Sistemas Baseados em Microprocessadores

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

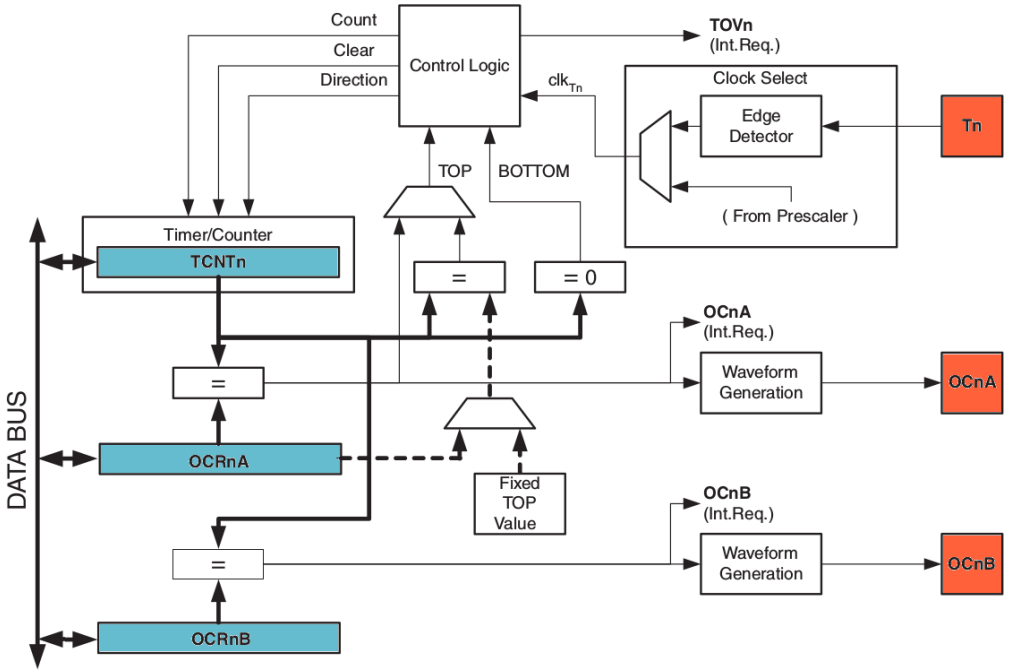
Faculdade de Engenharia

ATmega328p – Timers

João Paulo de Sousa

General Timer concepts

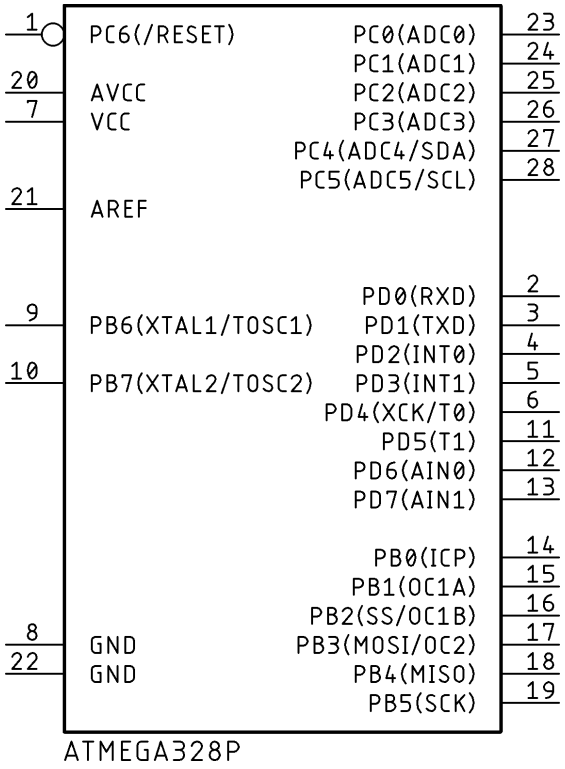
- Counts pulses:
  - Internal, derived from CLK
  - External, at pins **Tn**
- Can generate interrupt requests:
  - When **TCNTn** overflows
  - At given values of **TCNTn**, stored in **OCRnA, OCRnB**
- Can generate PWM signals at pins **OCnA** e **OCnB**.



Note: n = 0,1,2 identifies the Timer

Associated pins

- There are 3 timers...
- External pulse count:
  TC0: T0(PD4), TC1: T1(PD5)
- Input capture:
  TC1 only: ICP1(PB0)
- PWM outputs:
  TC0: OC0A(PD6), OC0B(PD5)
  TC1: OC1A(PB1), OC1B(PB2)
  TC2: OC2A(PB3), OC2B(PD3)



