How to develop an ES? (from here: session 11)

Amir Mahdi Hosseini Monazzah

Room 332,
School of Computer Engineering,
Iran University of Science and Technology,
Tehran, Iran.

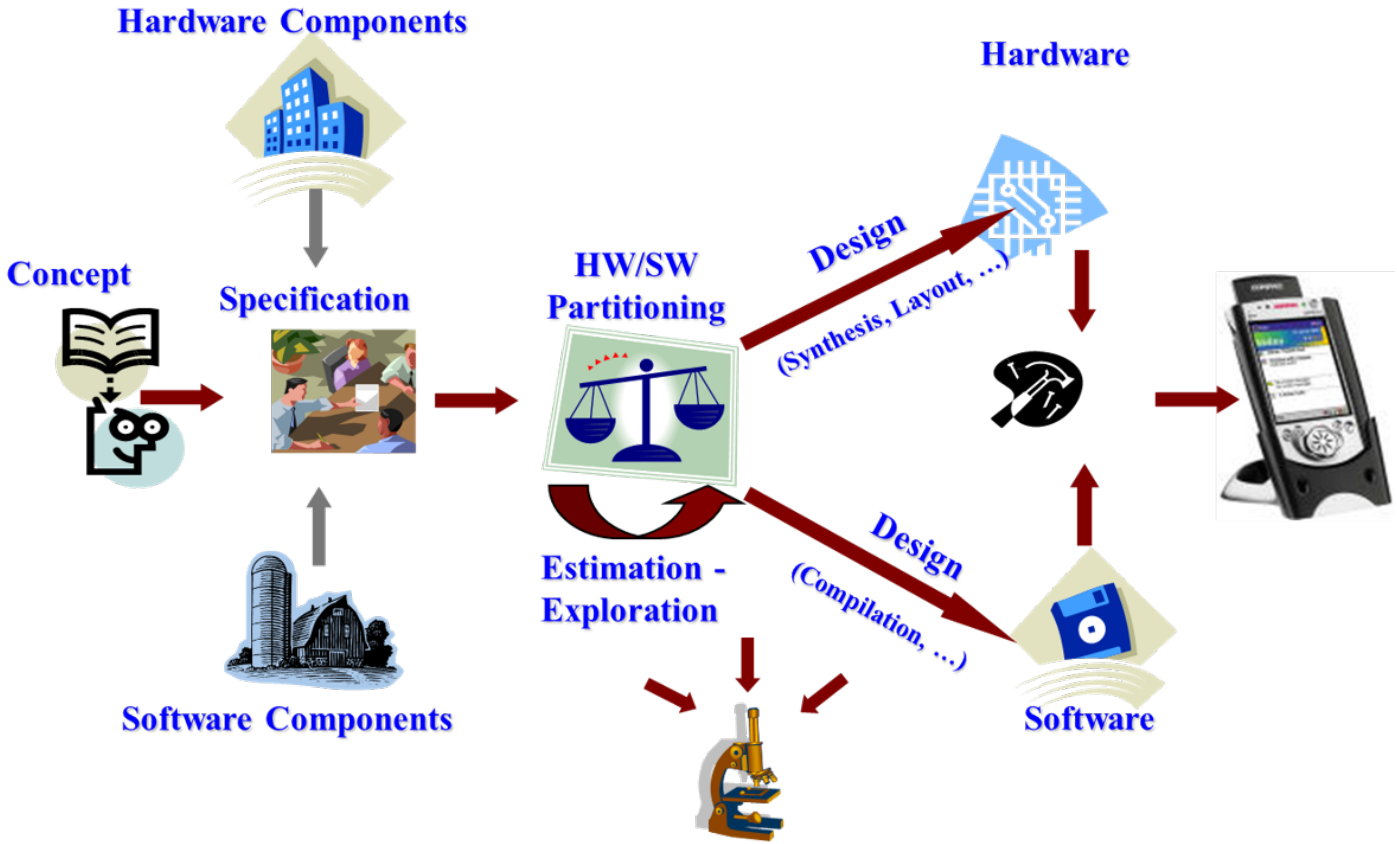
monazzah@iust.ac.ir

Fall 2024

Outline

- ES development preliminaries
  - Software
  - Hardware
- State-off-the-art development boards
  - Exploring the markets
  - Arduino
- Automata-based programming
  - Preliminaries
  - FSM
  - Statechart language

Introduction



Introduction

- Developing an application via microcontroller
  - Software development
    - We will consider CodeVision AVR
  - Hardware development
    - Simulation : We will use Proteus
    - Implementation : We will consider ATmega32

CodeVision AVR

- Opening Code vision
- Creating new project

Figure 3-1. New Project Dialog.

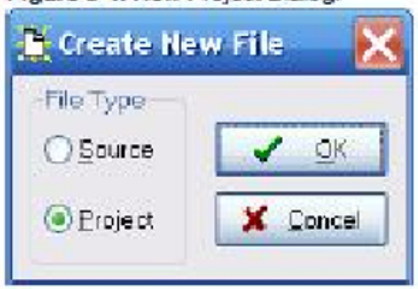
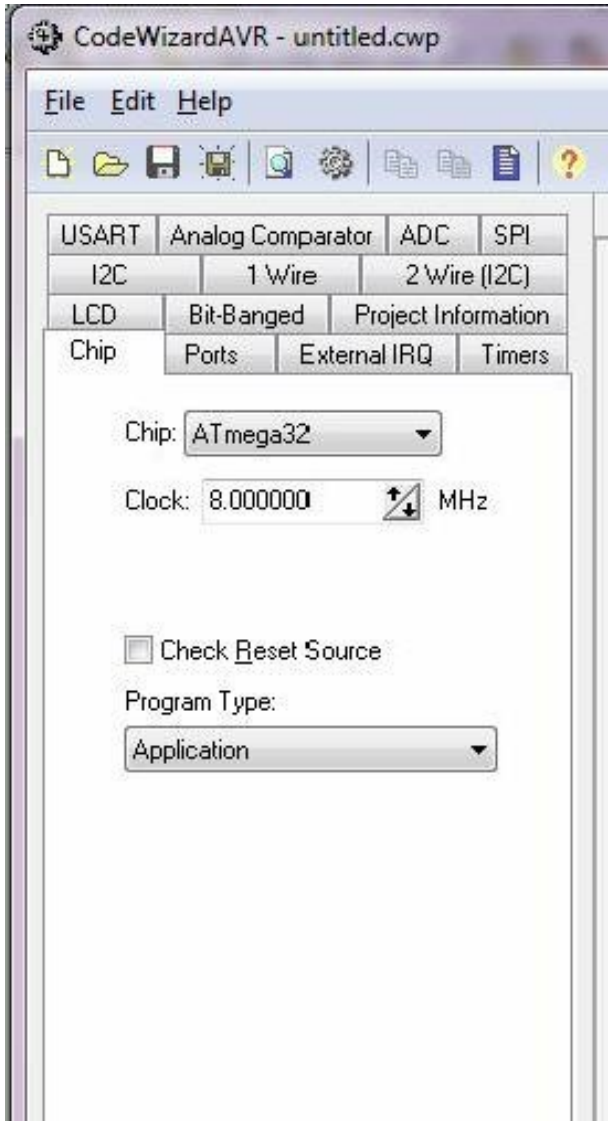


Figure 3-2. Confirmation Dialog.

CodeVision AVR

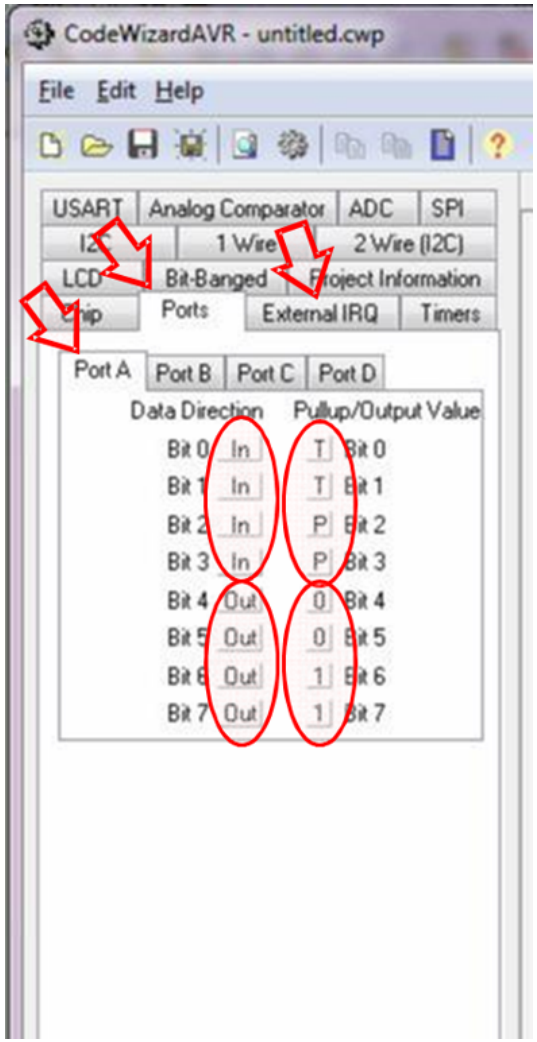
- Chip selection



- Clock setting

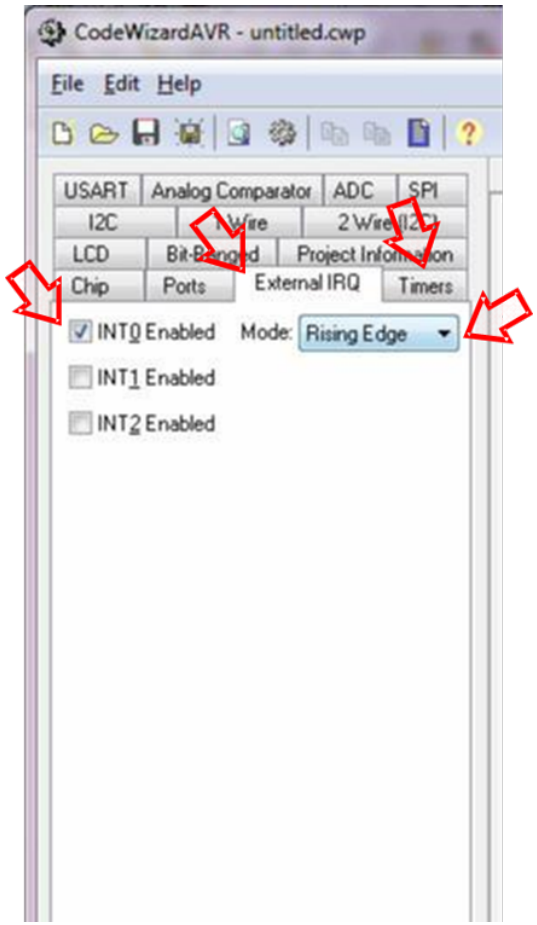
CodeVision AVR

- Port setting
  - Selecting port
  - Input/output selection (DDRx)
  - Pull up / Output value selection (PORTXD)



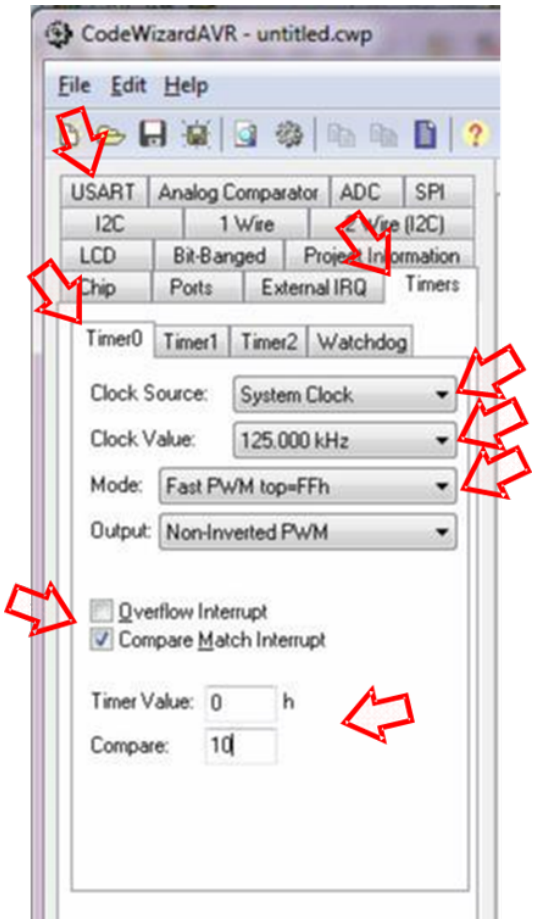
CodeVision AVR

- External interrupt setting
  - Interrupt number selection
  - Interrupt mode determination
    - Low level
    - Any change
    - Falling edge
    - Rising edge



