Embedded Systems

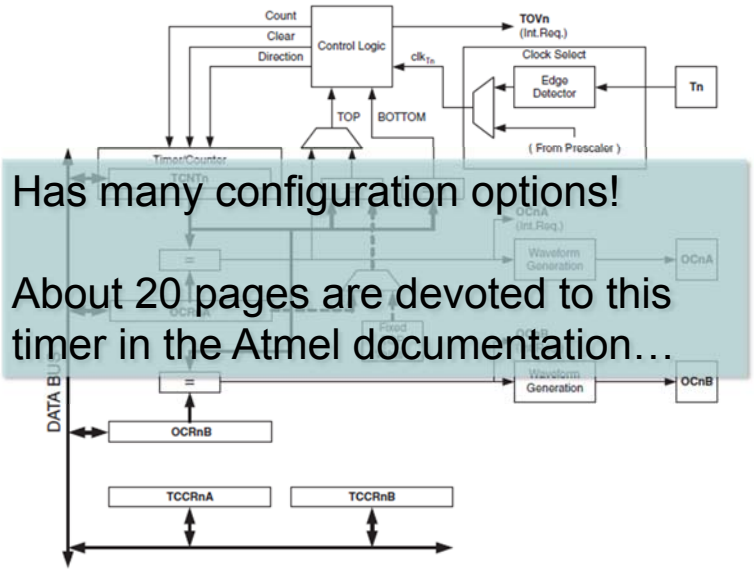
Lecture 8: Timers

F-35 Lightning II Electro-optical Targeting System (EOTS)

Timers and Counters

- **Very important peripherals on microcontrollers**
- **ATtiny45 has two 8-bit timers**
- **ATmega88PA has**
  - **Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode**
  - **One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture**
- **We will explore the ATtiny45 timers. Later in course, students will use timers on ATmega88PA**

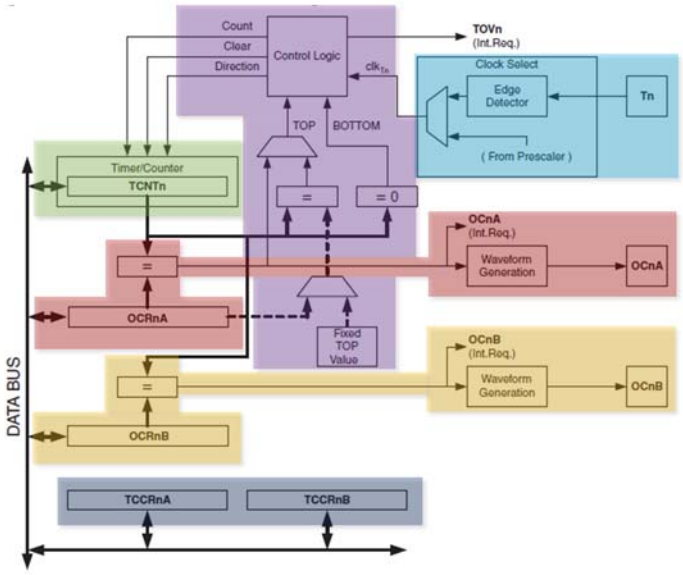
8-Bit Timer/Counter for ATtiny45



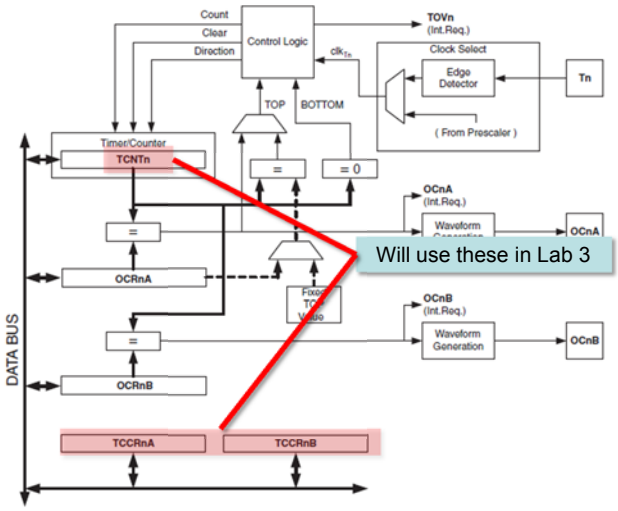
Has many configuration options!