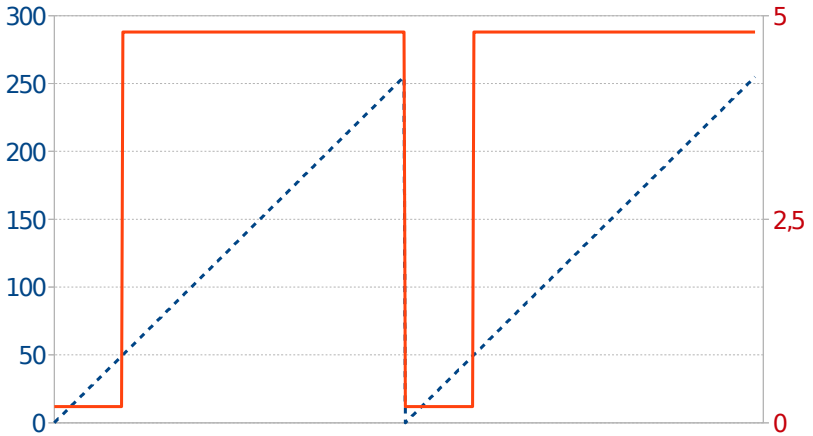
Note: not all alternate pin functions are indicated on the Eagle CAD symbol

Working modes overview

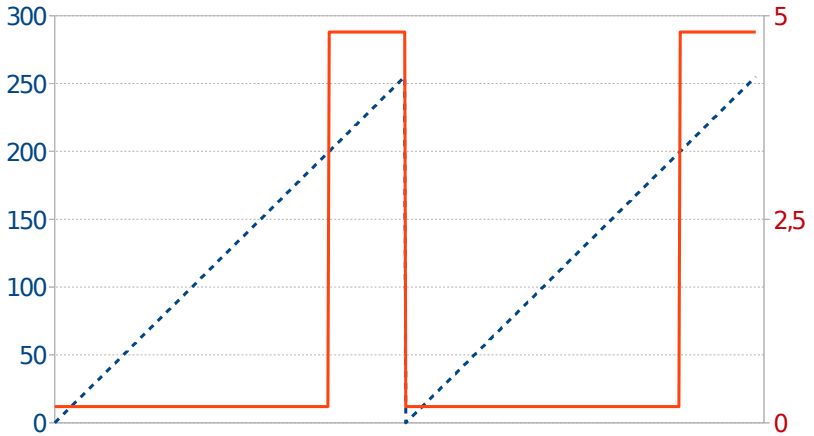
- NORMAL mode:
  - Counts at TCNTn up to a maximum value (255/65535)
  - Overflows to zero whithout stopping the count
  - Generates an interrupt request at overflow of TCNTn
- CTC mode (*Clear Timer on Compare*):
  - Counts at TCNTn up to OCRnA
  - Resets to zero whithout stopping the counting
  - Generates an interrupt request when TCNTn = OCRnA

Working modes overview

- PWM modes (several)
  - Fast: up counting
  - Slow: bidirectional counting
- Logic value at pin OCnX depends on the counting:



OCRnA=50



OCRnA=200

Working modes overview

- Input capture mode (only for Timer/Counter 1)
  - Used to time stamp events
  - Used to measure elapsed time
- Count value is copied to an auxiliary register when an external event occurs
  - At pin ICP1
  - On the analog comparator

Operation Registers

- **TCNTn**: impulse counter
  - TCNT0 and TCNT2 are 8-bit wide, TCNT1 is 16-bit
- **OCRnA**, **OCRnB**: intermediate values
  - OCR0A/B, OCR2A/B are 8-bit wide, OCR1A/B is 16-bit
- **TIFRn**: stores the interrupt request status
- Access to 16-bit registers:
  - Two 8-bit accesses (read: L,H; write: H,L) ← Handled by the C compiler
  - Interrupt service might have to be disabled

(12 Operation registers in total: n = 0,1,2 identifies the Timer)

Configuration Registers

- **TCCRnA, TCCRnB, TCCR1C** : main configuration
  WGM: working mode, **CS**: Clock Select,
  COM: behaviour of PWM, **ICN**: behaviour of input capture

TCCR0A, TCCR1A, TCCR2A:

COMnA1	COMnA0	COMnB1	COMnB0	-	-	WGMn1	WGMn0

TCCR0B, TCCR2B, (below: TCCR1B):

FOCnA	FOCnB	-	-	WGMn2	CSn2	CSn1	CSn0
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

TCCR1C:

FOC1A	FOC1B	-	-	-	-	-	-

(7 Configuration registers in total: n = 0,1,2 identifies the Timer)

Interrupt request enable

- **TIMSKn**: enable individual interrupt requests
  - **TOIEn:** Interrupt request when overflow of TCNTn
  - **OCIEnA:** Interrupt request when TCNTn = OCRnA
  - **OCIEnB:** Interrupt request when TCNTn = OCRnB
  - **ICIE1:** Interrupt request when event captured

TMSKn, n=0,2:

-	-	-	-	-	OCIEnB	OCIEnA	TOIEn

TIMSK1:

-	-	ICIE1	-	-	OCIE1B	OCIE1A	TOIE1

Status of interupt requests

- **TIFRn**: Timer n Interrupt Flag Register
  - TOVn: Pending request of type overflow
  - OCFnA, OCFnB: Pending request of type TCNT = OCRA/B
  - ICF1: Pending request of type Capture

TIFRn, n=0,2:

-	-	-	-	-	OCFnB	OCFnA	TOVn

TIFR1:

-	-	ICF1	-	-	OCF1B	OCF1A	TOV1

- Clear the pending request by writing a one

Interrupt vectors

Timer	Condition	Address	Symbolic name
TC2 (8-bit)	Compare match A	0x000E	TIMER2_COMPA_vect
	Compare match B	0x0010	TIMER2_COMPB_vect
	Overflow	0x0012	TIMER2_OVF_vect
TC1 (16-bit)	Capture	0x0014	TIMER1_CAPT_vect
	Compare match A	0x0016	TIMER1_COMPA_vect
	Compare match B	0x0018	TIMER1_COMPB_vect
	Overflow	0x001A	TIMER1_OVF_vect
TC0 (8-bit)	Compare match A	0x001C	TIMER0_COMPA_vect
	Compare match B	0x001E	TIMER0_COMPB_vect
	Overflow	0x0020	TIMER0_OVF_vect

Working Mode Bits

Mode	WGM3	WGM2	WGM1	WGM0	Mode (TC0/TC2)	Mode (TC1)
0	0	0	0	0	**Normal (255)**	**Normal (65535)**
1	0	0	0	1	Slow PWM (255)	Slow PWM (255)
2	0	0	1	0	**CTC (OCRnA)**	Slow PWM (511)
3	0	0	1	1	Fast PWM (255)	Slow PWM (1023)
4	0	1	0	0	Reserved	**CTC (OCR1A)**
5	0	1	0	1	Slow PWM (OCRnA/B)	Fast PWM (255)
6	0	1	1	0	Reserved	Fast PWM (511)
7	0	1	1	1	**Fast PWM (OCRnA/B)**	Fast PWM (1023)
8	1	0	0	0	Reserved	Ph/Fr PWM (IC1A)
9	1	0	0	1	Reserved	Ph/Fr PWM (OCR1A/B)
10	1	0	1	0	Reserved	Phase PWM (ICR1)
11	1	0	1	1	Reserved	Phase PWM (OCR1A/B)
12	1	1	0	0	Reserved	CTC (ICR1)
13	1	1	0	1	Reserved	Reserved
14	1	1	1	0	Reserved	Fast PWM (ICR1)
15	1	1	1	1	Reserved	**Fast PWM (OCR1A/B)**

Clock Select bits (Prescaler)

CSn2..0			Meaning (TC0/TC1)
0	0	0	Stop Counting
0	0	1	Prescaler 1
0	1	0	Prescaler 8
0	1	1	Prescaler 64
1	0	0	Prescaler 256
1	0	1	Prescaler 1024
1	1	0	External, falling edge
1	1	1	External, rising edge

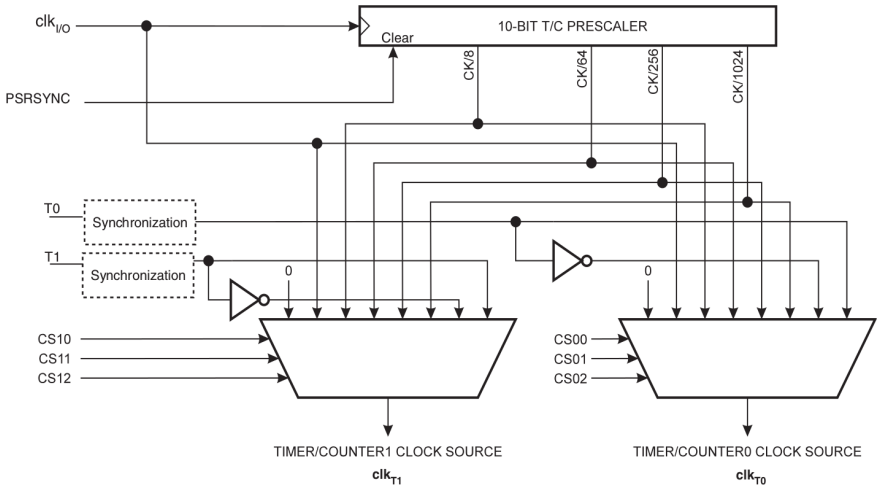
CS22..0			Meaning (TC2)
0	0	0	Stop Counting
0	0	1	Prescaler 1
0	1	0	Prescaler 8
0	1	1	Prescaler 32
1	0	0	Prescaler 64
1	0	1	Prescaler 128
1	1	0	Prescaler 256
1	1	1	Prescaler 1024

Possible values: 1 – 8 – 32 – 64 – 128 – 256 – 1024

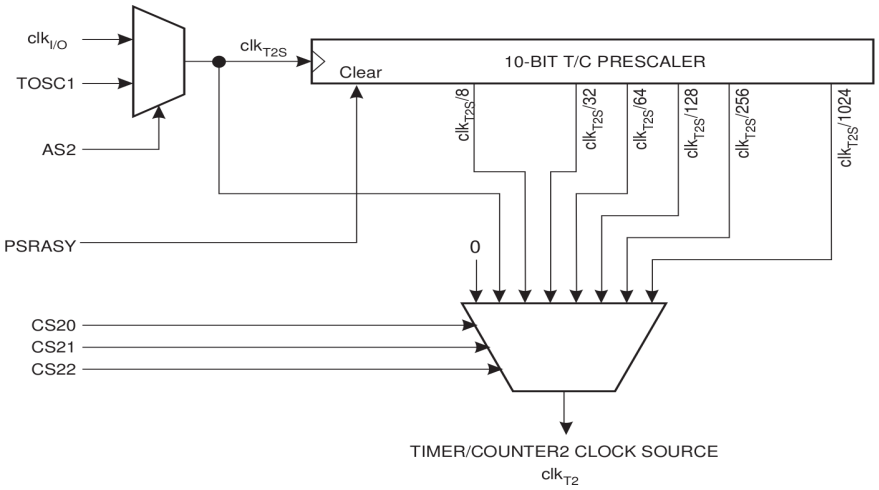
(Red: TC2 only)

