Lab : standalone AVR uC ATmeg328 – arduino as a programmer

codes (commented/adapted) from: Make AVR – Elliot Willams

materials (digikey):
-breadboard power supply (1738-1057-ND) - to power the uC it needs a 9V battery
-uC atmega238 without bootloader (ATMEGA328P-PN-ND) – standalone uC
-resonator 16MHz (X908-ND) – we use an external clock
-two 22pF capacitor (399-9721-ND) and one 0.1uF (399-13978-1-ND)
-holder with barrel connector for 9V battery (holder (1528-1116-ND)
-arduino R3 (1050-1024-ND) to be used as a programmer
-FTDI chip (1528-1644-ND) this is a USB -UART converter. For serial communication
-LED, 10K resistor, 330 resistor, jumper wires, breadboard, 9V battery
-0.1uF capacitor and 5K resistor for low-pass filter
-47uF and speaker for audio set ups.
-Oscilloscope – voltmeter

topics:

- blink an LED (output)

- push button and LED (input and output)
- Serial communication – python or serial monitor
- using the ADC – free-running mode – multiplexer

- externally triggered interrupt

- timer to measure time

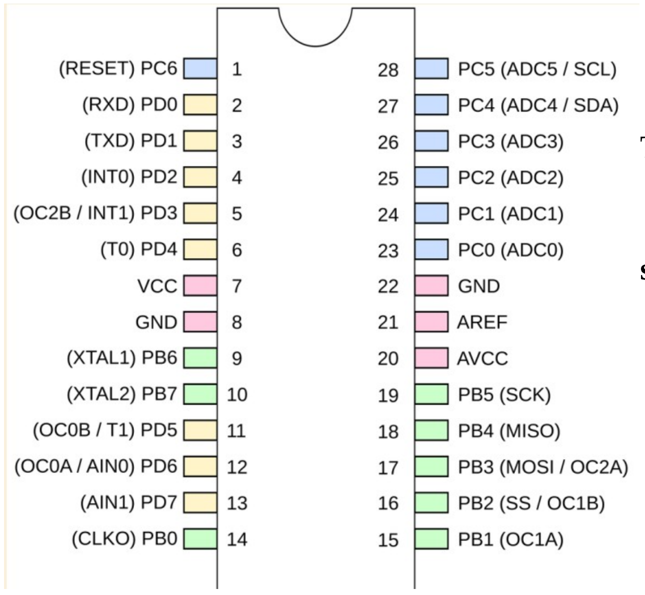
- timer to trigger event after a given time CTC mode

- timer and interrupt

- PWM mode

The uC you will use has no bootloader (the firmware that allows you to upload sketch using an USB cable) so you need to use the SPI protocol with a programmer. Instead of using an atmel programmer we will configure an arduino as a programmer and you will keep this arduino-based programmer to upload sketch into the uC. We will use the arduino IDE to edit our C codes and to compile the code before sending it to the uC standalone through the programmer. The IDE is convenient because it includes the compiler for C (gcc ompiler) and the avrdude (software to talk to the uC). There is no need to make a makefile. The codes are written in C. We don't use the arduino C++ functions.

Some macro used in the codes for bits manipulation are from AVR and are defined in io.h:
https://www.microchip.com/webdoc/AVRLibcReferenceManual/ch20s22s02.html
Macro bit_is_set
Macro bit_is_clear
Macro loop_until_bit_is_set
Macro loop_until_bit_is_clear



The pins and their functions

source: Make: AVR programming – Elliot Williams

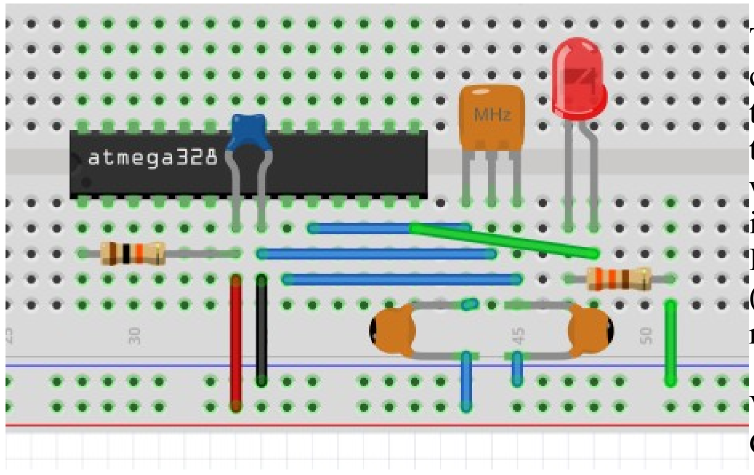
part I: blinking sketch

step1: turn an arduino as a programmer.

