ROD – Realtid Og Datopsamling

Jens Dalsgaard Nielsen
Automation & Control
ES
AAU

Its all about responsiveness

Standard case

Fixed sampling rate

ex: run algorithm every 10 msec

Code must bi finished before next samplings periode

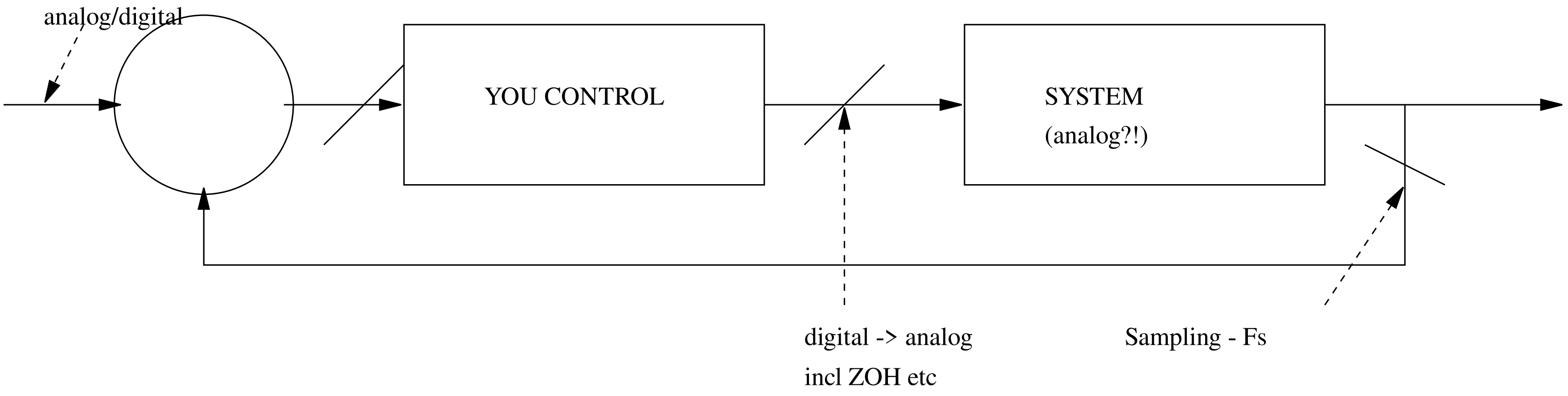
--oOo--

not to late, not to early

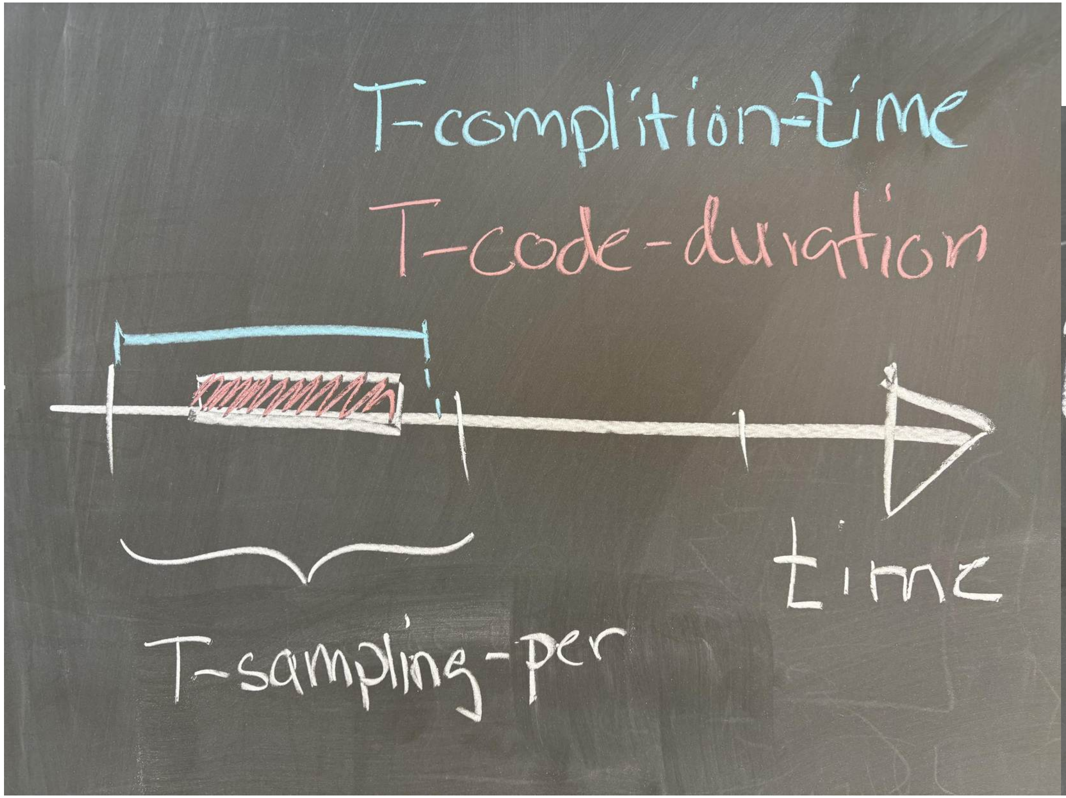
Just within the sampling period

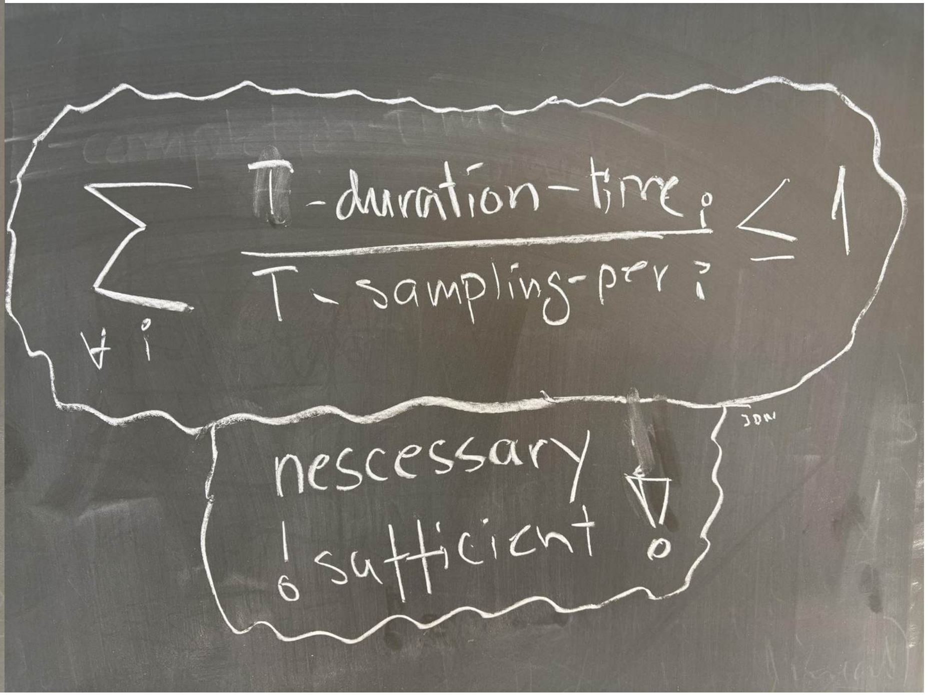
Its all about time – precise time/timing

fast rep from beginning of course



strict cyclic code - a very simple requirement
Later in ROD this will be elaborated





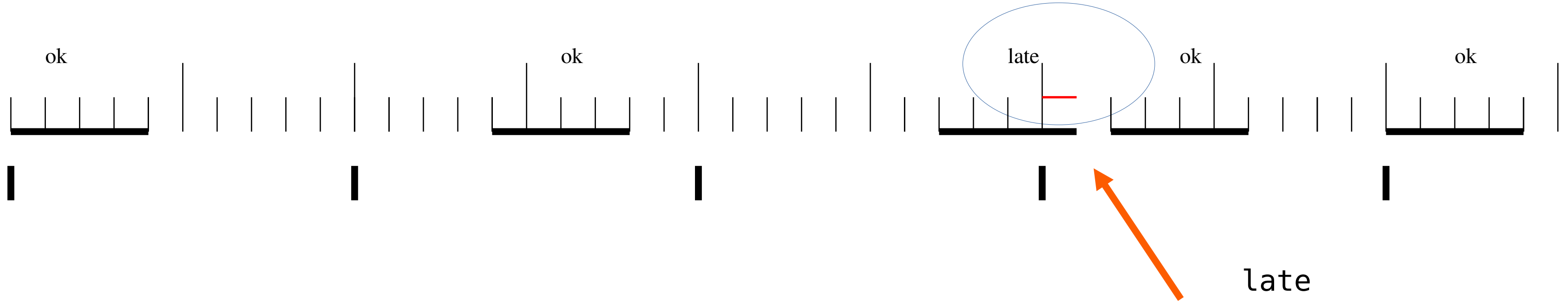
EXAMPLE: SAMPLING AT TIME OR .. TOO LATE

sample ok or too late
my/your definition
or just dont care :-(

time axis 1 msec

Ts = 10 msec   (periodof Sampling)
Td = 4 msec (duration of exection)

NB: CPU time can/will be be used other places (ISR, threads,…)
depending on your system



The (not so) obvious way ?