About 20 pages are devoted to this timer in the Atmel documentation…

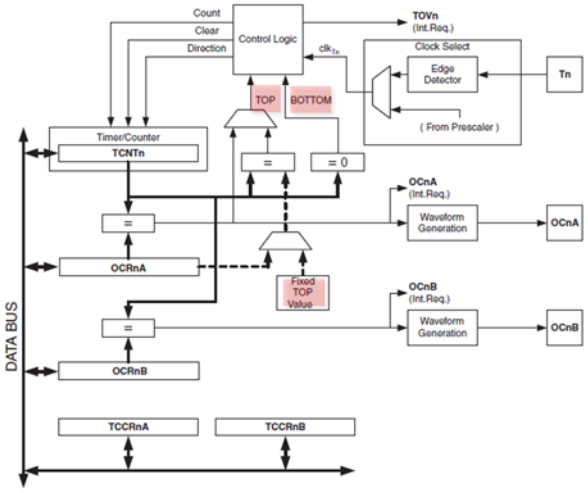
ATtiny45 8-Bit Counter/Timers – Basic Components



Many register and bit references in this section are written in general form. A lower case "n" replaces the Timer/Counter number, in this case 0. A lower case "x" replaces the Output Compare Unit, in this case Compare Unit A or Compare Unit B. However, when using the register or bit defines in a program, the precise form must be used, i.e., TCNT0 for accessing Timer/Counter0 counter value and so on.

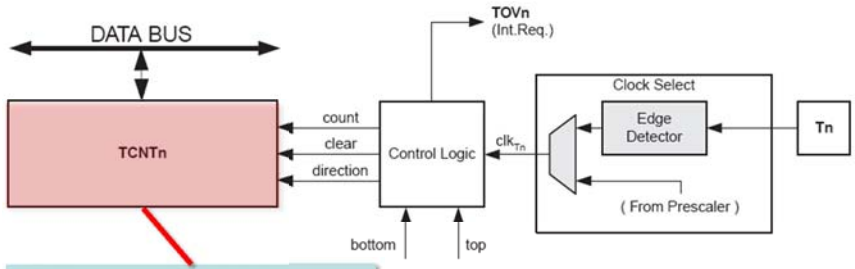


Constant	Description
BOTTOM	The counter reaches BOTTOM when it becomes 0x00
MAX	The counter reaches its MAXimum when it becomes 0xFF (decimal 255)
TOP	The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0A Register. The assignment depends on the mode of operation



The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit.

Figure 11-4. Counter Unit Block Diagram



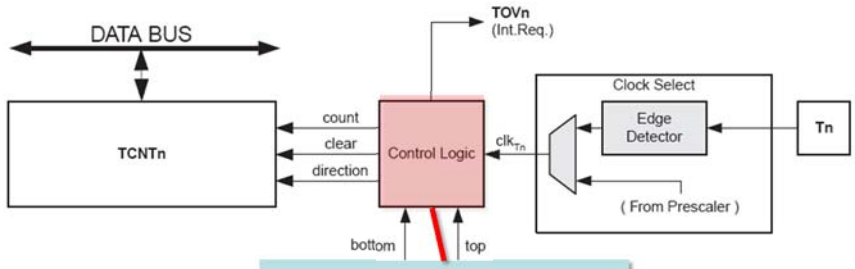
8-bit register that holds the count
TCNT0 → Timer/Counter 0
TCNT1 → Timer/Counter 1

S

count	Increment or decrement TCNT0 by 1.
direction	Select between increment and decrement.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk_{T0} in the following.
top	Signalize that TCNT0 has reached maximum value.
bottom	Signalize that TCNT0 has reached minimum value (zero).

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit.

