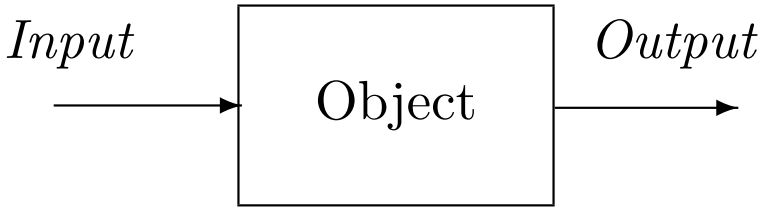
Digital systems and basics of electronics

Adam Szmigielski
aszmigie@pjwstk.edu.pl
materials: $ftp(public) : //aszmigie/SYC/ENG$

Microcontrollers in control systems - lecture 15

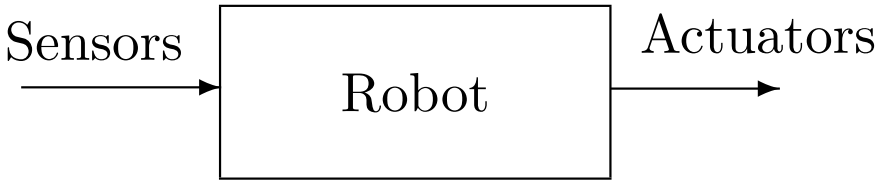
Control object



Control object - the object that performs the process (scheduled).

- *Physical object* (process, device) is an integral part of the control problem,
- For designing the control, knowledge of the physical object is necessary,
- *Input* - an input signal, controls our object,
- *Output* - the output signal, determines the state of the object feature we are interested in.

Object control on the example of a robot



- Sensory
  - distance sensors
  - orientation sensors,
  - camera
  - other.
- Actuators
  - driving mechanisms,
  - grippers and manipulators,
  - other devices.

Control goal

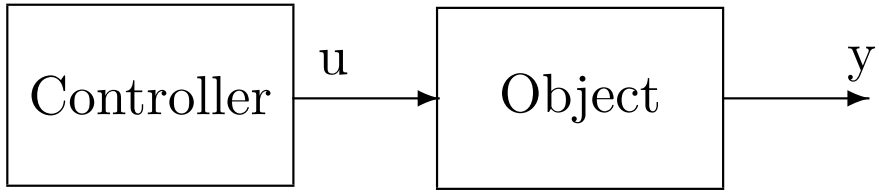
Before we select the sensors, actuators, design the architecture of the regulation system, we must define *goals* - effects to be achieved in the control process or after its completion.

- *What do we want to achieve* (energy reduction, profit increase, ...)?
- *What sizes* should you control to achieve your goals?
- What are the requirements (speed, accuracy, ...)?

Control rule

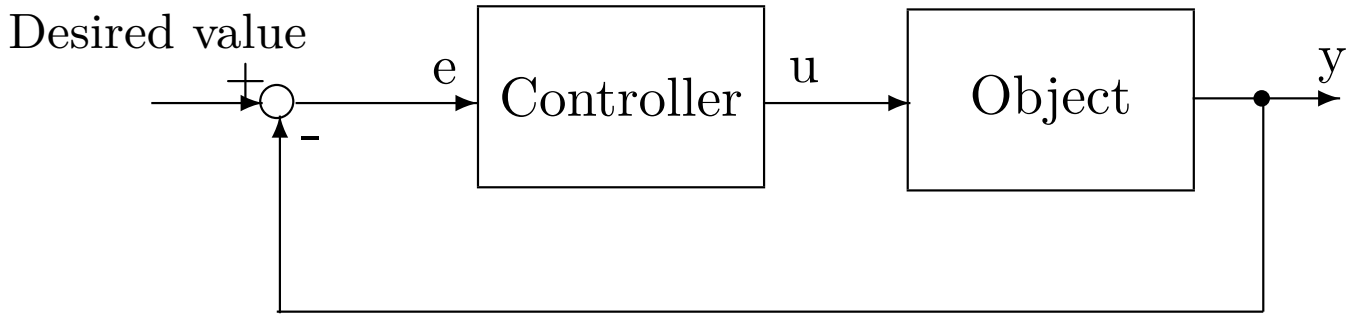
- *Principle (algorithm)* of processing information about the state of the object into executive control signals.
- Basic control principles:
  - Open system control,
  - Closed loop control system.