- Facts: Ts = 10, Td=3   index s: periode, d: duration
- Left (code) solution: Ts = 10+3 (delay(Tp))
- We need to compensate for Td
- Right (code) solution Ts = 10   (delay(Ts – Td))
- Ok ? Well ok but
- Compensate delay – well
  - Don’t take ISR etc into acount code
- Changes of duration in yourCode must be reflected in de
  - Test and measure
  -

```
void setup() {
}

#define TS 10

void yourCode() {
  // takes 3 msec
}

void loop() {
  yourCode();
  delay(TS);
}
```

```
void setup() {}

#define TS 10
#define TP  3

YourCode()
{

}
void loop() {
  yourCode();
  delay(TS - TD);
}
```

The primitivo
Cyclic execution of code within sampling constraints

- Some metrics

- Td: code-duration - effective duration of cyclic code
  - Code/algorithm is strict cyclic with a known longest duration

- Ts: sampling-per - sampling period
  - fixed so if 1 msec now in sampling frquency it will the same tomorrow

- Tc: completion-time
  - time in sampling period where code must be finish
  - if T-completion-time is equal to T-sampling-per its pushing it to the limits.
    - Must be finalised before start of next sampling period
  -

- T-code-duration is effective execution time.
- So if an interrupt (duration 2msec) is handled during running the code. Then ...
- Completion time will be prolonged with 2 msec (obvious)

A much better and stable solution

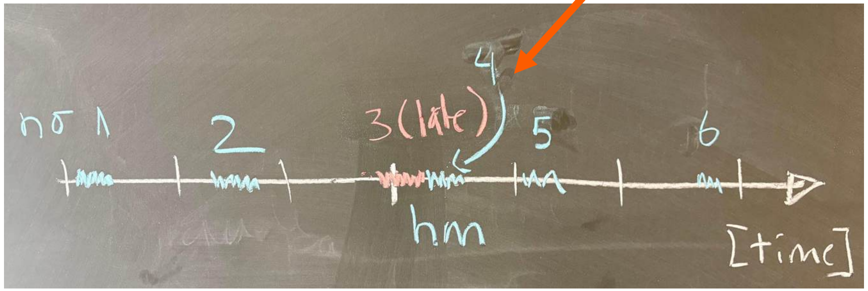
Skeleton code for arduino/stm32/PIC/… etc without RT kernel

- PRO
  - Independent of Td (duration)
  - ONLY TS (samplings period)
  - Independent of “disturbing” ISR etc
  - Robust
  - Easy

- CON
  - If to much noise you might be too late
  - Difficult to handle

- Principle
  - How long time has gone since last run
    - (millis() - tLast)

- If a sample I delayed so it comes in next period you will observe a double firing




```
void run_your_periodic_code() {
  readPorts();
  algorithms();
  writePorts();
}

#define TS = 3
unsigned long tLast;

void setup() {
  tLast = 0;  //uint max
}

void loop() {

  if (TS <= (millis() - tLast)) {
    tLast += TS;
    run_your_periodic_code();
  }
}
```

Problem 1: Integer wrap around
”everything has an end”

- Tlast += TS; grow, grow and grow, and ...

- Limit for unit16_t value [0 – 63535] (0 → 2^16 – 1) binary 1111.1111.1111.1111 + 1 = 0 :-)
- Limit for unit32_t value [0 – 4294967295] (0 → 2^32 – 1)

- (uint32_t) unsigned long (@ Arduino world)

- So what comes after 4294967295 ?

- Tlast += TS;

- Millis internal counter uint32_t
- After +489.7 days(24h) millis goes from 4294967295 to 0
  - Aka Wrap around
  - 4294967295 + 1 = 0

- What does the “if” line react?

- Same problem for 8,16,32,64 bit unsigned integers

- *(64 bit wrap around every 6,8 year)*

```
if (TS <= (millis() - tLast)) {
  tLast += TS;
  run_your_periodic_code();
}
```

Wrap around 3 bit (unsigned) int - an example

Binary (3 bit int)	Unsigned int 3 bit	*Signed int 3 bit*
111	**7**	*-1*
110	**6**	*-2*
101	**5**	*-3*
100	**4**	*-4*
011	**3**	*3*
010	**2**	*2*
001	**1**	*1*
000	**0**	*0*

Red arrow: **increment** uint value by 1 - when reaching max 111(7) next is 111+001 = 000(0) - carry evaporate – no bits for it
Blue arrow: **decrement** uint value by 1 - when reaching min 000(0) next is 000-001 = 111(7) - we can always borrow

Ex: 2: (2-1 = 1) (1 - 1 = 0) (0-1 = 7) (7-6 = 1)

(5 + 1 = 6) (6 + 1 = 7) (7 + 1 = 0) no carry (0 + 1 = 1)

001,001 etc is binary representation of the 3 bit uint3-t

Still in **3 bit** world :-)

max of “uint3_t” 2^3 -1 = 7 (wrap value)

```
  110    (6)
+ 001
-----
  111    (7)
+ 001
-----
(1)000   (0) no carry bq only 3 bit
+ 001
-----
  001    (1)
...
```

```
  010
- 001
-----
  001   (1)
- 001
-----
(1)000  (0) always possible to borrow
- 001
-----
  111   (7)
- 001
-----
  110   (6)
```

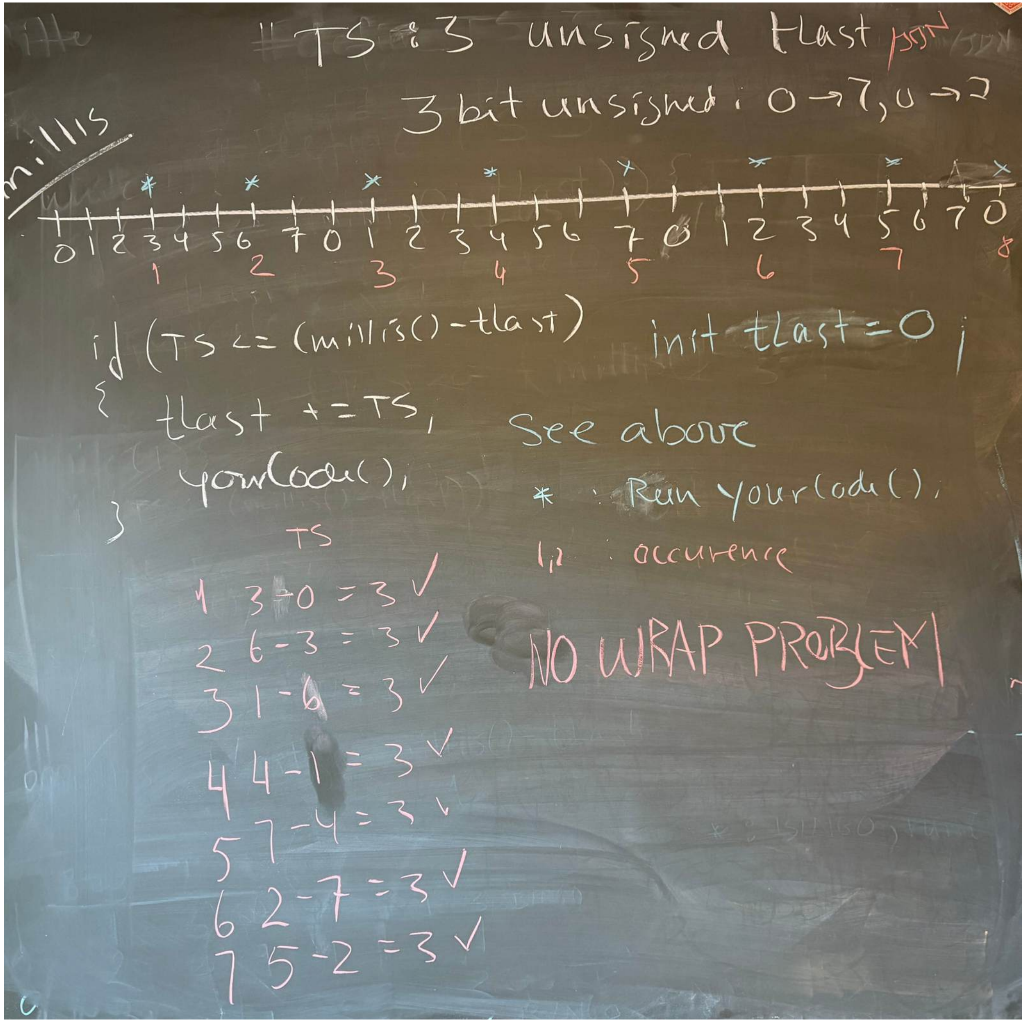
```
1 - 1 = 0  →  0 + 1 = 1
0 - 1 = 7  →  7 + 1 = 0

7 – 1 = 6  →  6 + 1 = 7
```

Wrap II

- All integers wrap around at uint max
- 1 msec ticks
  - uint8_t (unsigned char)   255   (2^8 – 1)   wrap every 256 msec
  - uint16_t   65535   (2^16 – 1)   wrap every 65,5 sec
  - uint32_t(arduino unsigned long) 4294967295   (2^32 – 1)   wrap every 1,2 hour
- Example:
  - uint8_t (unsigned char) values 0,1,2,3,4,…,255   (2^8-1 = 255 as max -   -1 bq we have a 0)
  - What after 255 ??
  - Add 1
    - 252+1 = 253 ,   253 -1 = 252   ... 255+1 = 0   , 255 + 4 = 3 3 - 4 = 255
  - Subtract 1
    - 3-1 = 2 , 2-1 = 1 , 1-1=0, 0-1 = 255
- REMEMBER UNSIGNED NUMBERS so   0-1 != -1 ..
- Same behaviour for 31,32,34,64,128,...bit – just a bit longer out in the future (time)

Wrap around – unsigned integer math - 3 bit unsigned int example

- Potential problem overflow ?
- 3 bit unsigned int 0,1,2,3,4,5,6,7,0,1,2,3,4...



