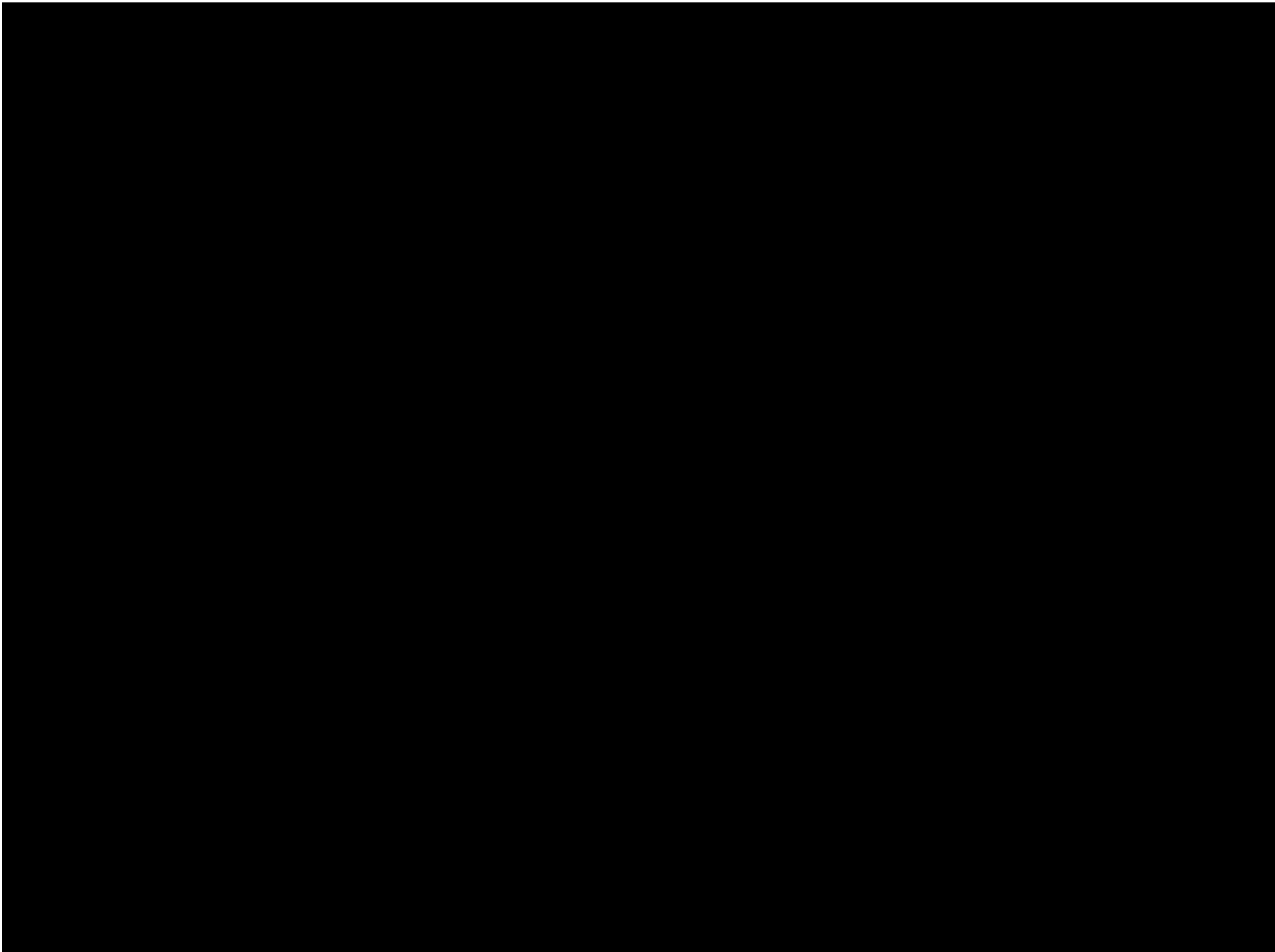
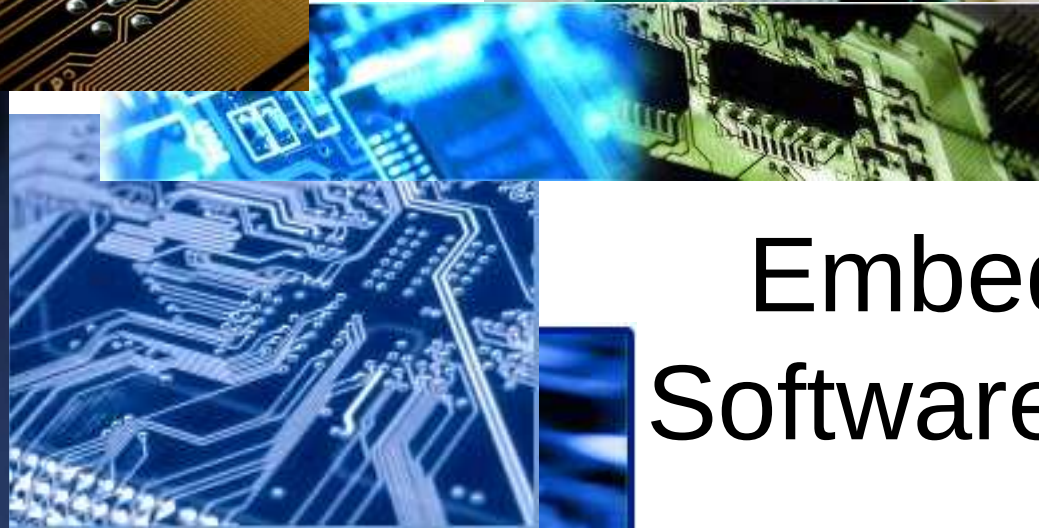
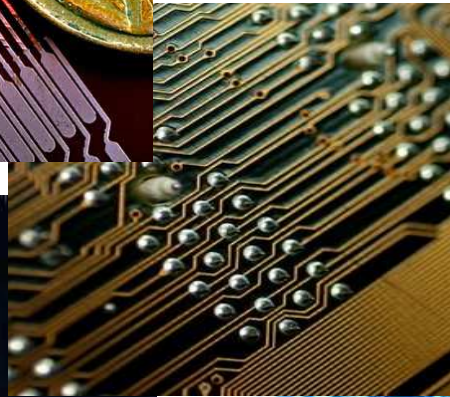
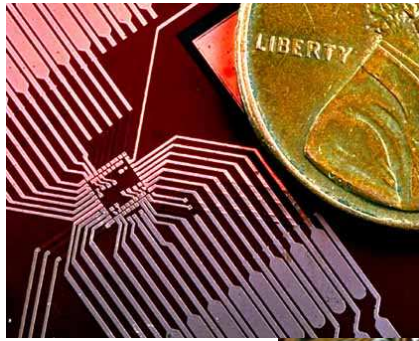






