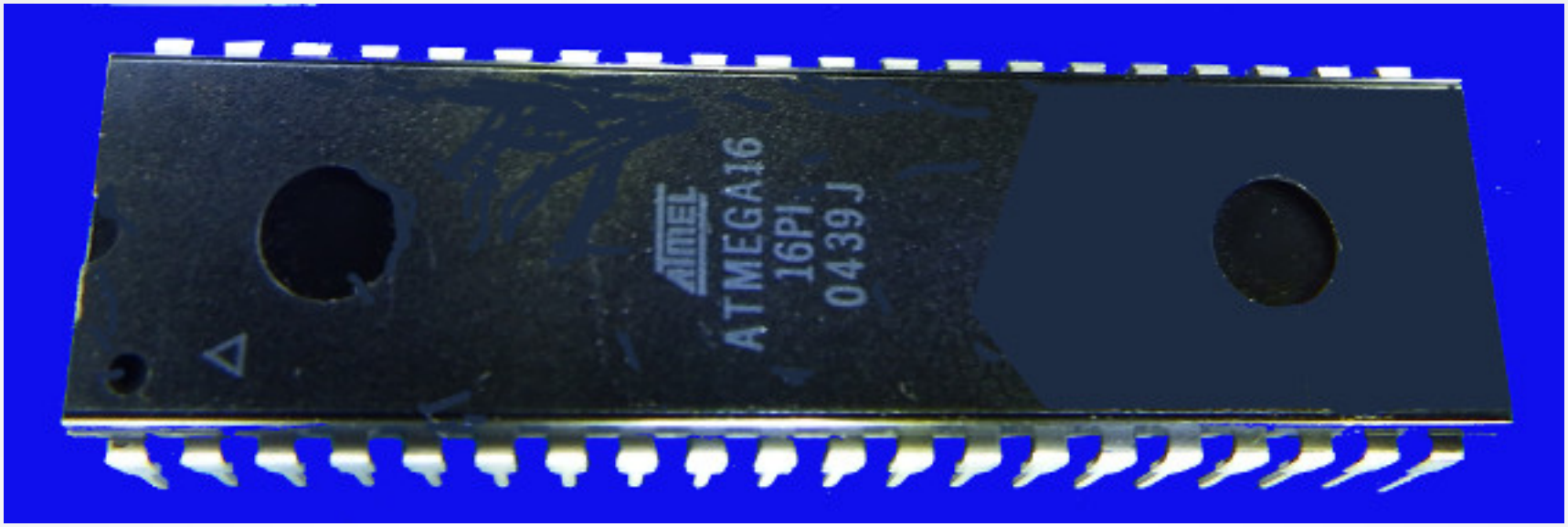


Lecture 6: Timer/Counter

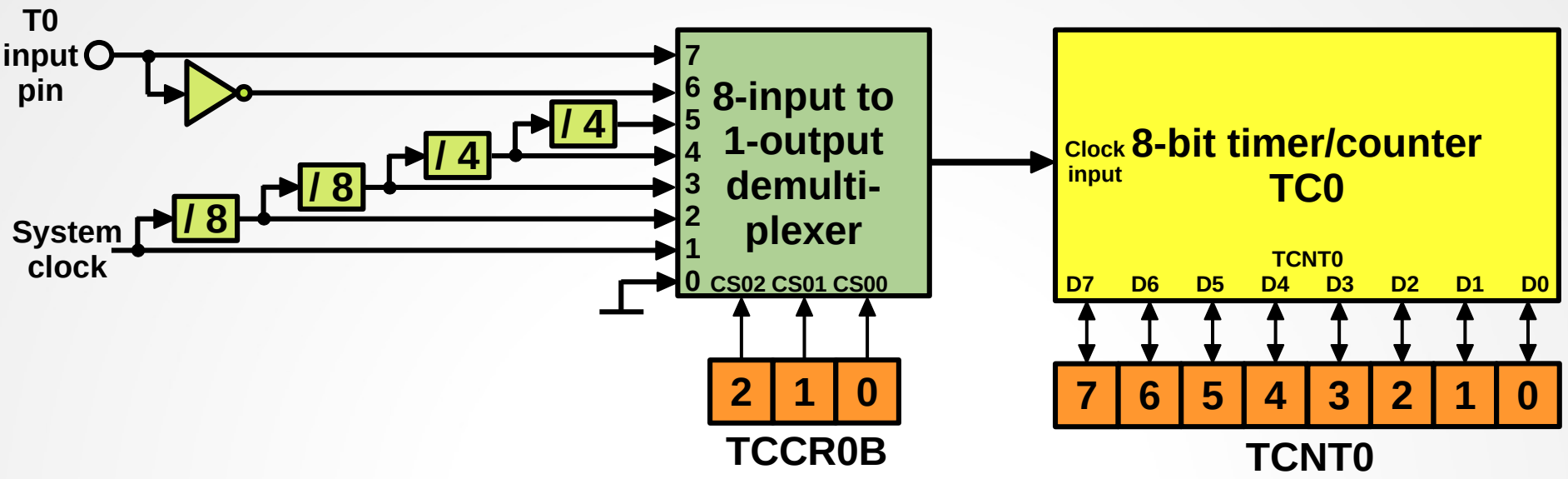
Hardware, Internals and Programming of AVR Microcontrollers in Assembler