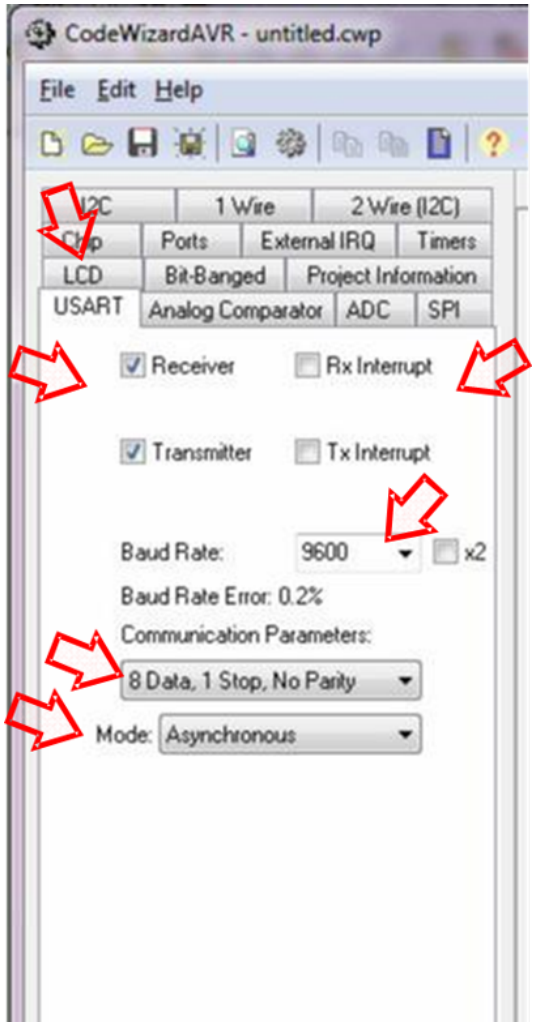
CodeVision AVR

- Timers setting
  - Timer number selection
  - Timer or counter determination
  - Timer frequency determination (CSxx)
  - Timer mode determination (WGMxx)
  - Timer interrupt (OCIEx / TOIEx)
  - Timer initial values (TCNTx / OCRx)



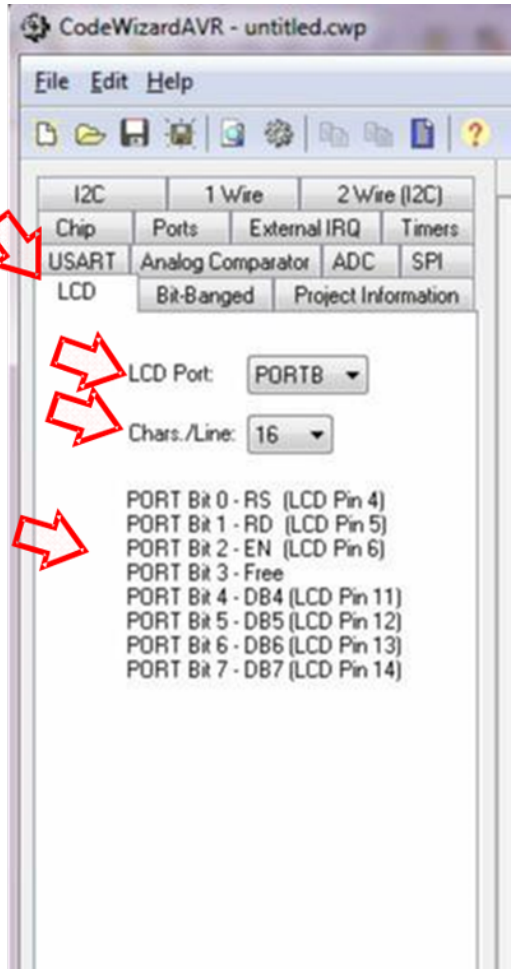
CodeVision AVR

- Serial USART setting
  - Serial direction selection
  - Serial interrupt enabling (RXCIE / TXCIE)
  - Serial baud rate determination (UBRR)
  - Serial communication parameters determination (UCSRC)
  - Serial mode selection



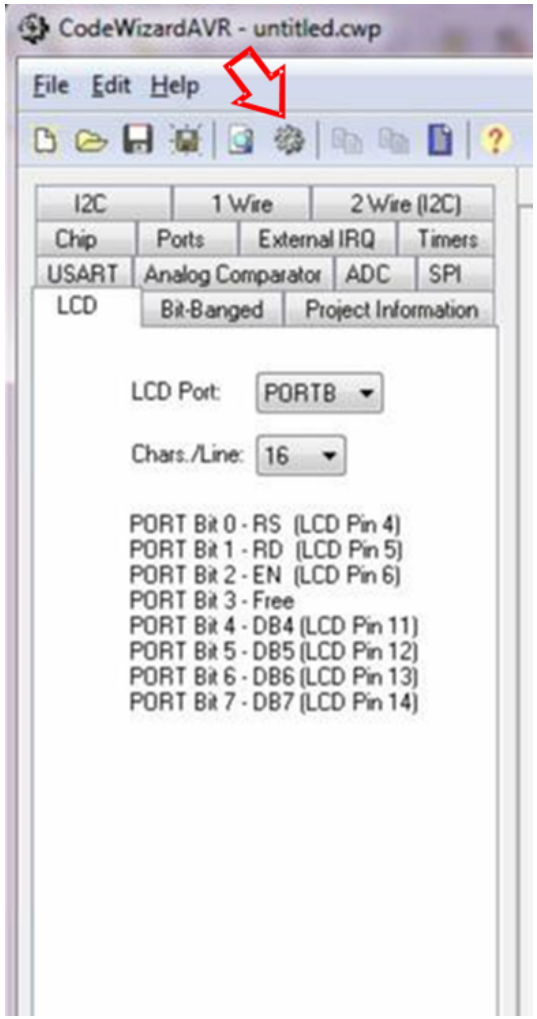
CodeVision AVR

- LCD setting
  - LCD port selection
  - LCD selection
  - LCD pin out explanation

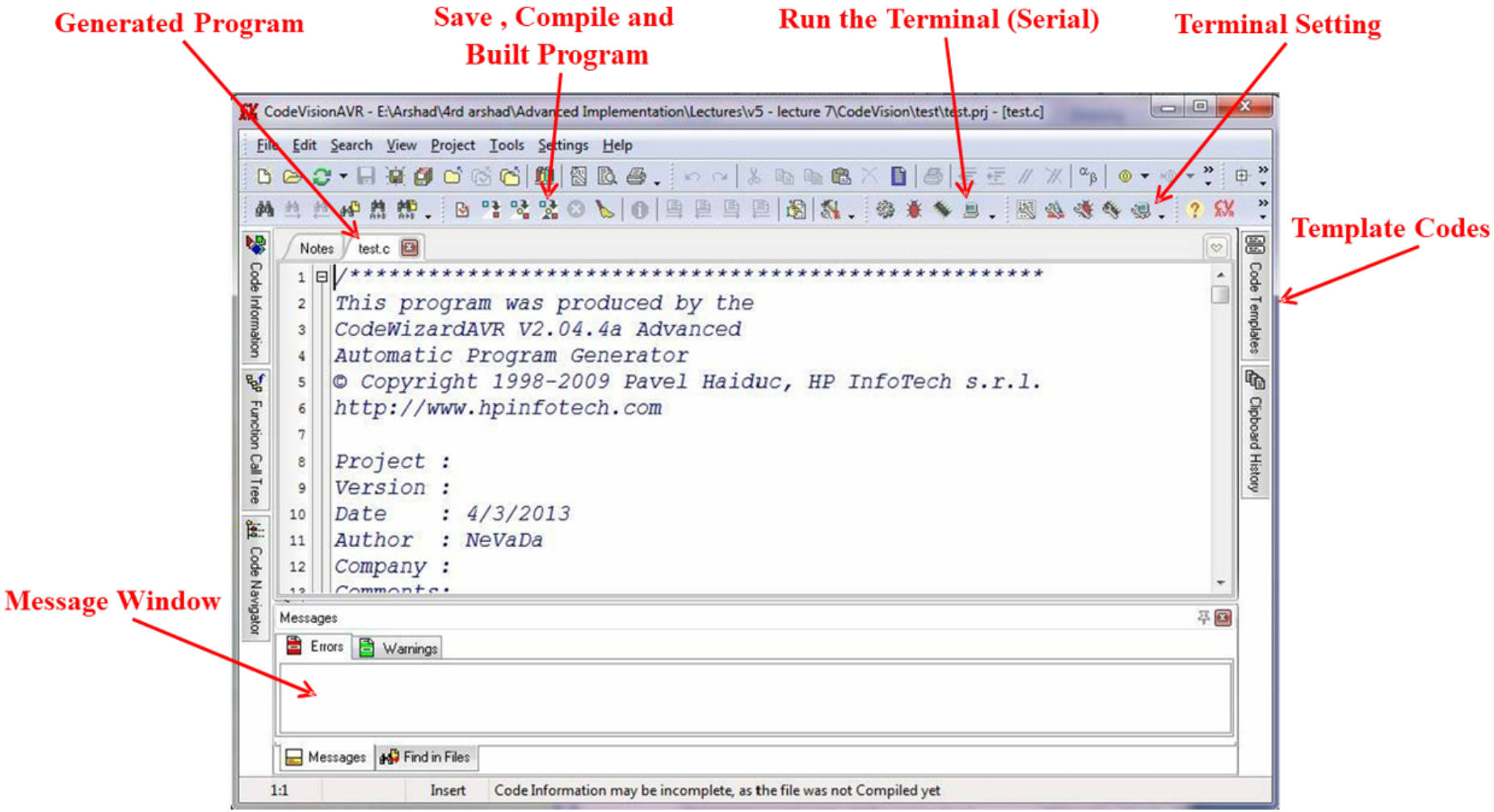


CodeVision AVR

- Completing the project
  - Save source file
  - Save project file
  - Save Code Wizard AVR project file



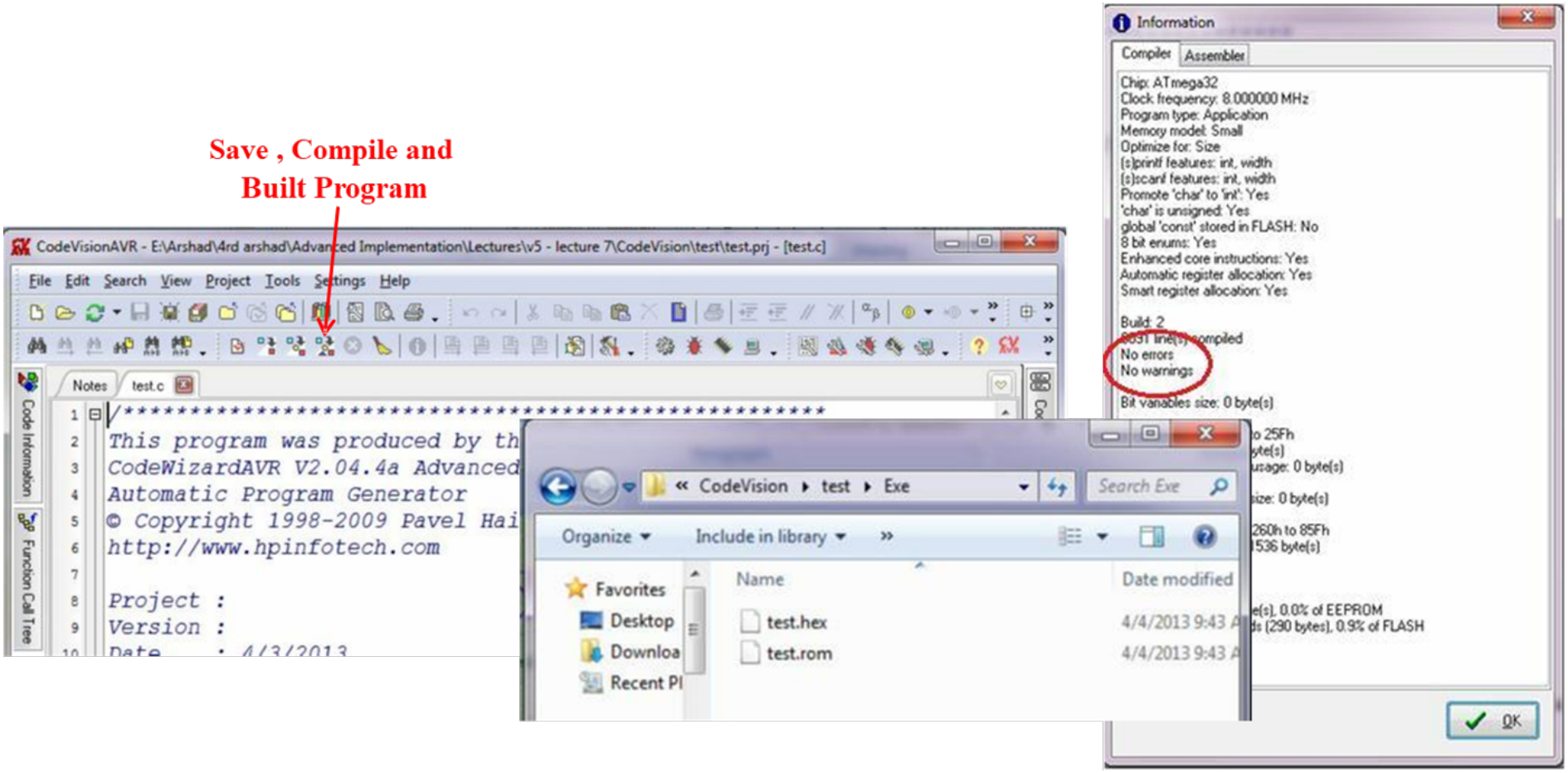
CodeVision AVR



CodeVision AVR

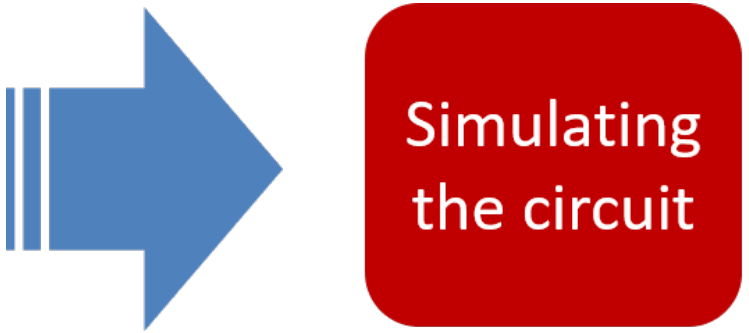
- Place your code in determined places
- Save, compile and build your project
- Information window shows the occurred warnings and errors
- Now the Hex file of your code is generated ins
  - Project path/Exe/