Prescaler internal details

- TC0, TC1:



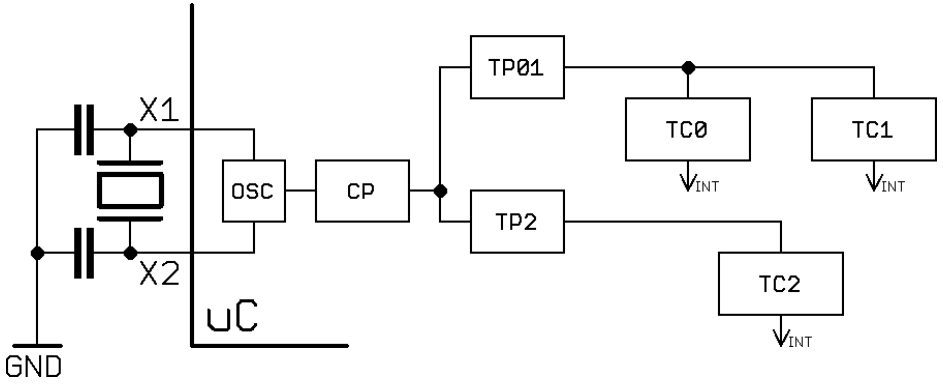
- TC2:



$$f_{TC} = f_{CLKIO}/TP \qquad f_{CLKIO} = f_{CLK}/TP$$

Frequency of internal clock pulses

- Facts:
  - Two prescalers (CP, TP)
  - Interrupt request at the end of each complete count (CNT)
  - Defaults: CP=8 (Atmega, factory), CP=1(Arduino)
  - Avoid changing CP (Why?)



$$F_{INTR} = \frac{F_{CLK}}{CP \times TP \times CNT}$$

$$CP \times TP \times CNT = \frac{F_{CLK}}{F_{INTR}} = \frac{T_{INTR}}{T_{CLK}}$$

Prescalers and count values

Design space exploration tool:

Fosc (MHz)	Timer Prescaler (TP)	Values in red only applicable to TC2						
16								
TINTR (ms)	001/001	010/010	---/011	011/100	---/101	100/110	101/111	
100	1	8	32	64	128	256	1024	
CLK Prescaler (CP) (2⁰)	1				25000		6250	1562,5
(2¹)	2				12500		3125	781,25
(2²)	4		50000		6250		1562,5	390,63
(2³)	8		25		3125		781,25	195,31
(2⁴)	16		12500		1562,5		390,63	97,66
(2⁵)	32	50000	6250		781,25		195,31	48,83
(2⁶)	64	25000	3125		390,63	195,31	97,66	24,41
(2⁷)	128	12500	1562,5		195,31	97,66	48,83	12,21
(2⁸)	256	6250	781,25	195,31	97,66	48,83	24,41	6,1