Embedded Software Cycle

Heba Sadek



Agenda

Day one

- Software Cycle overview
- Requirements Analysis
- LAB1
- Static Design
- LAB2

Day Two

- Dynamic Design
- LAB3

Day Three

- File structure (Coding)
- Testing
- LAB4

Software Cycle



To build a software system we go through **phases**.

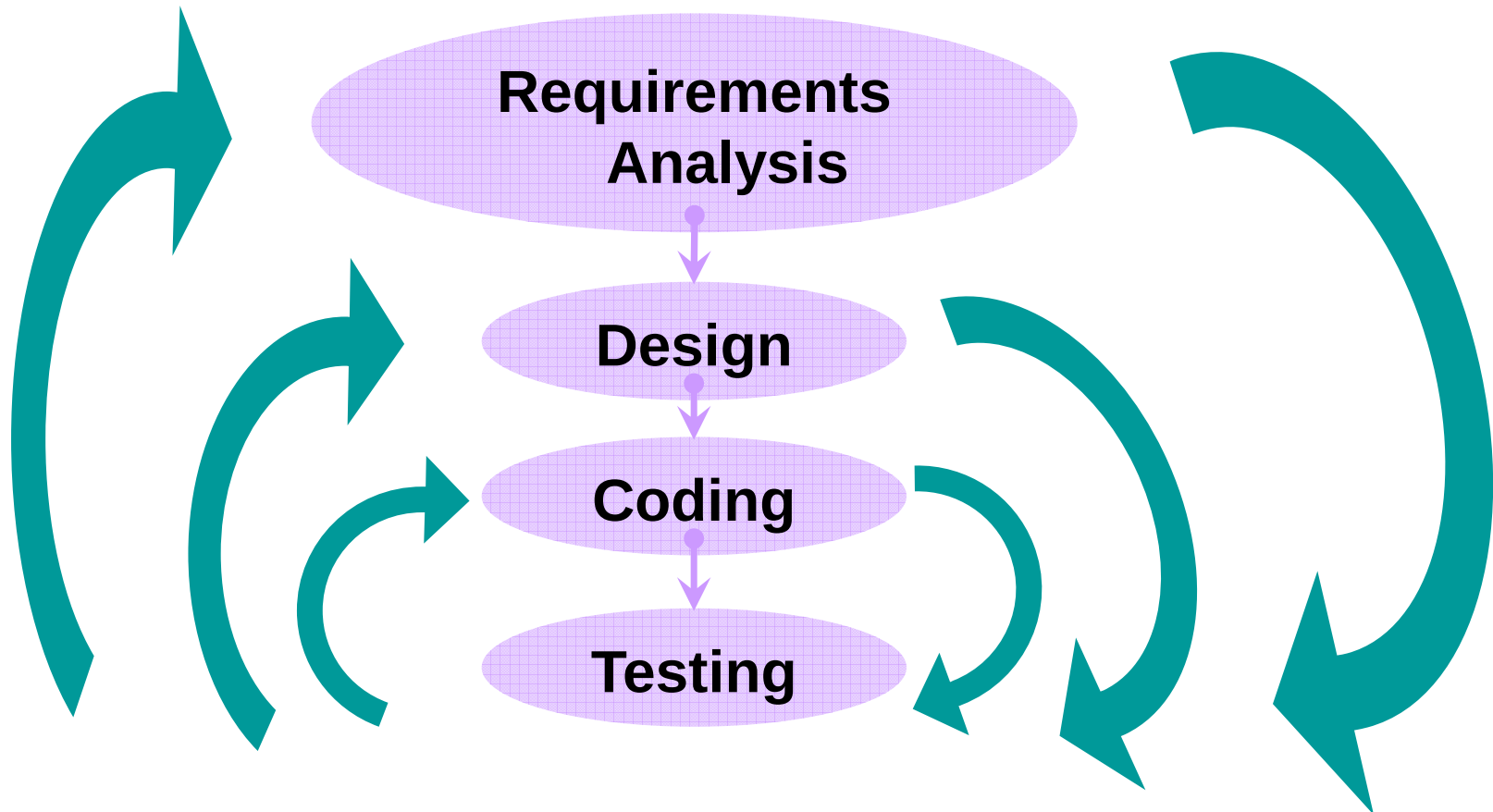


They are ordered steps that we follow to make our idea a real working software. There are different ways to do these steps and order them.

Common steps to make a software:

1. **Requirements analysis:** What will the software do?
2. **Design:** How will the software do it?
3. **Coding:** Write the software **code** from design
