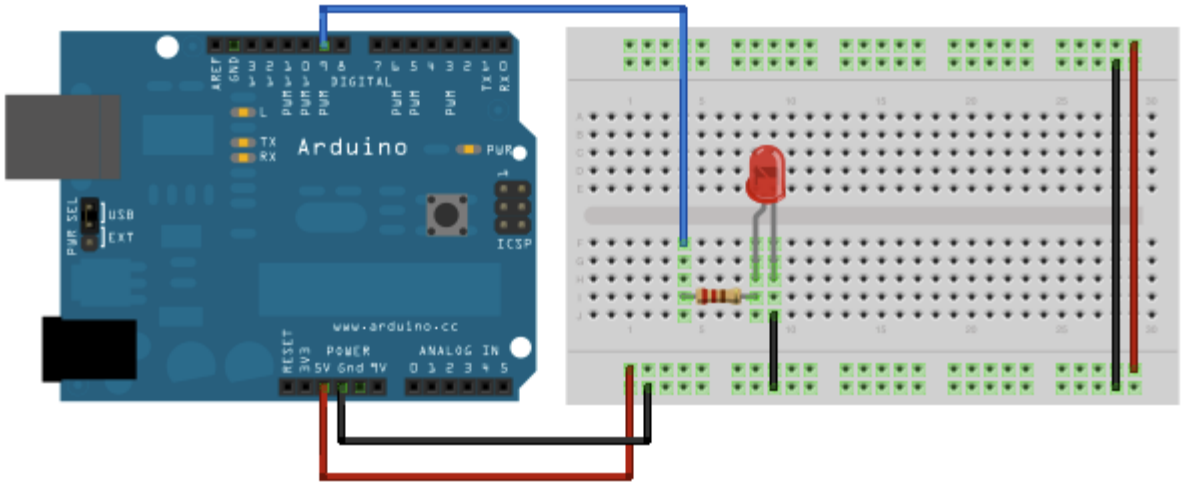
$$D = \frac{value}{255} \times 100[\%]$$



Signal generation (PWM)

Exemple 7: fade-in, fade-out with an external LED
http://arduino.cc/en/Tutorial/Fade





Signal generation (PWM)

Exemple 7: fade-in, fade-out with an external LED
http://arduino.cc/en/Tutorial/Fade

```
int led = 9;                // the pin that the LED is attached to
int brightness = 0;         // LED brightness ≈ D (duty cicle)
int fadeAmount = 5;         // fade stepping

void setup()  {
   pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop()  {

 // set the brightness of pin 9 (Duty cycle)
  analogWrite(led, brightness);

  // update the brightness for the next step of the loop
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see (visualize) the dimming effect
  delay(30);
}
```

Signal generation

Variable frequency and fixed duty cycle (50%): *tone()* function

(~ CTC timer mode / L5)

- **tone(pin, frequency)** – generates a signal with ‘frequency’ on ‘pin’
- **tone(pin, frequency, duration)** – generates a signal with ‘frequency’ on ‘pin’ in a limited time period: ‘duration’ [ms]
- **noTone(pin)** stops the signal generation on ‘pin’.

Only one pin at a time can generate a signal using tone function. To generate a tone signal on another pin you should stop any active tone: **noTone(activePin)**

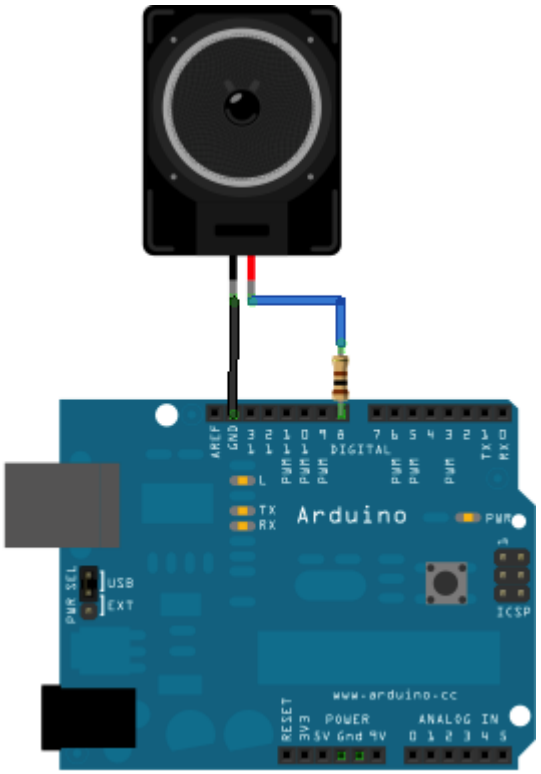
For some Arduino boards, the tone function/generation can alter PWM signal generation !