CodeVision AVR

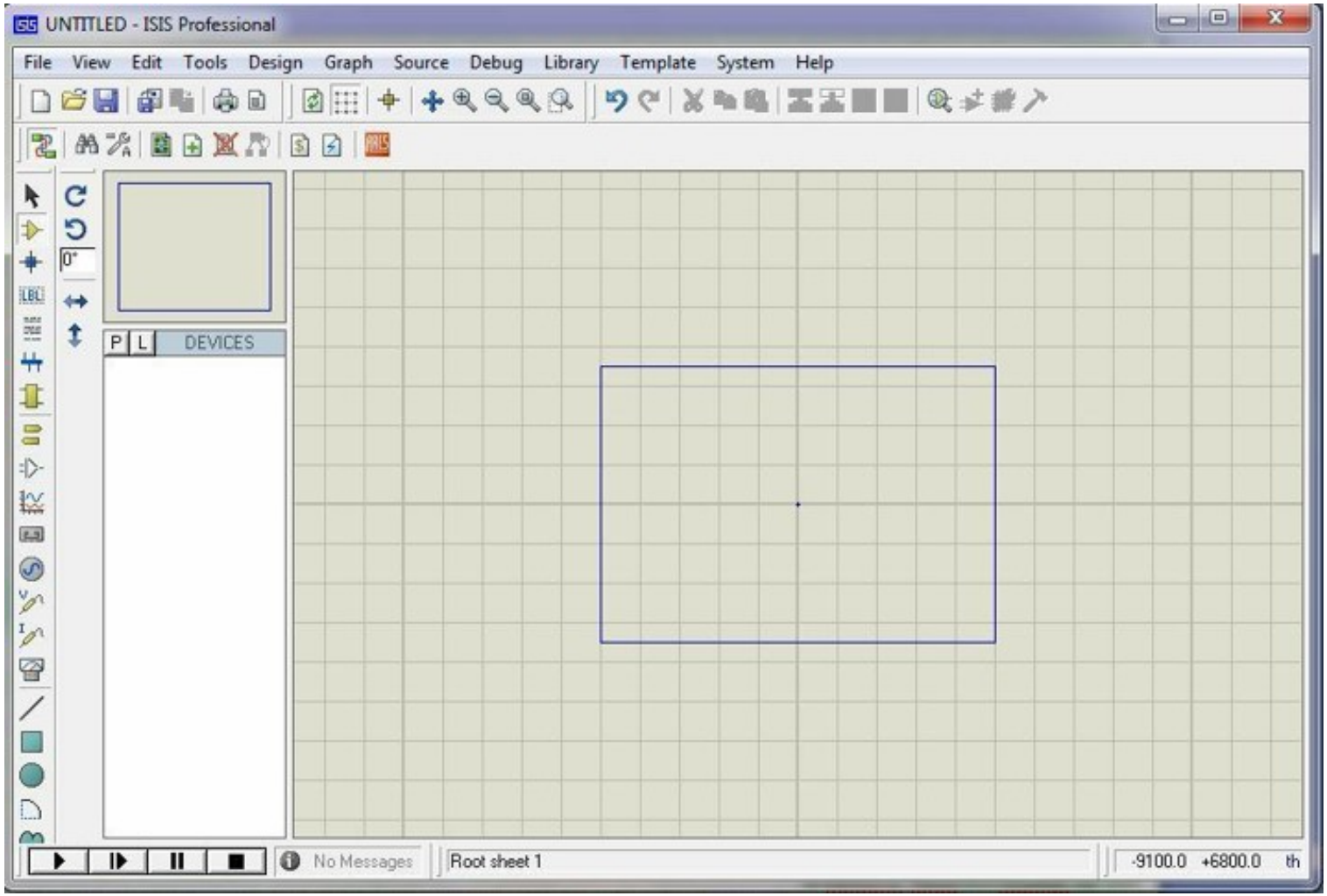


Proteus

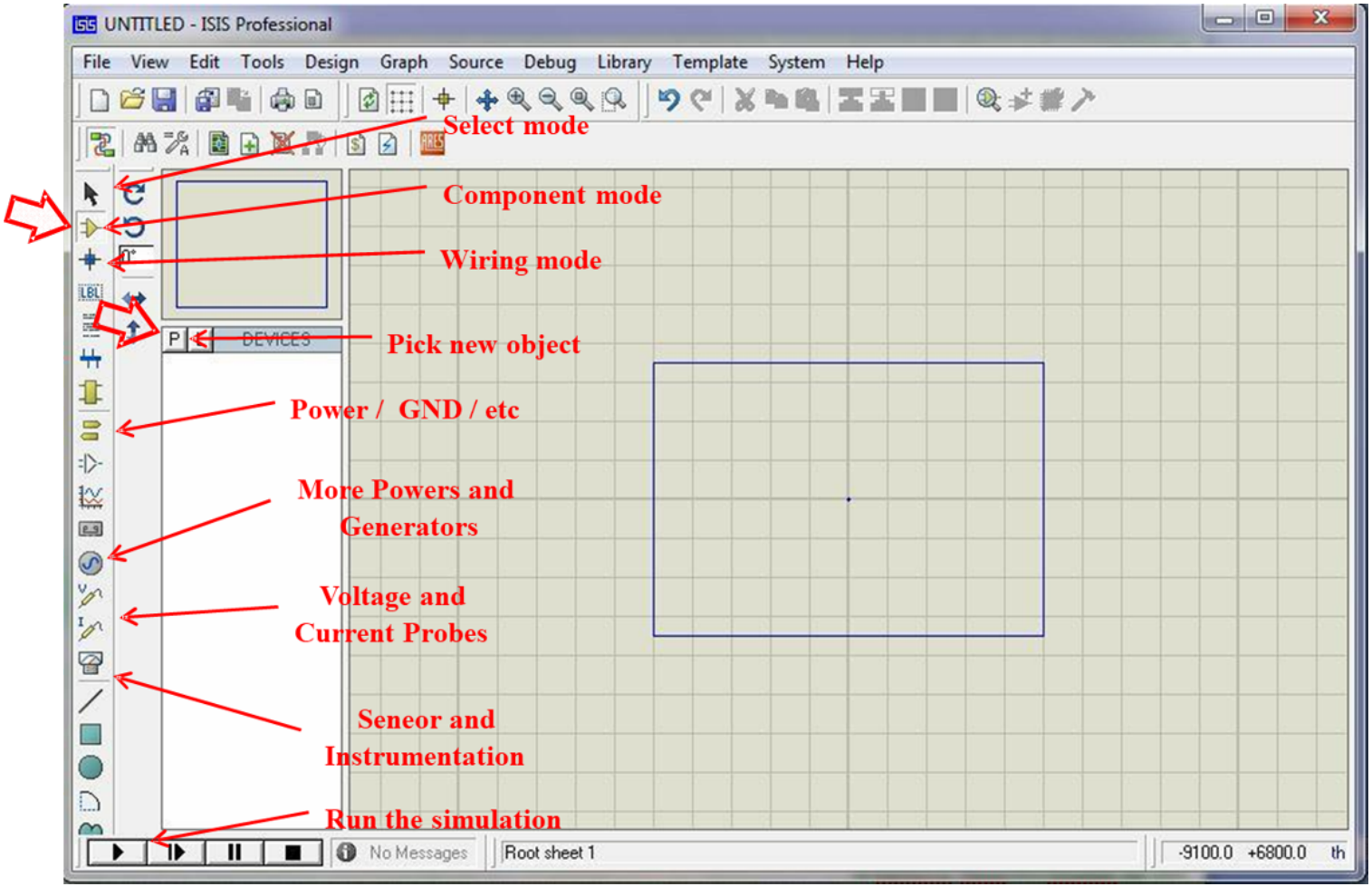
- Procedure for simulating an embedded system
  - Opening Proteus
  - Placing the devices of the circuit
  - Wiring the circuit
  - Programming the micro
    - ATmega32
  - Adding probes and sensors if needed



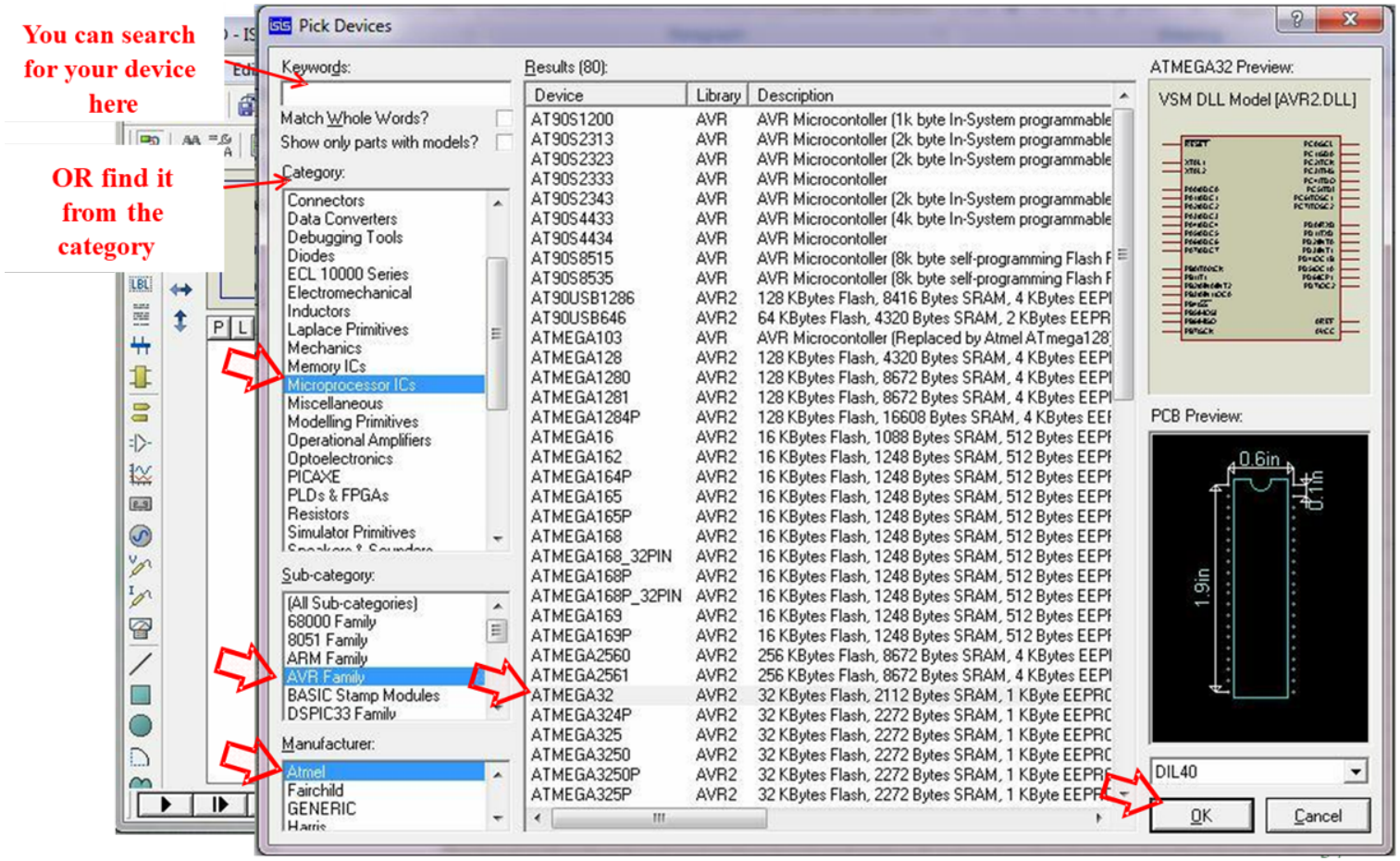
Proteus


UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template System Help
DEVICES
No Messages
Root sheet 1
-9100.0 +6800.0