4. **Testing:** Make sure it really **works**

Software Cycle



Requirements Analysis

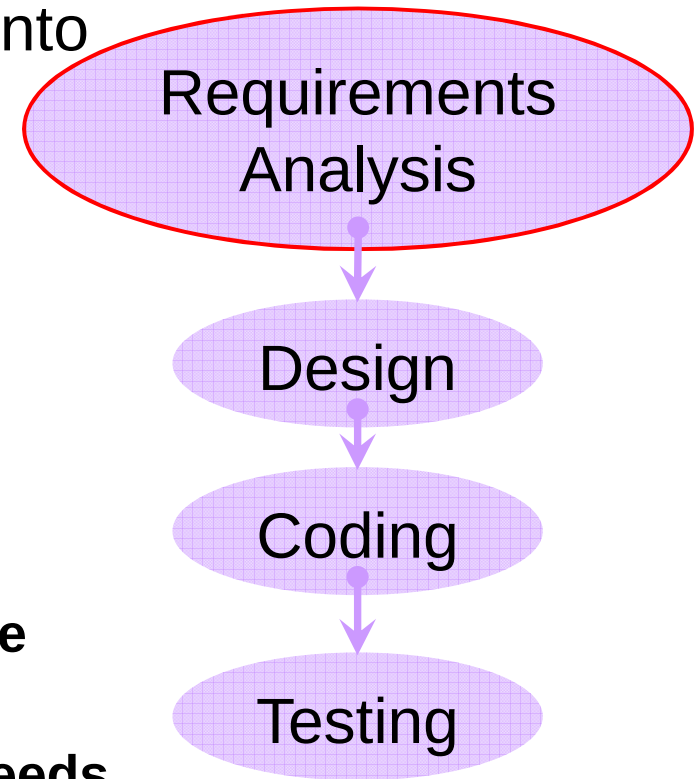
- Divide the big idea of the system into small pieces
- Each piece is a requirement

Example:

Control temperature inside a tank

The system is required to

1. **Display temperature on LCD**
2. **Be adjusted to the wanted temperature using a keypad**
3. **Activates an alarm if temperature exceeds 90c**



Design

What the software should do to satisfy the requirements

How the software handles different inputs and scenarios

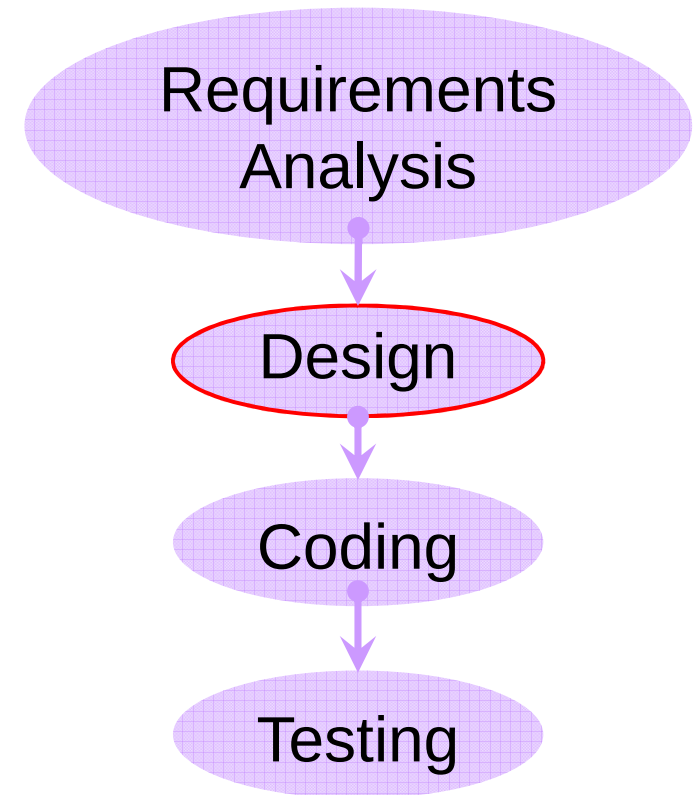
Choose hardware to use (which microcontroller)