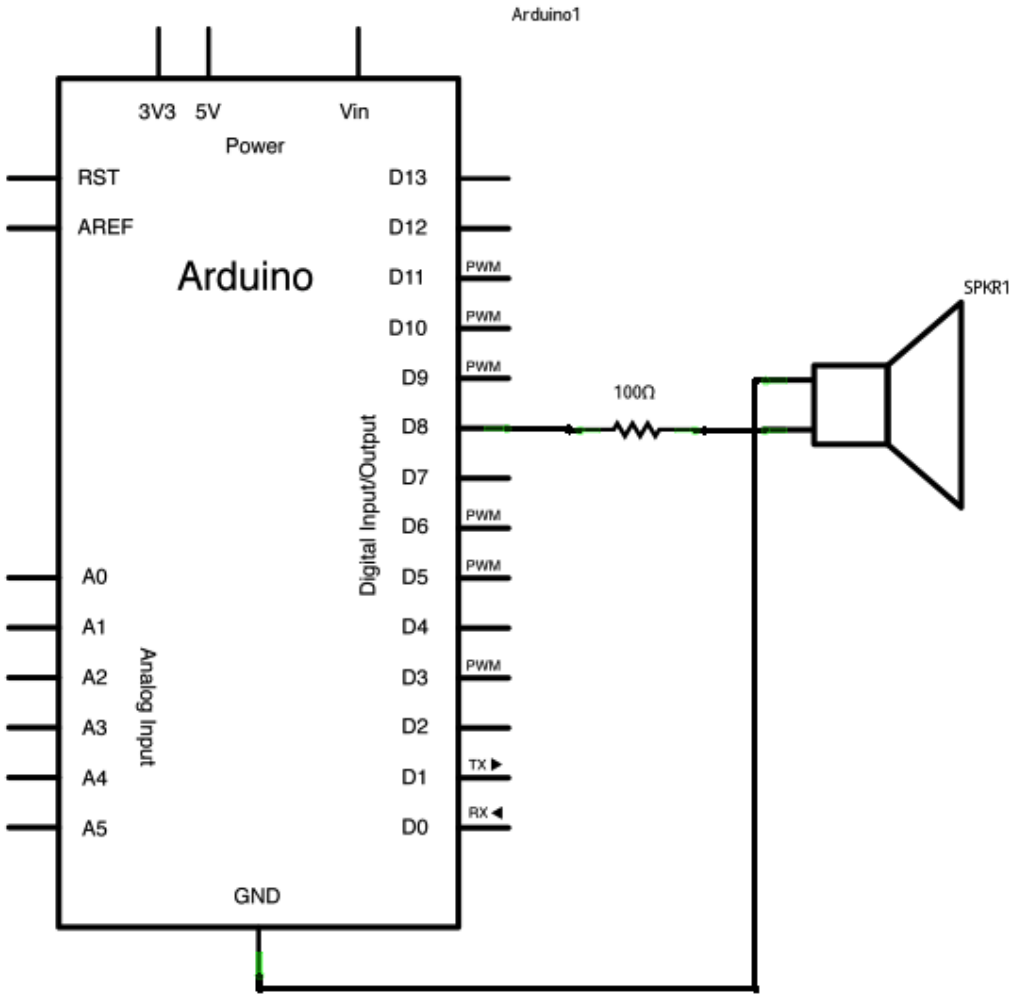
No output pin setup is required.