Concl2 on unsigned int wrap around - OK

- No problem
- 8 bit unsigned char example on right
- Tlast = 250 tNow = 3
- 3 – 250 = 9
- Counting
- 251,252,253,254,255,0,1,2,3 = 9
- Yes no wrap around problem

```
/home/jdn> cat xx.c
#include <stdio.h>

int main()
{
unsigned char v3,v4;

v3 = 4; v4 = 250;

printf("4 - 250 (8 bit)   %u\n", (unsigned char)(v3-v4));

v3 = 250; v4 = 10;

printf(" 250 + 10 (8 bit) %u\n", (unsigned char)(v3+v4));



return 0;
}
/home/jdn> ./a.out
4 - 250 (8 bit)   10
 250 + 10 (8 bit) 4
/home/jdn> 
```

Resume – real time execution on no OS

- No kernel running
- Only your code and interrupts/timers
- One one time service: your inface: millis
  - (function micros is different)
- Is inter time between millis tick stable ?
- How does it compare to real time ?
- uint math in line 16
- coding ....

```
1  void run_your_periodic_code() {
2    readPorts();
3    algorithms();
4    writPorts();
5  }
6
7  #define TS = 3
8  unsigned long tLast;
9
10 void setup() {
11   tLast = 0;  //uint max
12 }
13
14 void loop() {
15
16   if (TS <= (millis() - tLast)) {
17     tLast += TS;
18     run_your_periodic_code();
19   }
20 }
```

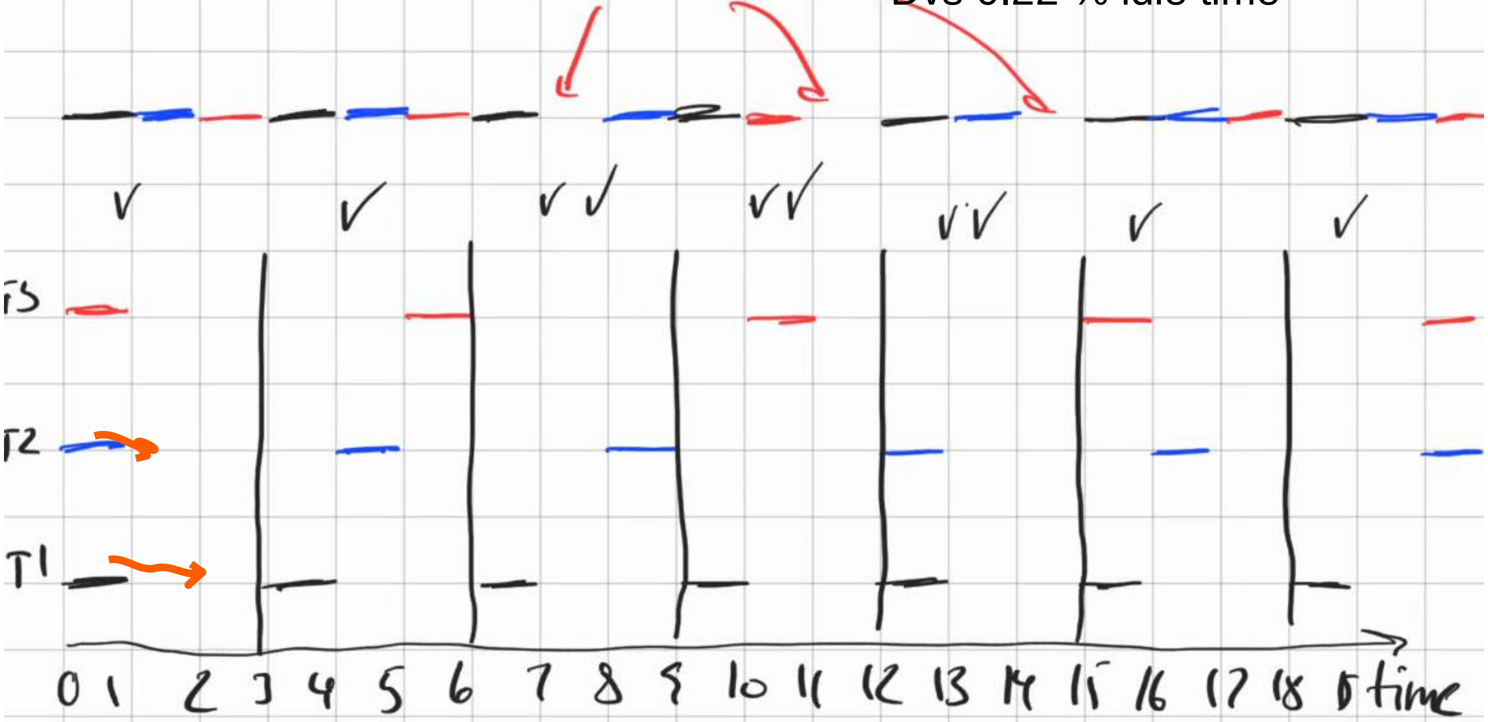
Task	T_P	T_D
1	3	1
2	4	1
3	5	1

CHECK $\sum T_D = 1 + 1 + 1 <= T_{P1}$

$3 <= 3$ ✓

1/3 + ¼ + 1/5 = 0.78 < 1

Dvs 0.22 % idle time



Other problems ?

OTHER PROBLEMs WITH TIME ?

CASE: Arduino AVR examples (UNO,MEGA,...) - bq I have one

- **The question : can we trust time and especially stable sampling time ?**

- **Why Arduino ?** because no operating system or real kernel running
- “Methods” can carefully be used on nearly every architecture

- WE NEED A TIMER

- We do have millis

- We need to check millis ?!
- obvious Question: is it stable an correct in time ?
- No kernel running
- Only your code and interrupts/timers
-
- One one time service: your inface: millis
- (function micros is different)
-
- Is inter time between millis tick stable ?
- How does it compare to real time ?

```
void run_your_periodic_code() {
  readPorts();
  algorithms();
  writPorts();
}

#define TS = 3
unsigned long tLast;

void setup() {
  tLast = 0;  //uint max
}

void loop() {

  if (TS <= (millis() - tLast)) {
    tLast += TS;
    run_your_periodic_code();
  }
}
```

For testing - leap detection code

```
unsigned long tickDelta, tickNow, tickLast;

unsigned int ticks = 0;
void setup() {
  Serial.begin(115200);
  while (!Serial);  // wait until Serial is initialised
  tickLast = millis();
}

void loop() {
  tickNow = millis();
  tickDelta = tickNow - tickLast;

  if (0 == tickDelta) {  // has a tick passed ? NOT ?!
    return;
  }
  ticks++;
  tickLast = tickNow;  // a tick has passed so update tickLast

  if (1 < tickDelta) {    // more than 1 count in between == leap millisecond
    Serial.print(ticks);  // print how many ticks in between leap milli seconds
    Serial.print(" ");
    Serial.println(tickDelta);
    ticks = 0;
  }
}
```

```
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
42 2
41 2
42 2
42 2
```

leap milli second ?