Choose the language to use

Example:

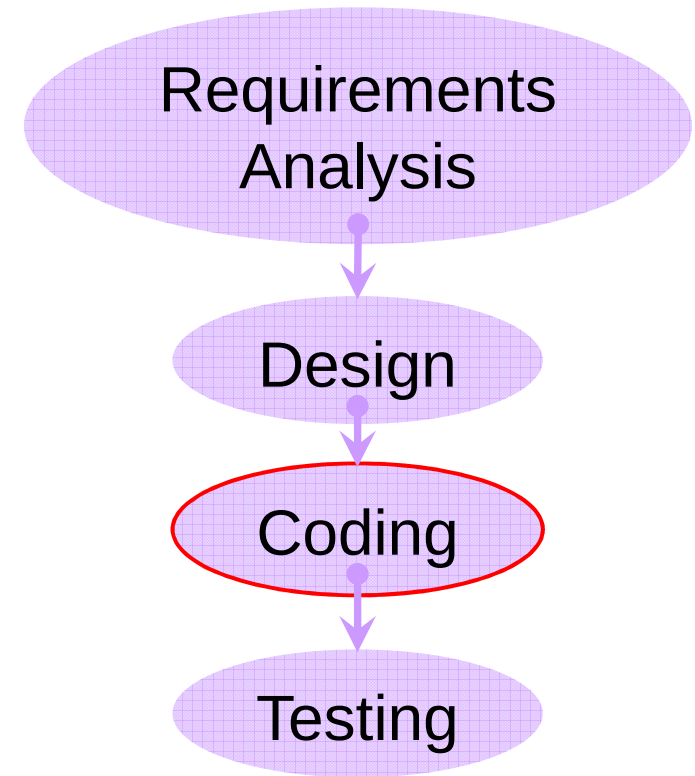
- Inputs: Sensor, Keypad
- Outputs: Heater enable, Cooler enable, LCD, Alarm enable
- Hardware: AVR – ATmega32
- Language: C

- Check temperature every 20 msec
- Calculate average over 1 sec
- Output the updated temperature to the LCD
- If the average temperature is over 90c set alarm signal to high
- When keypad is pressed read the value of wanted temperature
- If it is higher than current average turn on heater
- If it is higher than current average turn off heater



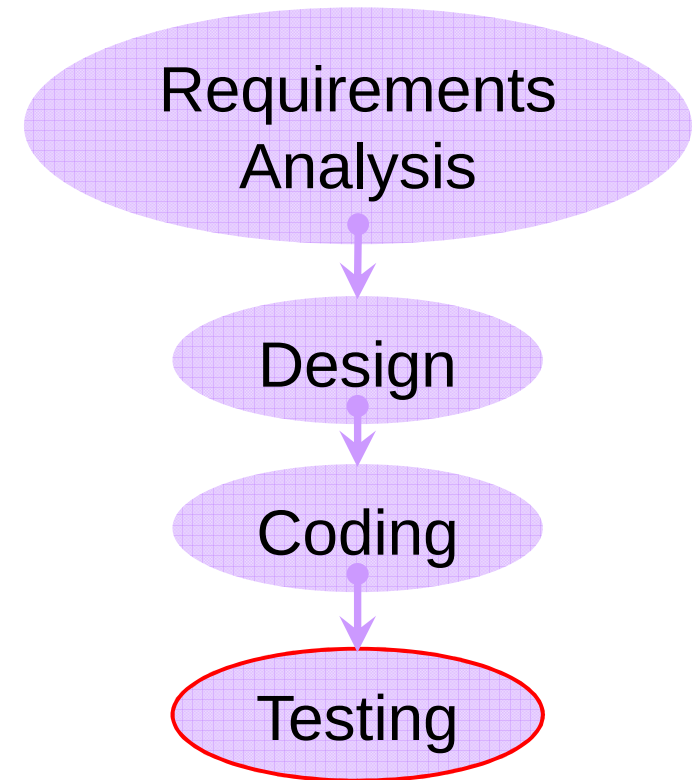
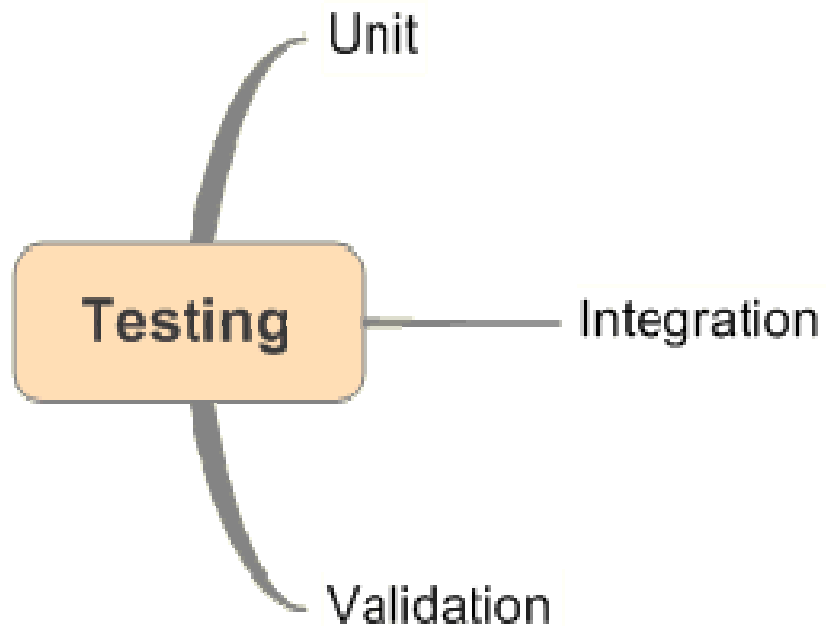
Coding