CNT

CP

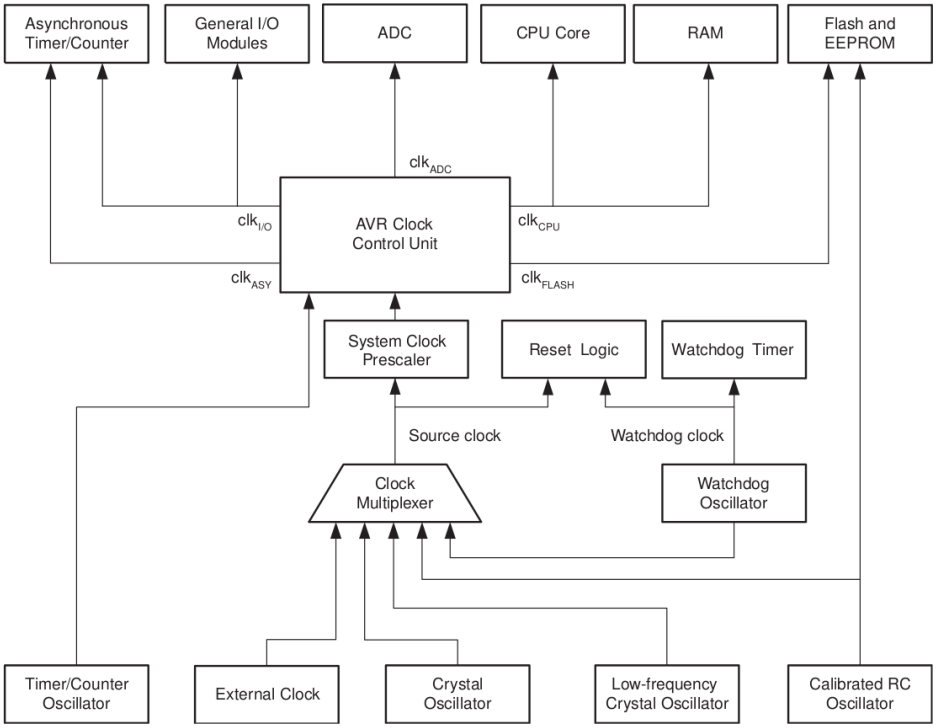
TP

Error (%)						
8	32	64	128	256	1024	
0	0	0	0	0	0,03	
0	0	0	0	0	0,03	
0	0	0	0	0,03	0,16	
0	0	0	0,03	0,03	0,16	
0	0	0,03	0,03	0,16	0,68	
0	0,03	0,03	0,16	0,16	1,7	
0	0,03	0,16	0,16	0,68	1,68	
0	0,03	0,16	0,16	0,68	1,7	1,72
0	0,03	0,16	0,68	1,7	1,68	1,64

Homework: Build your own spreadsheet...

Clock sources & internal distribution chain

- Several sources:
  - External oscillator
  - External crystal
  - Other...
- Several internal clocks:
  - CPU and RAM
  - FLASH and EEPROM
  - ADC, Timers, IO Ports...
- Each internal clock can be Individually disabled...



Clock Prescaler

- CLKPR register (protected access)

CLKPCE	-	-	-	CLKPS3	CLKPS2	CLKPS1	CLKPS0

N

- To write on this register:
  - Write 1 on bit CLKPCE and 0 on the other bits (0x80)
  - Write 0 on bit CLKPCE and define the other bits

```
CLKPR = 0x80;     /* Set CLKPCE bit, clear the others */
CLKPR = N;        /* CLK prescaler is 2^N */
```

Clock Prescaler

- CLKPR register (protected access)

CLKPCE	-	-	-	CLKPS3	CLKPS2	CLKPS1	CLKPS0

N

- Restrictions:
  - Writings can not be more than 4 clock cycles apart (need to activate compiler optimizations)
  - Only combinations 0..8 of CLKPS bits are allowed
- Factory default: N=3, (Arduino: N=0)

How to configure a timer

- Initialization (six steps)
  1. Stop the timer
  2. Clear all pending interrupt requests
  3. Define the working mode
  4. Define the start and end values for the count
  5. Configure how interrupt requests will be issued
  6. Start the timer with the adequate prescaler

```
void tc0_init(void) {
  TCCR0B = 0;             // Stop TC0
  TIFR0 |= (7<<TOV0);     // Clear pending intr
  TCCR0A = 0;             // Mode NORMAL
  TCNT0 = T0BOTTOM;       // Load BOTTOM value
  TIMSK0 = (1<<TOIE0);    // Enable Ovf intrpt
  TCCR0B = 4;             // Start TC0 (TP=256)
}
```

How to serve a periodic interrupt request

- Periodic interrupt service:
  - If necessary, reinitialize the count value
  - Execute any other task required by the application

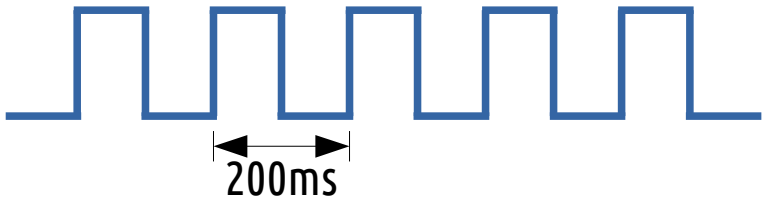
```
ISR(TIMER0_OVF_vect) {
  TCNT0 = T0BOTTOM;              // reload TC1
  PORTB = PORTB ^ (1<<LED);      // toggle LED
}
```

To further explore...

- ATmega328P datasheet (Moodle)
  - TC0/TC1: Ch. 19/20, Prescaler TC0/TC1: Ch. 21
  - TC2 & Prescaler TC2: Ch. 22
  - Clock signal distribution: Ch. 13
- Application note:
  - AVR130 Setup and Use the AVR® Timers

Timers – Example 1

- Problem:
  Blink a LED at a rate of 5Hz without using loop delays



Timers – Example 1

- Problem:
  Blink a LED at a rate of 5Hz without using loop delays

- Solution:
  A periodic interrupt service routine will toggle the LED every 100ms
- Questions: which timer? which mode? which prescaler? which count value?

(Answers here)

Timers – Example 1

- Facts:
  - T_{INTR} = 100ms
  - F_{CLK} = 16MHz
- Strategy:
  - T_{CLK} = 62,5ns
  - $T_{INTR} = 1.600.000 \cdot T_{CLK}$
- Goal: find CP, TP and CNT that verify:

  CP x TP x CNT = 1.600.000
- Restrictions:
  - CNT < 65535 (TC1)
  - CNT < 255 (TC0, TC2)
  - CP: 1,2,4,8,16,32,64,128
  - TP: 1,8,32,64,128,256,1024

Timers – Example 1

- CP=1 to avoid changing other peripherals, so:

$$TP \times CNT = 1600000$$

- Some values, by trial and error:

1600000 = 1 x 1600000 (a)		Too large count, even for 16-bit
= 8 x 200000 (b)		Too large count, even for 16-bit
= 32 x 50000 (c)		Illegal (needs a 16 bit number in TC2)
= 64 x 25000 (d)		Ok: CP=1, TP=64, CNT=25000 (16-bit)
= 128 x 12500 (e)		Illegal (needs a 16 bit number in TC2)
Best choice (why?):	= 256 x 6250 (f)	Ok: CP=1, TP=256, CNT=6250 (16-bit)
= 1024 x 1562,5 (g)		Rounding error

Timers – Example 1

- Solution (timer 1): CP=1, TP=256, CNT=6250
- Mode NORMAL
  - Counts from X to 65535, interrupt on overflow
  - X = 65536 - 6250 = 59286
  - Needs to reinitiate X
- Mode CTC
  - Counts from X to OCR1A, interrupt when TCNT1=OCR1A
  - X = 0, OCR1A= 6250
  - Don’t need to reinitiate X

Timers – Example 1

```
/*********************************************
 * example1.c
 *  A simple demo using periodic interrupts.
 *   Purpose: Blink an LED at 5Hz (T=200ms)
 *            without using delays.
 *  Solution: The LED has to toggle every 100ms.
 *            This will be accomplished by the
 *            ISR of a periodic interrupt
 *            request.
 *********************************************
 * If Fosc=16MHz, Tosc=62,5ns. To generate an
 * interrupt request every 100ms we need to
 * count 1600000 clock periods.
 * We need to find combinations of CP, TP and
 * COUNT verifying CPxTPxCOUNT=1600000.
 *
 * Let's start with CP=1 to avoid disturbing
 * other peripherals:
 *
 * 1600000 = 1x1024x1562,5 = 1x256x6250
 *         = 1x128x12500   = 1x64x25000
 *         = 1x32x50000
 *
```

```
 * The first combination will introduce a
 * timing error because the count value is not
 * an integer. All the others have zero error.
 * We need a 16-bit counter/timer (TC1) which
 * immediately eliminates TP=32 and TP=128
 * that are only valid for the 8-bit timer TC2.
 *
 * We are then limited to 2 combinations:
 *  1600000 = 1x256x6250 = 1x64x25000
 *
 * Let's choose CP=1, TP=256, COUNT=6250 since
 * the timer input frequency will be lower
 *
 *********************************************
 * For the mode:
 * Mode 0: TC1 starts at a given BOTTOM value,
 * counts up to 65535, and overflows to zero
 * without stopping.
 *
 * Mode 2: TC1 starts at a given BOTTOM value,
 * counts up to the value in OCR1A and returns
 * to zero without stopping
 *
```

Timers – Example 1

```
 *
 * The other modes are for PWM generation
 *
 * Let's choose mode NORMAL. Mode CTC is
 * left as an exercise for the student
 *
 ********************************************
 * Created: Oct 12, 2014
 *  Author: jpsousa@fe.up.pt (eclipse+gcc)
 ********************************************/

#include <avr/io.h>
#include <avr/interrupt.h>

#define LED PB5

/* 100ms = 6250 clock cycles @ 16MHz/(1*256)  */
#define T1BOTTOM 65536-6250
```

```
/********************************************
 * The main loop is empty since everything
 * is handled by the ISR of Timer 1 which
 * is executed every 100ms
 ********************************************/

void main(void) {
  DDRB |= (1 << LED);  // LED as output
  tc1_init();          // Init Timer 1
  sei();               // Enable global int

  while(1);            // Main loop is empty!
}
```

Timers – Example 1

```
/*********************************************
 * Timer 1 initialization in NORMAL mode
 *********************************************
 * - Stop TC1 and clear pending interrupts
 * - Define mode of operation & BOTTOM value
 * - Set the required interrupt mask
 * - Start timer with the proper prescaler
 *********************************************/
void tc1_init(void) {
  TCCR1B = 0;           // Stop TC1
  TIFR1 = (7<<TOV1)     // Clear all
        | (1<<ICF1);    //   pending interrupts
  TCCR1A = 0;           // NORMAL mode
  TCNT1 = T1BOTTOM;     // Load BOTTOM value
  TIMSK1 = (1<<TOIE1); // Enable Ovf intrpt
  TCCR1B = 4;           // Start TC1 (TP=256)
}
```

```
/*********************************************
 * Timer 1 ISR is executed each 100ms
 *********************************************
 *    - Reload BOTTOM value
 *    - Toggle LED
 *********************************************/
ISR(TIMER1_OVF_vect) {
  TCNT1 = T1BOTTOM;             // reload TC1
  PORTB = PORTB ^ (1<<LED);     // toggle LED
 }
```

Example 1 – Design space exploration

Goal: CP x TP x COUNT = 1.600.000

Fosc (MHz)	Timer Prescaler (TP)	Values in red only applicable to TC2						
16								
TINTR (ms)	001/001	010/010	---/011	011/100	---/101	100/110	101/111	
100	1	8	32	64	128	256	1024	
CLK Prescaler (CP)								
(2^0) 1				25000		6250	1562,5	
(2^1) 2				12500		3125	781,25	
(2^2) 4		50000		6250		1562,5	390,63	
(2^3) 8		25000		3125		781,25	195,31	
(2^4) 16		12500		1562,5		390,63	97,66	
(2^5) 32	50000	6250		781,25		195,31	48,83	
(2^6) 64	25000	3125		390,63	195,31	97,66	24,41	
(2^7) 128	12500	1562,5		195,31	97,66	48,83	12,21	
(2^8) 256	6250	781,25	195,31	97,66	48,83	24,41	6,1	