1. Plug in your Arduino UNO through USB to your computer. (WITH NOTHING CONNECTED BUT USB)
2. Open de IDE
3. Open > Examples > ArduinoISP
4. Select Arduino UNO from Tools > Board
5. Select your serial port from Tools > Board (Mine is usually COM3, but it may change.)
6. Upload sketch.

Now arduino is a programmer. Keep it as such

Step2: Build your own development board
refer to the mapping of the pins in the front page.



The reset pin (PC6 or pin1 of the chip) is connected through a 10K resistor to 5V so the pin does not float. It is a HIGH unless the chip is reset (to restart a program or when the code is uploaded). When the chip is reset then it is a LOW (done by the uC). If you let the pin float it might reach a LOW (below 2.5V) and the code that is running is randomlu reset.

VCC (pin 7 of chip) is connected to 5V
Gnd (pin 8) is connected to Gnd

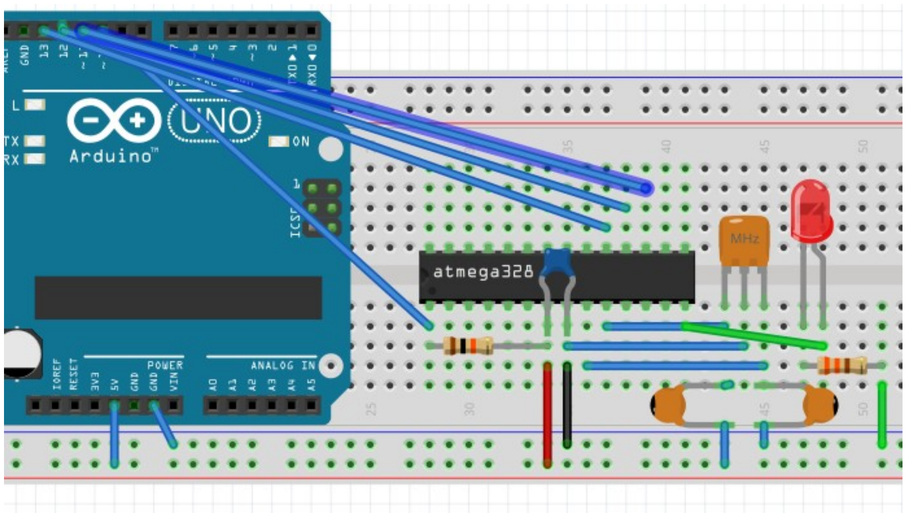
The oscillator (16MHz) has 3 legs. The middle one is connected to Gnd. One leg is connected to XTAL1/PB6 (pin 9) and the other onw to XTAL2/PB7 (pin 10). Those are the pins for external oscillator.

Add 22pF capacitor between XTAL1 and the ground and between XTAL2 and the ground.

Add also a 0.1uF capacitor between Vcc and Gnd to smooth the signal.

The long leg of the LED is connected to PB0 (pin14 of chip). The short leg to a 220 ohms resistor and the resistor to the ground. This is the LED to blink.

Step3: programmer wired through SPI.



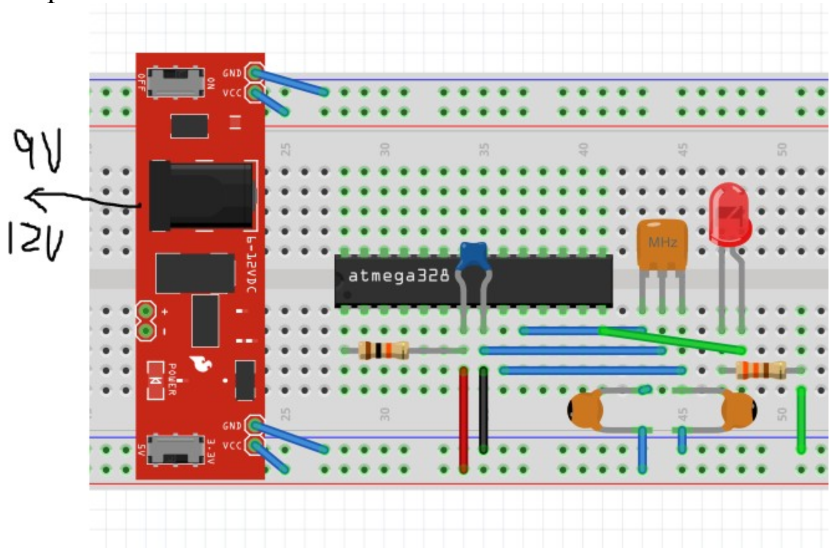
Connect :
arduino/pin 10 to RESET/PC6 (pin 1).
arduino/pin 11 to PB3/MOSI (pin 17).
arduino pin 12 to PB4/MISO (pin 18)
arduino pin 13 to PB5/SCK (pin 19)
power the chip with Arduino. Connect arduino to computer through the USB cable.