Signal generation

Example 8: Play a Melody using the tone() function

http://arduino.cc/en/Tutorial/Tone





Signal generation

Example 8: Play a Melody using the tone() function

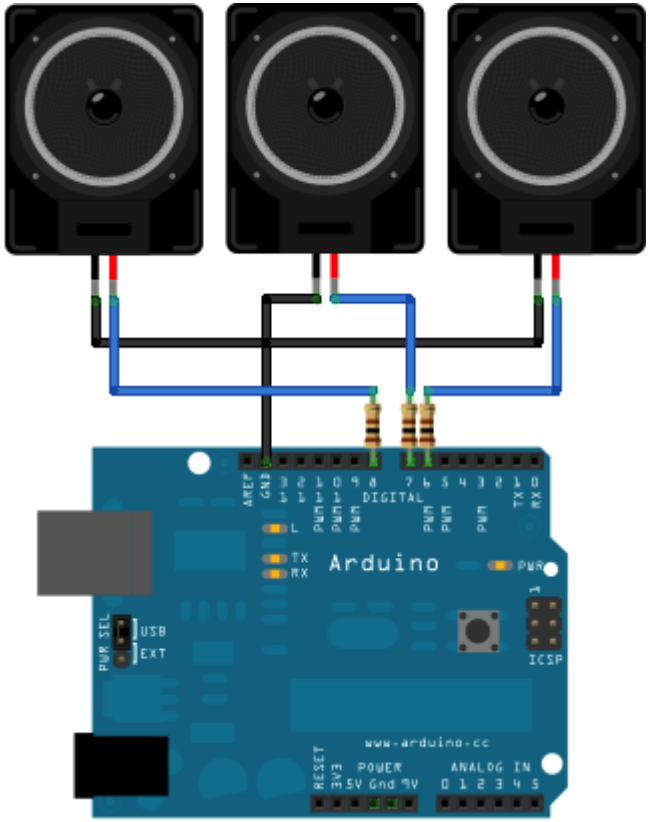
http://arduino.cc/en/Tutorial/Tone

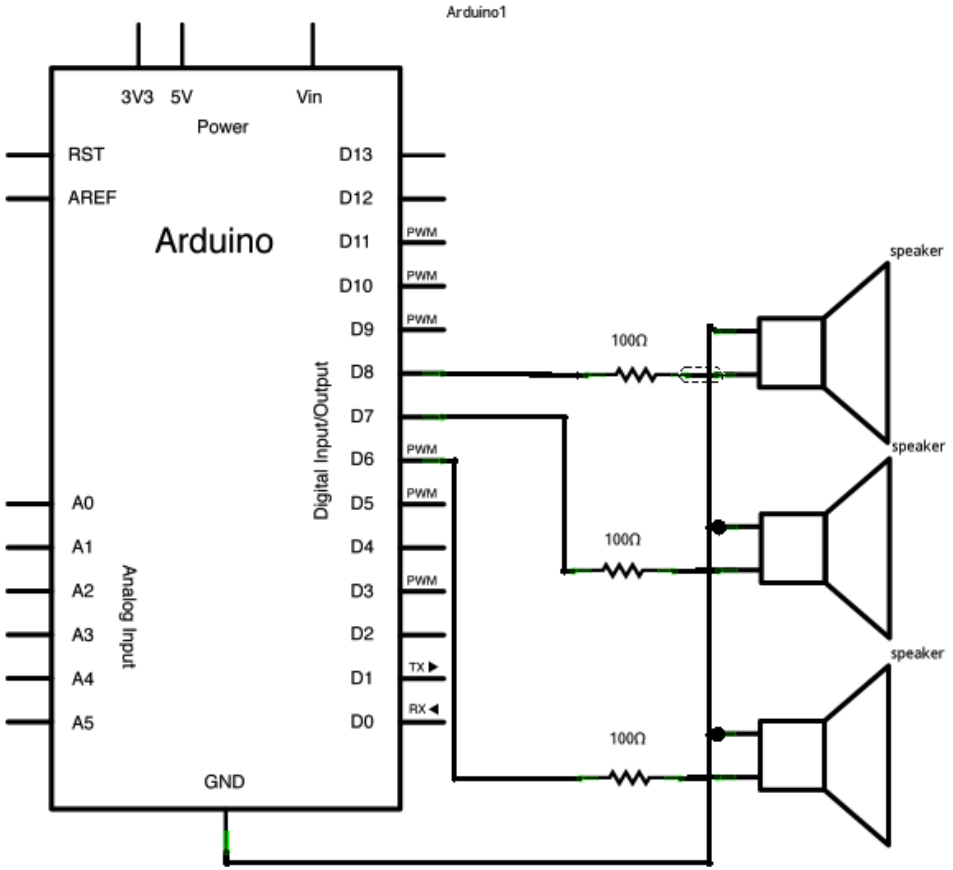
```
#include "pitches.h“ // containes the frequency values for all the notes

// notes in the melody – constant values defining frequency for each used note
int melody[] = {  NOTE_C4, NOTE_G3,NOTE_G3, NOTE_A3, NOTE_G3,0, NOTE_B3, NOTE_C4};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = { 4,8,8,4,4,4,4,4 };

void setup() {
  for (int thisNote = 0; thisNote < 8; thisNote++) { // iterate over the notes of the melody:
    // to calculate the note duration, take one second divided by the note type.
    // e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000/noteDurations[thisNote];
    tone(8, melody[thisNote],noteDuration);

    // to distinguish the notes, set a minimum time between them: note's duration + 30%
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);

    noTone(8); // stop the tone playing for current note
  }
}

void loop() { }   // no need to repeat the melody
```