- Write actual code that runs on the hardware
- Code files are structured according to static design



Testing

- Try the software in different scenarios and check the result



Traffic Light Controller Project



Goals:

Work with UART

Work with Interrupts

Work with timers

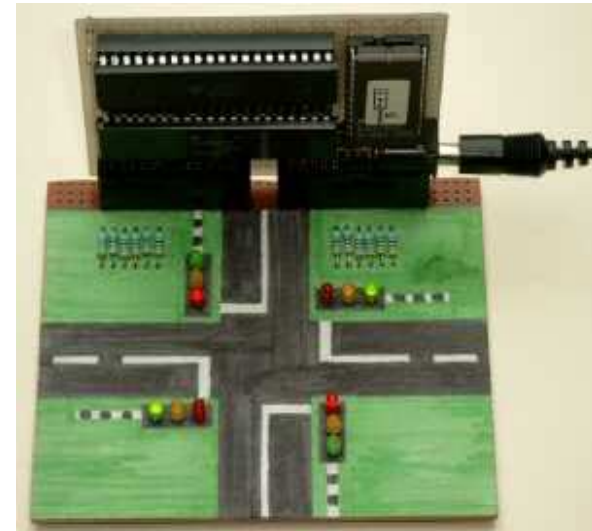
Learn design



Idea: Control traffic light according to state of the street: cars and pedestrian crossing the sign



Use AVR and C



Requirements Analysis

Traffic Light Controller

LAB 1

The system consists of

- One traffic pole with 2 LEDs (Red & Green)
- One car sensor
- One walk push button

When a walking man needs to cross the road he presses the push button, the traffic light should switch to RED for cars and stay for some time.

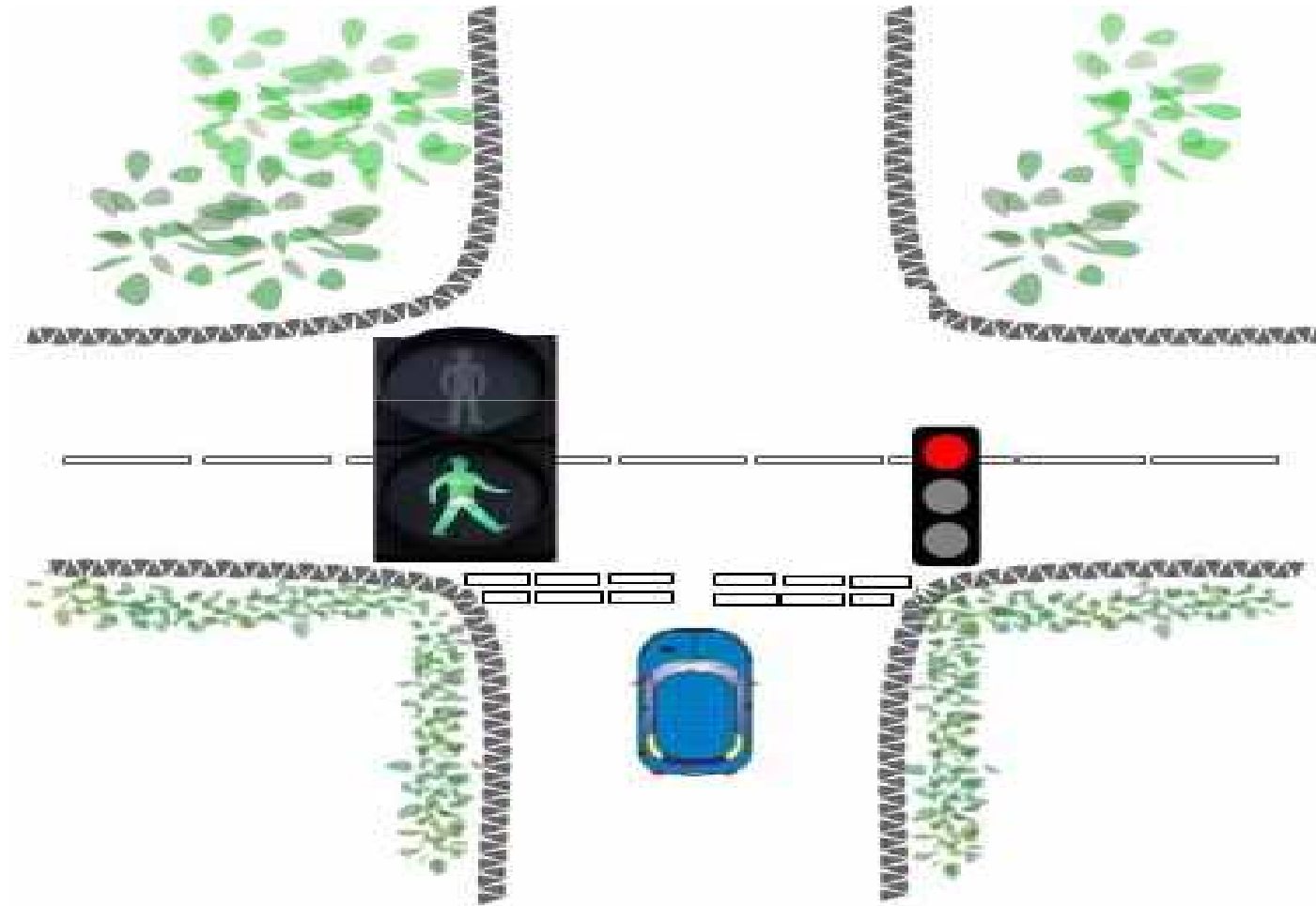
When a car is present at the crossing the car sensor detects it,
The traffic light should switch to GREEN for cars and stay for some time.