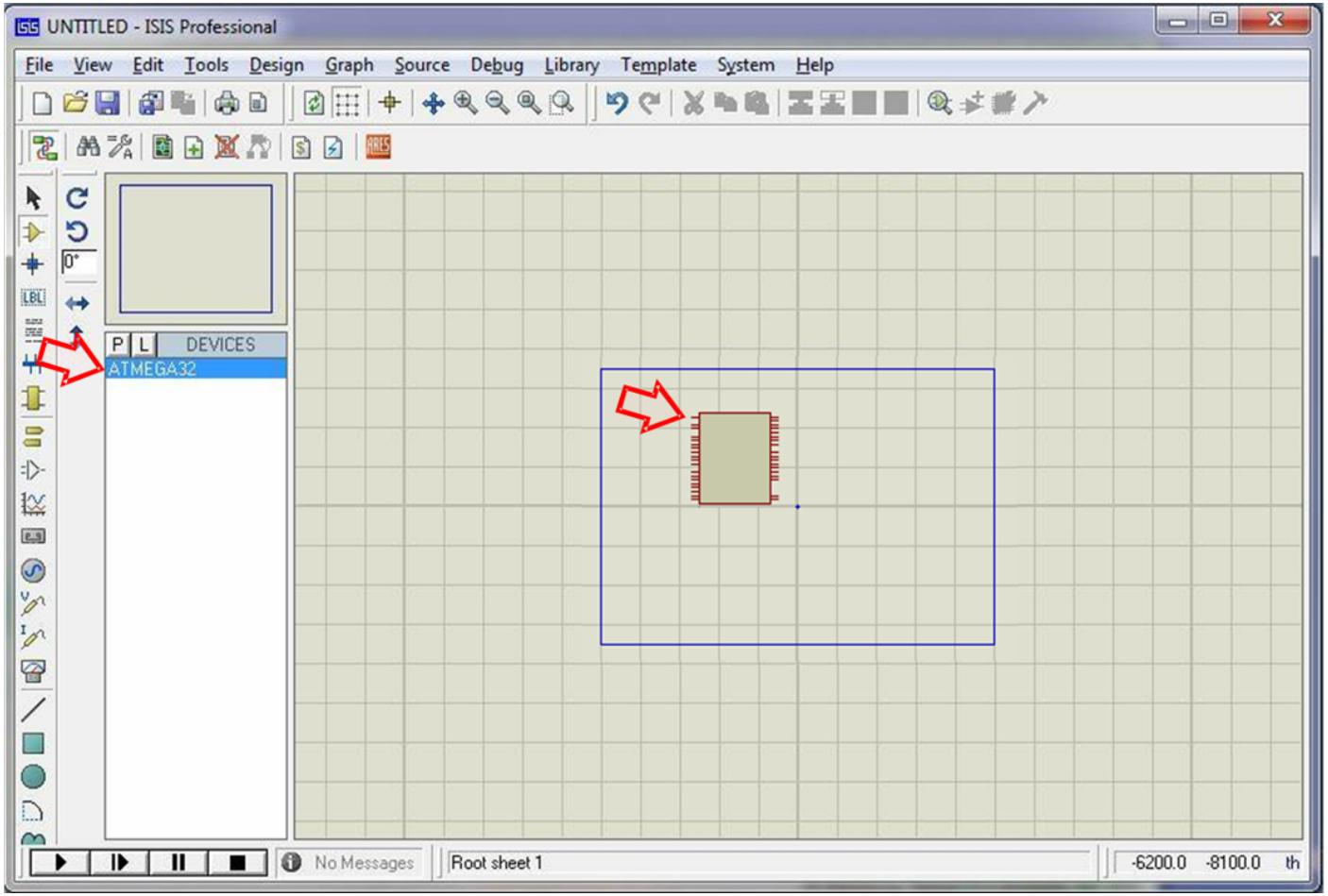
Proteus


UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template System Help
Select mode
Component mode
Wiring mode
Pick new object
DEVICES
Power / GND / etc
More Powers and Generators
Voltage and Current Probes
Seneor and Instrumentation
Run the simulation
No Messages
Root sheet 1
-9100.0 +6800.0 th

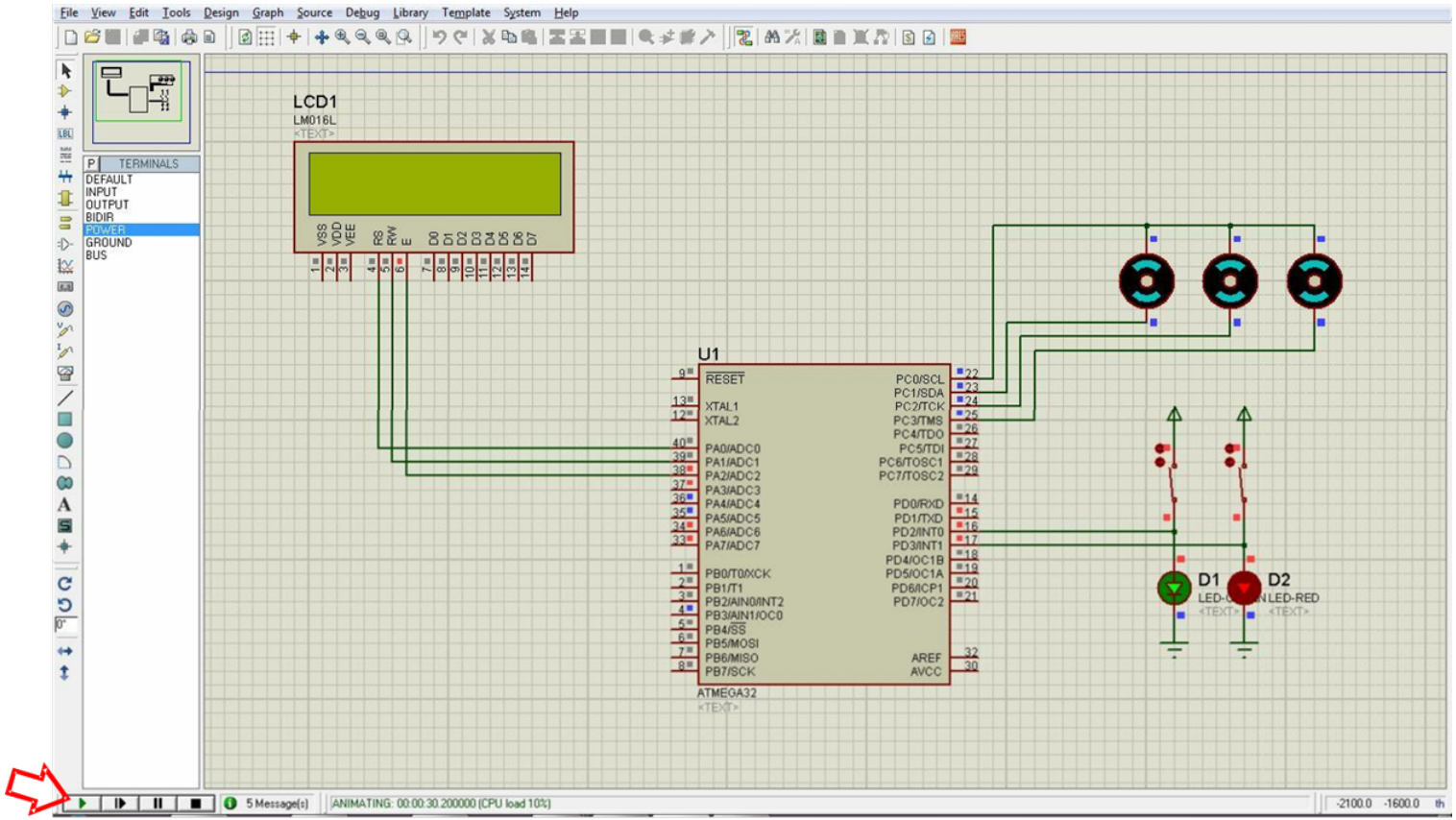
Proteus


You can search for your device here
OR find it from the category
Pick Devices
Keywords:
Match Whole Words?
Show only parts with models?
Category:
Connectors
Data Converters
Debugging Tools
Diodes
ECL 10000 Series
Electromechanical
Inductors
Laplace Primitives
Mechanics
Memory ICs
Microprocessor ICs
Miscellaneous
Modelling Primitives
Operational Amplifiers
Optoelectronics
PICAXE
PLDs & FPGAs
Resistors
Simulator Primitives
Sub-category:
(All Sub-categories)
68000 Family
8051 Family
ARM Family
AVR Family
BASIC Stamp Modules
DSPIC33 Family
Manufacturer:
Atmel
Fairchild
GENERIC
Results (80):
Device
Library
Description
AT90S1200 AVR AVR Microcontoller (1k byte In-System programmable
AT90S2313 AVR AVR Microcontoller (2k byte In-System programmable
AT90S2323 AVR AVR Microcontoller (2k byte In-System programmable
AT90S2333 AVR AVR Microcontoller
AT90S2343 AVR AVR Microcontoller (2k byte In-System programmable
AT90S4433 AVR AVR Microcontoller (4k byte In-System programmable
AT90S4434 AVR AVR Microcontoller
AT90S8515 AVR AVR Microcontoller (8k byte self-programming Flash F
AT90S8535 AVR AVR Microcontoller (8k byte self-programming Flash F
AT90USB1286 AVR2 128 KBytes Flash, 8416 Bytes SRAM, 4 KBytes EEPI
AT90USB646 AVR2 64 KBytes Flash, 4320 Bytes SRAM, 2 KBytes EEPR
ATMEGA103 AVR AVR Microcontoller (Replaced by Atmel ATmega128
ATMEGA128 AVR2 128 KBytes Flash, 4320 Bytes SRAM, 4 KBytes EEPI
ATMEGA1280 AVR2 128 KBytes Flash, 8672 Bytes SRAM, 4 KBytes EEPI
ATMEGA1281 AVR2 128 KBytes Flash, 8672 Bytes SRAM, 4 KBytes EEPI
ATMEGA1284P AVR2 128 KBytes Flash, 16608 Bytes SRAM, 4 KBytes EEI
ATMEGA16 AVR2 16 KBytes Flash, 1088 Bytes SRAM, 512 Bytes EEPI
ATMEGA162 AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA164P AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA165 AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA165P AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA168 AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA168_32PIN AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA168P AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA168P_32PIN AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA169 AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA169P AVR2 16 KBytes Flash, 1248 Bytes SRAM, 512 Bytes EEPI
ATMEGA2560 AVR2 256 KBytes Flash, 8672 Bytes SRAM, 4 KBytes EEPI
ATMEGA2561 AVR2 256 KBytes Flash, 8672 Bytes SRAM, 4 KBytes EEPI
ATMEGA32 AVR2 32 KBytes Flash, 2112 Bytes SRAM, 1 KByte EEPRC
ATMEGA324P AVR2 32 KBytes Flash, 2272 Bytes SRAM, 1 KByte EEPRC
ATMEGA325 AVR2 32 KBytes Flash, 2272 Bytes SRAM, 1 KByte EEPRC
ATMEGA3250 AVR2 32 KBytes Flash, 2272 Bytes SRAM, 1 KByte EEPRC
ATMEGA3250P AVR2 32 KBytes Flash, 2272 Bytes SRAM, 1 KByte EEPR
ATMEGA325P AVR2 32 KBytes Flash, 2272 Bytes SRAM, 1 KByte EEPR
ATMEGA32 Preview:
VSM DLL Model [AVR2.DLL]
PCB Preview:
0.6in
0.1in
1.9in
DIL40
OK
Cancel

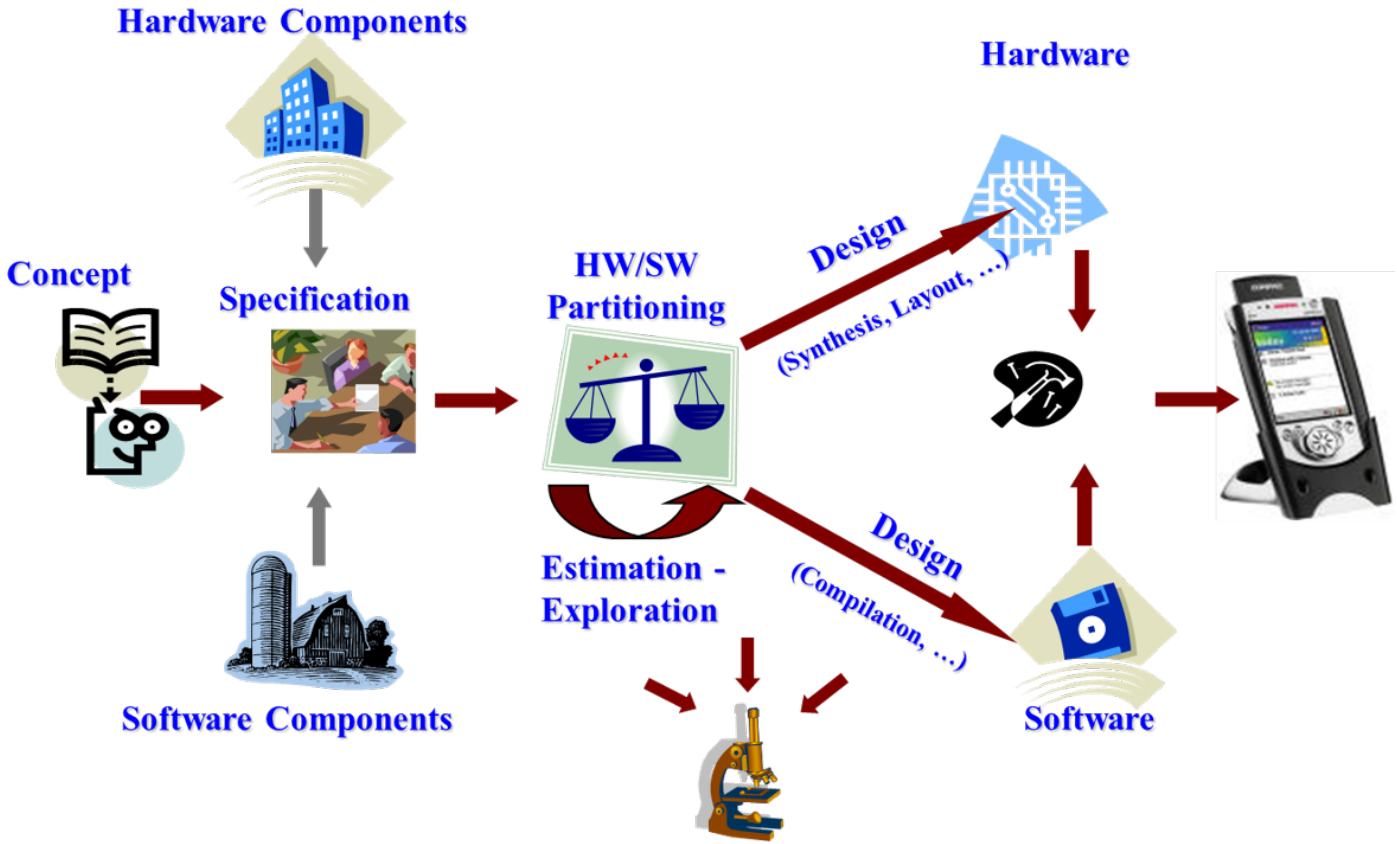
Proteus


UNTITLED - ISIS Professional
File View Edit Tools Design Graph Source Debug Library Template System Help
P L DEVICES
ATMEGA32
No Messages
Root sheet 1
-6200.0 -8100.0 th