Figure 11-4. Counter Unit Block Diagram



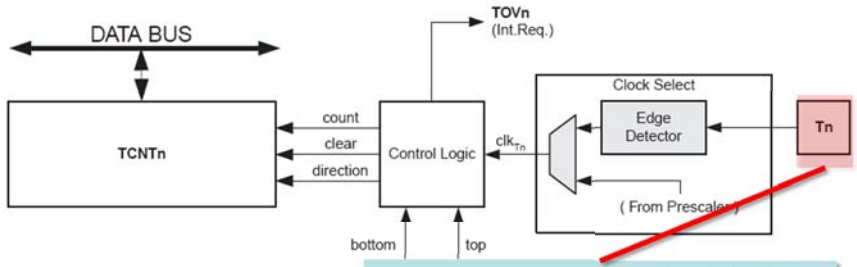
Determines if it counts up or down. One configures this by programming various configuration registers

Signal description (internal si

count	Increment or decrement TCNT0 by 1.
direction	Select between increment and decrement.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk_{T0} in the following.
top	Signalize that TCNT0 has reached maximum value.
bottom	Signalize that TCNT0 has reached minimum value (zero).

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit.

Figure 11-4. Counter Unit Block Diagram

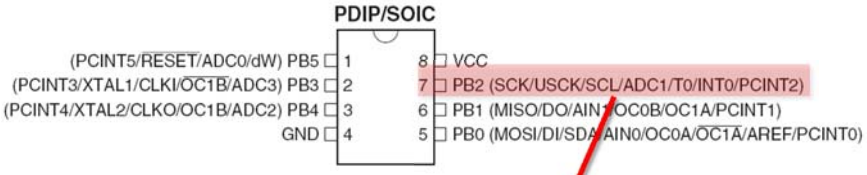


Input pulses can come from external pin **T0**. This is **PB2**, assuming it is properly configured.

Can be either **T0** falling edge or **T0** rising edge.

Signal description (internal signals):

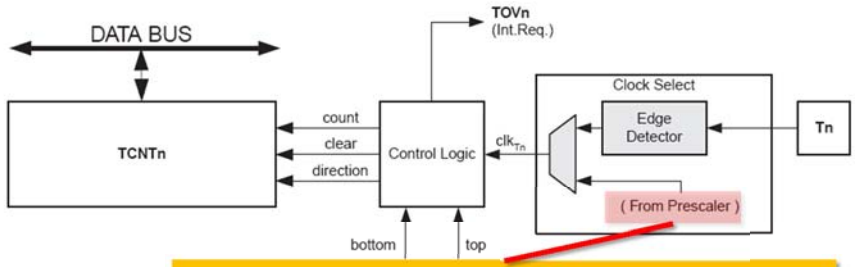
count	Increment
direction	Select between increment and decrement.
clear	Clear TCNT0 (set all bits to zero).
clk_{Tn}	Timer/Counter clock, referred to as clk_{T0} in the following.
top	Signalize that TCNT0 has reached maximum value.
bottom	Signalize that TCNT0 has reached minimum value (zero).



Input pulses can come from external pin **T0**. This is **PB2**, assuming it is properly configured.

The main part of the 8-bit Timer/Counter is the programmable bi-directional counter unit.

Figure 11-4. Counter Unit Block Diagram



Input pulses can come from main system clock, optionally divided or prescaled by system clock frequency:

$f_{CLK\_I/O}$, $f_{CLK\_I/O}/8$, $f_{CLK\_I/O}/64$, $f_{CLK\_I/O}/256$, or $f_{CLK\_I/O}/1024$.

Assuming a 10 MHz clock this would be

10 MHz, 1.25 MHz, 156.25 KHz, 39.0625 kHz, and 9.765625 kHz

Signal description (i

count	
direction	
clear	
clk$_{Tn}$	
top	
bottom	Signalize that TCNT0 has reached minimum value (zero).

TCNT0 – Timer/Counter Register

TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x32	TCNT0[7:0]								TCNT0
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	

The Timer/Counter Register gives direct access, both for read and write operations, to the Timer/Counter unit 8-bit counter.

```
; Wait for TIMER0 to roll over.
delay:
      in    r24,TCNT0    ; Get counter value
      cpi   r24,0x00     ; is it zero?
      brne  delay        ; no --> wait some more
      …                  ; yes --> continue on
```

"Naive Approach!"

