HOCHSCHULE
RHEIN-WAAL
Rhine-Waal University
of Applied Sciences

LABORATORY

Microcontroller

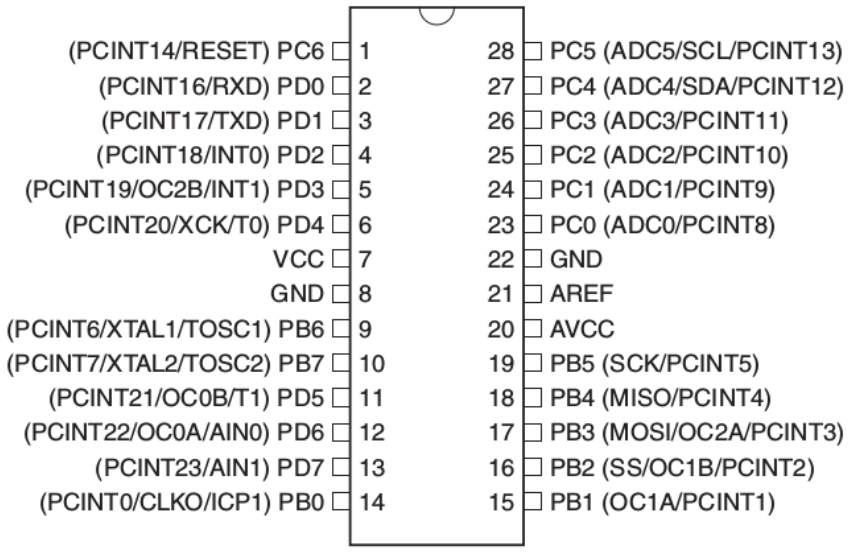
3

EXPERIMENT:

Timer and Counter

1 System clock

The processor on the MyAVR board uses an external clock source of 8 MHz (it's a crystal oscillator, connected to the pins 9 and 10, see Figure 1). This clock is used to synchronize the internal processes. All microcontrollers use clocks. A basic instruction is processed when a tick from the clock passes. Just like these programs we are writing, as clock ticks pass, instructions are processed in time with the ticks of the clock.



External crystal oscillator is connected to PIN 9 and 10.

Figure 1: Atmel Mega88PA Microcontroller

2 Timer and Counter

The timer and counter functions in the microcontroller simply count in sync with the microcontroller clock. There are 3 Timers/counter in the AtMega88PA. Timer0 and Timer2 can only count up to 255 (8-bit counter), Timer1 can count up to 65535 (16-bit counter). That's far from the 8000000 ticks per second that the AVR on our board provides. The microcontroller provides a very useful feature called prescaling. Prescaling is simply a way for the counter to skip a certain number of microcontroller clock ticks. The AVR microcontrollers allow prescaling numbers of: 1, 8, 64, 256 and 1024. If –for example- 64 is set as the prescaler, then the counter will only count every time the clock ticks 64 times. That means, in one second (where the microcontroller ticks 8000000 million times) the counter would only count up to 125000.

Timers have a register for control and another register that holds the count number. The control register contains some switches to turn on and off features. One of the features is which prescaling to select. The control registers are called TCCR0A and TCCR0B (for Timer0), TCCR1A and TCCR1B (for Timer1). Here's a snippet from the datasheet (Page 133 and 134), see Figure 2 and Figure 3. The bits 0 to 2 determine the Timer1 prescaler.

The TCCR0 is the 8-bit control register for Timer0. There are only 8 switches to turn on and off. TCCR1 is 16-bit, so it has 16 switches to turn on and off, but it comes in two 8-bit registers labeled A and B (TCCR1A and TCCR1B). The register, that holds the count is called the TCNT register. And there is an 8-bit version (TCNT0) and a 16-bit version (TCNT1). The TCNT1 register actually gets its number from two other 8-bit

Timer/Counter 1 Control Register B – TCCR1B

Bit	7	6	5	4	3	2	1	0	
	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	

Figure 2: Timer/Counter1 Control Register B

Table 16-5. Clock Select Bit Description

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.