by
Gerhard Schmidt
Kastanienallee 20
D-64289 Darmstadt

Timers/Counters in AVRs

- **Doing timing tasks with counting loops is not very effective: the AVR can do nothing else than counting and counting can better be done with timers.**
- **All AVRs have at least one timer on board that can do that.**

8-bit timer/counter with prescaler



- 8-bit timers/counters count from 0 up to 255, then restart.
- They are switched on by setting bits 0 to 2 in the port register TCCR0B to a non-zero value and off by setting 0 to.
- The timer can be clocked by the prescaled system clock (divided by 1, 8, 64, 256 or 1,024), the counter can be clocked by positive or negative pulses on the T0 input pin (pin 10 in ATtiny24‘s PDIP package).

Achievable TC frequencies

- **The following timer clock frequencies can be achieved with different prescaler values:**

Prescaler	System clock Internal RC oscillator			System clock crystal oscillator		
	1 MHz	**4 MHz**	**8 MHz**	**2.048 MHz**	**3.2768 MHz**	**4.194304 MHz**
0	Timer off					
1	1 MHz	4 MHz	8 MHz	2.048 MHz	3.2768 MHz	4.194304 MHz
8	125 kHz	500 kHz	1 MHz	256 kHz	409.6 kHz	524.288 kHz
64	15.625 kHz	62.5 kHz	125 kHz	32 kHz	51.2 kHz	65.536 kHz
256	3.90625 kHz	15.625 kHz	31.25 kHz	8 kHz	12.8 kHz	16.384 kHz
1,024	976.563 Hz	3.90625 kHz	7.8125 kHz	2 kHz	3.2 kHz	4.096 kHz