Step4: upload the code

download test_avr_blink from my github. Open the code with the arduino IDE.
Select Tools → programmer → Arduino as ISP to program through the arduino-programmer
select Tools → Board → Arduino Duemilanove
shift-click on the upload button (shift-Ctrl-U) to flash your code in the AVR target. If you just click the upload button you get an error.

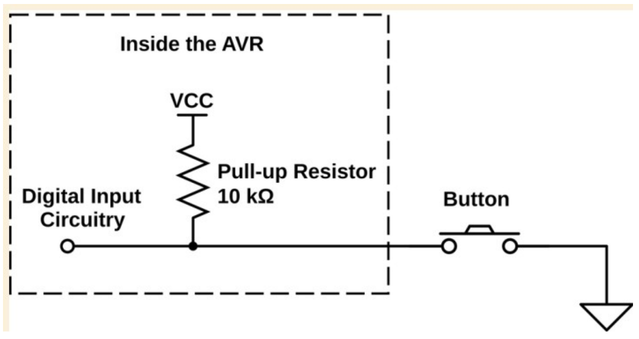
The code is flashed. You can disconnect the programmer. Explain the code.

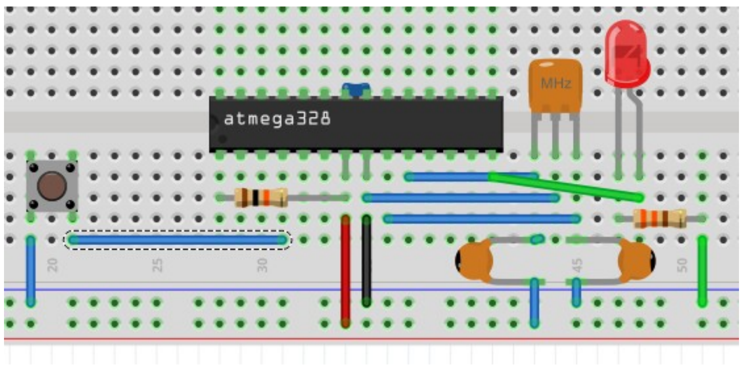
Step5: independent breadboard
Power the AVR with a power breadboard and the LED should blink.



Part II: simple push button

Here we want to have a simple push button connected at pin PD2. The push button is open when not pushed. We activate a pull-up resistor inside the uC so when the button is not pushed PD2 is HIGH. When it is pushed PD2 is low. A LED attached at PB0 is lit when the button is pushed.





Source:
AVR programming (Elliot Williams)

PART III: Serial communication with FTDI chip (USB bridge)

Now building on the previous set up. We will try to send data from the uC to the PC through the USB cable using the two lines Tx/RX connected to the UART module of the uC. For this we will use a FTDI chip that makes the communication possible between the UART module and a driver inside the PC. Instead of using a serial monitor to “catch” the data we will use python to catch the data using a python library. The sketch is test_avr_button_boss.

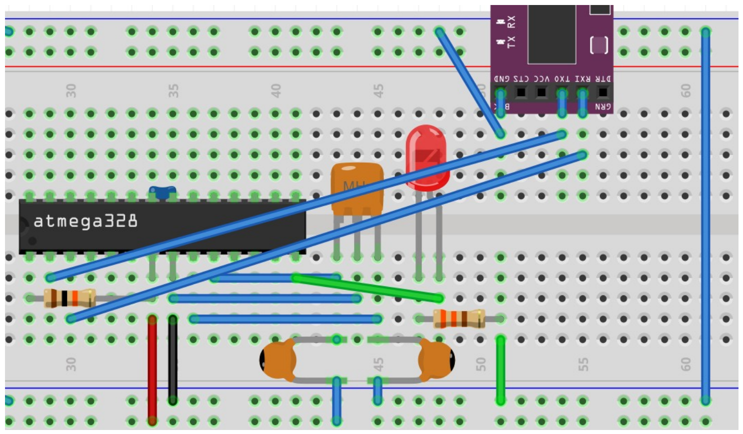
Open the sketch and understand what it does.

Upload the code test_avr_button_boss in the uC using arduino as a programmer (see previous step) using the SPI protocol.

Next we will use a USB bridge (FTDI chip) so the uC can communicate with the PC.