Proteus

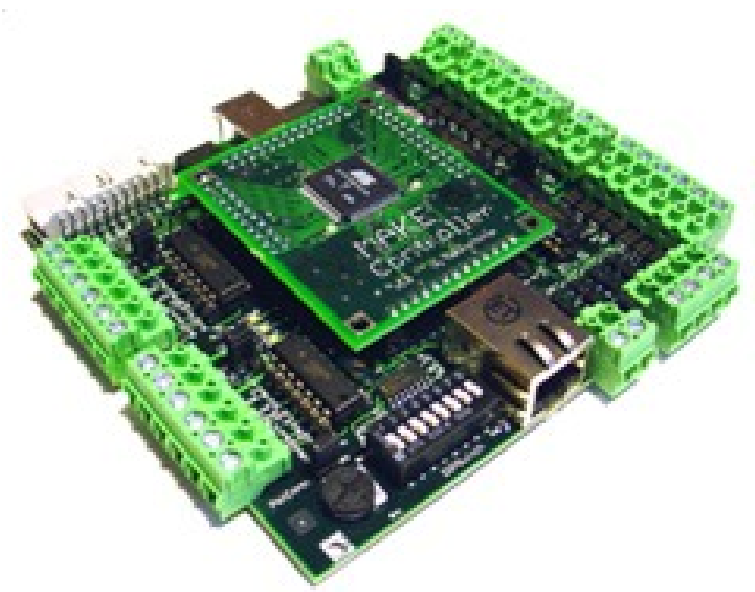

File View Edit Tools Design Graph Source Debug Library Template System Help
LCD1
LM016L
<TEXT>
VSS VDD VEE RS RW E D0 D1 D2 D3 D4 D5 D6 D7
TERMINALS
DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
BUS
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA32
<TEXT>
D1
D2
LED-RED
<TEXT>
5 Message(s)
ANIMATING: 00:00:30.200000 (CPU load 10%)

Introduction



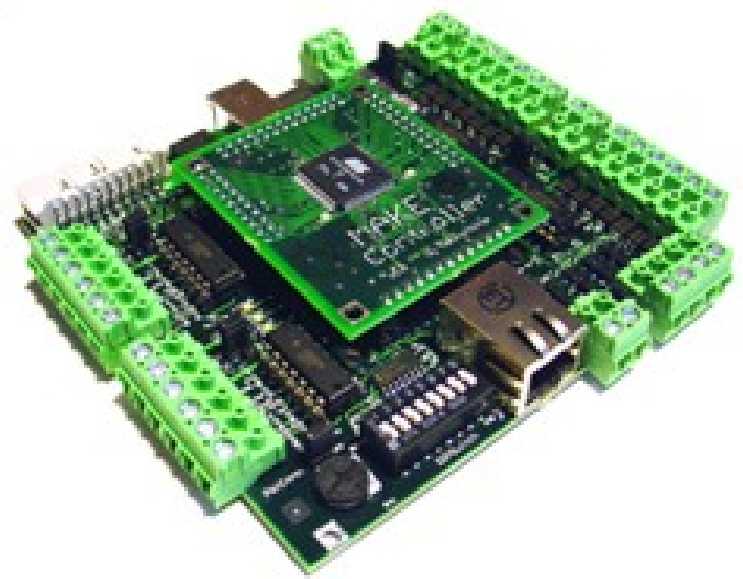
Introduction

- What is an embedded development board?
  - A microcontroller built onto a single printed circuit board
  - Provides all of the circuitry necessary for a useful control task
    - A microprocessor
    - I/O circuits
    - A clock generator
    - RAM
    - Stored program memory
    - Any necessary support ICs



Introduction

- Goal
  - The board is immediately useful to an application developer
  - Do not require to spend time and effort to develop controller hardware
- Pros
  - Low-cost
  - Education friendly

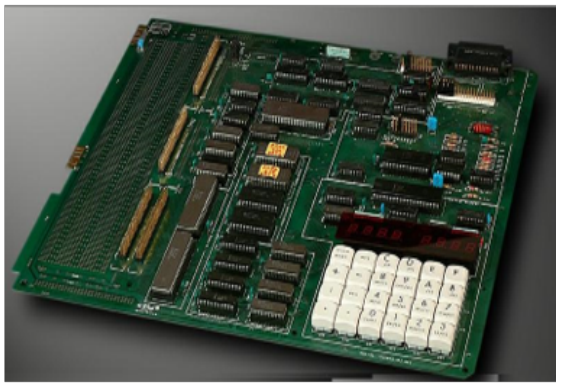


Introduction

- History
  - Development boards appeared in the late 1970s
    - When the appearance of early microprocessors : Such as the 6502 and the Z80
    - Made it practical to build an entire controller on a single board
    - More affordable than dedicating a computer to a relatively minor task

Intel SDK-51

Intel SDK-80



Intel SDK-86

Most popular ES development boards

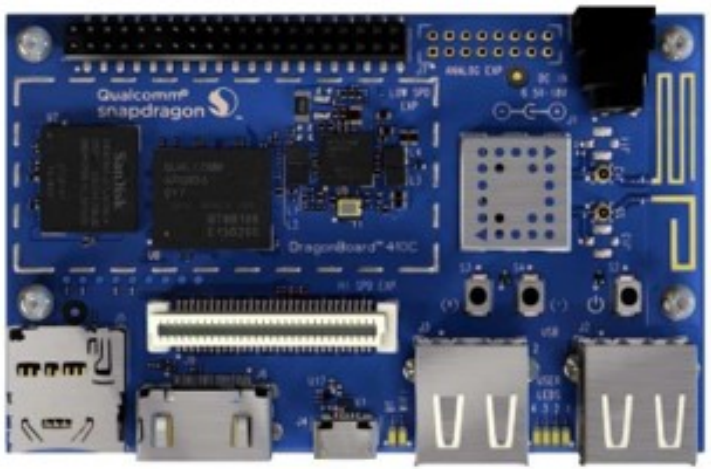
- Raspberry Pi 3 B+
  - A small pocket-sized computer
  - Running the Raspbian operating system
    - A variant of Debian Linux
  - Has a Broadcom processor
  - Low-cost embedded board with high reliability

Most popular ES development boards

- Raspberry Pi 3 B+
  - Supports various on-board peripherals
    - I2C
    - SPI
    - HDMI interface
    - Camera interface
    - UART interface
    - SDIO (Secure digital input output) for a SD card interface
  - The board comes up with
    - Software APIs and routines for application programming
    - It also has OTG (On the Go programming) for USB applications
  - There are various raspberry pi models

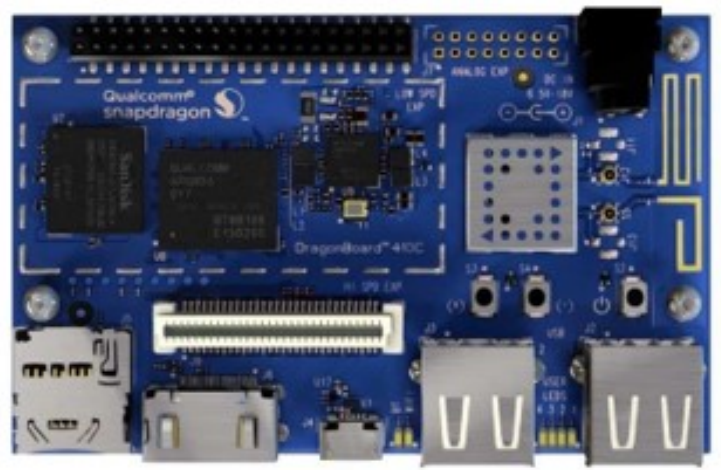
Most popular ES development boards

- Qualcomm Snapdragon
  - Single Board Computer (SBC)
  - Has powerful Qualcomm Snapdragon processor
  - It supports various interfaces like
    - Wi-Fi
    - Bluetooth
    - Global Positioning System (GPS)
  - It is well suited for
    - Internet of Things (Iot) applications
    - Medical applications
    - Robotic applications



Most popular ES development boards

- Qualcomm Snapdragon
  - Capability of connecting
    - Keyboard
    - Mouse
    - USBMouse
    - Interfaces
  - Some smartphones are coming equipped with this processor



Most popular ES development boards

- BeagleBone Black
  - Has Sitara processor
    - Running on ARM Cortex-A8 core
    - From Texas Instruments (TI)
  - 1GHz clock frequency
    - Supports various operating systems Windows-Linux-Android

Most popular ES development boards

- BeagleBone Black
  - The development board supports
    - Ethernet
    - Micro SD card
    - UART
    - Flash
    - HDMI interface for audio and video
  - The weight of the system is around 40 grams
    - Making portable and easy to carry

Most popular ES development boards

- PandaBoard
  - Low-power and low-cost
  - Based on TI's OMAP4460
    - Open media application platform
  - Supports different operating systems
  - Has a Cortex-M3 processor
    - Running at 1.2GHz clock frequency
  - Well suited for image processing applications
  - The processor is accompanied by a 384MHz GPU (Graphics Processing Unit)

Most popular ES development boards

- PandaBoard
  - The development board includes
    - Two USB ports
    - Ethernet and wireless connectivity using bluetooth
  - The video processing capability of the PandaBoard makes it a good for 1080 High Definition (HD) applications.
  - Well-suited for entertainment applications; however, it is quite expensive when compared with other development board

Most popular ES development boards

- Intel Galileo Gen 2
  - Has Intel Quark SoC X1000 processor
  - Designed using Pentium technology
  - Compatible with shields for the Arduino Uno R3
  - Has 8Mb Flash
  - Has different interfaces
    - USB
    - SD card
    - UART
    - Ethernet

Most popular ES development boards

- Intel Galileo Gen 2
  - Has a rich set of software libraries for developing applications
    - Thus, it is well suited for students and electronic hobbyists

Most popular ES development boards

- Arduino Mega 2560
  - Has an 8-bit ATmega2560 microcontroller
    - Running at 16MHz
  - Has 54 digital Input/output pins
  - Has 16 analog inputs
  - Has four UARTs
  - Can be programmed using the Arduino IDE

Most popular ES development boards

- Arduino Mega 2560
  - It is compatible with other variants of Arduino shields
  - One of the best platforms in electronic projects
  - Ideal if you are a beginners
    - Quickly develop applications
      - With less effort than it is compatible with other variants of Arduino shields
      - Supported by thousands of active users and contributors

Most popular ES development boards

- Banana Pi M2+
  - Supports various interfaces
    - Bluetooth
    - Wi-Fi
    - Ethernet
  - Great computing performance
  - Has quad-core ARM Cortex-A7 processor
    - Running at 1.2GHz

Most popular ES development boards

- Banana Pi M2+
  - There are various versions of the Banana Pi M2+
    - H3
    - H2+
    - EDU
    - H5
    - For example, the Banana EDU is well-suited for students and engineers to learn the functionality of small embedded applications, but it has no Wi-Fi or bluetooth on-board!

Most popular ES development boards

- CubieBoard6
  - Runs operating systems like
    - Linux
    - Android
  - Has a quad core Cortex-A9 processor
  - Can be powered using
    - A LiPo (lithium polymer) battery

Most popular ES development boards

- CubieBoard6
  - It comes with
    - An infrared (IR) sensor
    - A Real Time Clock (RTC) module
    - Wi-Fi
    - Bluetooth
    - An audio input and output via a 3.5mm jack
    - HDMI

Most popular ES development boards

- Odroid-C2
  - Has a 64-bit quad-core processor
    - Suitable for applications like
      - Multimedia
      - Gaming
      - Consumer electronics
  - Has Amlogic S905 processor
    - Which is based on an ARM Cortex-A53
    - Make it a resource for wearable applications
  - Boasts 2GB 32-bit DDR3 RAM

Most popular ES development boards

- Odroid-C2
  - 40 GPIO pins for connecting external peripherals
  - Can also work as a standalone computer
    - With available open source software packages
  - Has a built-in temperature sensor
  - Supports different interfaces
    - Four USB ports
    - Two UART
    - I2C interfaces

Most popular ES development boards

- HummingBoard Gate
  - Has been described as
    - " The device you've been waiting for to fulfil all of your IoT needs."
  - Has an NXP quad-core processor
  - Comes with an integrated mikroBUS socket
    - Suitable for MikroElektronika development boards and external peripheral modules
  - Supports 2GB of DDR3 RAM
  - Useful for building modular projects and providing proof-of-concept

Arduino overview

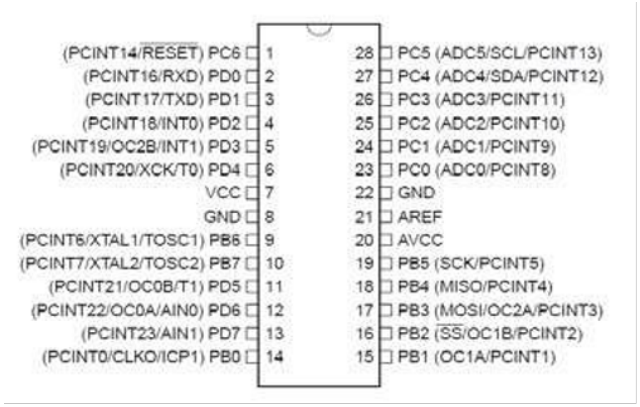
- The Arduino project was started in Italy to develop low cost hardware for interaction design
- The Arduino hardware comes in several flavors
- With the Arduino board, you can write programs and create interface circuits to read switches and other sensors, and to control motors and lights with very little effort
- The Arduino programming language is a simplified version of C/C++

Arduino overview

- An important feature of the Arduino is that you can create a control program on the host PC, download it to the Arduino and it will run automatically

ATmega328P in Arduino

- AVR 8-bit RISC architecture
- Available in DIP package
- Up to 20 MHz clock
- 32kB flash memory
- 1 kB SRAM
- 23 programmable I/O channels
- Six 10-bit ADC inputs three timers/counters
- Six PWM outputs



What is Arduino not?