If WALK button is pressed and a car is detected the walking man is allowed to cross first then the car

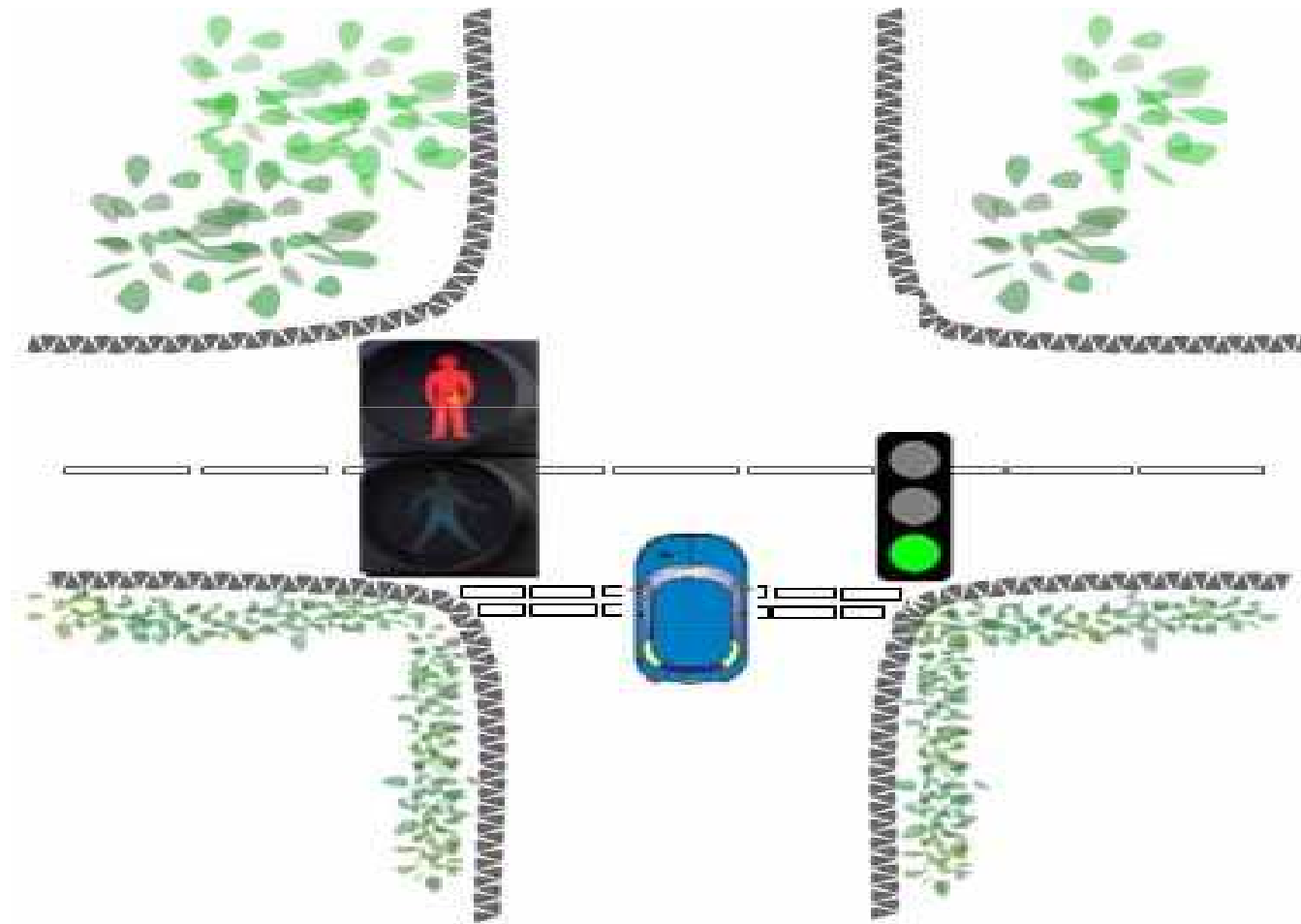
The system can be connected to a PC to check its status