For the connections:

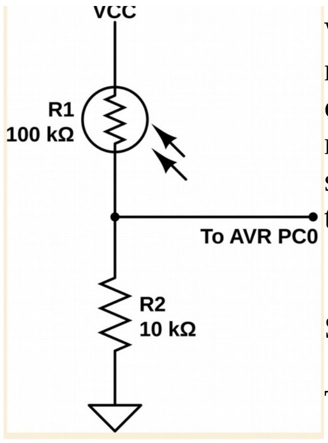
Tx uC or PD1 or pin 3 of chip → Rx FTDI
Rx uC or PD0 or pin 2 of chip → Tx FTDI
Gnd → Gnd



Install the drivers for the FTDI chip. Follow the instructions here:
https://learn.adafruit.com/ftdi-friend/installing-ftdi-drivers
Connect the FTDI to the PC through an USB cable.

Download the python code bossButton and try the set up.

PART IV the 10-bits ADC with hardware registers



We will use a LDR to demonstrate the reading of analog values using hardware registers. The analog reading is done by the pin PC0 (ADC0) / pin 23 of the chip. The pin reads the voltage across the resistor which voltage depend on the resistance of the LDR that is on the intensity of light. The measurement will be sent through Tx/Tx lines (USB) cable to a python program. You can also collect the measurement on a serial monitor

Source: AVR programming – Elliot Williams.

This is a voltage divider. The voltage across R2 = (R2/(R1+R2)) x 5

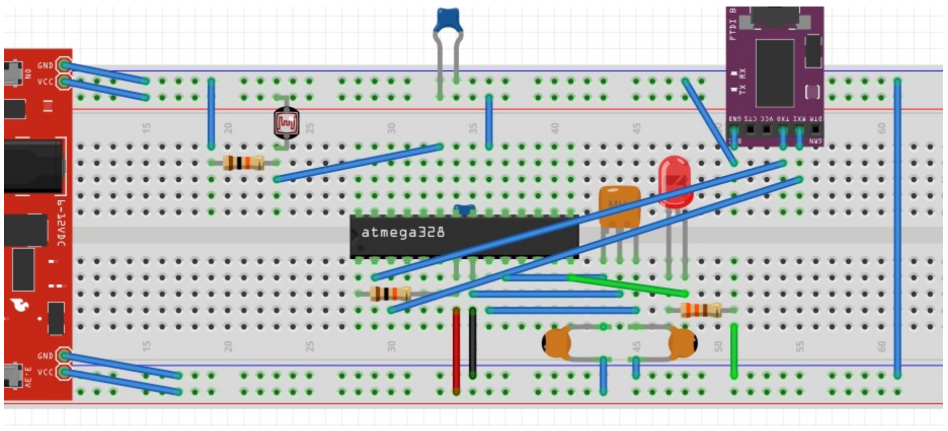
R1 depends on the amount of light collected by the LDR. More light means less resistance. Dark means more resistance (about 100K).

The value of R2 depends on the LDR we use.
We can connect the LDR to an ohmmeter and find the resistance with bright light and in the dark (finger on it). We can use the equation:

R2 = sqrt (R1light x R1dark).

I use a value of 33K.

Build this set up. You also need to provide the ADC some power by connecting AVCC (pin 20 of the chip). You can add a 0.1uF between AVCC and Gnd.



We will read the reading across R2. As the light intensity increases the voltage across R2 increases because R1 decreases. Make all the connections. You can also add 8 LEDs connected to PORT B (don’t forget the resistors to protect them). The LED will give the measurement as a 3 bits number (between 0 and 7) in digital form. So 7 is 5V and all the LEDs are on. 4 in digital form is 2.5V. The measurement of the voltage is sent to the USB cable and can be caught by a python program or can be displayed on a serial terminal.

Upload the sketch test_avr_light_sensor in the uC. Make sure you understand how the registers are tuned. Launch the Python program reading_values_serial. Connect the USB cable to the FTDI chip. You can connect briefly the reset pin to Gnd to restart the program and python should read the voltages

To catch the serial data you can also launch a serial monitor (CoolTerm) and display the values. Reset the uC before the values are displayed.

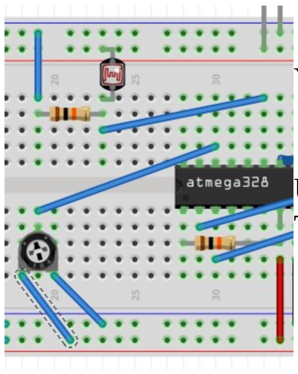
Reading analog values non stop – free running mode

The set up is the same. Upload the code test_avr_light_scope into the uC. Study the code.
Launch the python program serialScope and see if it works.