Yes

Every 41 or 42 ticks

leap milli seconds on UNO

- (42+42-41)/3 = 41,666 tick between leap seconds
-
- We got an extra millisecond as leap
  - 24 extra leap “milli seconds” every second
  - 16 MHz is “divided” down to 1024 usec used for millis (1 tick = 1024 usec)
  - So millis is to slow
  - loosing 24 usec @ 1 msec -> 1000/24 = 41.667 ticks between adding a leap msec !
  - Millis 1 msec = 1000/1024 msecs in real life
- So the internal timer is loosing approx 24 msec/sec
- XTAL discussion
  - By inspection the timer is based on the 16,000 MHz cpu xtal
  - standard (cheap) xtals has 50 ppm (parts pr million) max deviation
  - example:
    - 50 ppm => 50 msec deviation every 1 million msec ~ 1 leap msec/20 sec
    - ==> 83 leap seconds @ 24h
- Still 50ppm XTAL can not be used for a long time stable “atomic alike” watch/clock
- For sampling its ok

```
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
41 2
42 2
42 2
42 2
41 2
42 2
42 2
```

millis

Millis increment – one extra count every 41,66666 time tick

11111111111111111111111111111111111111111211111111111111111111111111111111111111111112
1111111111111111111111111111111111111111211111111111111111111111111111111111111111112
11111111111111111111111111111111111111111211111111111111111111111111111111111111111112

Can we live it ?   Yes

Better be without ?   a little bit Yes

“Only major” problem might be measuring short intervals.(one digit milliseconds)

Testing AVR/Arduino runtime system

- Is millis delivering stable time ticks ? 1,2,3,4,5,6,…
- Test it

- Is millis service accurate ? (atomic time,…)
- In advance answer is no.
- millis is based on counting on the 16 MHz XTAL which drives the cpu
- Std accuracy on cheap XTALS: max deviation is 50 ppm (parts pr million)
  - Every million seconds the second count is within +- 50
  - Every million/50 seconds the second count is within +- 1 sec
  - 10^6/50 = 5,55 hour → so for every 24 hour precision is within +- 4 seconds (do the math)
  - 1 week = 4*7 = 28 seconds
  - Not good even for timestamping
- Can we live with it ? Yes ! But not as a watch
  - +- 50 ppm on samplings freq is not much (0,05 o/oo) (*confess: this is not the whole truth but ...)*

FYI FYI FYI: Wiring.c – file is in your installation interrupt service routine for built in timer called every timertick of 1 (!) millisecond

- Hmm “leap milliseconds” in line 57

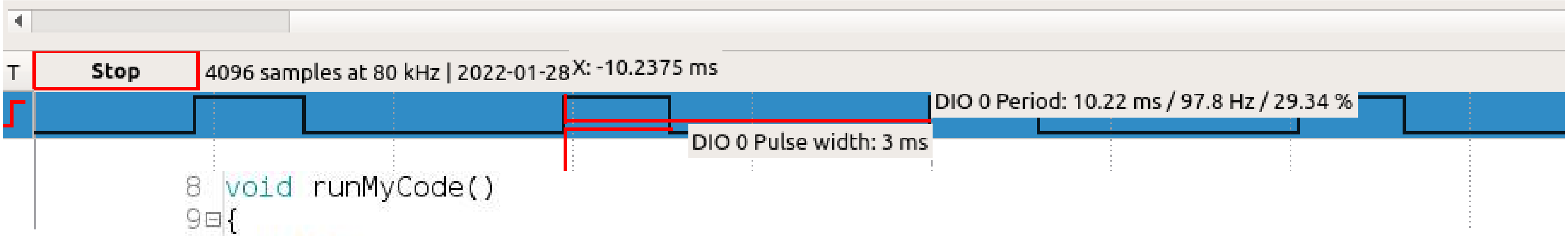
- Millis is slower than 10msec
- 10 msec is act ~10.25msec
- Thats why line 57..
- So in average millis steps in milliseconds
- So at approx every 40 msec a leap millisec is added

```
53      m += MILLIS_INC;
54      f += FRACT_INC;
55      if (f >= FRACT_MAX) {
56          f -= FRACT_MAX;
57          m += 1;
58      }
59
60      timer0_fract = f;
61      timer0_millis = m;
62      timer0_overflow_count++;
```

```
42 #if defined(TIM0_OVF_vect)
43 ISR(TIM0_OVF_vect)
44 #else
45 ISR(TIMER0_OVF_vect)
46 #endif
47 {
48      // copy these to local variables so they can be stored in registers
49      // (volatile variables must be read from memory on every access)
50      unsigned long m = timer0_millis;
51      unsigned char f = timer0_fract;
52
53      m += MILLIS_INC;
54      f += FRACT_INC;
55      if (f >= FRACT_MAX) {
56          f -= FRACT_MAX;
57          m += 1;
58      }
59
60      timer0_fract = f;
61      timer0_millis = m;
62      timer0_overflow_count++;
63 }
64
65 unsigned long millis()
66 {
67      unsigned long m;
68      uint8_t oldSREG = SREG;
69
70      // disable interrupts while we read timer0_millis or we might get an
71      // inconsistent value (e.g. in the middle of a write to timer0_millis)
72      cli();
73      m = timer0_millis;
74      SREG = oldSREG;
75
76      return m;
77 }
78
```

Pin13 (LED) on analog Discovery II CPU usage approx 30%

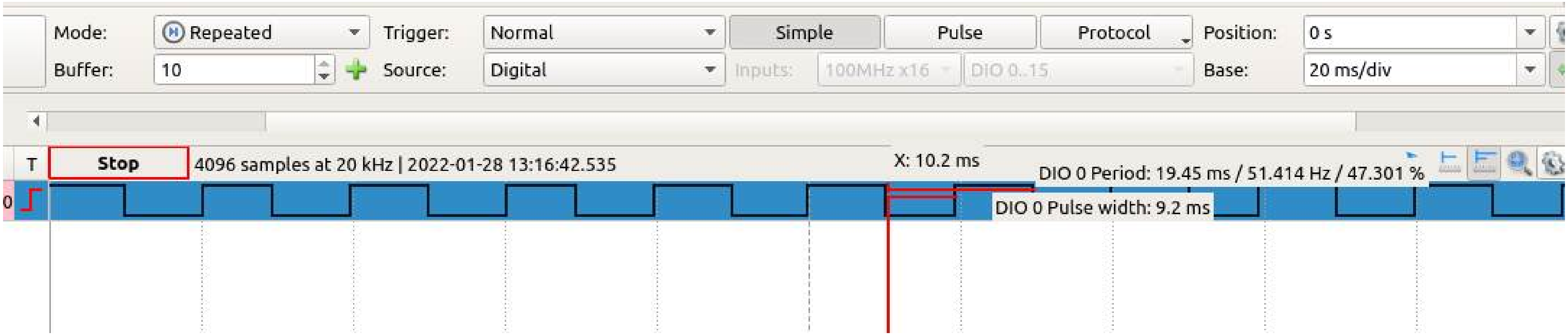
- Mission completed :-)




```
void runMyCode()
{
  delay(3);
}

const unsigned long sampleInterval = 10;  // 10 msec == 100 Hz
unsigned long  tNow=0;

loop() {
  if (tNow <= millis() ) {
    digitalWrite(13, HIGH);
    Serial.println(millis());
    tNow += sampleInterval;
    runMyCode();
    digitalWrite(13, LOW);
  }
```

TESTING time/millis HOUSTON analog discovery II (Ts10 msec)

- Hmmm - One period is only 9 msec – from time to time
- What ????
- Cant we trust millis ????   well