Configuring Timers

GTCCR – General Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C	TSM	PWM1B	COM1B1	COM1B0	FOC1B	FOC1A	PSR1	PSR0	GTCCR
Read/Write	R/W	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x2A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
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	

This configures some of the more advanced features of the timers, see documentation. We may return to this later in course. Sensible default/initial values are provided.

Configuring Timers

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x33	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

These bits configure some of the more advanced features of the timers, see documentation. We may return to this later in course.

These are "Clock Select" bits

Table 11-6. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(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 T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

0x02 will clock the timer at 1/8 the system clock

Configuring Timers

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x2A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
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	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x33	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 11-5. Waveform Generation Mode Bit Description

Mode	WGM 02	WGM 01	WGM 00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX(1)
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM(2)
2	0	1	0	CTC	OCRA	Immediate	MAX(1)
3	0	1	1	Fast PWM	0xFF	BOTTOM(2)	MAX(1)
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM(2)
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM(2)	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Configuring Timers – Normal Mode

Table 11-5. Waveform Generation Mode Bit Description

Mode	WGM 02	WGM 01	WGM 00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on
0	0	0	0	Normal	0xFF	Immediate	MAX(1)
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM(2)
2	0	1	0	CTC	OCRA	Immediate	MAX(1)
3	0	1	1	Fast PWM	0xFF	BOTTOM(2)	MAX(1)
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM(2)
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM(2)	TOP

Notes: 1. MAX = 0xFF
2. BOTTOM = 0x00

Normal Mode

The simplest mode of operation is the Normal mode (WGM0[2:0] = 0). In this mode the counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (TOP = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. There are no special cases to consider in the Normal mode, a new counter value can be written anytime.

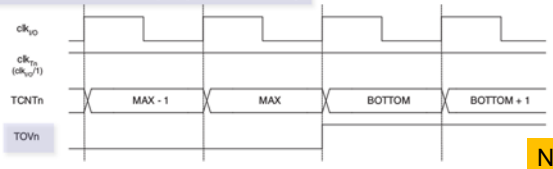
The Output Compare Unit can be used to generate interrupts at some given time. Using the Output Compare to generate waveforms in Normal mode is not recommended, since this will occupy too much of the CPU time.

Configuring T/Cs – "TOVn" in Normal Mode

11.8 Timer/Counter Timing Diagrams

The Timer/Counter is a synchronous design and the timer clock (clk_{T0}) is therefore shown as a clock enable signal in the following figures. The figures include information on when Interrupt Flags are set. Figure 11-10 contains timing data for basic Timer/Counter operation. The figure shows the count sequence close to the MAX value in all modes other than phase correct PWM mode.

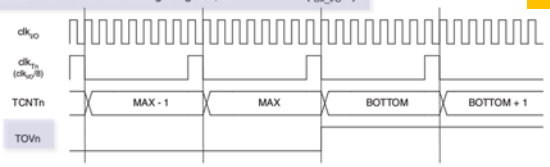
Figure 11-10. Timer/Counter Timing Diagram, no Prescaling



Normal Mode:
MAX=0xFF
BOTTOM=0x00

Figure 11-11 shows the same timing data, but with the prescaler enabled.