Open loop control system



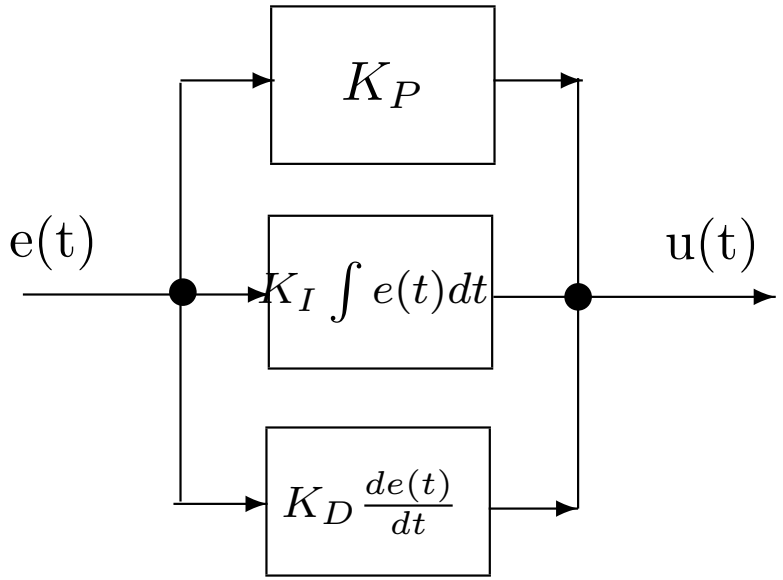
- The controller accomplishes *purpose of* by implementing *control algorithm*,
- The controller has no feedback on the course of control,
- Lack of *feedback* information makes the driver insensitive to control errors.

Closed loop system



- The control systems work with *negative feedback*,
- The purpose of the control system is to achieve setpoint by output y,
- The "how far" to reach the regulation goal is determined by *regulation error* - e,
- The regulator selects u control depending on the e control error,

PID Controller - *Proportional-Integral-Derivative*)



PID Controller mplements the PID algorithm, described by the formula:

$$u(t) = K_P \cdot e(t) + K_I \cdot \int e(t) + K_D \cdot \frac{de(t)}{dt}$$

The controller consists of *proportional elements P*, *integral I* and *differential D*. *Controller settings* - gain values $\{K_P, K_I, K_D\}$.

Discrete PID controller

- **PID controller** with discrete time it reacts in discrete moments of time.
- The interval between two praises T_N and T_{N-1} in which the regulator reacts is called *sampling period* $t_p = T_N - T_{N-1}$.
- The discrete PID controller implements the PID algorithm in the discrete version, i.e.

$$u(nT) = K_P \cdot (nT) + K_I \cdot \sum_{k=0}^{n} e(kT) + K_D \cdot (e(nT) - e(n-1))$$

- The sampling period should be as small as possible.

Characteristics of microprocessor control systems

- The use of advanced electronic technology - replacement of analog and electromechanical solutions,
- Idea of stabilizing feedback - the basic principle of operation of control systems using microprocessor systems,
- *The way the systems work* - sampling the state of the process at discrete time intervals and influencing the process at certain intervals, applying binary logic.
- *Accuracy* - a discrete signal form resistant to noise of measuring devices, possibility of transmission over long distances.
- *Cost* - technological development, decreasing microcontroller production costs.
- *New algorithms* - discrete systems can reach the set value in finite time.
- *Flexibility* - easy configuration of controllers - software.

- *Processing errors* - operations: addition, subtraction, negligible errors compared to analog circuits.

Control - example

The purpose of the control is to allow the robot to quickly move a given distance. For simplicity, we assume that the robot moves in a straight line, there are no obstacles and the elements used to build the robot are perfect.