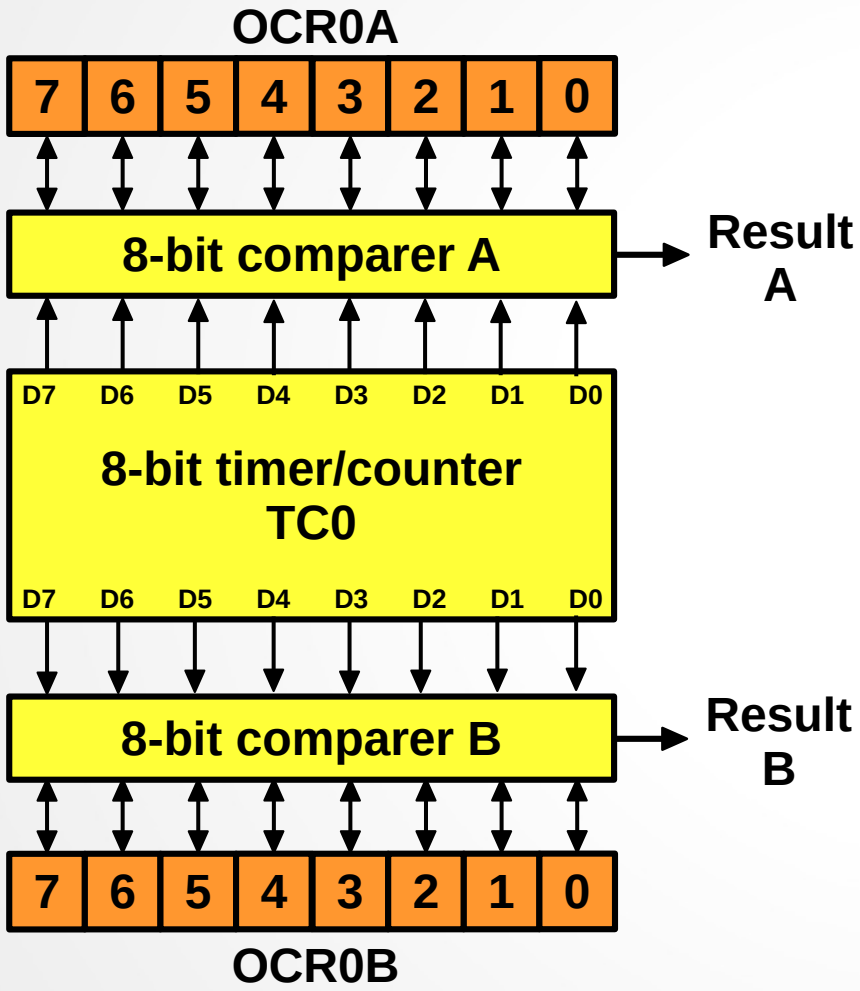
Achievable frequencies TC overflow

- **The timer overflows after 256 TC clock cycles. The following shows the frequencies at which this happens:**

Prescaler	System clock Internal RC oscillator			System clock crystal oscillator		
	1 MHz	4 MHz	8 MHz	2.048 MHz	3.2768 MHz	4.194304 MHz
0	Timer off					
1	3.90625 kHz	15.625 kHz	31.250 kHz	8 kHz	12.8 kHz	16.384 kHz
8	488.28125 Hz	1.953125 kHz	3.90625 kHz	1 kHz	1.6 kHz	2.048 kHz
64	61.03515625 Hz	244.140625 Hz	488.28125 Hz	125 Hz	200 Hz	256 Hz
256	15.2587890625 Hz	61,03515625 Hz	122.0703125 Hz	31.25 Hz	50 Hz	64 Hz
1,024	3.814697265625 Hz	15.2587890625 Hz	30.517578125 Hz	7.8125 Hz	12.5 Hz	16 Hz