You can also add the 8 LEDs at PORT B and display the 8 bits number are previously. Try that.

Using the multiplexer of the ADC

The ADC has a multiplexer or multiple ways switch so it can read from different channels (up to 6). Here we are going to read from two channels Channel 0 and channel 3. Channel 0 (PC0/ADC0) is still reading the voltage across R2. Channel 3 (PC3/ADC3) will read from a 10Kpot. So the set up is the same but you need to add a 10Kpot. The wiper is connected to PC3 (pin 26 of the chip).



You don't need the FTDI chip/ There is no data to send to the computer.

Upload the code test_avr_night_light.
Try to understand how the registers are tuned.

PART V Interrupts

It is a section in the hardware so the uC can stop what it is doing and run a short routine. AVR uCs have only one core so it can not do 2 things at the same time. Thanks to the interrupt mode the uC will save what is is doing (execution backed up in the stack pointer) and the program counter is loaded with the routine (Interrupt Service Routine or ISR) address which is stored in the interrupt vector space address.

So we can have external interrupt attached to pin PD2 and PD3 (digital 2 and digital 3). For example, when a voltage (from a button) changes, rises or fails a routine is run (ISR). We can have an external interrupt at the other pins as well but it can only react to a change in voltage. It can not tell a rising voltage from a falling voltage. Special registers have to be tuned in order to use the other pins as interrupt pins. First we enable one of the pin as an interrupt pin (like INIT0 which is PD2). The uC will check its flag constantly. If the flag is on (change in voltage) then the ISR routine is run. This routine does not take any variable and does not return any variable but takes a "vector or pointer" (like INT0_vect) which points to the memory address where the ISR is found. The vector contains a command that makes the uC " jumps" to the location of the ISR. The hardware keeps a " vector space" or map with all the ' vectors' that points to an interrupt handler. This map is found in the datasheet. In the sketch you need to include the interrupt routine: ISR(INIT0_vect) { what to do if interrupt occurs }. When the ISR is run a global flag is turned off so no other interrupt can occur. You need to set that flag on before, in the setting of the registers. When the ISR is done running the global flag is on again and the INIT0 flag (for example) is cleared. You also have to switch a bit in a register to tell the uC to react to a voltage change, voltage rise or fail (only for PD2/INIT0 and PD3/INIT1).

You also have internally trigger interrupts that respond to the internal AVR hardware peripherals. For example, the interrupts that can run the ISR code when the ADC / serial communication peripherals have got new data They can also be triggered by timers. The datasheet gives a map to show the priority of the interrupts. The uC constantly checks the flags of the interrupts. Is it set? If yes it runs the corresponding ISR. Then the uC clears the interrupt flag again. You can clear the flag manually. The uC runs the interrupt with the highest priority according to the map. INIT0 has the highest priority (attached to PD2).

Interrupt triggered externally by a change in voltage at PD0

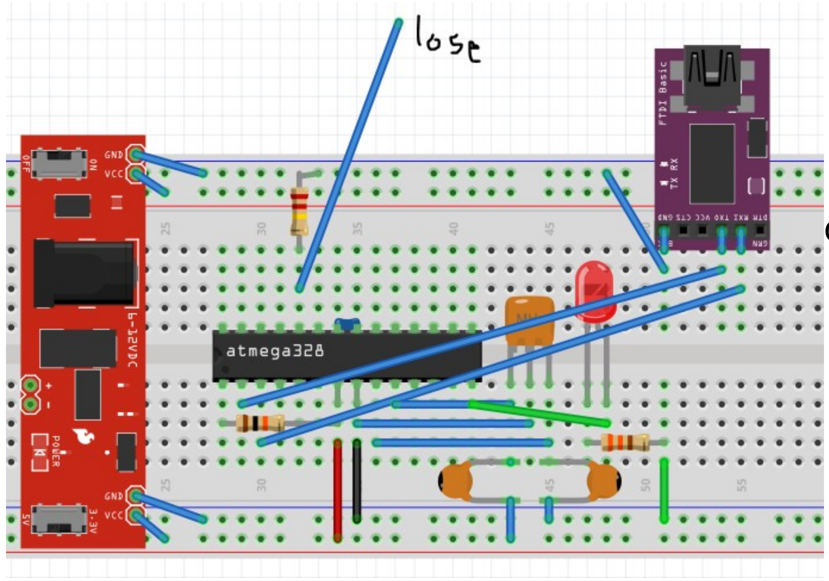
PD2 and PD3 can trigger interrupt if the voltage rises/fails/changes.
A button is attached to PD2 (like previously PD2 – button - Gnd) and we have 2 LEDs. One attached to PB0 and one attached to PB1. (see front page for map of pins). The LED at PB0 blinks all the time (main function) and when we press the button (change in voltage at PD2 pressing or releasing the button) the LED at PB1 turns on or off. So the uC interrupts what it is doing to handle a button press and runs the ISR function.