- Control - voltage applied to engines,
- Output - the distance traveled is proportional to the diameter of the circle and the duration of the movement

Open loop control

We can only turn on the engines for a given moment.

- We need to calculate how long the engines should be on,
- Apply voltage to the motors and start the timer for the set time,
- Turn off the engines after the set time.

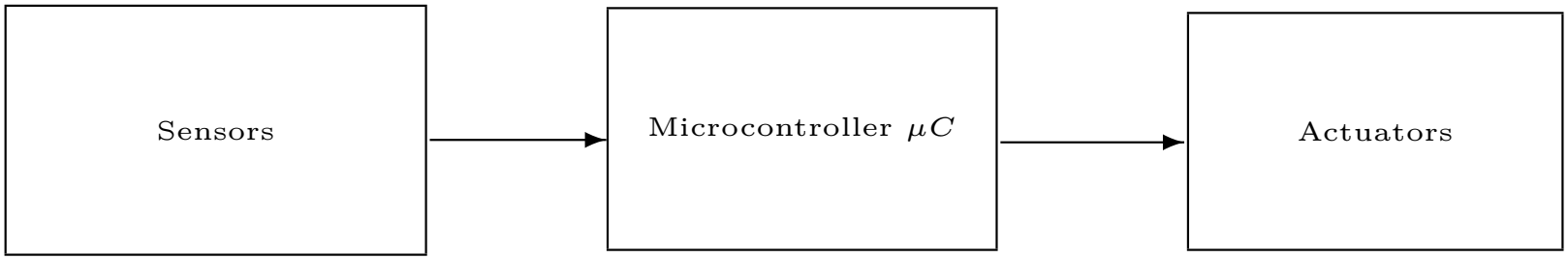
Problems:

- Sensitive to interference,
- Engine inertia. Turning off motor voltages does not stop the motor immediately (the engine is a dynamic system),
- The inertia of the engine can be included in the calculations before moving.

Closed loop control

- Specify the sampling frequency - The sampling period must be at least twice less than the time needed to travel the route.
- Select the control algorithm
- Specify the controller settings - the controller settings are determined depending on the motors, speed and control accuracy.
- Enter the setpoint for closed loop system input.

Implementation of control using μC



- Programmable μC performs control tasks
- Sensor signals require matching to μC
- Actuators - the digital signal must be converted into a control signal,
- You can easily change the control algorithm,
- It's usually difficult to add new sensors or actuators

Event handling methods

- *Interrupt* - control change, regardless of the one currently being performed program due to the occurrence of a interrupt signal. The appearance of an interrupt causes the currently executing program to be suspended and the controller to execute the interrupt handling procedure.
- *Polling* - active, periodic, sampling (checking) status external devices via the controller.

Polling

- The *polling* technique is most often used in the context of I / O device support,
- In *polling* the central computer periodically checks the status of the external device in waiting for the readiness of this device,
- *Polling* is used in situations where the computer connects to external devices to collect (refresh) data, while this cooperation takes place in *off-line* mode,
- *Polling* can be used to exchange information with external devices when, for some reason, these devices cannot start communication,
- On systems that support one task *polling* it may also apply. Most of the processor time would then be wasted on checking the readiness of the device,
- On systems that require **multiple tasks** *polling* **is not very effective** compared to interrupts.

Types of interrupts

1. *Hardware:*
   - *External* interrupt signal comes from an external source. These interrupts are used for communication with external devices.
   - *Internal* - derived from the timer
   - *Internal exceptions* - reported by the processor for signaling exceptional situations (e.g. division by zero)
2. *Software:* the interrupt service routine is called from the program code (up to communication with the operating system).