Comparers

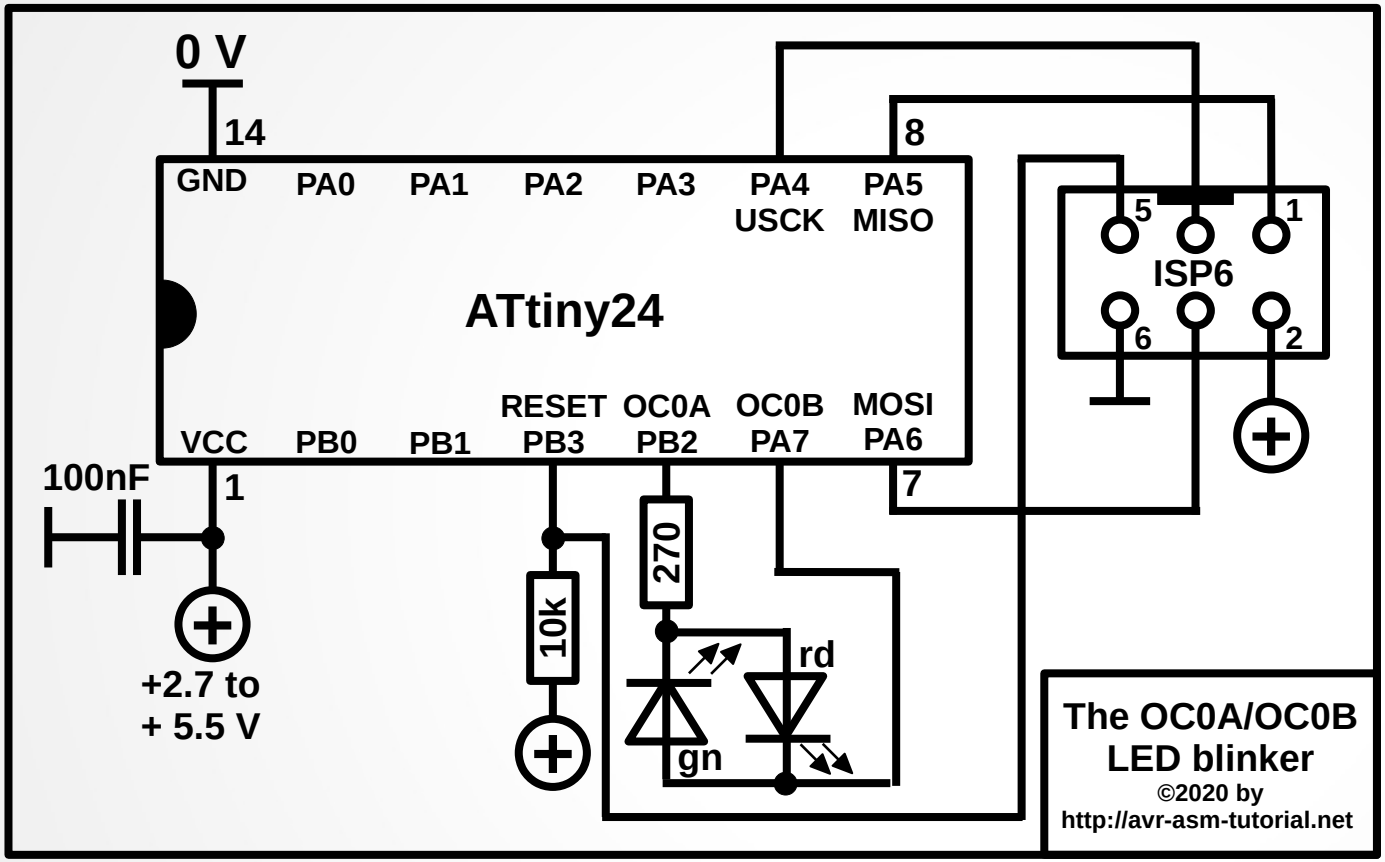
- Most timers/counters have two comparers on board. Those compare the timer/counter‘s state continously with compare match values that were written to the comparer‘s port register.



- In the timer/counter clock cycle following the compare match, one or two output pins can either be set, be cleared or be toggled.
- Most timers/counters have two comparers on board. Those compare the timer/counter‘s state continously with compare match values that were written to the comparer port registers.

The hardware of the toggles

- **Build the following hardware with a red/green Duo-LED:**



Toggling OC0A and OC0B

- The following source code
  - sets the comparer port registers to 255,
  - enables toggling of the OC0A and OC0B output pins, and
  - starts TC0 with a prescaler value of 1,024.

```
; Timer toggles OC0A and OC0B
      ; Configure pins as output (switching output driver on)
      SBI DDRB, DDB2 ; The OC0A output pin is PB2, set as output
      SBI DDRA, DDA7 ; The OC0B output pin is PA7, set as output
      SBI PORTA, PORTA7 ; Set OC0B = PA7 high
      ; Set output pins to toggle mode
      LDI R16, (1<<COM0A0) | (1<<COM0B0) ; Toggle output pins A and B
      OUT TCCR0A,R16 ; to TC0 control register A
      ; Start TC0 in normal mode with its prescaler
      LDI R16, (1<<CS02) | (1<<CS00) ; Prescaler to 1,024
      OUT TCCR0B, R16 ; to TC0 control register B
      ; Endless loop
Loop:
      RJMP Loop
```