Upload the code test_avr_interrupt1 in the uC. We don't need the FTDI chip here (USB to Serial hardware). Just 2 LEDS and one button. Try to understand the code.

Interrupt triggered externally at PC1

Interrupts can be triggered at any other pin (other than PD2 and PD3) but it will only detect change in voltage (PD1 and PD2 can tell a rise from a fall). Here we will attach an interrupt (externally triggered) at PC1. PC1 will be connected to a capacitor (here a lose wire) and to a 200 K resistor → Gnd. The wire is a capacitor that can hold charges. If we touch it we increase the capacitance.
The set -up is to detect if the wire (or other capacitor) is touched or not. If it is touched the LEDs at PORT B are turned on.

For 50ms, the idea is to keep charging the capacitor (CP → output HIGH) and let it discharge through 200K (CP1 → input). When the capacitor discharges through the resistor (from HIGH to LOW) the interrupt routine is called (there is a fall in voltage) and we count the number of discharges (ChargeCycleCount) during 50ms. After 50ms we have the number of discharges the capacitor had and this will depend on the capacitance. If the wire was touched we expect a number (chargeCycleCount) smaller than when it is not touched (chargeCycleCount<THRESHOLD). Larger is the capacitance, longer it takes to discharge. THRESHOLD is the number of discharges when the capacitor is not touched. It takes some trial and error to find this number. For that we use the USB serial converter (FTDI) to send the numbers of discharges to the PC. The data can be caught by python of by a serial monitor.



CP1 is also called CAP_SENSOR

upload the sketch test_avr_capacitor2. Try to understand the code and try it.

PART VI timers and more Interrupts

Atmega328 has 3 timers. Timer0 (8-bits), timer1 (16 bits), timer 2 (8 bits). We can use the timers to time event. To start an event when a value is reached (using interrupt). To generate waveforms (modulated or not). An 8-bits timer counts from 0 to 255, overflows then counts again. See appendix B. We can use prescalers to slow down the clock and the time it takes to increase by one bit.

Using timer to time an event – reaction time

We use timer1 which is a 16-bits timer. It can count from 0 to 65,535. We can read the value of the counter in a register (TCNT1) so we can time an event. Here we will time a reaction time. We pick a prescaler of 1028. With a clock of 16MHz, the clock slows down to 15,564Hz. This means that the time between 2 clicks is 64us = 1/16 ms. So when the counter increases by one bit 1/16 ms has elapsed. The counter can therefore count from 0 to 4s.

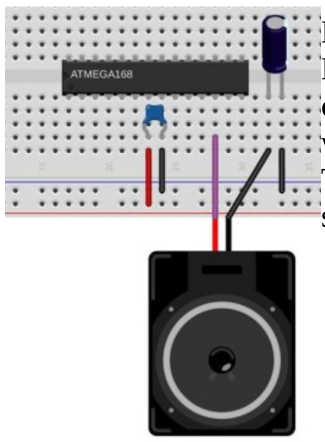
For the set-up, the person needs to push a button when the LEDs are on. Its reaction is displayed through the serial bus. The set up is:
button at PD2 (like before) and LED at PORT B (or just at PB0).

Upload the code test_avr_reaction_time and try to understand it. Try it.

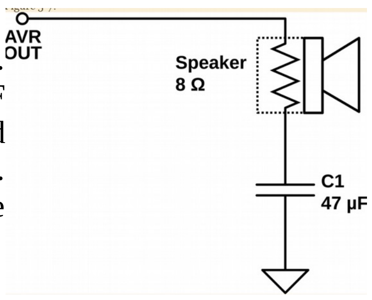
Using the timer to toggle a pin when a value (of the counter) is reached.

CTC mode (clear on compare mode)

Counters have each 2 pins that can be toggled. Timer0 has PD6/OC0A and PD5/OC0B. Timer1 has PB1/OC1A and PB2/OC1B. Timer2 has PB3/OC2A and PD3/OC2B. We can toggle those pins to create square wave with a given frequency (to be tuned with the prescalers). Those pins are also used to create PWM waveforms (next section). To create square wave we use the CTC mode. The counter counts until it reaches a value stored in a register (in OCR0A for timer0 pin PD6) then a pin (PD6) is toggled and the counter is reset. That produce a square wave with a period = twice the value in OCROA. The pin is on then off then on… We will attach a speaker and produce some notes. The speaker is push and pull at a given frequency. The notes are defined by the length of the period (on and off). The precaler is 1028 so a click if of a clock is 64us. To produce the note A0 the counter value to reach is 238. so the time for the counter to go from 0 to 238 is 64us x239=15ms. A period (on and off) is 30ms. Or a frequency of about 30Hz. The notes and their corresponding counter values are in scale8.h. The set up will generate different notes for different times.