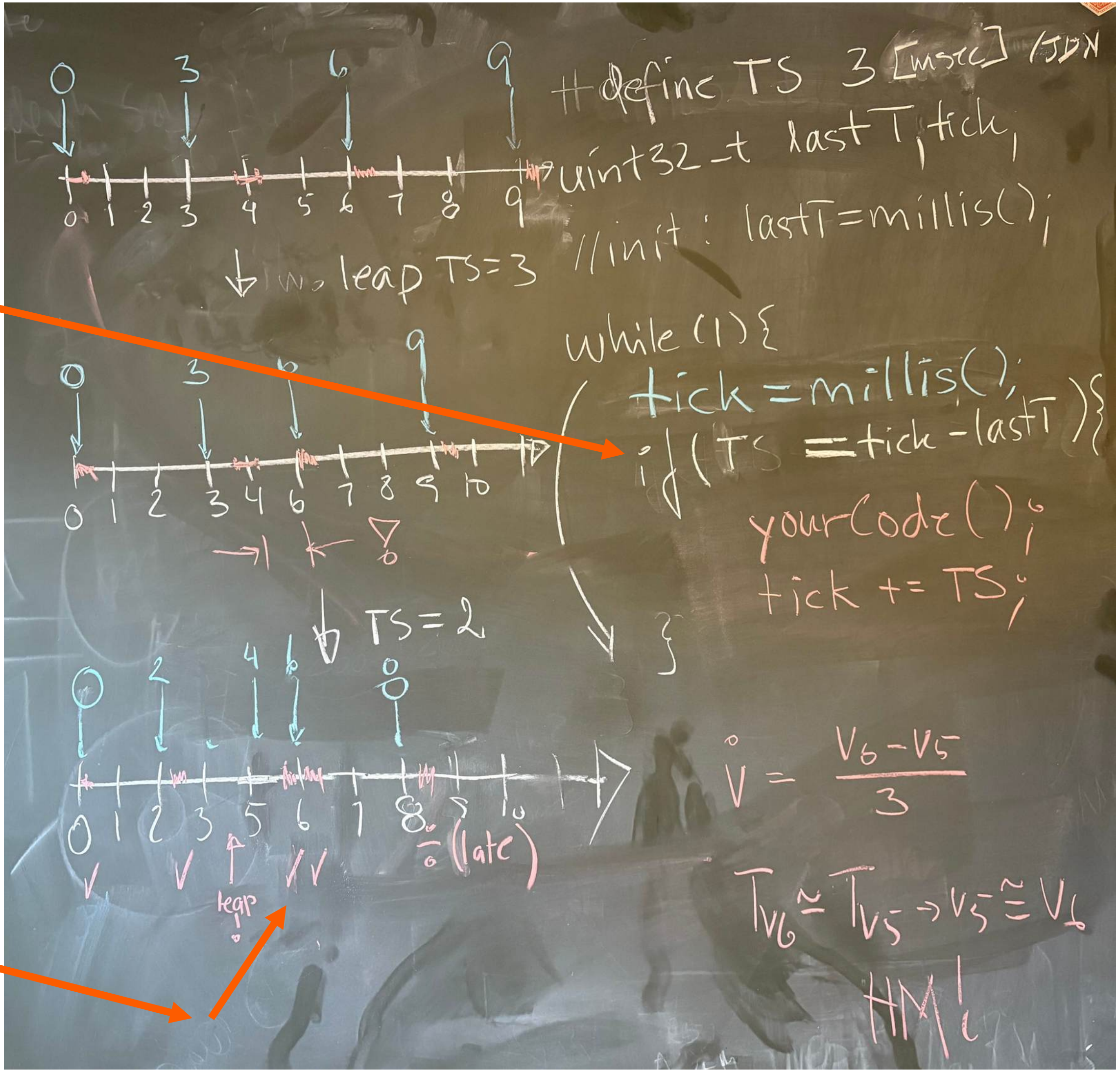


Arduino RT/time

- The Arduino timer library could have been much better concerning accuracy
- https://forum.arduino.cc/t/millisecond-precise-timing/995239/24?page=2 post #25
- https://www.best-microcontroller-projects.com/arduino-millis.html
- YOU MIGHT FIND SIMILAR PROBLEMS IN OTHER SYSTEMS
- MORALE: test & code inspection

RT CODE – not accurate

- TS < 3 - hmmm
- TS = 2 - hmmmmmmmmmmmm
- TS = 1 - just wait until tick change
- Only happens 41 times / second
- can you live it
- yes/maybe I think
- But ask your local controller !
- Leap msec @ 3→5
- At time 6 1. order derivate ~= 0 hmmm



Comment

- **Testing is important**
- **Dont trust documentation blind**
- Reading critical library code might be nescessary
- Used Arduino as case because it is simple and open
- Same procedure for other “non operating system/RTkernel) systems
- Measuring time intervals
  - Millis returns time in msec – which can up to 1 msec old ..
  - And addition to that add msec leap “from time to time”

Reference > Language > Functions > Time > Millis

millis()

[Time]

Description

Returns the number of milliseconds passed since the Arduino board began running

Hidden time eaters in your system

- Communication and buffers (RS232,I2C,...)

- Can be on other medias like SPI, Canbus,
  - Whatif I am slave and want to inform the master/controlling node – hmmm

- A communication problem when using outgoing buffers

- Maybe take a look at
  - http://storepeter.dk/3d-printer/stress-testing-5-3mbit-sec-serial-debug-stream-from-arduino
  - 5Mbit/sec printing from ISRs on a 16 MHz arduino ...

An example rs232
Hidden time eater – can break your real time

```
/*
  HardwareSerial.h - Hardware serial library for Wiring
```

```
#if ((RAMEND - RAMSTART) < 1023)
#define SERIAL_TX_BUFFER_SIZE 16
#else
#define SERIAL_TX_BUFFER_SIZE 64
#endif
```

- TX Serial port
- Example TX 100 bytes @ 9600 baud (10bit == 1Byte)
- TX time: First 64 byte ~ 0 s
  64 bytes is pushed into outgoing buffer
  Serial interrupt takes care of TX
- Last 36 (100-64) bytes **we are busy waiting** so TX takes approx
  36*1.1 msec ~= **40 msec** **UUUUUHUUUUUUUU**
- **@115200 baud** time will be 36*0.096 = **3.3 msec**
- CONCLUSION
  - Baudrate up
  - Data amount down
  - Avoid/minimize print
    if possible
  - & THINK !!!!

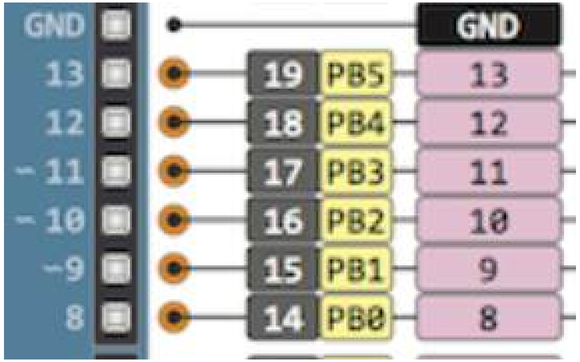
    Think before print
  - **Inspect instead of suspect**

```
tx_buffer_index_t i = (_tx_buffer_head + 1) % SERIAL_TX_BUFFER_SIZE;

// If the output buffer is full, there's nothing for it other than to
// wait for the interrupt handler to empty it a bit
while (i == _tx_buffer_tail) {
  if (bit_is_clear(SREG, SREG_I)) {
    // Interrupts are disabled, so we'll have to poll the data
```

Goto Lab - Measure time – real time

- Dig pin HIGH and LOW **and measure on it from outside**

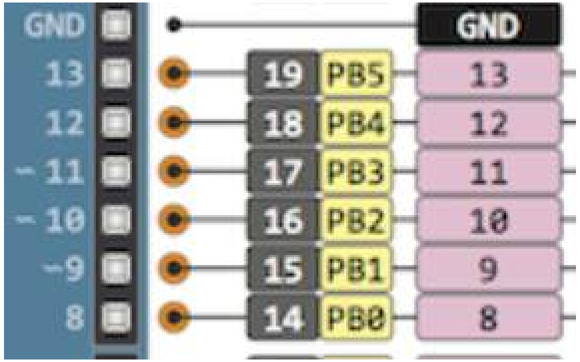
- PRO easy
- CON slow (digitalWrite)




```
// UNO
void setup() {
  pinMode(8, OUTPUT);
}

void loop() {

  // set pin 8 HIGH
  digitalWrite(8, HIGH);
  // someCode();

  // set pin 8 LOW
  digitalWrite(8, LOW);
  // measure of pin by oscilloscope, dig analyser,...
}
```

General - shown UNO specific (https://jensd.dk/doc/arduino/pins/)

- Pin 8 is bit 0 on PORTB
- digitalWrite(8,HIGH) : PORTB |= PORTB (1<<0) bitwise OR
- digitalWrite(8,LOW) : PORTB &= ~(1<<0)
- Similar operations on other architectures
- 10-20-30 times faster
- Similar on other architectures




```
// UNO
void setup() {
  pinMode(8, OUTPUT);
}

void loop() {

  // set pin 8 HIGH
  PORTB |= 1;  // (1<<0) ;
  //digitalWrite(8, HIGH);
  // someCode();

  // set pin 8 LOW
  PORTB &= ~1;// (1<<0)
  //digitalWrite(8, LOW);
  // measure of pin by oscilloscope, dig analyser,...
}
```

Even more tight

- Testing very short duration code, interruption by ISRs might influence on measurement of duration
- One Solution – turn off interrupt handling temporarily
- Only iff – your code is shorter in duration then the ISRs on your HW
- System ISRs can be
  - Timer (like 1 msec)
  - Serial ports
  - I2C ports
  - …
  -
- Advise Code must be shorter than 500 usec or so
- Some architectures has more complex ISR systems than others
- Arduino is simple – we can switch it ON/OFF by to machine instructions DI(disable) SEI(enable)
- Most/all architecture has similar instructions
- LIFE IS NOT EASY – we cant learn all about life in just one session

Measure short duration time without being disturbed target Arduino UNO – just as example

- Mea from outside

```
// UNO
void setup() {
  pinMode(8, OUTPUT);  // just the lazy way
}

#define DISABLEINTR() __asm__ volatile("cli")
#define ENABLEINTR() __asm__ volatile("sei")

void loop() {
  DISABLEINTR();
  PORTB |= 1;  // (1<<0)  pin 8(PORTB bit 0) HIGH

  yourVeryShortInTimeCode();

  PORTB &= ~1;  // (1<<0)  pin 8(PORTB bit 0) LOW
  ENABLEINTR();

  doSomethingElse();
}
```

Interrupt

- Basic CPU functionallity

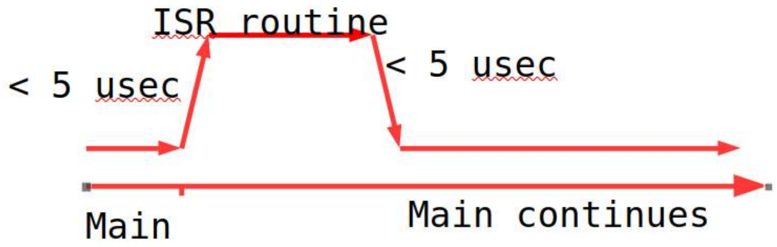
- External/internally HW can change level on a digital CPU pin to indicate …