Hints to the source code

- **Please refer to the device‘s data book, chapter 11.9 for the port register descriptions.**
- **The formulation (1<<COM0A0) | (1<<COM0B0) works as follows:**
  1. **The assembler picks the 1 and shifts it left (<<) for COM0A0 times. As COM0A0 is 6 (taken from the def.inc file), the 1 is shifted 6 times to the left and 0b0100,0000 results for the first bracket.**
  2. **Then the assembler picks the other 1 and shifts it left (<<) for COM0B0 times. COM0B0 is 4, the result is 0b0001,0000 for the second bracket.**
  3. **The | means „Binary OR“ and combines both with or‘ing them bit-by-bit. The result is 0b0101,0000.**
  4. **The reason why this formulation is used is that COM0A0 is simpler to remember than 0b0100,0000 and the position of the two bits is always correct, even if moved to somewhere else inside this port register.**

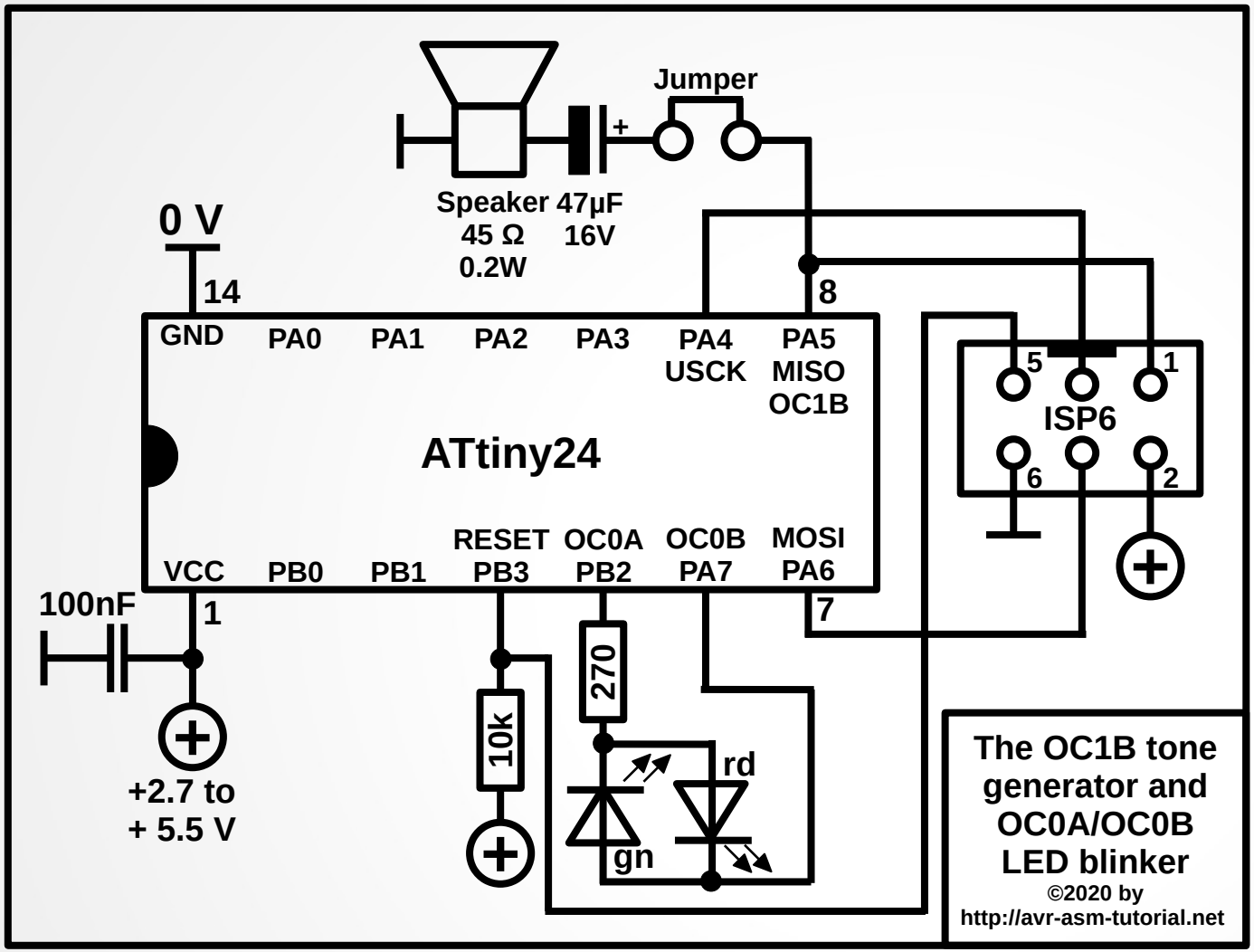
Further hints on the source code

- **The reason why PA7 on start-up is set is as follows:**