Interrupt vectors

- *The interrupt vector* is the address of the beginning of the interrupt handling,
- *Interrupt vector*, when an interrupt occurs, it is entered into the *instruction counter* - the PC register, and the contents of the PC register are put on *stack*,
- Addresses of interrupt service routines are stored in *interrupt vector table*,
- It stores the addresses of individual interrupt handling routine

Vector No.	Address	Source	Interrupt Definition
1	0x000	RESET	External Pin, Power-on, Brown-out, and Watchdog
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	PCINT0	Pin Change Interrupt Request 0
5	0x004	PCINT1	Pin Change Interrupt Request 1
6	0x005	PCINT2	Pin Change Interrupt Request 2
7	0x006	WDT	Watchdog Time-out Interrupt
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow
18	0x011	SPI, STC SPI	Serial Transfer Complete
19	0x012	USART, RX	USART Rx Complete
20	0x013	USART, UDRE	USART, Data Register Empty
21	0x014	USART, TX	USART, Tx Complete
inne	...	...	...

Maskable and non-maskable interrupts

- *Maskable interrupts* that can be blocked and unlocked in software,
- *Non-maskable interrupts* - interrupts that cannot be blocked programmatically. These are interrupts, the occurrence of which each time causes an unconditional jump to the function of handling this interrupt, e.g. reset

Interrupt handling

- Interrupt handling procedure - a sequence of orders that perform the desired response to an interrupt,
- *Main program* - a sequence of microprocessor operations (orders) carried out when there are no interrupts,
- Interrupt handling cannot make any changes to the main program.

Interrupt handling procedure

1. Identification of the reason for the interrupt (implementation can be hardware),
2. Delete the reason for the interrupt (implementation can be hardware),
3. Lock interrupt,
4. Storage on the working register stack,
5. Proper interrupt handling,
6. Restore working records from the stack,
7. Unlock interrupt,
8. Return to the suspended program.

Stack

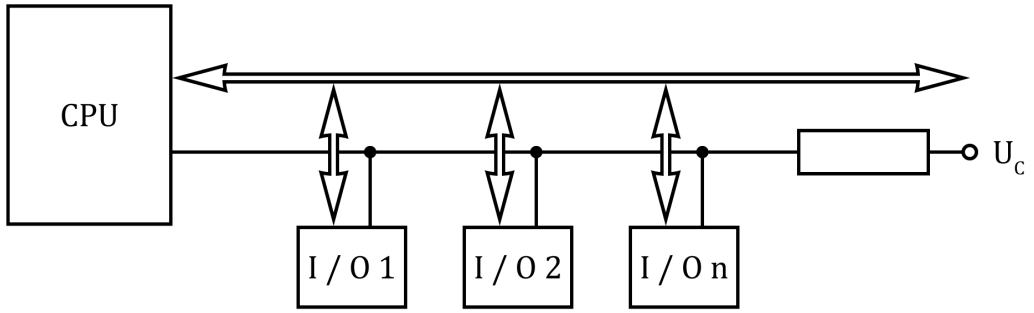
- When the interrupt is called, the address is put on the stack,
- The stack pointer should be positioned where the stack is located.

Interrupt priority

- *Interrupt priority* - diversification as to the importance (urgency) of tasks carried out by the microprocessor system,
- In particular, these tasks can be interrupt handling procedures, by varying their urgency, priority is given to each interrupt,
- For AVR, the interrupt handling system is flat (no hierarchy). All interrupts are equally valid.