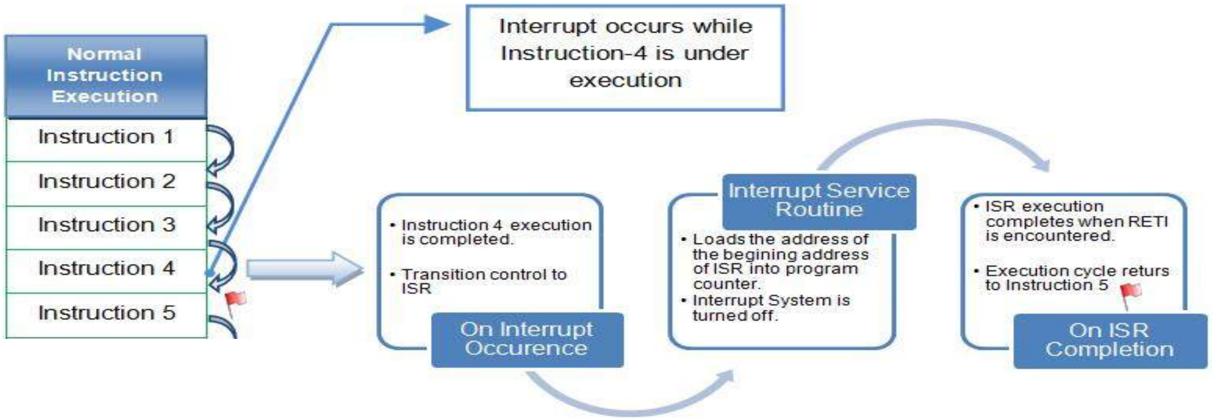
1) Disable interrupt – to prevent more interrupts from other sources
2) Save status for the interrupted code (save CPU machine registers)
3) Execute the code associated to this specific interrupt
4) Inverse of 2)
5) Inverse of 1)

Time:
- 1) and 5) normally takes 1 machine instruction (on UNO 1/16MHz)
- 2) and 4) takes 5 to 50 machine instructions depending on

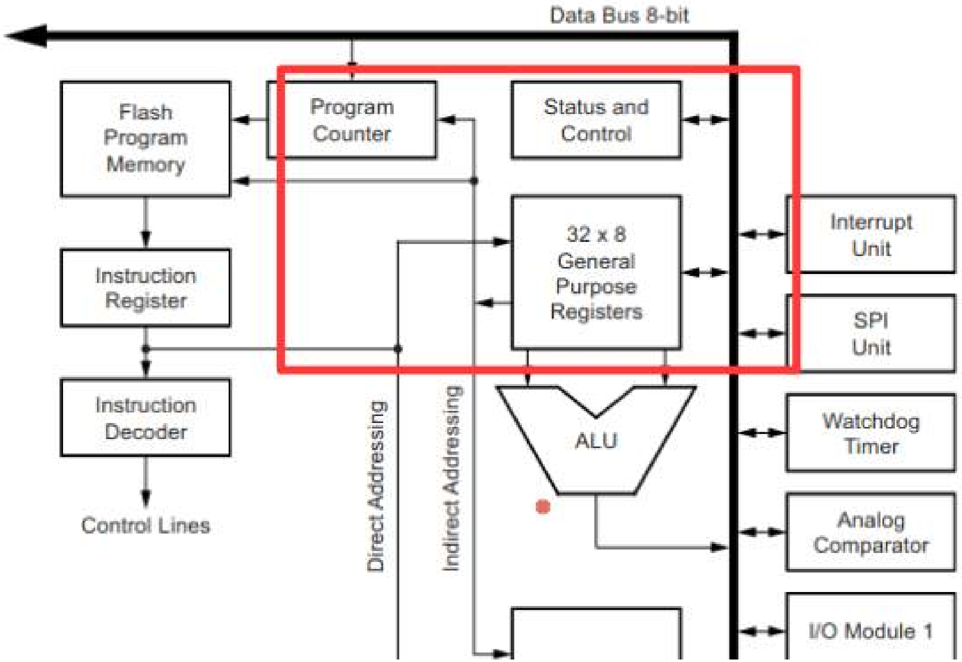
ISR

- Disable interrupt globally and save program counter (where we are in the code)
- Save context – state of machine
- “call” interrupt service routine
- Reestablish context   (revers of 2)
- Reestablish program counter (+1 : next instruction)
- Re-enable interrupt





Block Diagram of the AVR Architecture



The stack for preserving information (CPU)

- The 32 general purpose registers holds ALL info for running program.

- Where to save data/registers ?
- **Just pile them up**

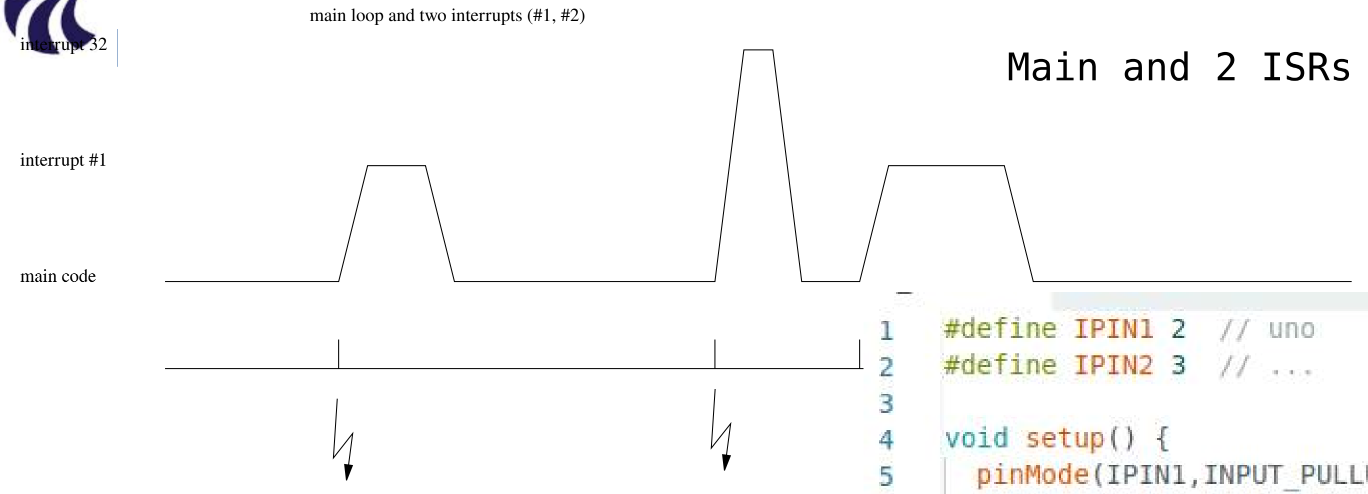
- The stack concept
  - **Push “item” on top of stack**
  - **Pop “item” on top back from stack**
  - **You cant get element nr 3 from**
  - No name space for “variables”
  - So just remember push order of elements

- In Interrupt Service Routine we do push all in start
- And pop back when leaving the ISR code :-)

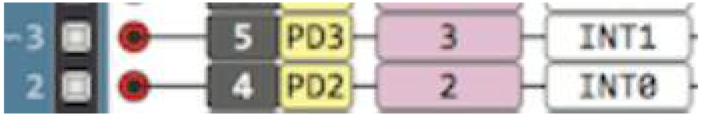
- NB we have in principle 32 registers to take care in the AVR


```
void INTERRUPT()
{
  // PUSHREGISTERS();

  // now you do whatever you want
  //
  // Interrupt is disabled when we are here
  // so running code for 1 sec means
  // we are loosing all other interrupts
  //
  // millis timer generate 1000 intr/sec
  //
  // so please below 1 msec (with a mragin)

  //POPREGISTERS();
}
```