Here is the set up. A speaker is attached to PD6. Between the speaker and Gnd there is a 47uF capacitor. The speaker has a 8 ohms resistance and with a 5V output that would mean too much current. The capacitor block the DC current but let go the small AC current.

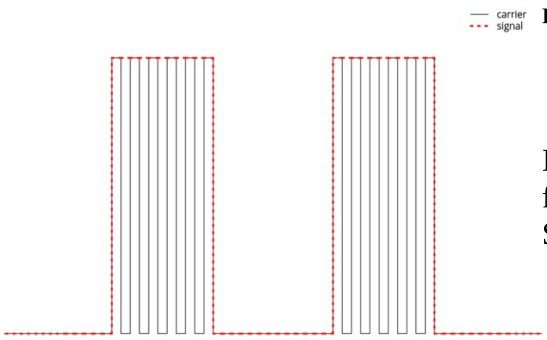


Upload the code test_avr_counter_waveform and see if you can understand it.

More on the CTC mode here: http://maxembedded.com/2011/07/avr-timers-ctc-mode/

Using interrupt when a value in the counter is reached. CTC mode with interrupt

The idea is to create a radio signal that can be picked up at a station. To create a radio signal we need a carrier (1MHz signal) that can be picked up by a radio receiver and the signal (a note like A2) that modulates the volume of the carrier.

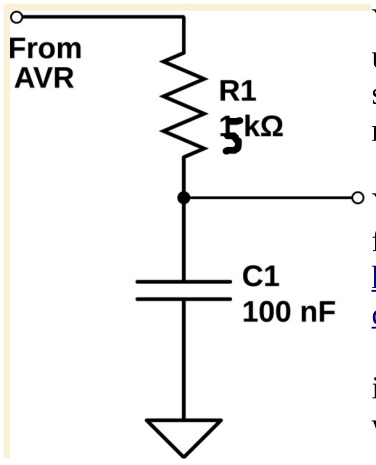


If we want the note A2 the carrier will be modulated at a frequency of 330Hz.
Source picture: Elliot Williams. Programming AVR.

To produce the carrier we will toggle the pin PD5 attached to timer 0 (like previously). The pin will toggle at a frequency of 1Mz. The clock is 16Mhz. The time between each click is 0.063us. If the value of the counter to reach is 7 then the period is 8 x 2 x 0.063us= 1us which is a frequency of 1MHz. The mode is CTC again. An antenna (wire) is connected to PD5. If the direction is 1 then the carrier is emitted. If the direction is 0 (input) the carrier can not be emmited.

In addition timer 1 (16-bits) will trigger an interrupt when a value in its counter is reached (stored in 0CR1A). The interrupt turns on and off PD5 thus modulating the carrier. The frequency at which the modulation occurs will define the pitch of the note. The timer 1 can count to 65,535. If the value of the counter to reach is 1487 and if the clock runs at 1MHz then we get about a frequency of 330Hz which is the note A2. (1/1000,000 x 1488 x 2 for the period) . This was the original code with a clock of 8MHz and a prescaler of 8. But we use an external clock of 16MHz so we will multiply the original pitch by 2 to get the right frequency of the note.

Upload test_avr_Amradio and try to understand it. You need to plugg a wire at PD5/OCOB. This is your antenna.



You can check the AM signal with a radio receiver (tuned at 1Mhz). If you use an oscilloscope, it will pick up only the 1MHz signal and not the AM signal. To remove the 1Mhz signal, you can use a low pass filter. PD5 – resistance 5K – cap 0.1uF – Gnd and read the signal across the capacitor.

You can check that the A0 note signal has a frequency of 84Hz. (the cut off frequency is about 300Hz according to the calculator: http://www.learningaboutelectronics.com/Articles/Low-pass-filter-calculator.php)
You can change the resistor to change the cut off frequency. For example if you want a cut off frequency of 20,000Hz (max frequency we can detect with our hears) then the resistor is about 80 ohms (see calculator).