- It is not a chip (IC)
- It is not a board (PCB)
- It is not a company or a manufacturer
- It is not a programming language
- It is not a computer architecture

Although it involves all of these things...

What is Arduino?

- It's a movement, not a microcontroller
  - Founded by Massimo Banzi and David Cuartielles in 2005
  - Based on "Wiring Platform", which dates to 2003
  - Open-source hardware platform
  - Open source development environment
    - Easy-to learn language and libraries (based on wiring language)
    - Integrated development environment (based on processing programming environment)
    - Available for Windows / Mac / Linux

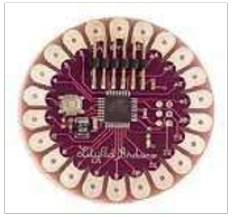
What is Arduino?

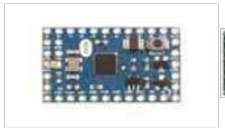
- Arduino is a popular "open source" single board microcontroller. It is designed to make the process of using electronics in multidisciplinary projects more accessible
- Open source hardware, you can make your own board, or buy one
- Cheap, easily available
- Open source software
- Very widespread, many projects openly available
- Extra HW (shields) available

This idea began in Italy and its initial purpose was to make STUDENT design projects more affordable than other prototyping projects at the time.

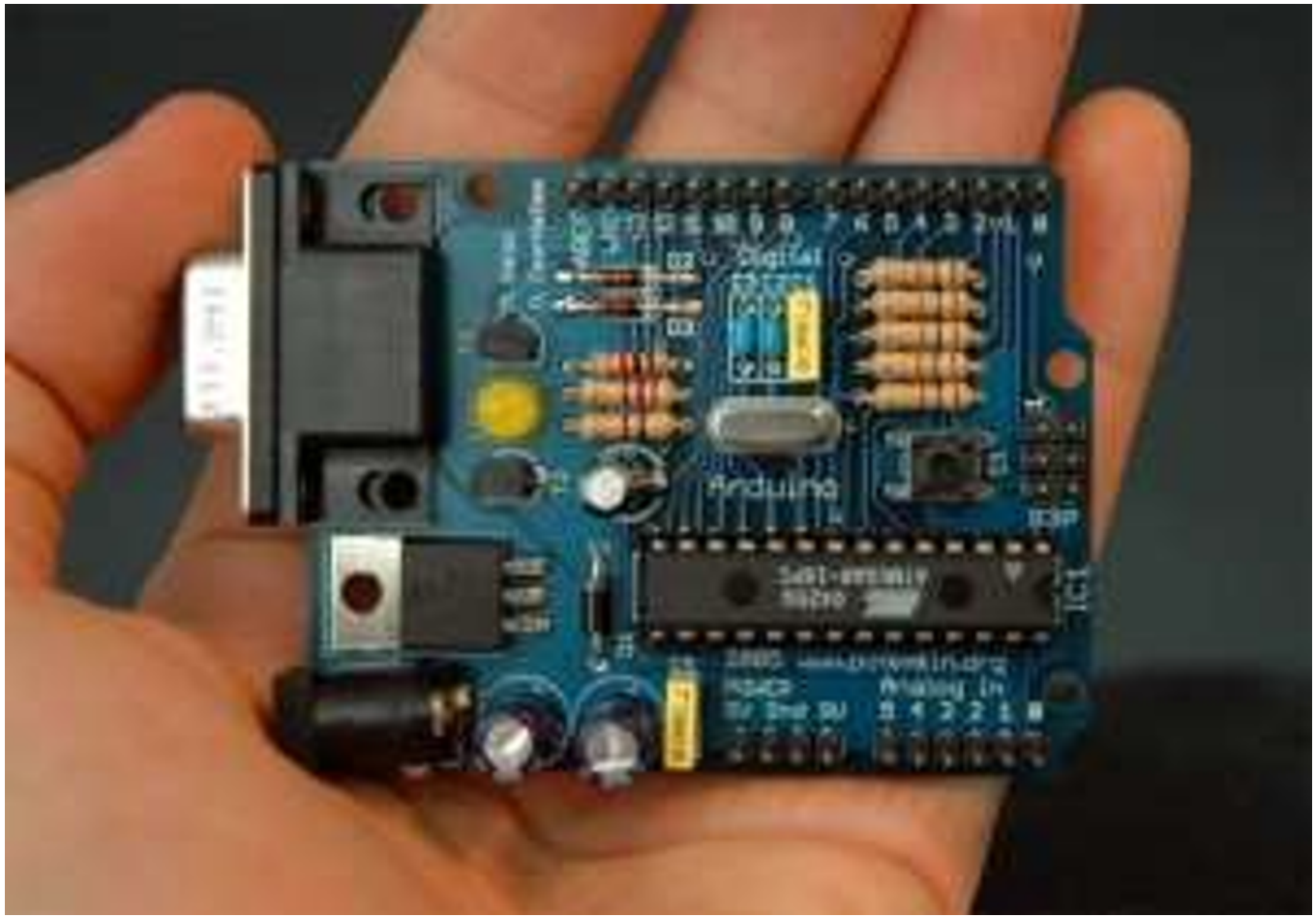
The many flavors of Arduino

- Arduino Uno
- Arduino Leonardo
- Arduino LilyPad
- Arduino Mega
- Arduino Nano
- Arduino Mini
- Arduino Mini Pro
- Arduino BT

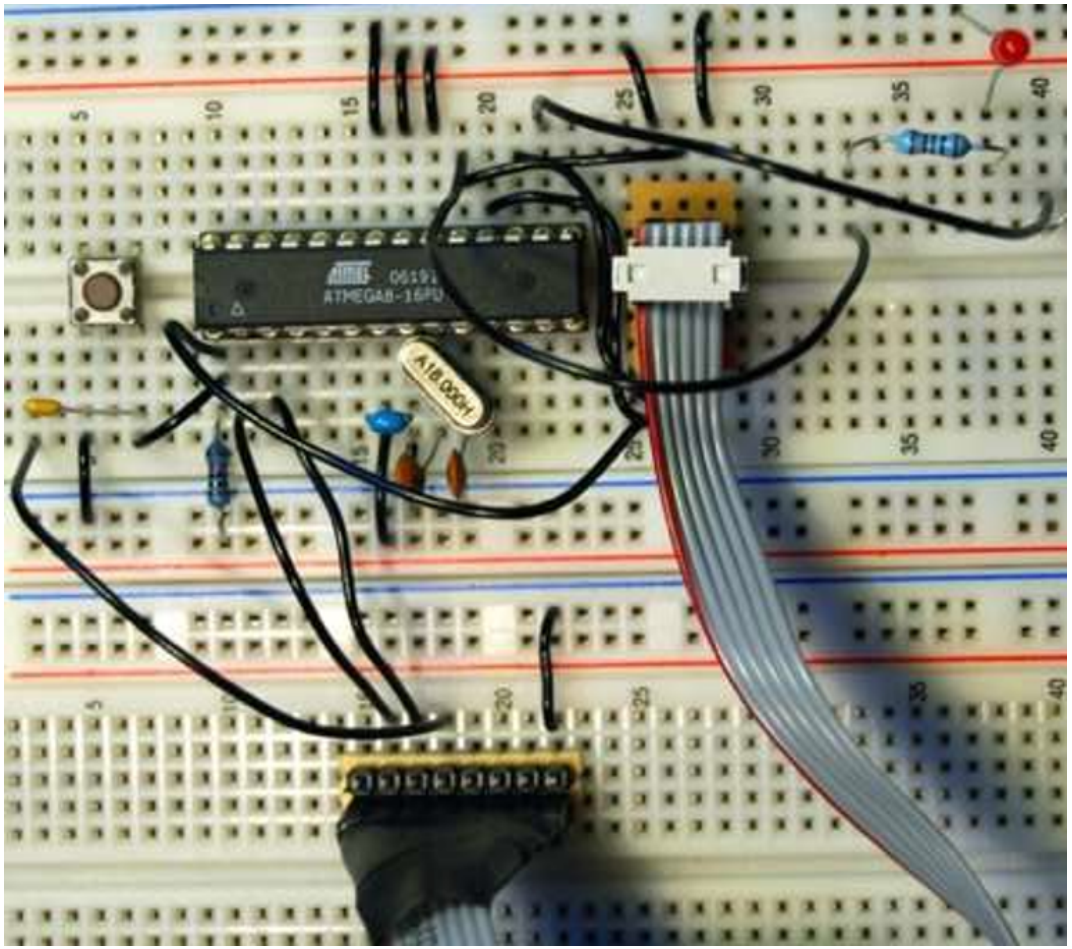




Original Arduino with RS-232



Arduino on breadboard



Arduino add-ons (shields) (up to/from here: session 11/12)

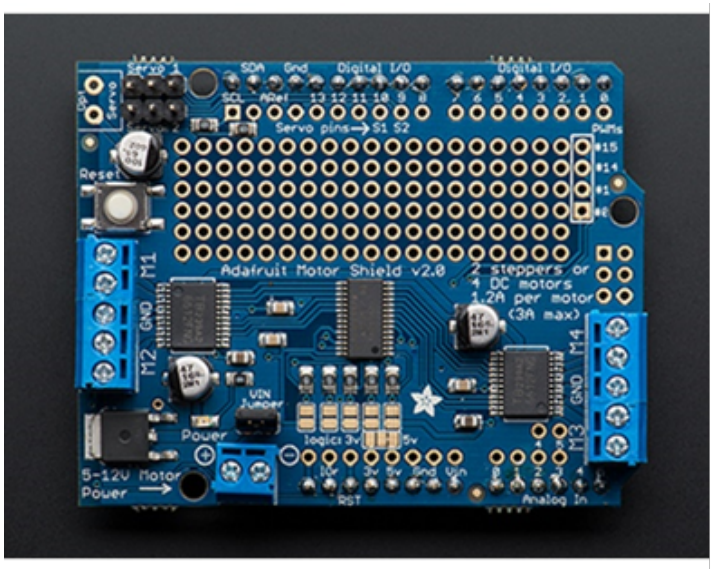
- TFT touch Screen
- Data logger
- Motor/Servo shield
- Ethernet shield
- Audio wave shield
- Cellular/GSM shield
- WiFi shield
- ...many more

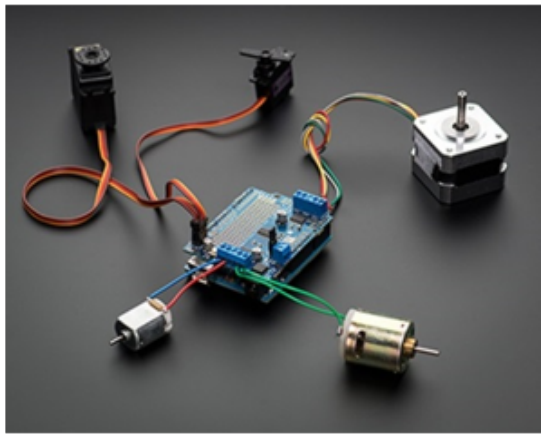
Other hardware choices-sheilds

- Xbee shield
  - Allows an Arduino board to communicate wirelessly using Zigbee
  - Can communicate up to 30.5m indoors or 91.5m outdoors (with line-of-sight)
  - Can be used as a serial/usb equipment and you can put it into a command mode and configure it for a variety of broadcast and mesh networking options

Other hardware choices-sheilds

- Adafruit servo/stepper/DC motor shield
  - A shield that can control 2 hobby servos and up to 2 unipolar/bipolar stepper motors or 4 bi-directional DC motors



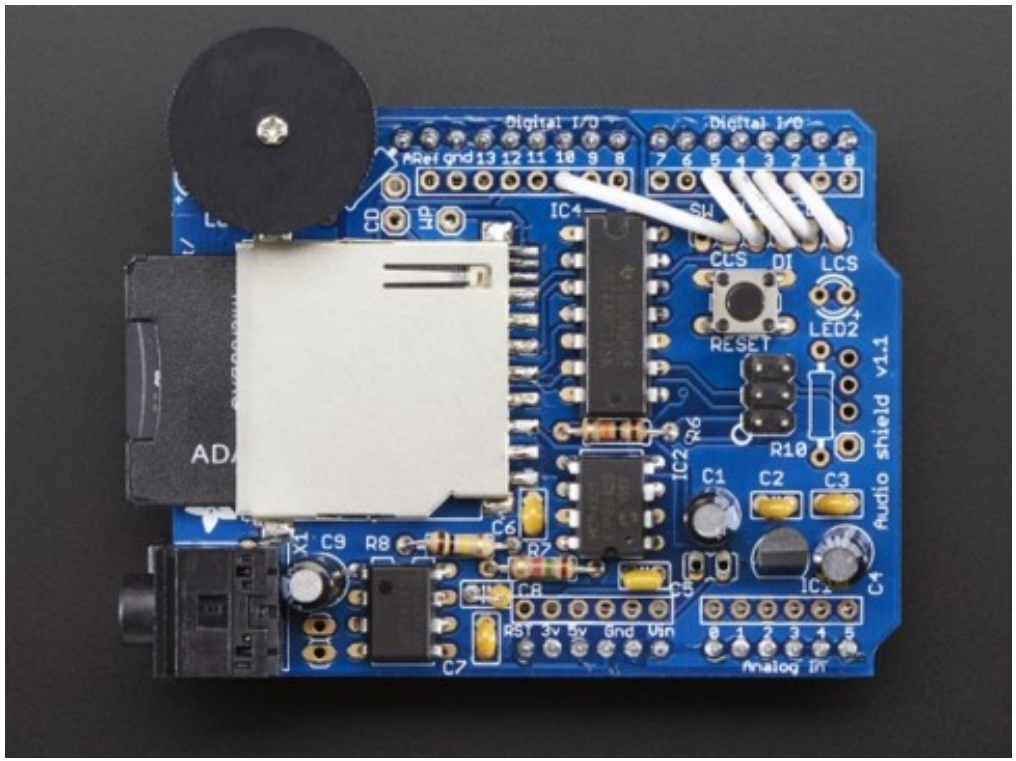


Other hardware choices-sheilds

- Battery shield
  - A shield from Liquidware that connects to the back of the Arduino, with a USB-rechargable lithium ion battery that can power an Arduino for 14-28 hours depending on the circuit
- Liquidware touch shield
  - OLED touch screen shield