Main and 2 ISRs on UNO

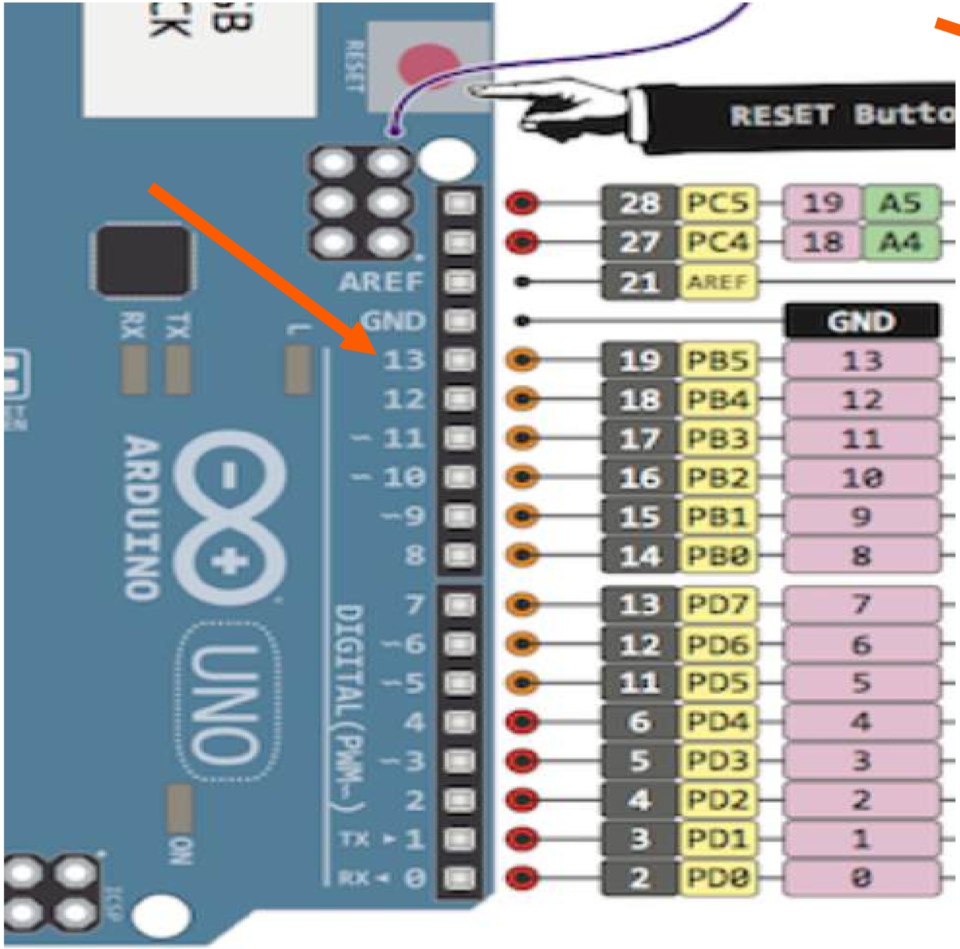


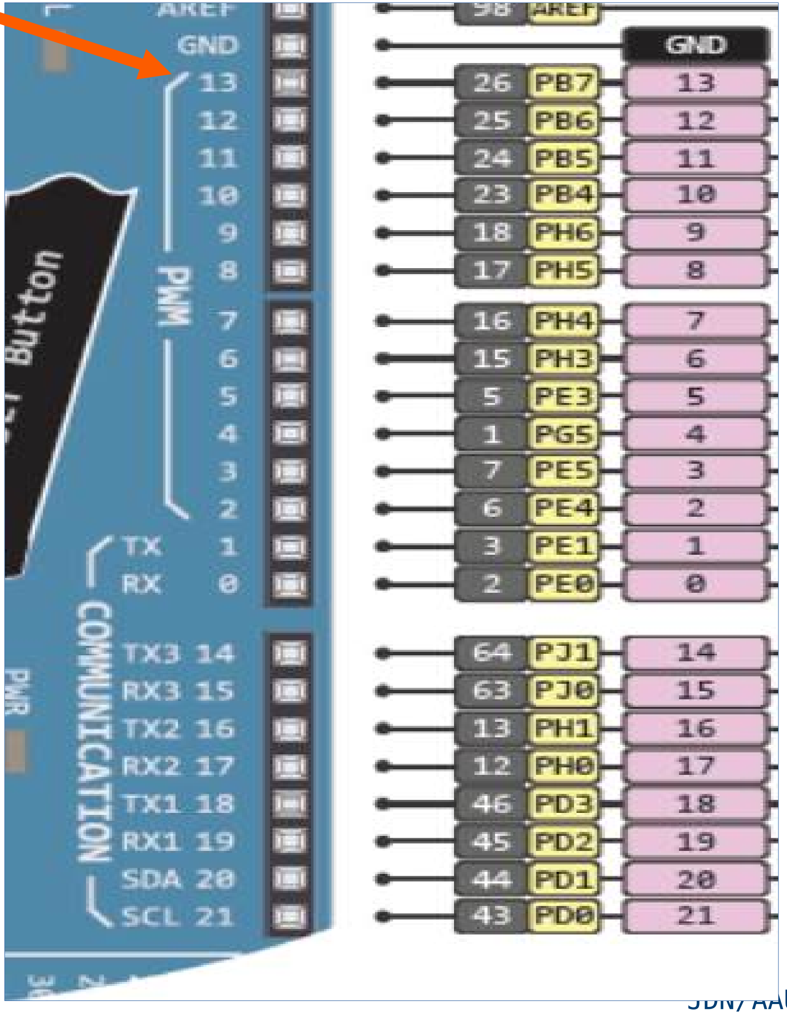
- The Arduino easy way
- Arduino uses “the Trampoline” way :-(
  - A system ISR calling yours
- Con: Small extra overhead because runtime system handles interrupts as an extra SW layer
- Pro: easy to use and portable code




```
#define IPIN1 2  // uno
#define IPIN2 3  // ...

void setup() {
  pinMode(IPIN1,INPUT_PULLUP);
  pinMode(IPIN2,INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(IPIN1), isr1, FALLING);
  attachInterrupt(digitalPinToInterrupt(IPIN2), isr2, FALLING);
}

void isr1() // called when IPIN1 goes low
{
}

void isr2() // called when IPIN2 goes low
{
}

void loop() {
}
```

UNO & MEGA (PB5 - PB7)





The AVR hardcore way

```
void setup() {
  // put your setup code here, to run once:
  pinMode(2, INPUT_PULLUP);
  pinMode(3, INPUT_PULLUP);

   // Enable interrupts INT0 & INT1 on CHANGE
  EIMSK |= (1 << INT0) | (1 << INT1); // enable int0 and int 1
  EICRA |= (1 << ISC11) | (1 << ISC01);  //int 1 nd int0 on falling edge
  //https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
  // p54
}

// EIMSK : enable interrupt mask
// EICRA : External Interrupt Control Register A
// can set pin to react on rising, falling, changing - we go for falling due to pullup on pin

volatile unsigned long int0 = 0, int1 = 0;

ISR(INT0_vect) {
  int0++;  // killroy was here
}

ISR(INT1_vect) {
  int1++;
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

to run once:

Just info – no comments -Just scratching in the surface

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..2 – Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega328P, and will always read as zero.

- **Bit 1 – INT1: External Interrupt Request 1 Enable**

When the INT1 bit is set (one) and the I-bit in the status register (SREG) is set (one), the external pin interrupt is enabled. The interrupt sense control1 bits 1/0 (ISC11 and ISC10) in the external interrupt control register A (EICRA) define whether the external interrupt is activated on rising and/or falling edge of the INT1 pin or level sensed. Activity on the pin will cause an interrupt request even if INT1 is configured as an output. The corresponding interrupt of external interrupt request 1 is executed from the INT1 interrupt vector.

Table 12-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

EICRA – External Interrupt Control Register A

The external interrupt control register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7..4 – Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega328P, and will always read as zero.

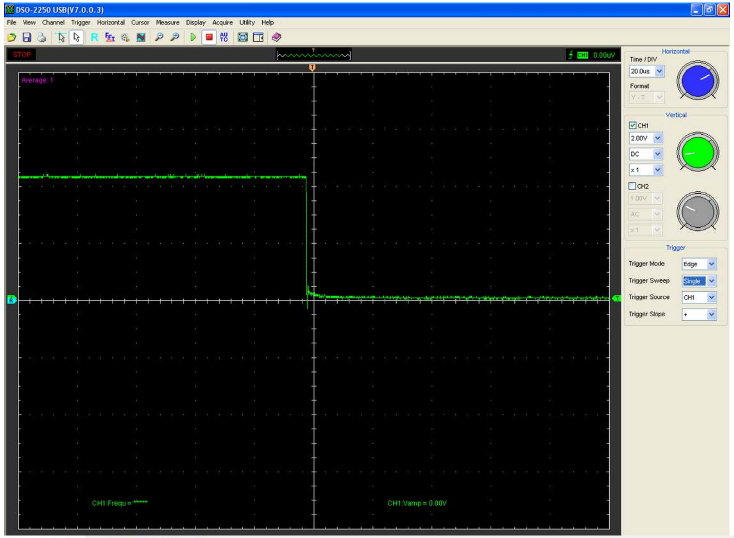
- **Bit 3, 2 – ISC11, ISC10: Interrupt Sense Control 1 Bit 1 and Bit 0**

The external interrupt 1 is activated by the external pin INT1 if the SREG I-flag and the corresponding interrupt mask are set. The level and edges on the external INT1 pin that activate the interrupt are defined in Table 12-1. The value on the INT1 pin is sampled before detecting edges. If edge or toggle interrupt is selected, pulses that last longer than one clock period will generate an interrupt. Shorter pulses are not guaranteed to generate an interrupt. If low level interrupt is selected, the low level must be held until the completion of the currently executing instruction to generate an interrupt.

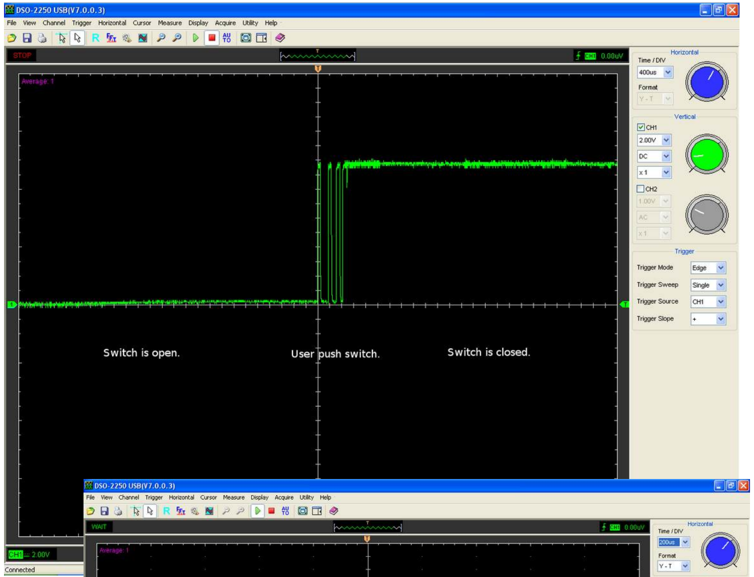
Bouncing inputs

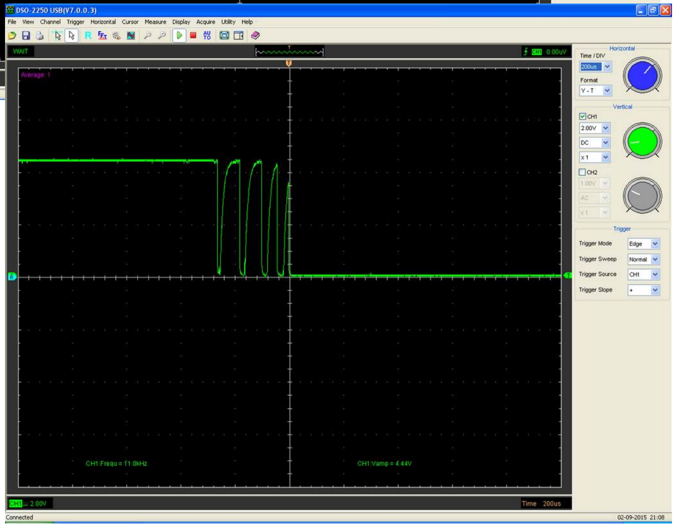
(https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/)

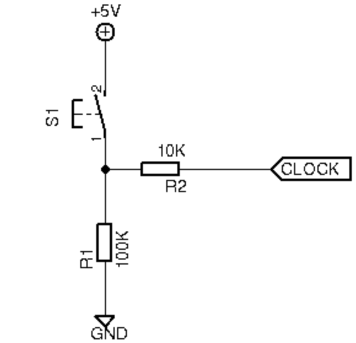
- Due to mechanics and physics:

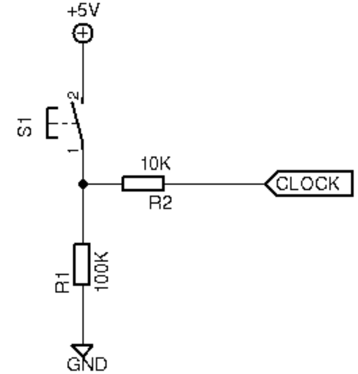