  - **The port register PB2 (=OC0A) is cleared on RESET.**
  - **The same with PA7 (=OC0B). As both are zero on start-up, the LED would be off.**
  - **As both toggle their PORT state after 256 TC clocks, they both are set to one, the LED still will be off.**
  - **By setting the PA7 port bit, the state of this is always the opposite of PB2, and the LED is always on. The two I/O-pins reverse the voltage of the LED on every toggle.**

The 16-bit counter TC1

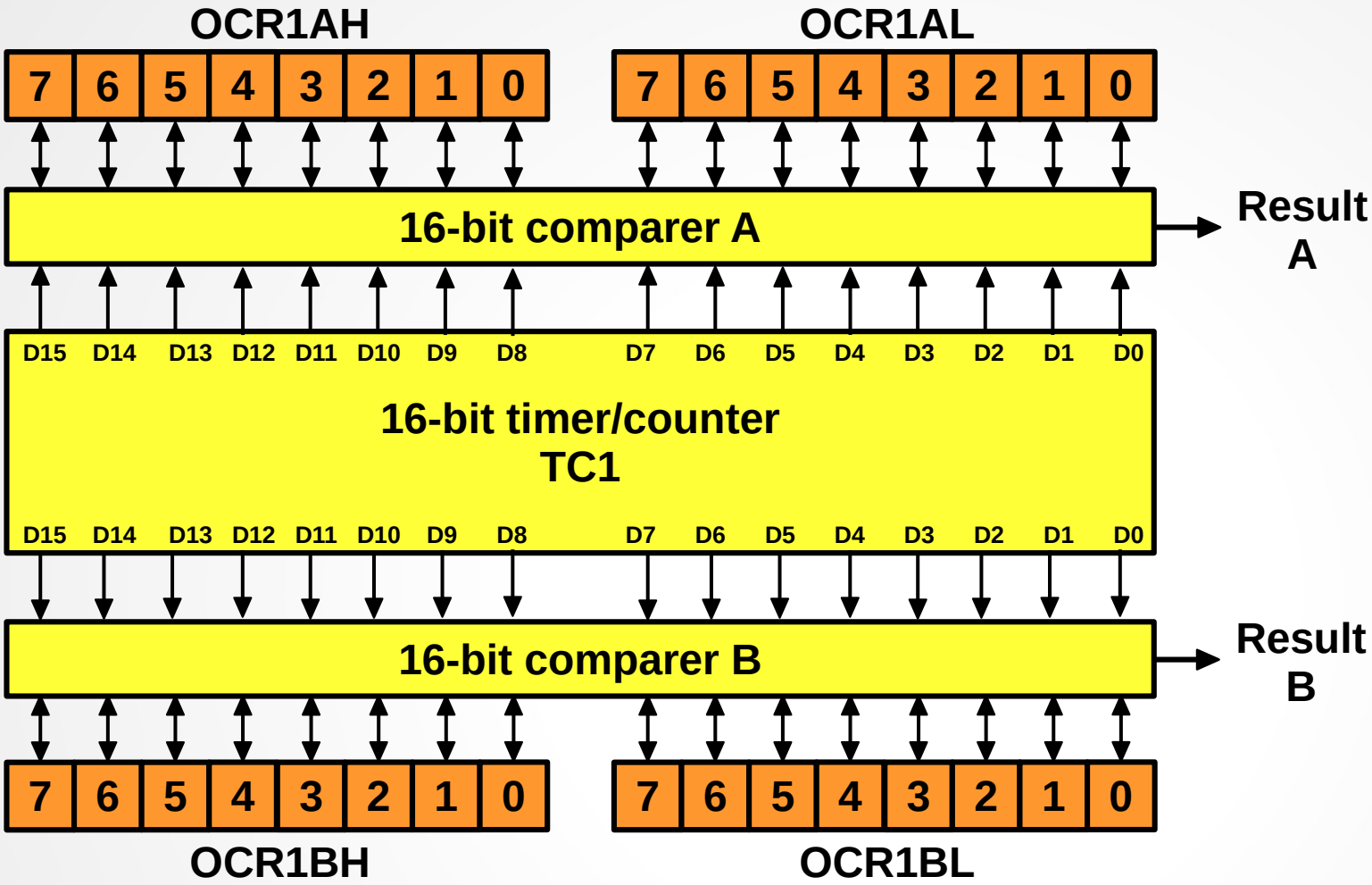
- **The ATtiny24 has a second counter on board, which can count with 16 bits up to 65,535.**
- **16-bit counters are a little bit different in handling, this is why we have to learn some new stuff here.**
- **In this section we work with the TC in a second mode: CTC. That means: the TC counter is cleared (Clear Timer on Compare, restarts at zero) in the clock cycle following the compare match.**
- **This allows to generate a wider variety of frequencies.**