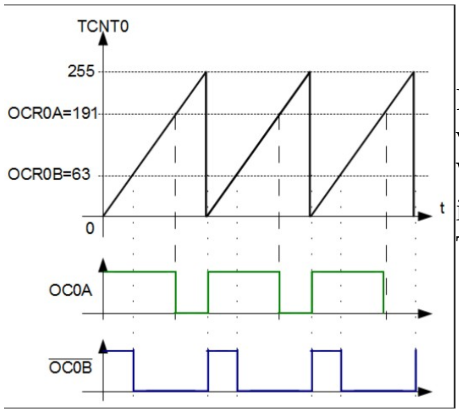
Source picture: MAKE AVR – Elliot Willams

PART VII using the timers to generate PWM waveforms to simulate analog values

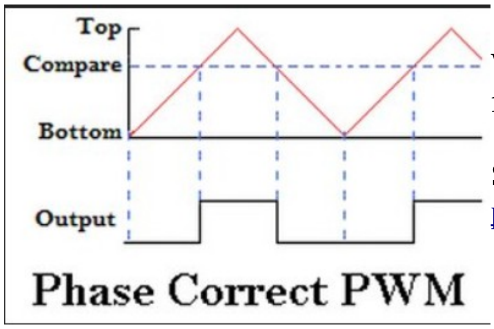
We can use the counter to generate square waves with different duty cycle. If a square wave is 20%on and 80% off, its duty cycle is 20% and a voltmeter reads the average of 0.2 x 5V= 1V. We can simulate analog values this way given than the sensor does not react quick enough and can only see “ averages”.

See the tutorial : https://learn.sparkfun.com/tutorials/pulse-width-modulation/all

To generate those signals we use a timer and a value (stored in a register OCR0A) to compare the counter to. When this value is reached a pin attached to the timer (PD5 for example) is is toggled. There are different modes but below is an example with timer 0. See appendix B to use PWM with the arduino IDE.



For example timer 0 counts from 0 to 255 (the seesaw) and the value to compare is stored in the register OCR0A. Say 191. When the counter is above 191 the waveform is LOW otherwise it is HIGH. 191 is 75% of 256 so a voltmeter reads about 3.75 V. This is called the fast PWM.



We can use the mode phase correct PWM and the counter goes from 0 to 255 and 255 to 0 for one cycle of the waveform.

See http://maxembedded.com/2011/08/avr-timers-pwm-mode-part-i/

Simple code to simulate analog output with PWM waveforms

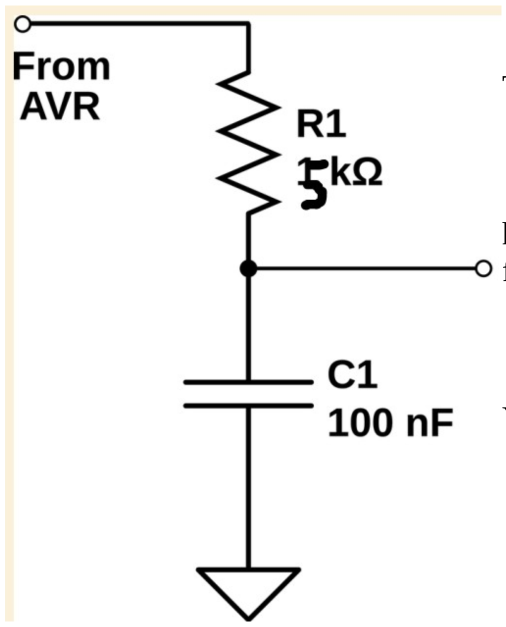
We will try a simple code to change the brightness of 3 LED. The LED are attached to PB1 (can be toggled by timer1), PB2 (timer1) and PB2 (timer2). We can try both phase correct and fast PWM to change the brightness of the LED. The number to reach is in between 0 and 255. 255 is 5V.

Upload the code test_avr_pwm1 and connect the LED. You can change the brightness to see the difference. Instead of LEDs you can try a motor. Try both fast PWM mode and phase correct mode. Use an oscilloscope to see the difference. Use different prescaler. By default the frequency of the PWM waveform is 490Hz (prescaler of 64). You can increase the prescaler and change the PWM frequency.

Sine wave with PWM

This is a program to make a sine wave at PB1 using PWM waveforms. We use 100 samples to create the sine wave and wait for 50us between outputing the value. So the period of the sine wave is 50us x 100. The frequency is 200Hz. Upload the program and try it: test_avr_pwm3

We can not see the sine wave with an oscilloscope because we see the pulses. To remove the pulses we can use a low pass filter and keep the 200Hz signal.



This one cut anything above 320Hz.

http://www.learningaboutelectronics.com/Articles/Low-pass-filter-calculator.php

You can try to make ramp signals.