Figure 3: Clock Select Bit Description

registers (TCNT1H and TCNT1L, Low and High byte) to create a complete 16-bit number. If we want to stick to the example and turn on the Timer1 with a prescaler of 64, we would set CS11 and CS10 in the TCCR1B register to 1. This is done in the same way, like setting bits in the PORT register (without affecting the other bits), using the logic OR (bitwise OR) operator |. The counter value (which can be read by using "TCNT1") would then be incremented every 8 microseconds, Figure 4 explains this visually. We can not only read the counter value in TCNT, we can also write (e.g. setting it back to zero).

2.1 Flashing LED

Task 1:
Use the template files (in "Templates/Experiment3"), modify the init function in such a way, that Pin 0 on PortB is used as the only output. There will be no input necessary, at the moment.
Connect an LED to this pin (Figure 5) and make it blink every second by using Timer1.

For this, you can use a simple compare (if counter value is equal or greater to.... do something), to trigger the action on the outputs.

2.2 Audio output

Task 2:
Generate a 300 Hz signal (50% duty cycle) using the same setup (Timer1, LED).

Maybe it is necessary to change the prescaler. The LED should now be 50% dimmer than before. You are applying a 300 Hz software PWM signal (with fixed 50% duty cycle) to the LED, see Figure 6. The Atmel Mega88PA has an internal hardware to generate PWM signals, do not use this here. The LED only appears to be dimmer, because the human eye cannot recognise the LED being turned on and off very fast. Now connect the piezo buzzer instead of the LED, see Figure 7. You should hear a 300 Hz (not very clear) tone.

If all at this point is complete and works fine, inform the staff. You must be able to show and explain all steps you have done. The next steps are optional.