Signal generation

Example 9: Playing tones on Multiple outputs using the tone() function

http://arduino.cc/en/Tutorial/Tone4





Signal generation

Example 9: Playing tones on Multiple outputs using the tone() function

http://arduino.cc/en/Tutorial/Tone4

Only one pin at a time can generate a signal using tone function. Ton is generated / cycled over the 3 different pins in a sequential order !!!

```
void setup() { }

void loop() {

  noTone(8);              // turn off tone function for pin 8
  tone(6, 440, 200);      // play a note on pin 6 for 200 ms
  delay(200);

  noTone(6);               // turn off tone function for pin 6
  tone(7, 494, 500);       // play a note on pin 7 for 500 ms
  delay(500);

  noTone(7);               // turn off tone function for pin 7
  tone(8, 523, 300);       // play a note on pin 8 for 500 ms
  delay(300);
}
```

Advanced usage of timers

Options: usage of a dedicated library (i.e. Timer1) or configure directly the AVR timer/counter registers (see C5)

- **Timer1 library**: http://playground.arduino.cc/Code/Timer1
  - Functions for the 16 bit Timer 1 (recommended for Arduino UNO.
  - For Arduino Mega, only OCR1A, OCR1B1 are supported by the library and is recommended to use Timer3 library (with the same functions) http://playground.arduino.cc/uploads/Code/TimerThree.zip.

Most important methods of Timer class:

- **initialize(period)** – initialize the timer with 'period' [us]. Period is the time interval in which the timer performs a complete counting cycle.
- **setPeriod(period)** – modifies the period of an already initialized timer.
- **pwm(pin, duty, period)** – generates a PWM signal on 'pin' with the duty cycle value 0 .. 1023 and an optional 'period' [us]. For 'pin' only values at which the Timer/counter outputs are physically connected are allowed: Timer 1 connected at 9 and 10, Timer 3 connected at 2, 3 and 5 for Arduino Mega.
- **attachInterrupt(function, period)** – attaches an ISR 'function' to be called every time when the timer finishes a cycle (OVFi) or at time intervals specified by the optional parameter 'period' (COMPi).
- **detachInterrupt()** – de-attaches the ISR
- **disablePwm(pin)** – deactivates the PWM generation on the specified pin
- **read()** – returns the time interval passed from the last saturation of the counter