The hardware



- This is the combined tone generator and blinker.
- The tone is generated with TC1‘s OC1B pin.
- To avoid interference with the MISO function, a jumper disconnect the speaker during programming.

16-bit timers



- 16-bit timers have all count and compare port registers in 2 bytes each: Low and High.
- The two bytes have to written/read in two single operations.

- To ensure that the high and low byte belong to the same counter state, reading and writing use an interim buffer where the first byte is stored until the second byte is written or read.

Assessing 16-bit port registers

- During read access, the low byte must be read first. This stores the associated high byte in the temporary register, which can be read next.

```
; Reading the 16-bit counter
      IN R16, TCNT1L ; Read low byte first, this stores the high byte
      IN R17,TCNT1H ; Then the high byte
```

- During write access, the high byte must be written first. This stores the high byte in the temporary register.

```
; Writing to the 16-bit counter
      LDI R16, High(count) ; The high byte
      OUT TCNT1H, R16 ; Write high byte first, this stores the high byte in the interim
      LDI R16, Low(count) ; The low byte next
      OUT TCNT1L, R16 ; Then the low byte, the high byte taken from interim
```

- Note that the writing of the 16 bits simultaneously takes place only after the low byte has been written.

CTC mode of the timer

- The following shows the CTC mode of the timer. CTC modes can be used in 8- and in 16-bit timers/counters.
- CTC modes use a compare register (normally A) to restart the timer in the next following timer clock cycle that follows compare match.
- If your counter shall only run from 0 to 5, set its compare register to 5. When the sixth counter pulse occurs, the counter will restart at zero.
- The number of active counter clock cycles is always CompareMatchA plus one.
- The CTC mode requires that the WGM bits in the control registers A and B are set to the appropriate combination.

Timer modes TC1

Table 12-5. Waveform Generation Modes

Mode	WGM 13:10	Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0000	Normal	0xFFFF	Immediate	MAX
1	0001	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0010	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0011	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0100	CTC (Clear Timer on Compare)	OCR1A	Immediate	MAX
5	0101	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0110	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0111	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1000	PWM, Phase & Freq. Correct	ICR1	BOTTOM	BOTTOM
9	1001	PWM, Phase & Freq. Correct	OCR1A	BOTTOM	BOTTOM
10	1010	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1011	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1100	CTC (Clear Timer on Compare)	ICR1	Immediate	MAX
13	1101	(Reserved)	–	–	–
14	1110	Fast PWM	ICR1	TOP	TOP
15	1111	Fast PWM	OCR1A	TOP	TOP