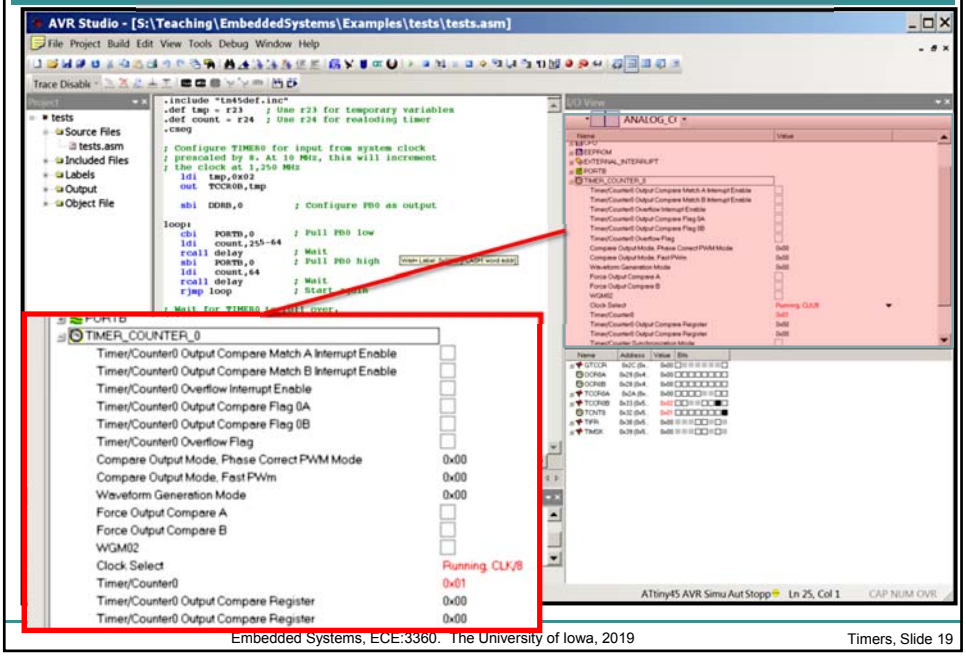
Figure 11-11. Timer/Counter Timing Diagram, with Prescaler ($f_{clk\_I/O}/8$)



Timer/Counter (T/C) Configuration

- **Key questions:**
  - **Clock Source?**
    - internal → μC clock selection, prescaler, …
    - external → rising/falling edge
  - **T/C Mode?**
    - Normal
    - Pulse With Modulation (PWM): fast, phase correct, …
    - Clear Timer on Compare match (CTC)
    - …
  - **T/C Integration into program?**
    - Polling
    - Interrupt
    - Waveform generation with T/C HW
  - **Initial values of TCNTn, OCRnA, OCRnB, … ?**
  - **Initialization sequence**
- Note: code snippets provided only illustrate concepts!
- Check configuration → debugging in Atmel Studio 7 → I/O View

Timer/Counter Configuration – I/O View



Configuring Timers

The timers have configuration/control registers (**GTCCR, TCCR0A,** and **TCCR0B**). Since they are located in I/O memory space, one uses **IN/OUT** instructions to access the registers.

```
// *** Example in C code ***
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 1250.000 kHz
// Mode: Normal top=FFh
// OC0A output: Disconnected
// OC0B output: Disconnected
TCCR0A = 0x00;
TCCR0B = 0x02;
TCNT0  = 0x00;
OCR0A  = 0x00;
OCR0B  = 0x00;
```

C code to configure ATtiny45 **TIMER0** to use the system clock, prescaled by 8

```
LDI  R30,0x02
OUT  0x33,R30
```

Assembly code to configure ATtiny45 **TIMER0** to use the system clock, prescaled by 8

Generating Square Waves – "Naive Approach!"

```
.include "tn45def.inc"
.def tmp = r23      ; Use r23 for temporary variables
.cseg

; Configure TIMER0 for input from system clock
; prescaled by 8. At 10 MHz, this will increment
; the clock at 1.250 MHz
  ldi   tmp,0x02
  out   TCCR0B,tmp

  sbi   DDRB,0      ; Configure PB0 as output

loop:
  cbi   PORTB,0     ; Pull PB0 low
  rcall delay       ; Wait
  sbi   PORTB,0     ; Pull PB0 high
  rcall delay       ; Wait
  rjmp loop         ; Start again

; Wait for TIMER0 to roll over.
delay:
  in    tmp,TCNT0   ; Get counter value
  cpi   tmp,0x00    ; is it zero?
  brne  delay       ; no --> wait some more
  ret               ; yes --> return
.exit
```

Note the **.def** directive

Note the **.cseg** directive

Load 0x02 into **TCCR0B**.