Error (%)						
1	8	32	64	128	256	1024
0	0	0	0	0	0	0,03
0	0	0	0	0	0	0,03
0	0	0	0	0	0,03	0,16
0	0	0	0	0,03	0,03	0,16
0	0	0	0,03	0,03	0,16	0,68
0	0	0,03	0,03	0,16	0,16	1,7
0	0	0,03	0,16	0,16	0,68	1,68
0	0,03	0,16	0,16	0,68	1,7	1,72
0	0,03	0,16	0,68	1,7	1,68	1,64

Timers – Example 2

- Problem: Add a second LED toggling every 440ms

Timers – Example 2

- Problem: Add a second LED toggling every 440ms

- Solution 1: Add a second timer
  - Main program initializes both timers
  - ISR of each timer takes care of one task
- Solution 2: use a single timer
  - Main program initializes the timer
  - ISR implements a common time base and toggles each LED at the proper moments in time

Timers – Example 2, solution 2

- Common time base using an 8-bit timer (why?)
  - Time bases: divisors of (100,440) = {20,10,5,**4**,2,1}
  - LED1 toggles each 25 time ticks (25x4=100)
  - LED2 toggles each 110 time ticks (110x4=440)

```
ISR(TIMER2_COMPA_vect) {
  if (time1) time1--;
  else {
    PORTB = PORTB ^ (1<<LED1);  // toggle LED1
    time1=220;
  }

  if (time2) time2--;
  else {
    PORTB = PORTB ^ (1<<LED2);  // toggle LED2
    time2=50;
  }
}
```

```
void tc2_init(void) {
  TCCR2B = 0;             // Stop TC2
  TIFR2 |= (7<<TOV2);     // Clear pending intr
  TCCR2A = 0;             // Mode NORMAL
  TCNT2 = 0;              // Load BOTTOM value
  OCR2A = T2TOP;          // Load TOP value
  TIMSK2 = (1<<OCIE2A); // Enable COMPA intr
  TCCR2B = 4;             // Start TC1 (TP=256)
}
```

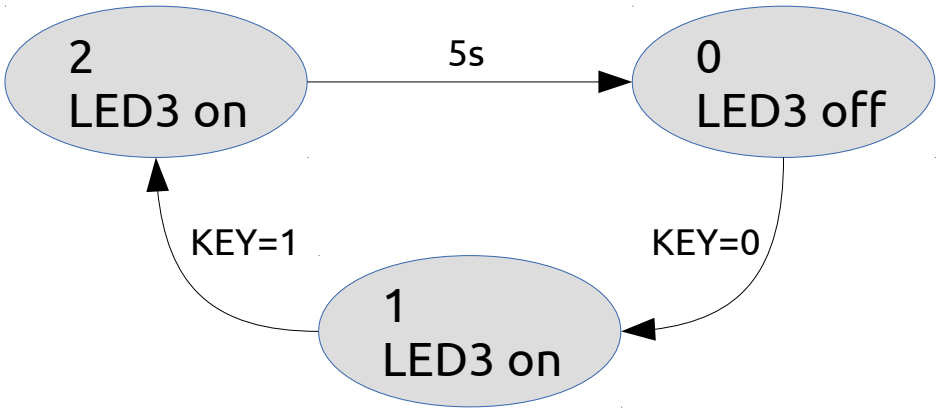
Timers – Example 3

- Problem: Whenever a key is pressed, a third LED switches on for 5s.
  Additional activations of the key during that period have no effect.

Timers – Example 3

- Problem: Every time a key is pressed, a third LED switches on for 5s.
  Additional activations of the key during that period have no effect.

- Solution: do it on main()
  - Simple state machine:
  - KEY handled using an external interrupt or by just testing the port



Example 3, solution

```
#define DEBUG

#include <avr/io.h>
#include <avr/interrupt.h>

#ifdef DEBUG
#include "serial.h"
#endif

#define LED1 PB5
#define LED2 PB4
#define LED3 PB3

#define KEY PB2

#define T1TOP 1250              // CTC mode
#define T1BOTTOM 65536-1250     // NORMAL mode

#define TIME1VAL 5      //20x5=100
#define TIME2VAL 22     //20x22=440
#define TIME3VAL 250    //20x250=5000

uint8_t state=0, ostate=1, nstate=0;
uint8_t time1,time2,time3=TIME3VAL;
```

```
/*******************************************
 * Timer 1 initialization in CTC mode
 *******************************************
 * - Stop TC1 and clear pending interrupts
 * - Define mode of operation, BOTTOM  & TOP
 * - Set the required interrupt mask
 * - Start timer with the proper prescaler
 *******************************************/
void tc1_init(void) {
  // Stop TC1 and clear pending interrupts
  TCCR1B = 0;
  TIFR1 |= (7<<TOV1);

  // Define mode CTC
  TCCR1A = 0;
  TCCR1B = (1<<WGM12);

  // Load BOTTOM and TOP values
  TCNT1 = 0;
  OCR1A = T1TOP;

  // Enable COMPA interrupt
  TIMSK1 = (1<<OCIE1A);

  // Start TC1 with a prescaler of 256
  TCCR1B |= 4;
}
```