Advanced usage of timers

Prescaler / counter increment / max counting interval for a 16MHz (16 bit timer):

In general:

Max Period = (Prescale)*(1/Frequency)*(2^17)
Time per Tick = (Prescale)*(1/Frequency)

Prescaler	**Time per tick**	**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	64uS	8388.608mS

Advanced usage of timers

Example 10 - set a timer of length 100000 microseconds (or 0.1 sec - or 10Hz => the LED will blink 5 times, 5 cycles of on-and-off, per second) (https://code.google.com/p/arduino-timerone/downloads/list)

```
#include <TimerOne.h>
void setup()
{
    pinMode(13, OUTPUT);           // Initialize the digital pin as an output
    Timer1.initialize(100000);     // set a timer (period=0.1 sec / 10Hz
                                   // LED will blink 5 times, 5 cycles of ON-and-OFF, per second)
    Timer1.attachInterrupt( timerIsr );  // attach the service routine here
}

void loop(){
    // Main code loop
    // TODO: Put your regular (non-ISR) logic here
}

/// -------------------------
/// Custom ISR Timer Routine
/// -------------------------
void timerIsr(){
    digitalWrite( 13, digitalRead( 13 ) ^ 1 ); // Toggle LED
}
```

Advanced usage of timers

Example 11 - Sets up PWM output on pin 9 with a 50% duty cycle, and attaches an interrupt that toggles digital pin 10 every half second.

```
#include "TimerOne.h“

void setup()
{
  pinMode(10, OUTPUT);
  Timer1.initialize(500000);            // initialize timer1, and set a 1/2 second period
  Timer1.pwm(9, 512);                   // setup PWM on pin 9, 50% duty cycle
  Timer1.attachInterrupt(callback);  // attaches callback() as a timer overflow interrupt (TOFI)
}

void callback()
{
  digitalWrite(10, digitalRead(10) ^ 1);
}

void loop()
{
  // your program here...
}
```

Advanced usage of timers

Example 12: function setPeriod from Timer1 library (https://code.google.com/p/arduino-timerone/downloads/list)

```
#define RESOLUTION 65536    // 16 bit timer

void TimerOne::setPeriod(long microseconds)
{
long cycles = (F_CPU / 2000000) * microseconds; // the counter runs backwards after TOP, interrupt is at BOTTOM so divide
    microseconds by 2

// check if no. of cycled fits the max. value of the counter
if(cycles < RESOLUTION)             clockSelectBits = _BV(CS10);                // no prescale, full xtal
else if((cycles >>= 3) < RESOLUTION) clockSelectBits = _BV(CS11);               // prescale by /8
else if((cycles >>= 3) < RESOLUTION) clockSelectBits = _BV(CS11) | _BV(CS10); // prescale by / 64
else if((cycles >>= 2) < RESOLUTION) clockSelectBits = _BV(CS12);               // prescale by / 256
else if((cycles >>= 2) < RESOLUTION) clockSelectBits = _BV(CS12) | _BV(CS10); // prescale by / 1024
else        cycles = RESOLUTION - 1, clockSelectBits = _BV(CS12) | _BV(CS10); // request was out of bounds, set as maximum

oldSREG = SREG;                                 // saves status register
cli();                                          // Disable interrupts for 16 bit register access
ICR1 = pwmPeriod = cycles;                      // configure ICR1 register
SREG = oldSREG;                                 // restore SREG (altered by CLI)
TCCR1B &= ~(_BV(CS10) | _BV(CS11) | _BV(CS12));   // erase clock select bits for Timer 1
TCCR1B |= clockSelectBits;                      // reset clock select register, and starts the clock

}
```

Advanced usage of timers

Example 12 – used config. registers (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.

ICR1 is a 16 bit register divided in ICR1H ICR1L. It is used to specify the TOP threshold value up to which the counter is incremented. After reaching the TOP value the counter is reset to 0 after the end of 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	