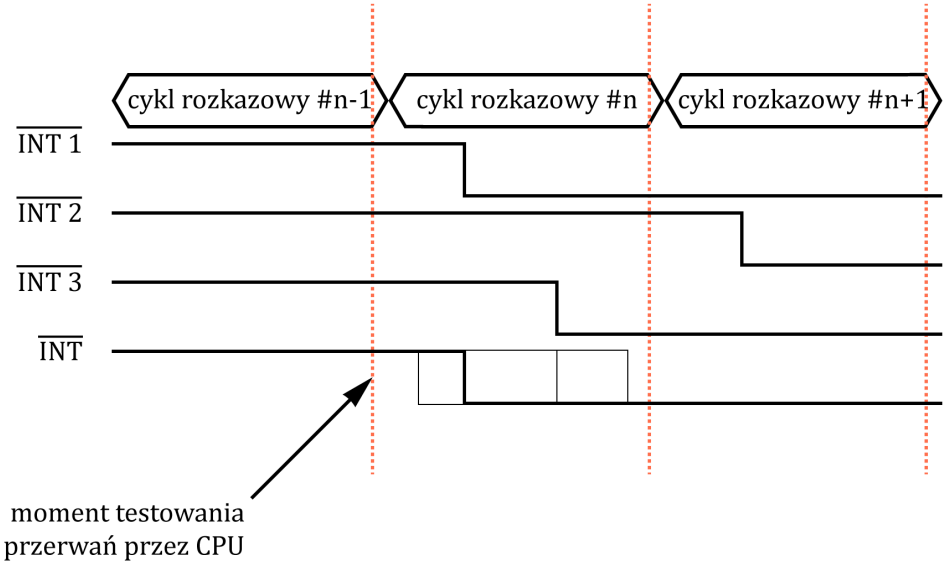
Implementation of the interrupt system

- *hardware* - *interrupt controller* determines the choice of interrupt.
- *software* - through a common interrupt handling procedure. He is an interrupt system arbiter (it recognizes the sources of current interrupts and decides about the order in which they are handled)



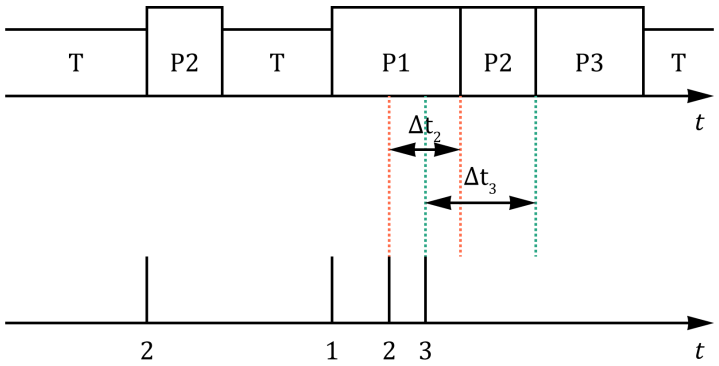
Asynchronity of interrupts

- Interrupts from various sources appear in any, independent of each other, moments of time,
- from the viewpoint of the interrupt processor 1 and 3 occurred simultaneously.



Interrupt system without priorities

- $\{\Delta t_2, \Delta t_3\}$ delays in response to interrupt handling,
- Possibility of losing an interrupt during these delays,
- The maximum waiting time for interrupt handling can be equal to the sum of the remaining interrupt handling times in the system,



Interrupt system with priorities

- There is a hierarchy of interrupt validity,
- Delays $\{\Delta t_1, \Delta t_2\}$ in response to interrupt handling,
- Interruptions with lower priorities may wait longer for service (in extreme cases they may not be served),

Types of interrupts

- Clock interrupts - measuring time,
- Interrupts from external devices - irregular,
- Interrupts from systems controlling system operation - with the highest priority. They indicate the status of work as:
  - power failure
  - CPU error / exception
  - other

Other interrupts

1. SPI, STC - Serial Transfer Complete,
2. USART, RXC USART - Rx Complete,
3. USART, TXC USART, Tx Complete,
4. USART, UDRE USART Data Register Empty,
5. ADC ADC Conversion Complete,
6. - etc.

Arduino interrupt handler structure - external interrupts only

```
const byte interruptPin = 2;
volatile byte state = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(interruptPin), blink, CHANGE);
}

void loop() {
  digitalWrite(ledPin, state);
}

void blink() {
  state = !state;
}
```

task for labs

1. Using the interrupt *int0* write a program that displays on the LED display the number of events causing the interrupt (proximity sensor),
2. Complete the task from 1 based on the querying technique,
3. The system, consisting of a servo and a proximity sensor, scans the space to detect an object (obstacle). When an object is detected (based on interrupt service *int0*), the system goes to the alarm procedure, i.e.
   - should emit a short, one-second beep,
   - display information about the azimuth (angle) of the obstacle encountered on the LED display,
   - send azimuth information to the serial link

   The alarm is reset after pressing the S1-A1 key