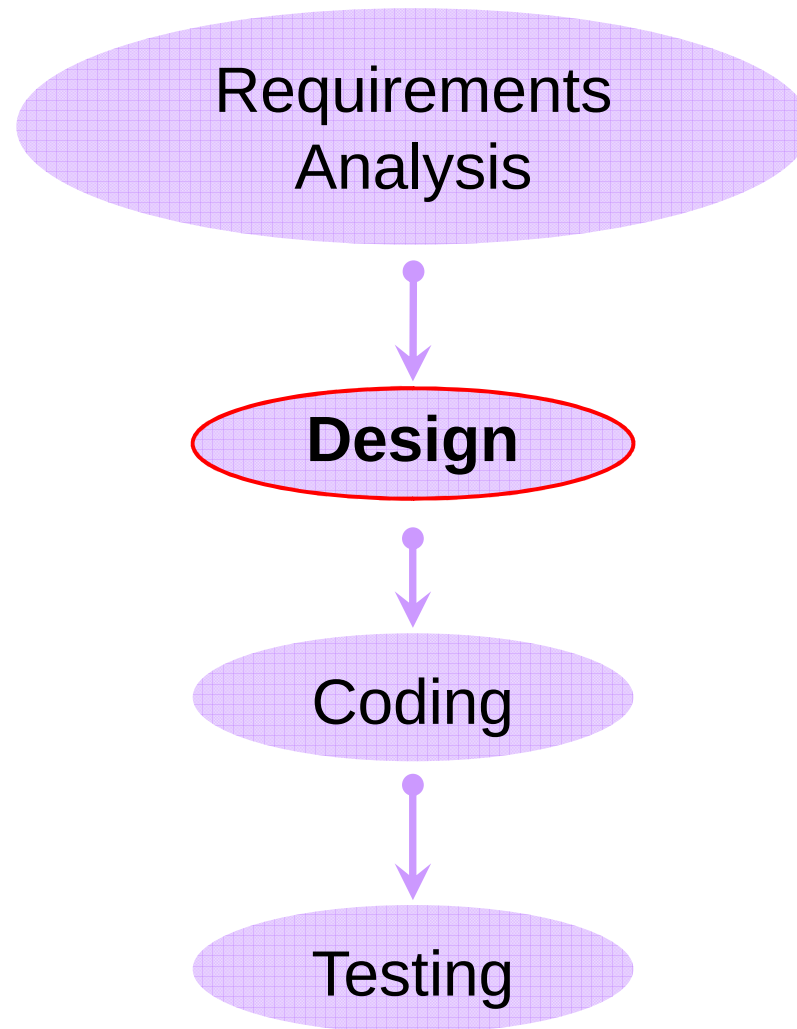
Write system requirements in the form of clear separate points