```
void setup() {
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(2), myISR,FALLING);
}

volatile unsigned long isrCount;

void myISR()
{
isrCount++;
}

void loop() {

}
```









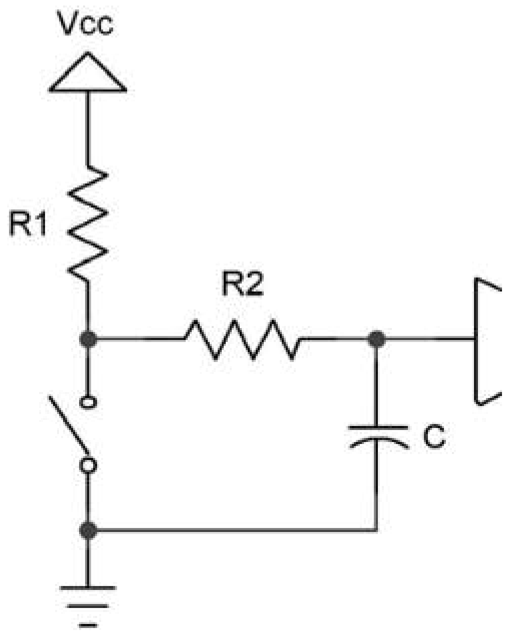
SW debounce

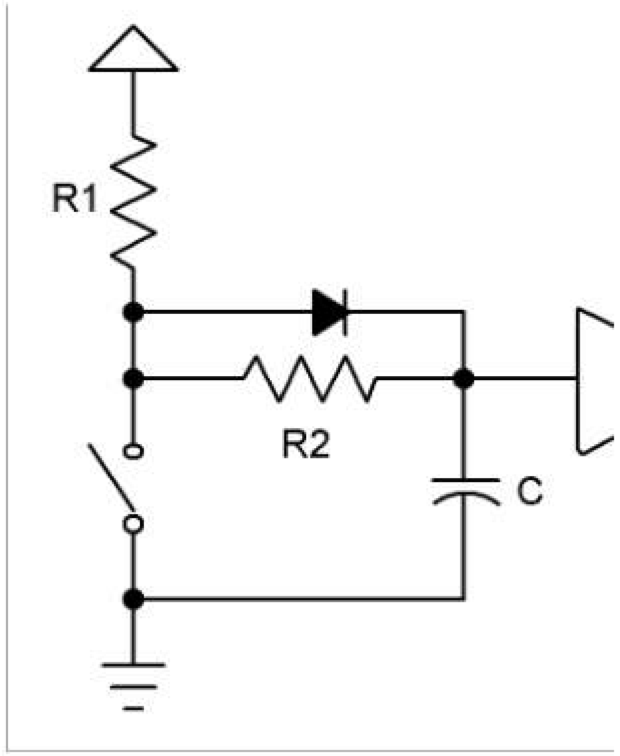
- MINWAIT might be critical

```
void setup() {
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(2), myISR,FALLING);
}

volatile unsigned long isrCount;
volatile unsigned long lastISR=0,tNow;

#define MINWAIT 100
void myISR()
{
  // SW DEBOUNCE
  tNow = millis();
  if (MINWAIT < (tNow-lastISR)) {
    lastISR = tNow;
    isrCount++;
  }
}
```

HW debounce

- https://my.eng.utah.edu/~cs5780/debouncing.pdf
- And other methods.





https://my.eng.utah.edu/~cs5780/debouncing.pdf around p 14

$$V_{cap} = V_{initial}\left(e^{\frac{-t}{RC}}\right)$$

where

V_{cap} is the voltage across the capacitor at time t,

$V_{initial}$ is the voltage initially on the cap,

t is the time in seconds,

R and C are the values of the resistor and capacitor in ohms and farads, respectively.

Most of the switches I examined last month had bounce times well under 10 msec. Use 20 to be conservative.

Rearranging the time constant formula to solve for R (the cost and size of caps vary widely so it's best to select a value for C and then compute R) yields:

$$R = \frac{-t}{C\ln\left(V_{th}/V_{initial}\right)}$$

Interesting

- Well
- But id not handled gives you problem by
  - Interrupt handling
  - Polling at input
  - Etc
- Beware !

Exercises - #1

- BYOD : BringYourOwnDevice – Arduino AVR architecture UNO/MEGA/…

- We need some code which eat cpu time to mimic critical code running
- The needed code must no rely on timers only couting CPU cycles.
- Here it goes

```
void do_eat_msec(unsigned int eatTime) {
  while (10 < eatTime) {
    delayMicroseconds(10000);
    eatTime -= 10;
  }
  delayMicroseconds(eatTime * 1000);
}

// delayMicroseconds is to a for loop
```

Ex #1

- Extend the code on the right so you can run three independent realtime “yourCode”

- YourCode #1   Tp = 200 msec Td = 40 msec
- YourCode #2   Tp = 150 msec Td = 70 msec
- YourCode #3   Tp = 100 msec Td = 40 msec

- Question 1: is this feasible (hint slide #5) ?
- If not correct shorten Td for #3 to 20 and #2 to 40

- UNO use pin 8 for #1, pin 9 for #2 and pin 3 for #3

- Based on slide 27 measure code time for all of them

- Stable calue ? Same value all time or varying

- Do you see “too late” completion from time to time ?

- Forgot to say – go to lab an measure :-)

```
void setup() {
}

void yourCode() {
  // takes ?? msec
}

unsigned long tStart, tStop;

void loop() {

  tStart = millis();
  yourCode();
  tStop = millis();

  Serialprintln(tStop - tStart);

  while (1);  // stop loop
}

```

Exercise #2 - interrupt

- On UNO or MEGA or similar

- Implement one ISR which is triggered by falling edge
- 1) by attachInterupt
- 2) by the hard way

- The ISR shall execute one line : nrInt++;   // unsigned int nrCnt=0;

- Go to lab with our arduino

- Impl an interrupt as depicted in the slides and let loop print isrCount out endless

- Togle an output pin for every interrupt so we can measure reaction time

- Test with a button

- Impl a SW debounce and test – remember to takes pictures at oscilloscope mea on dig pin
  - Delay from input to toogle pin

- Impl a HW debounce and test it – remember to takes pictures at oscilloscope mea on dig pin
  - Delay from input to toogle pin

! ! ! !