Pull low and wait

Pull high and wait

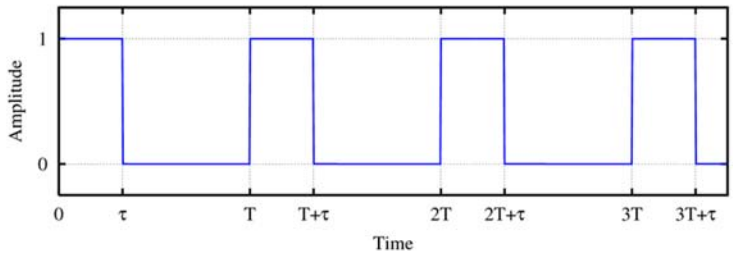
Timer rolls over and become zero and continue counting up

Excluding overhead this program generates a ~2.4 kHz signal. (Why?)

Square Waves & Duty Cycle

The **duty cycle** is the fraction of time that a system is in an "active" state or duty cycle is the proportion of time during which a component, device, or system is operated. Suppose a disk drive operates for 1 second, and is shut off for 99 seconds, then is run for 1 second again, and so on. The drive runs for one out of 100 seconds, or 1/100 of the time, and its duty cycle is therefore 1/100, or 1 percent.

For a square wave:



$$D = \frac{\tau}{T}$$

Generating Square Waves - "Naive Approach!"

```
.include "tn45def.inc"
.def tmp = r23      ; Use r23 for temporary variables
.def count = r24    ; Use r24 for reloading timer
.cseg

; Configure TIMER0 for input from system clock
; prescaled by 8. At 10 MHz, this will increment
; the clock at 1.250 MHz
   ldi   tmp,0x02
   out   TCCR0B,tmp

   sbi   DDRB,0         ; Configure PB0 as output

loop:
   cbi   PORTB,0        ; Pull PB0 low
   ldi   count,255-64
   rcall delay          ; Wait
   sbi   PORTB,0        ; Pull PB0 high
   ldi   count,64
   rcall delay          ; Wait
   rjmp loop            ; Start again

; Wait for TIMER0 to roll over.
delay:
   out   TCNT0,count    ; Load counter
wait:
   in    tmp,TCNT0      ; Get counter value
   cpi   tmp,0x00       ; is it zero (rolled over)?
   brne  wait           ; no --> wait some more
   ret                  ; yes, return
.exit
```

Note the **.def** directives

Load 0x02 into **TCCR0B**.

Note how we use the Assembler to do calculations for us

Excluding overhead this program generates approx. a 4.8 KHz wave.

The duty cycle is ~ 75%.

More Reliable Delay

The previous delay routine has some "**flaws**" since the timer is still counting while it is being reloaded. Further, checking for overflow by checking against zero while the counter is running may cause problems. A more reliable method is to check the timer overflow flag (**TOV0**) in the **TIFR** register:

TIFR – Timer/Counter Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x38	–	OCF1A	OCF1B	OCF0A	OCF0B	TOV1	TOV0	–	TIFR
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R	
Initial Value	0	0	0	0	0	0	0	0	

Set when timer 0 rolls over. To clear, write logic 1 (**yes "1" and not "0" …**)

The **TIFR** register is <u>not</u> in the lower 32 I/O registers, so we CANNOT use **CBI**, **SBI** etc. Rather, we must use **IN**, **SBR**, **OUT**:

```
; Clear over flow flag.
   in    tmp2,TIFR      ; tmp <-- TIFR
   sbr   tmp2,1<<TOV0      ; Clear TOV0, write logic 1
   out   TIFR,tmp2
```

More Reliable Delay

```
.def tmp1 = r23     ; Use r23 for temporary variables
.def tmp2 = r24     ; Use r24 for temporary values

…

; Wait for TIMER0 to roll over.
delay:

; Stop timer 0.
   in    tmp1,TCCR0B     ; Save configuration
   ldi   tmp2,0x00       ; Stop timer 0
   out   TCCR0B,tmp2

; Clear over flow flag.
   in    tmp2,TIFR       ; tmp <-- TIFR
   sbr   tmp2,1<<TOV0      ; Clear TOV0, write logic 1
   out   TIFR,tmp2

; Start timer with new initial count
   out   TCNT0,count     ; Load counter
   out   TCCR0B,tmp1     ; Restart timer

wait:
   in    tmp2,TIFR       ; tmp <-- TIFR
   sbrs  tmp2,TOV0       ; Check overflow flag
   rjmp  wait
   ret
```