Traffic Light Controller

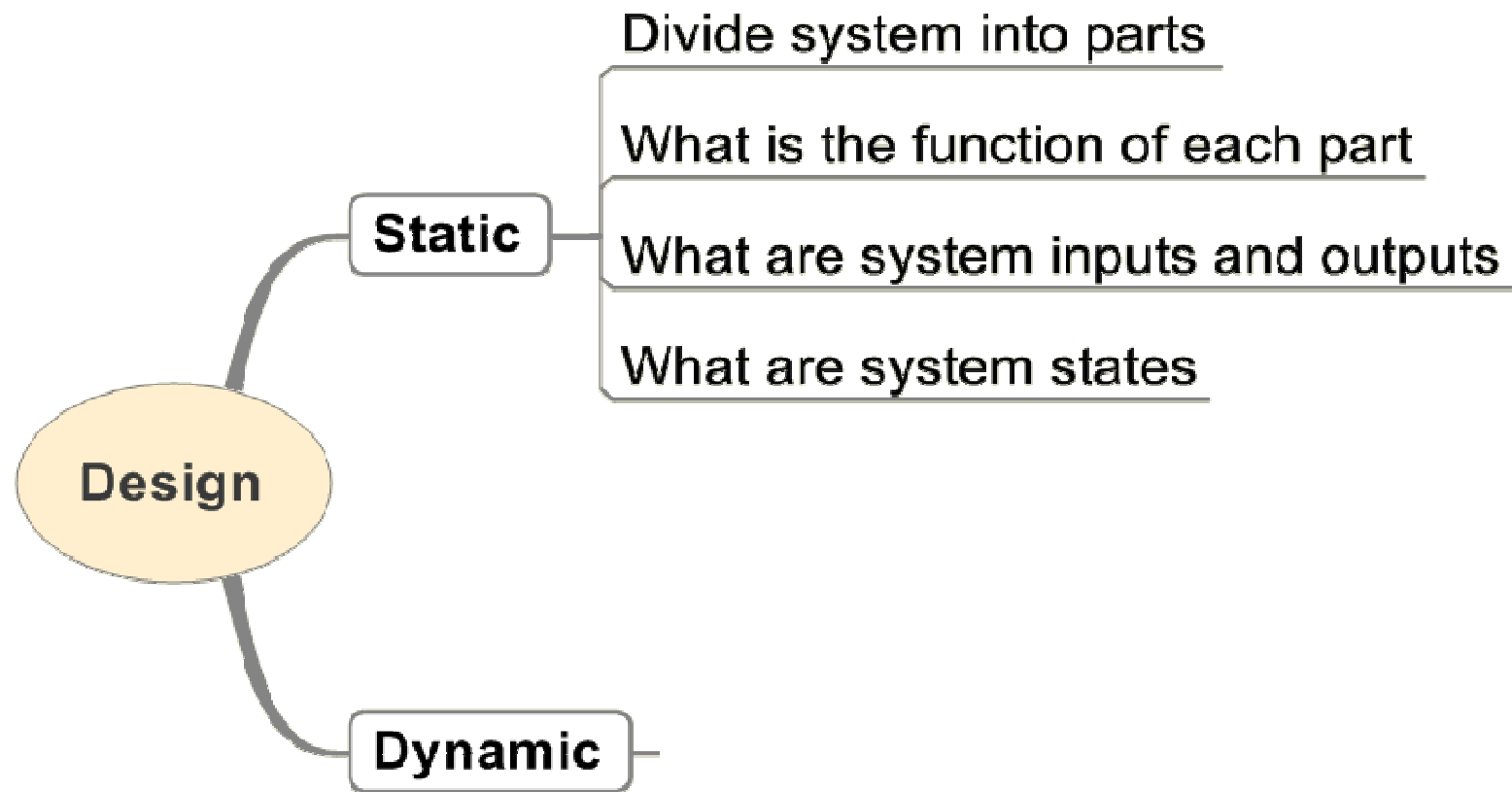


Traffic Light Controller





Design



Design

Static

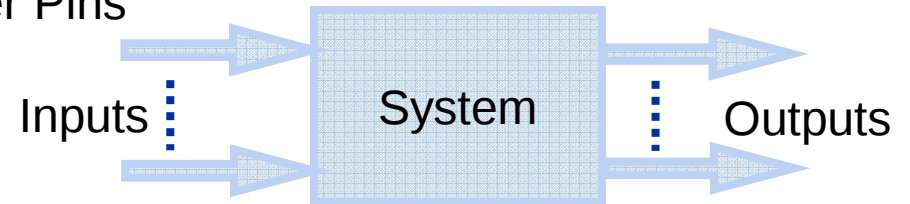
- ✓ Define parts (modules) of the software
- ✓ Define function of each part
- ✓ Define Inputs, Outputs and States

Static Design

Input/Output can be:

Analog then needs ADC/DAC that may be inside microcontroller or external

Digital connected directly to microcontroller Pins



Inputs

- Data required by the system to do its functionalities
- Inputs are defined by type, range of values and source

Outputs

- Data obtained from the system
- Inputs are defined by type, range of values and sink

Static Design

Software Modules

- A system is a big block with many functionalities
- Divide the big block into smaller manageable parts called Modules
- Modules are connected and work together to make the system
- Division is made according to functionalities
- Each Module is responsible for one or more functionality

Example:

Temperature control system Modules →

- Reading temperature input
- Reading keypad input
- Compare current temperature to required temperature
- Controlling the heater
- Output to the LCD
- Output to Alarm system

Static Design

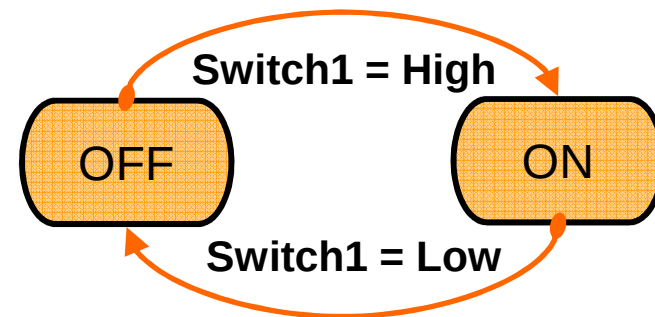
State

- According to current inputs or outputs, the system behavior may change
- A certain behavior is considered a state of the system
- A system should save its current state and calculate the next state with input/output changes
- System states values are saved as **global variables**

Example:

Simple state machine

System input is value of switch1



Software Layers

- Reading inputs (temperature, keypad)

Driver

- Compare current temperature to required temperature

Application

- Output to the LCD, Alarm system, heater

Driver

Static Design

Software Layers

- System components can be categorized into different levels
- Some components work directly with hardware → **Drivers** (depend on the microcontroller)
- Some components make calculations and manages logical states → **Applications** (depend on the system functionality)
- Application uses drivers to access hardware

Application

Drivers

Why layers?

- Layers makes changing the hardware used in the system possible without changing the application
- Also using same drivers for different applications in many systems is possible

Static Design

Traffic Light Controller

LAB 2

1. Determine system input/output
2. Divide the system into components (functions) and draw block diagram
3. Determine which components are drivers and which are application
4. Draw system state machine