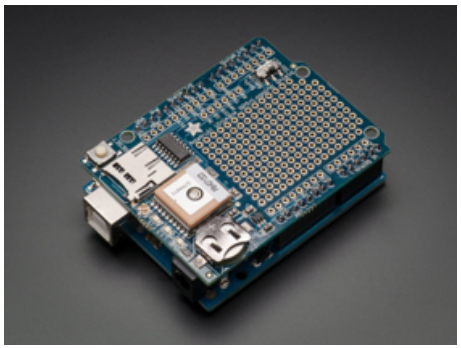
Other hardware choices-sheilds

- Adafruit wave shield
  - Plays any size 22KHz audio files from an SD memory card for music, effects and interactive sound art



Other hardware choices-sheilds

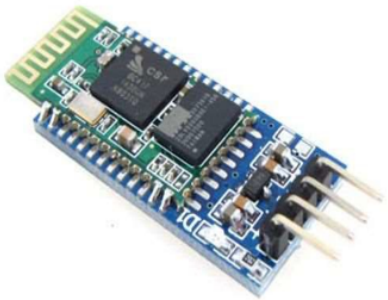
- Adafruit GPS data logging shield
  - Connects to a GPS module and can log location, time/date as well as sending data to an SD memory flash card



- Adafruit XPort\ethernet shield
  - Allows use of an XPort module for connecting to the internet as a client or server

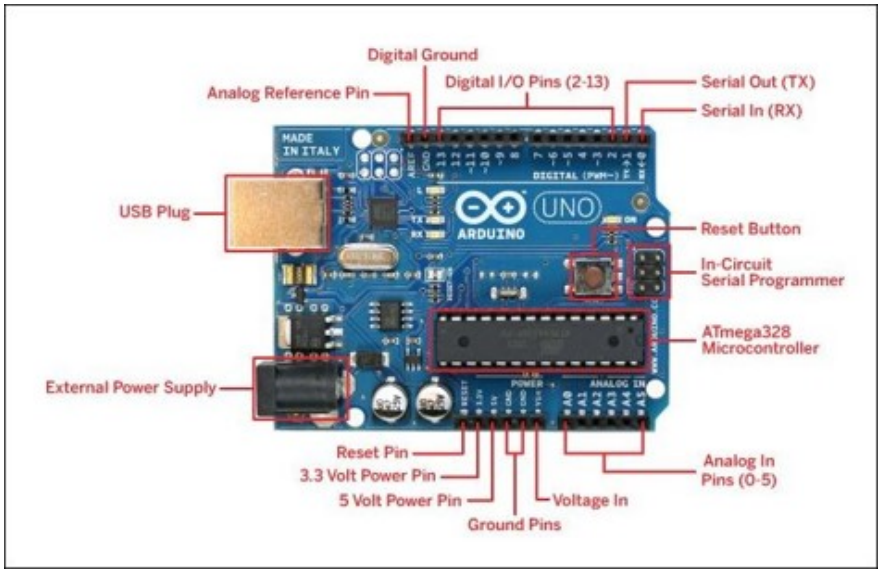
Other hardware choices-sheilds

- Bluetooth\USB\RS-232 to 5v TTL



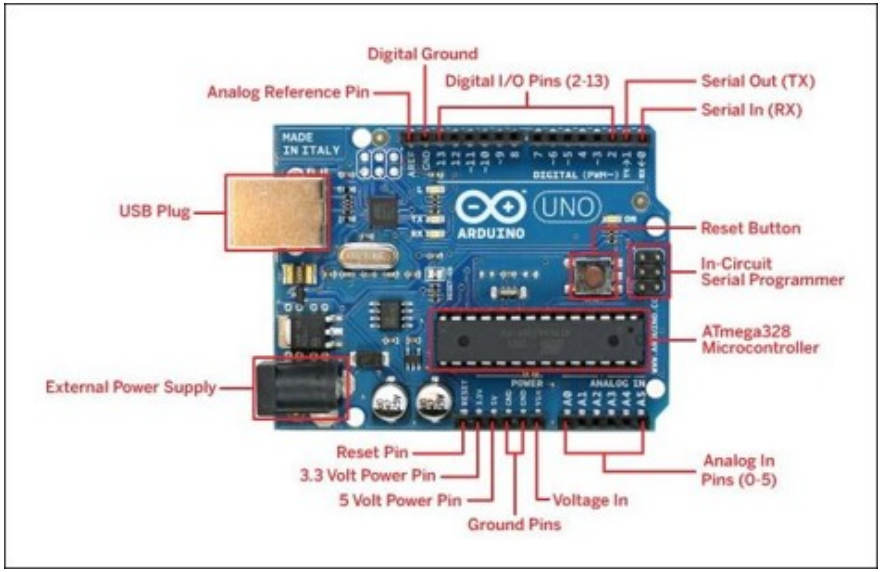
Arduino UNO pin diagram

- Analog reference pin
- Digital ground
- Digital pins 2-13
- Digital pins 0-1/serial in/out - TX/RX
  - These pins cannot be used for digital I/O (digital read and digital write) if you are also using serial communication (e.g. Serial.begin)



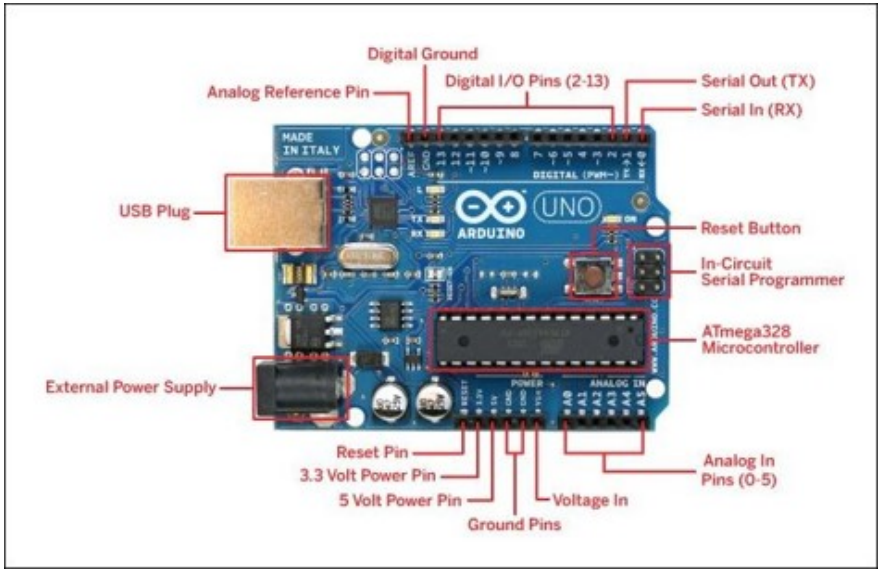
Arduino UNO pin diagram

- Reset button
- In-circuit serial programmer
- Analog in pins 0-5
- Power and ground pins



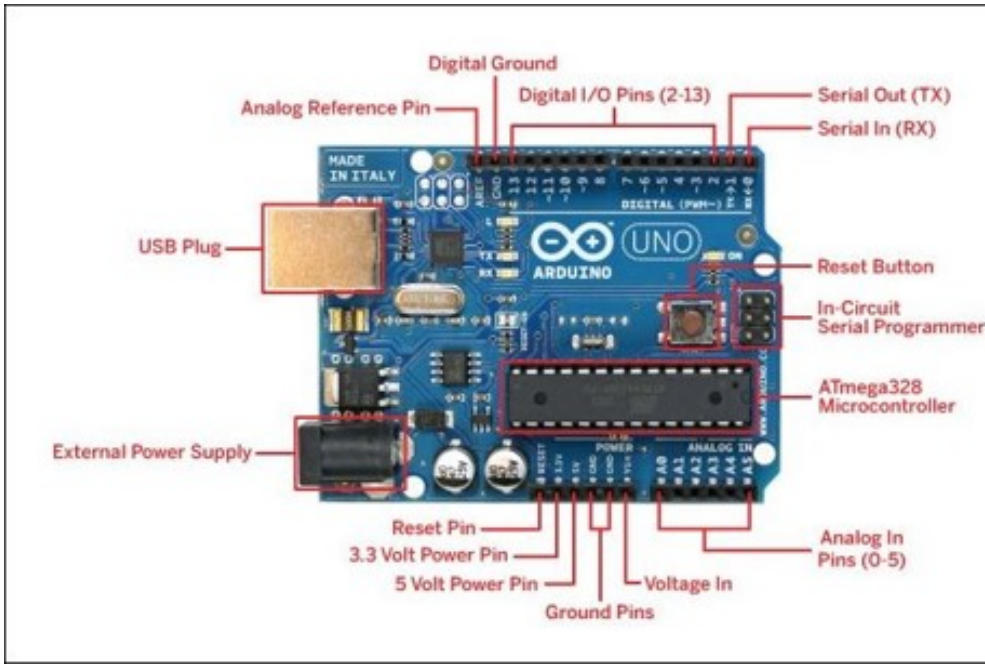
Arduino UNO pin diagram

- External power supply in (9-12VDC)
  - Toggles external power and USB power (place jumper on two pins closest to desired supply)



Arduino UNO pin diagram

- USB
  - Used for uploading sketches to the board and for serial communication between the board and the computer
  - Can be used to power the board



Arduino pins

- Digital pins in Arduino can be used for digital i/o
  - Command that can be used
    - pinMode()
    - digitalRead()
    - digitalWrite()
  - Each pin has an internal pull-up resistor
    - Can be turned on and off using digitalWrite() by writing a value of HIGH or LOW, respectively when the pin is configured as an input
  - The maximum current per pin is 40 mA

Arduino pins

- Serial: 0 (RX) and 1 (TX) used to receive (RX) and transmit (TX) TTL serial data in the Arduino
  - On the Arduino Diecimila, these pins are connected to the corresponding pins of the FTDI USB-to-TTL serial chip
  - On the Arduino BT, they are connected to the corresponding pins of the WT11 Bluetooth module
  - On the Arduino Mini and LilyPad Arduino, they are intended for use with an external TTL serial module (e.g. the Mini-USB Adapter)

Arduino pins

- External Interrupts: 2 and 3
  - These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value
  - See the attachInterrupt() function for details
- PWM: 3, 5, 6, 9, and 11
  - Provide 8-bit PWM output with the analogWrite() function
  - On boards with an ATmega8
    - PWM output is available only on pins 9, 10, and 11
- Reset: 7 (Arduino BT-only)
  - Connected to the reset line of the bluetooth module

Arduino pins

- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)
  - These pins support SPI communication, which, although provided by the underlying hardware
  - It allows serial communication with another interface by connecting it serially
- LED: 13
  - On the Diecimila and LilyPad, there is a built-in LED connected to digital pin 13
  - When the pin is HIGH value, the LED is on, when the pin is LOW, it is off

Arduino pins

- Analog pins
  - The analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function
  - Most of the analog inputs can also be used as digital pins
    - Analog input 0 as digital pin 14 through analog input 5 as digital pin 19
    - Analog inputs 6 and 7 (present on the Mini and BT) cannot be used as digital pins
- I2C: 4 (SDA) and 5 (SCL)
  - Support I2C (TWI) communication using the wire library

Power pins

- Power pins
  - VIN (sometimes labelled "9V")
    - The input voltage to the Arduino board when it is using an external power source
      - As opposed to 5 volts from the USB connection or other regulated power source
    - You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin
    - Note that different boards accept different input voltages ranges, please see the documentation for your board
    - Also note that the LilyPad has no VIN pin and accepts only a regulated input