Stop timer, clear overflow flag, reload, and start timer

Quiz

- **An engineer wants to utilize TC0 of an ATtiny45 with 8 MHz external (µC) clock to produce a delay of 7.5 ms.**
- **Q1: What prescaler should be used for highest accuracy?**
- **Q2: In "normal mode" (using TOV0), what value needs to be loaded into TCNT0?**
- **Q3: What will be the timing error in percent?**
- **Q4: What values need to be loaded into TCCR0A and TCCR0B?**

Solution → on blackboard!

Loading 16-Bit Timer on ATmega88PA

Many AVRs have 16-bit timer, meaning that the registers that hold the counts, are 16-bits (two bytes) wide. However, the AVR core is 8-bit, so loading the 16-bit registers must proceed byte-wise.

TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
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	

There is an internal **TEMP** register, **shared by all other 16-bit registers within each timer**, that facilitates loading.

The I/O instructions to the 16-bit timer registers involve this **TEMP** register.

However, on the Atmega88PA, **TCNT1H** and **TCNT1L** are outside the range of the IN/OUT instructions → use the **LDS**/**STS** instructions. (see ATmega88PA documentation)

To load the 16-bit timer, first **write** to **TCNT1H**. This does not load **TCNT1H**, but loads the internal **TEMP**. Next, **write** to **TCNT1L**. This loads **TCNT1L** and triggers the transfer from **TEMP** to **TCNT1H**.

Loading 16-Bit Timer on ATmega88PA

TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
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	

Step 1: Write the high byte. This places the content of r17 in the internal TEMP, and not TCNT1H.

STS TCNT1H, r17 → TEMP

TCNT1H

TCNT1L

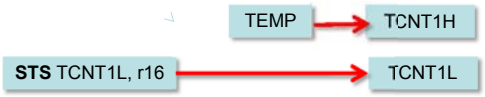
Loading 16-Bit Timer on ATmega88PA

TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
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	

Step 2: Write the low byte.

This copies the contents of r16 to TCNT1L, and also triggers the transfer from the internal TEMP to TCNT1H, all in one clock cycle



Loading 16-Bit Timer on ATmega88PA

```
; Set TCNT1 to 0x01FF
ldi r17,0x01
ldi r16,0xFF
sts TCNT1H,r17
sts TCNT1L,r16
```

First load TCNT1H. However, contents go to internal TEMP.

Next, load TCNT1L. This loads TCNT1L and triggers loading of TCNT1H.

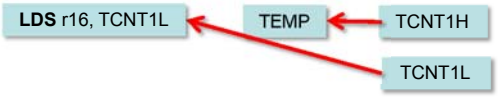
Reading 16-Bit Timer on ATmega88PA

TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
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	

Step 1: Read the low byte. Use the **LDS** instruction, since the registers are outside of the range of **IN.**

This copies the contents of TCNT1L to r16, and also triggers the transfer from the TCNT1H to the internal TEMP



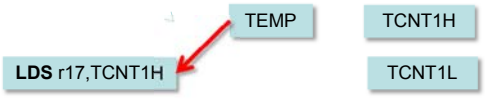
Reading 16-Bit Timer on ATmega88PA

TCNT1H and TCNT1L – Timer/Counter1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
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	

Step 2: Read the high byte.

This copies the contents of the internal TEMP to r17.



Reading 16-Bit Timer on ATmega88PA

To read the 16-bit timer, first read (using **LDS**) from TCNT1L. This also copies the contents of TCNT1H to the internal TEMP. Next read from TCNT1H.

```
; Read TCNT1 into r17:r16
LDS  r16,TCNT1L
LDS  r17,TCNT1H
...
```

First read TCNT1L. This also copies the contents of TCNT1H to the internal TEMP register.

Next, read TCNT1H.

From "m88PAdef.inc"

... EOL