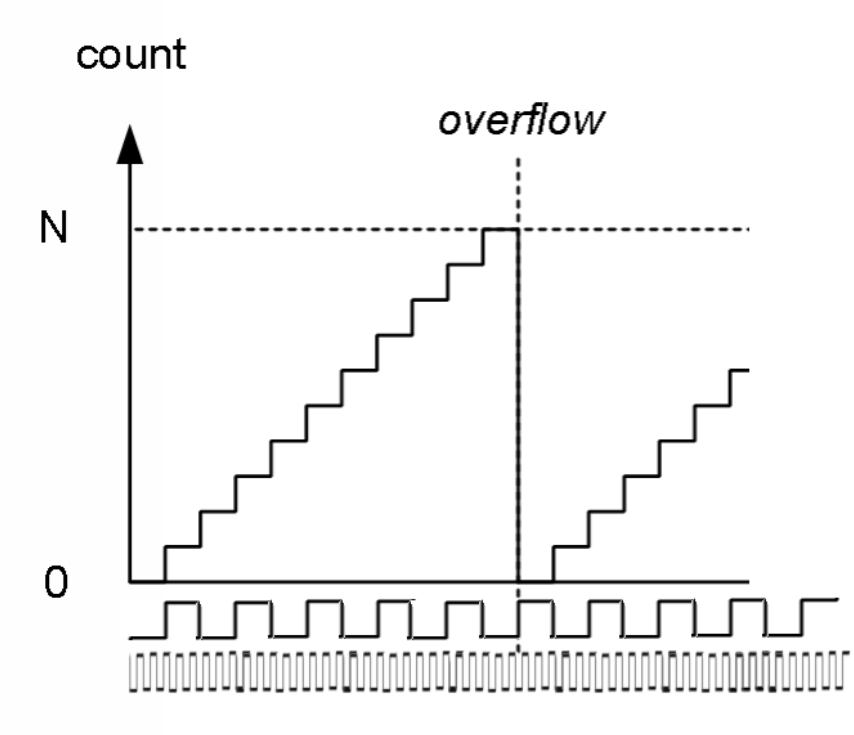


Figure 4: Counter overflow

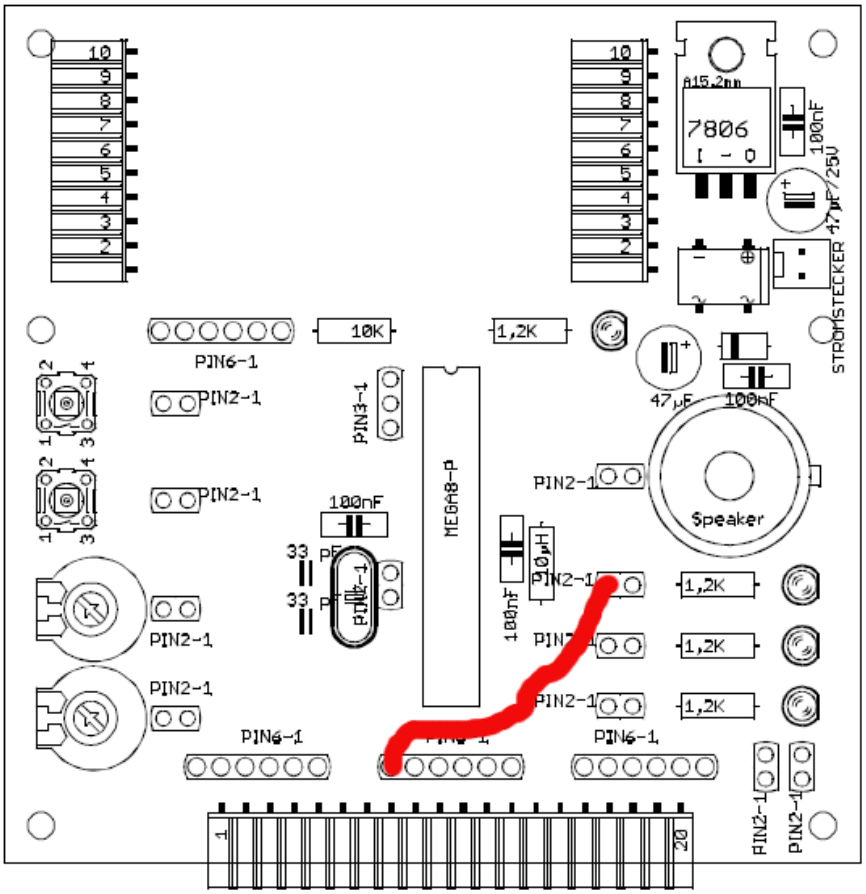


Figure 5: LED connection on the board

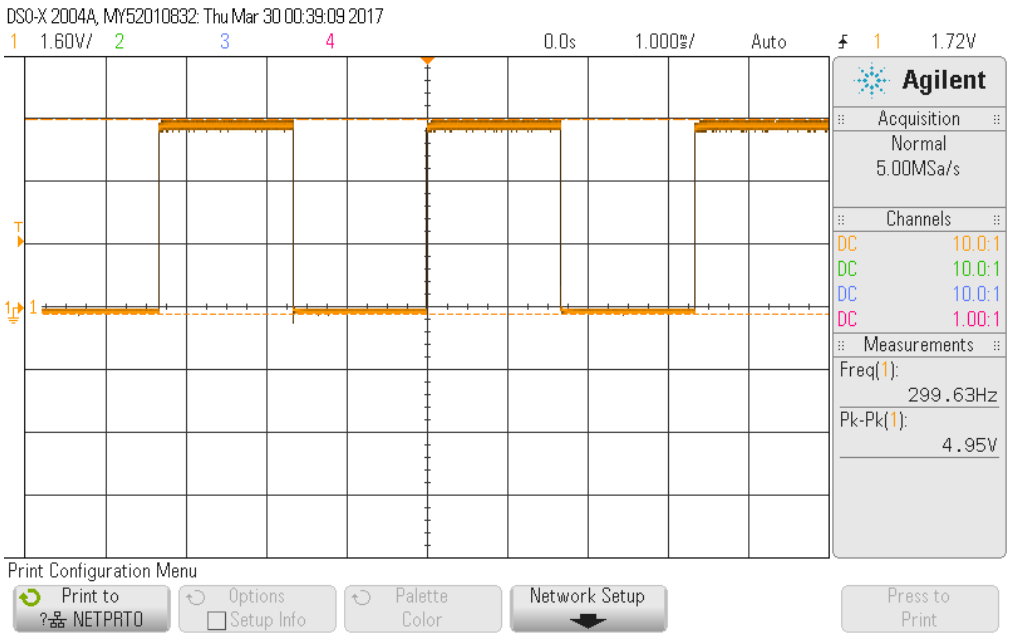


Figure 6: PWM signal 300 Hz captured with oscilloscope

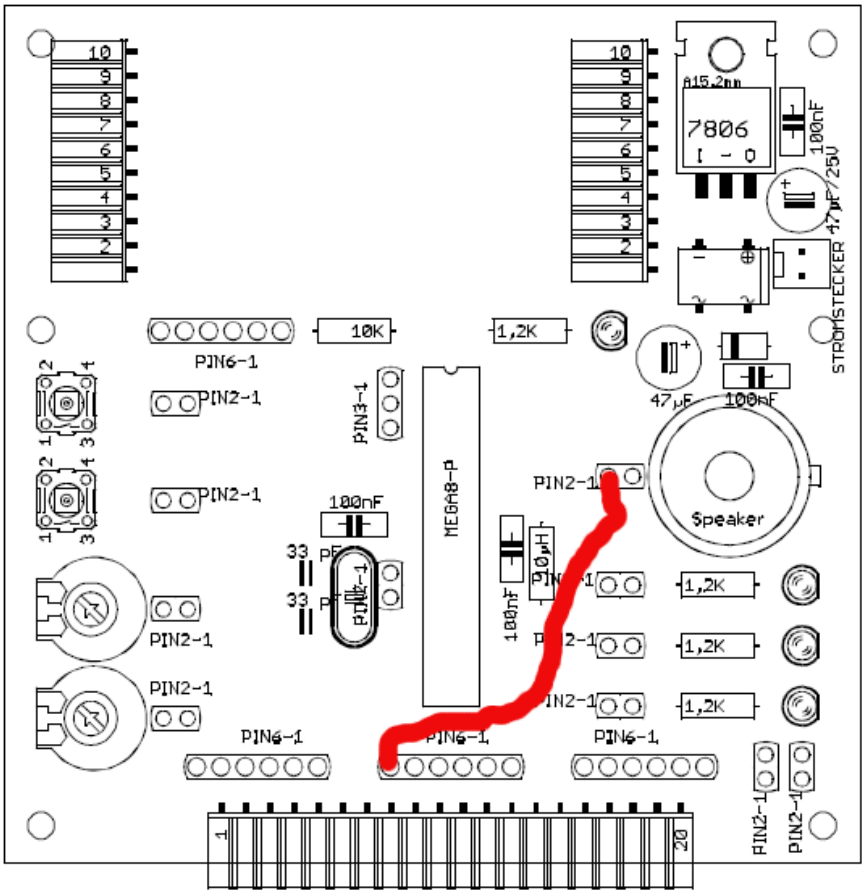


Figure 7: Audio output connection on the board

3 Using two Timer/Counter

Task 3 (Optional):
Use two Timer/Counter to play a song. One timer is to generate the tone pitch, and the other for the tone length.

The tones for ”silent night” are:

Tone	**Text**	**Duration(sec)**
G	Silent	0.34375
A		0.09375
G		0.21875
E	Night	0.71875
G	Holy	0.34375
A		0.09375
G		0.21875
E	Night	0.71875
G1	All	0.46875
G1	Is	0.21875
B	Calm	0.71875
C	All	0.46875
C	Is	0.21875
G	Bright	0.71875

Frequencies:

- A = 440 Hz
- B = 492,88 Hz
- C = 523,25 Hz
- E = 329,63 Hz
- G = 392 Hz
- G1 = 783,99 Hz

Here is a hint to make eventual changes more easy:

```
#define A 36
#define B 32
#define C 30
#define E 47
#define G 40
#define G1 20
#define END 1


uint8_t uchNote[] = {G, A, G, E, G, A, G, E, G1, G1, B, C, C, G, END}
```