Example 3, solution

```
/*******************************************
 * Timer 2 ISR for CTC mode
 * implements three independent sw timers
 *******************************************
 * This code is executed each 20ms:
 *   - decrement each timer and
 *   - if time reaches zero:
 *        - toggles a LED
 *        - resets timer value
 *******************************************/
ISR(TIMER1_COMPA_vect) {
  if (time1) time1--;
  else {
    PORTB = PORTB ^ (1 << LED2);  // toggle LED
    time1=TIME1VAL;
  }
  if (time2) time2--;
  else {
    PORTB = PORTB ^ (1 << LED1);  // toggle LED
    time2=TIME2VAL;
  }
  if (time3) time3--;
}
```

```
/*******************************************
 * Main
 *******************************************/
int main(void) {

  // Define outputs
  DDRB |= (1<<LED1) | (1<<LED2) | (1<<LED3);

  // Define input and activate internal pull-up
  DDRB &= (~(1<<KEY));
  PORTB |= (1<<KEY);

  tc1_init();                    // Init Timer 1
  sei();          // Enable global intr service

  #ifdef DEBUG
  usart_init();
  printf_init();
  printf("\n\n\rHello World!\n\r");
  #endif
```

Example 3, solution

```
while (1) {

    #ifdef DEBUG
    if (ostate!=state) {
      printf("%d ",state);
      ostate=state;
    }
    #endif

    switch (state) {
      case 0:{
        PORTB &= (~(1<<PB5));
        if (!(PINB & (1<<KEY))) nstate=1;
      }break;

      case 1:{
        PORTB |= (1<<PB5);
        if (PINB & (1<<KEY)) {
          nstate=2;
          time3=TIME3VAL;
        }
      }break;
```

```
      case 2: {
        PORTB |= (1<<PB5);
        if (!time3) nstate=0;
        //if (!(PINB & (1<<KEY))) time3=TIME3VAL;
      }break;

      default:
        nstate=0;
        break;
    }
    state=nstate;
  }
}
```

PWM generation

- Basic concepts here
- Associated pins here
- What happens:
  - Pins OCA/B will change state depending on the count and values of registers OCRA/B
  - Interrupts not needed

- If TCNT=OCRA:

  Pin OCA will change

- If TCNT=OCRB:

  pin OCB will change

- Change configurable by bits COMA and COMB

PWM behaviour

Mode	WGM3	WGM2	WGM1	WGM0	Mode (TC0/TC2)	Mode (TC1)
0	0	0	0	0	Normal (255)	Normal (65535)
1	0	0	0	1	Slow PWM (255)	Slow PWM (255)
2	0	0	1	0	CTC (OCRnA)	Slow PWM (511)
3	0	0	1	1	**Fast PWM (255)**	Slow PWM (1023)
4	0	1	0	0	Reserved	CTC (OCR1A)
5	0	1	0	1	Slow PWM (OCRnA/B)	Fast PWM (255)
6	0	1	1	0	Reserved	Fast PWM (511)
7	0	1	1	1	**Fast PWM (OCRnA/B)**	Fast PWM (1023)

Mode	COM0A1	COM0A0	Operation
0	0	0	Normal port operation, OC0A disconnected from timer
1	0	1	Toggle OC0A on Compare Match
2	1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM
3	1	1	Set OC0A on Compare Match, clear OC0A at BOTTOM

Example 4 - Variable duty cycle

```
/*******************************************
 * Timer 2 initialization in Fast PWM mode 7
 *******************************************
 * - Stop TC2 and clear pending interrupts
 * - Define mode of operation, BOTTOM  & TOP
 * - Define output behaviour
 * - disable interrupts
 * - Start timer with the proper prescaler
 *******************************************/
void tc2_init(void) {
  TCCR2B = 0;           // Stop TC2
  TIFR2 |= (7<<TOV2);   // Clear pending intr
  TCCR2A = (3<<WGM20)|(1<<COM2A0); // Fast PWM
  TCCR2B |= (3<<WGM02); // Set at TOP
  TCNT2 = 0;            // Load BOTTOM value
  OCR2A = T2TOP;        // Load TOP value
  TIMSK2 = 0;           // Disable interrupts
  TCCR2B = 6;           // Start TC2 (TP=256)
}
```

```
/*******************************************
 * Main
 *******************************************/
int main(void) {
  DDRB = (1<<LED);              // Outputs
  tc2_init();                   // Init Timer 2
  while (1){
    _delay_ms(50);
    OCR2A++;
  }

}
```

To further explore...

- ATmega328P datasheet (Moodle)
  - Timers: Ch. 19, 20, 22; Prescalers: Ch. 21, 22
  - Clock distribution: Ch. 13
- Application note:
  - AVR130 Setup and Use the AVR® Timers
- Most important:
  - Try all the examples at home
  - Explore the suggested variants