- **These are all available timer modes for TC1.**
- **The CTC modes are marked red.**

Setting the WGM bits

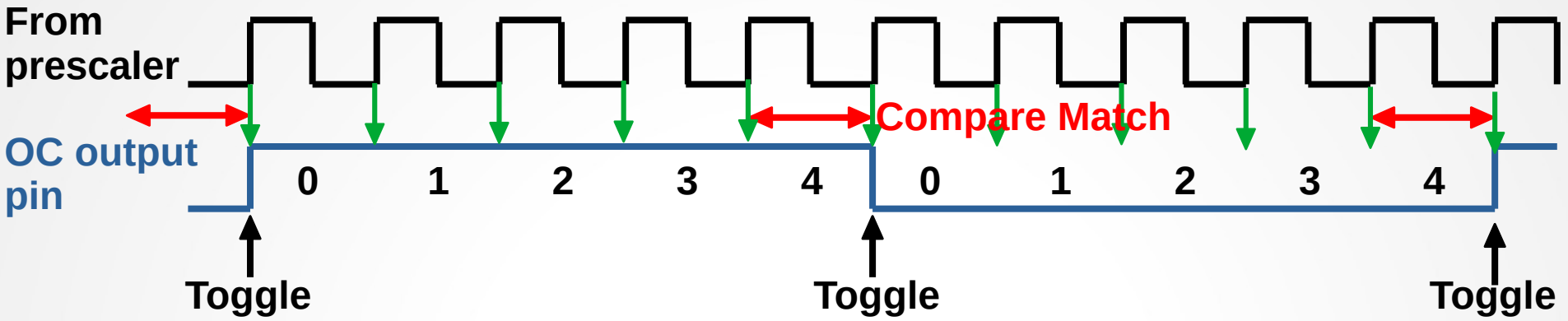
- **The WGM bits are located in the two control port registers TCCR1A and 1B:**

Bit	7	6	5	4	3	2	1	0	
0x2F (0x4F)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
Bit	7	6	5	4	3	2	1	0	
0x2E (0x4E)	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	

- **So WGM10 and WGM11 have to be written together with the COM bits, while WGM12 and WGM13 have to be written together with the clock selection bits CS.**

Generating tones

- The CTC mode and the OC pin works like this:



- The system clock, divided by the prescaler, advances the counter.
- When the counter reaches the compare value and the next counting pulse comes in, the OC pin toggles and the counter is cleared.
- The whole OC pin cycle requires two toggle events.

The frequency generated

- **Because of this, the generated frequency is:**
  $f = f_{system} / d_{prescaler} / (d_{compare} + 1) / 2$
- **For 1 MHz, a prescaler of 1 and a compare match value of 1,000 the generated frequency is 500 Hz.**
- **The lowest frequency at 1 MHz with a prescaler of 1 is 7.6 Hz, the highest is 500 kHz.**
- **So a prescaler of 1 allows generation of the complete audible frequency range.**

Questions and tasks in Lecture 6

Question 6-1: At which frequency does the red/green LED blink at 1 MHz and at 4 MHz clock? (Hint: do not forget that one cycle has two phases – one forward red and one backward green!)

Bonus question: At which clock frequency has the controller to be operated to get a 1 Hz blink frequency? How can this be made electronically (search for „External clock“ in the handbook)? And at which ISP clock frequency has the controller to be programmed then? Does your hard- and software allow this (test this!)?

Questions and tasks in Lecture 6 - Continued

Task 6-2: Write a program that generates a pre-defined frequency on the speaker (calculate the compare match value from the given frequency in Milli-Hertz) and that blinks in a second rhythm. When the LED is on, change the tone to be exactly one octave higher. Test your program on the breadboard and with the simulator (by enabling the timer view).

Bonus question: What are the compare match values for the gambit tones a to g (440, 493.883, 523.251, 587.330, 659.255, 698.456 and 783.991 Hz) and how accurate can those be generated with an xtal-driven ATtiny24 at 4 MHz? Is a crystal with 4.194304 MHz a better choice in this case? Why?