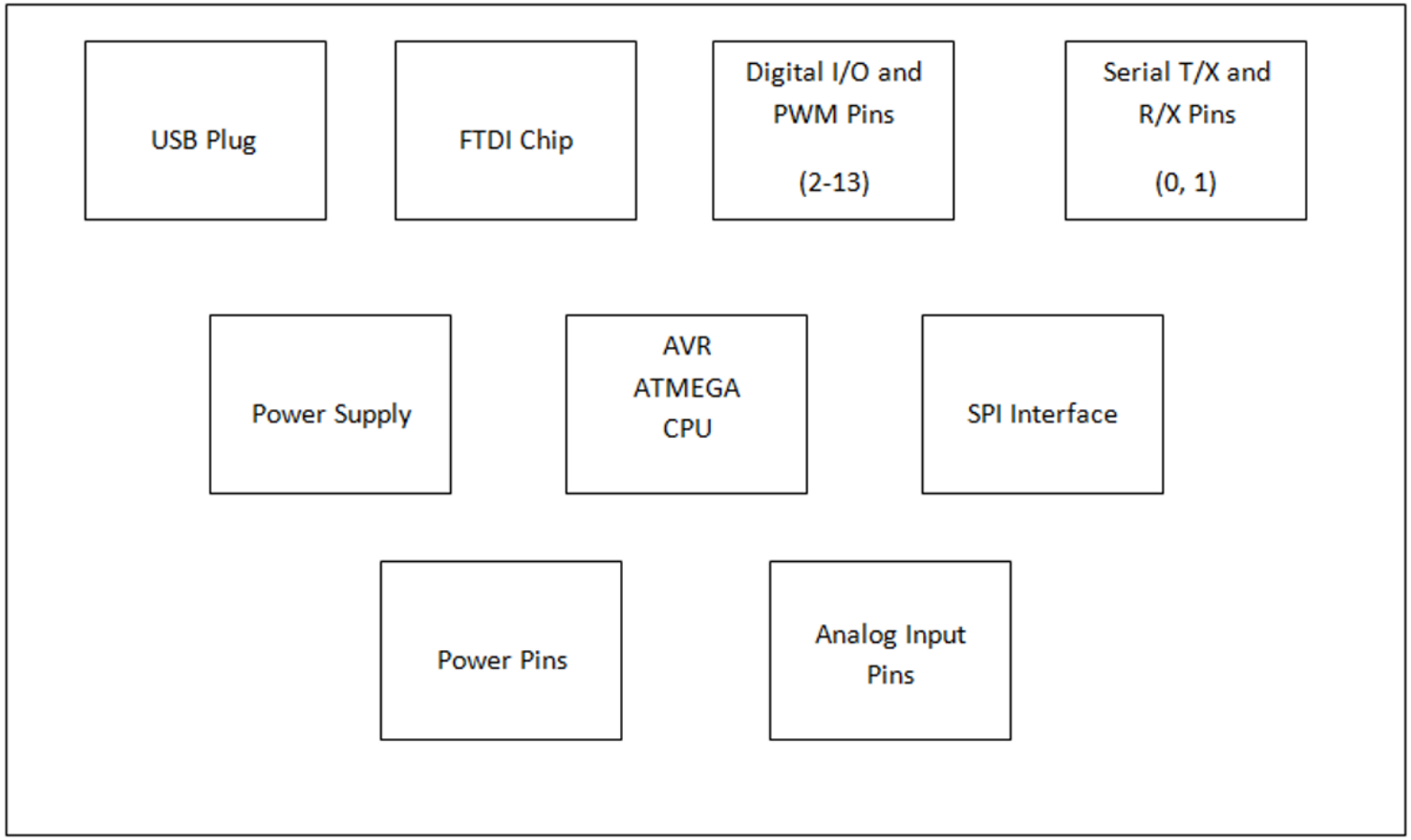
Arduino pins

- Power pins (continue)
  - 5V
    - The regulated power supply used to power the microcontroller and other components on the board
    - This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply
  - 3V3 (Diecimila-only)
    - A 3.3 volt supply generated by the on-board FTDI chip
  - GND
    - Ground pins

Arduino pins

- AREF
  - Reference voltage for the analog inputs
  - It provides the analog reference voltage for analog to digital conversion
- Reset (Diecimila-only)
  - Bring this line LOW to reset the microcontroller
  - Typically used to add a reset button to shields which block the one on the board

Block diagram of Arduino



Function of each block

- USB Plug
  - Download hex file of the program by connecting it with PC
  - Provide supply to Arduino board
  - This USB plug is connected with FTDI chip internally
- FTDI Chip
  - Is nothing but voltage level converter chip
  - Is used to convert USB voltage level to TTL level and vice versa

Function of each block

- Digital I/O and PWM section
  - Is used as for general purpose input output usage
  - Some pins are also used as PWM output pins
    - It is used for PWM based application like servo motors
- Serial Tx and Rx Section
  - These two pins are used for serial communication in Arduino
  - It is internally connected with USB plug through FTDI chip

Function of each block

- Power Supply
  - Power supply section provide different voltage level to different part of Arduino board
  - This part contain bridge rectifier and voltage regulator
- AVR Atmega CPU
  - Is the heart of the controller
  - Generally Atmega8, Atmega16, Atmega32, Atmega328 controller are used in Arduino board
  - Execute all the functions that written in Arduino program

Function of each block

- SPI Interface section
  - This section is used for serial peripheral interfacing
- Power pins section
  - This section include various voltage level pins including ground pins
- Analog input section
  - This section performs ADC operations

Arduino programming language

- Bare minimum code

```
void setup() {
    // put your setup code here, to run once:
}

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

- Setup
  - It is called only when the Arduino is powered on or reset
  - It is used to initialize variables and pin modes

Arduino programming language

- Bare minimum code

```
void setup() {
    // put your setup code here, to run once:
}

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

- Loop
  - The loop functions runs continuously till the device is powered off
  - The main logic of the code goes here
  - Similar to while(1) for micro-controller programming

Arduino programming language

- PinMode
  - A pin on Arduino can be set as input or output by using pinMode function
    - pinMode(13, OUTPUT); // sets pin 13 as output pin
    - pinMode(13, INPUT); // sets pin 13 as input pin

Arduino programming language

- Reading/writing digital values
  - digitalWrite(13, LOW); // Makes the output voltage on pin 13 , 0V
  - digitalWrite(13, HIGH); // Makes the output voltage on pin 13 , 5V
  - int buttonState=digitalRead(2); // reads the value of pin 2 in buttonState

Arduino programming language (up to/from here: session 17/18)

- Working with ADC
  - The Arduino Uno board contains 6 pins for ADC
  - 10-bit analog to digital converter
  - This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023
- Reading/Writing Analog Values
  - analogRead(A0); // used to read the analog value from the pin A0
  - analogWrite(2,128);

Arduino programming language

- Arduino timing
  - delay(ms)
    - Pauses for a few milliseconds
  - delayMicroseconds(us)
    - Pauses for a few microseconds
- More commands
  - https://www.arduino.cc/reference/en

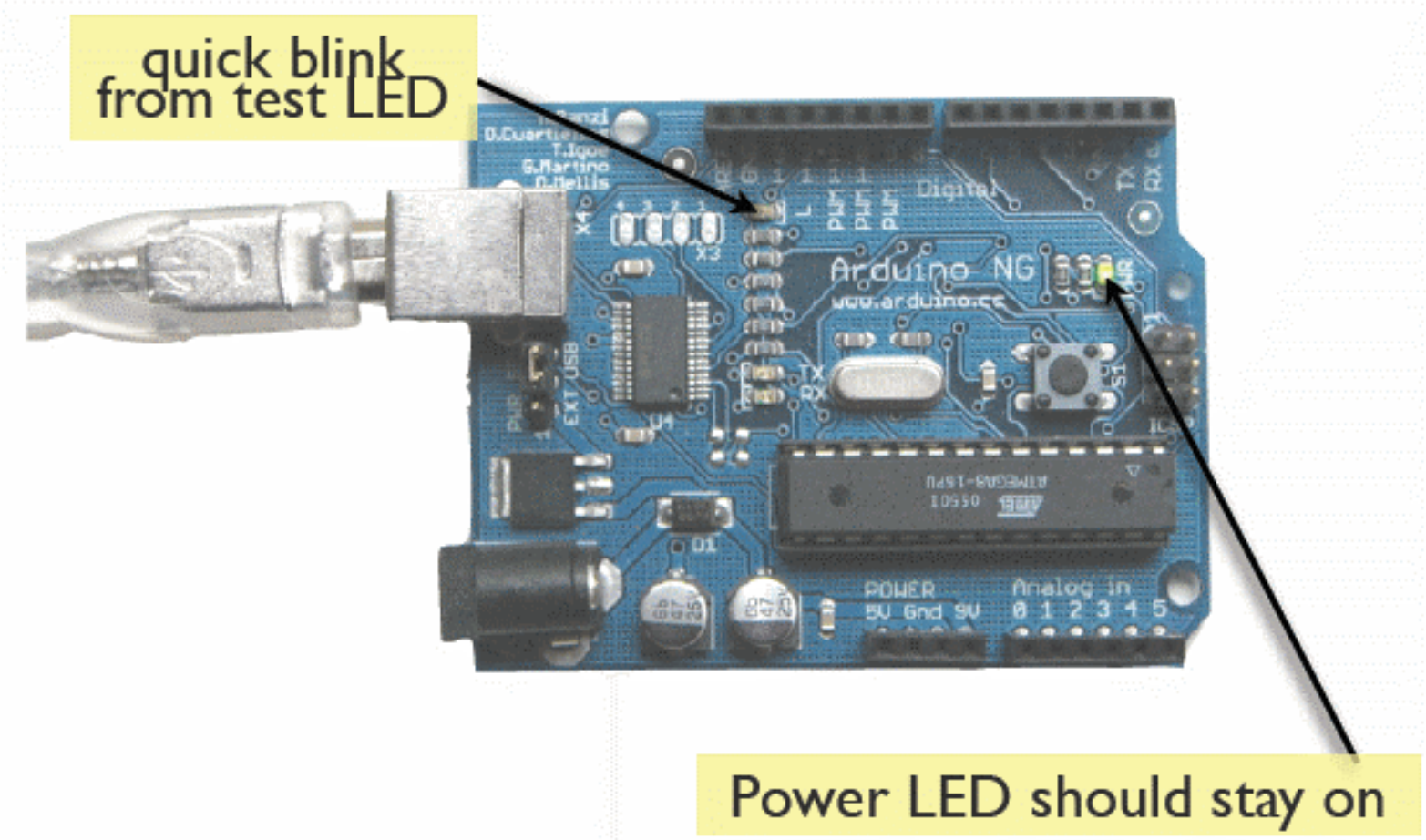
Getting started with Arduino

- Check out: http://arduino.cc/en/Guide/HomePage
  - Download and install the Arduino environment (IDE)
  - Connect the board to your computer via the USB cable
  - If needed, install the drivers
  - Launch the Arduino IDE
  - Select your board
  - Select your serial port
  - Open the blink example

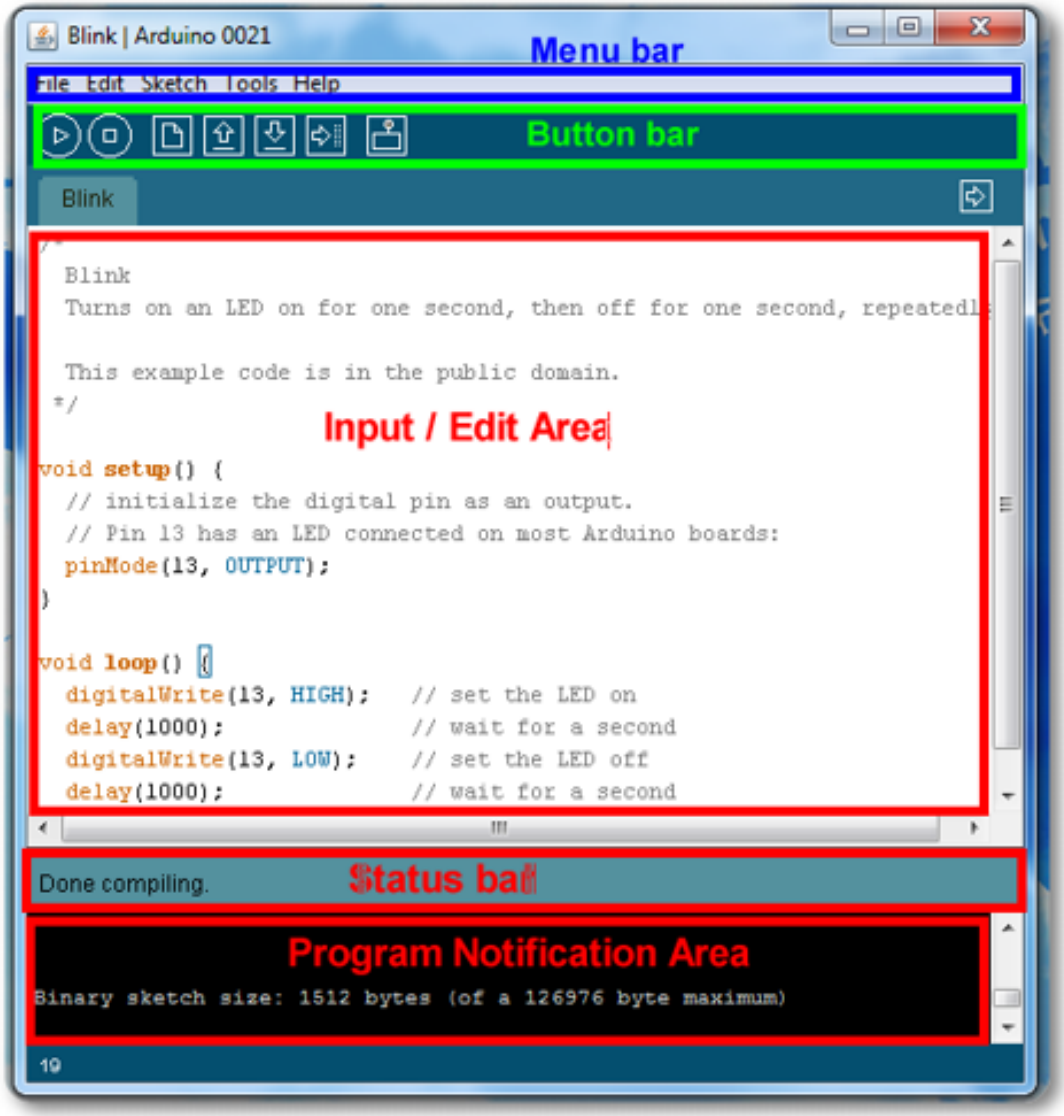
Getting started with Arduino

- Check out: http://arduino.cc/en/Guide/HomePage
  - Upload the program

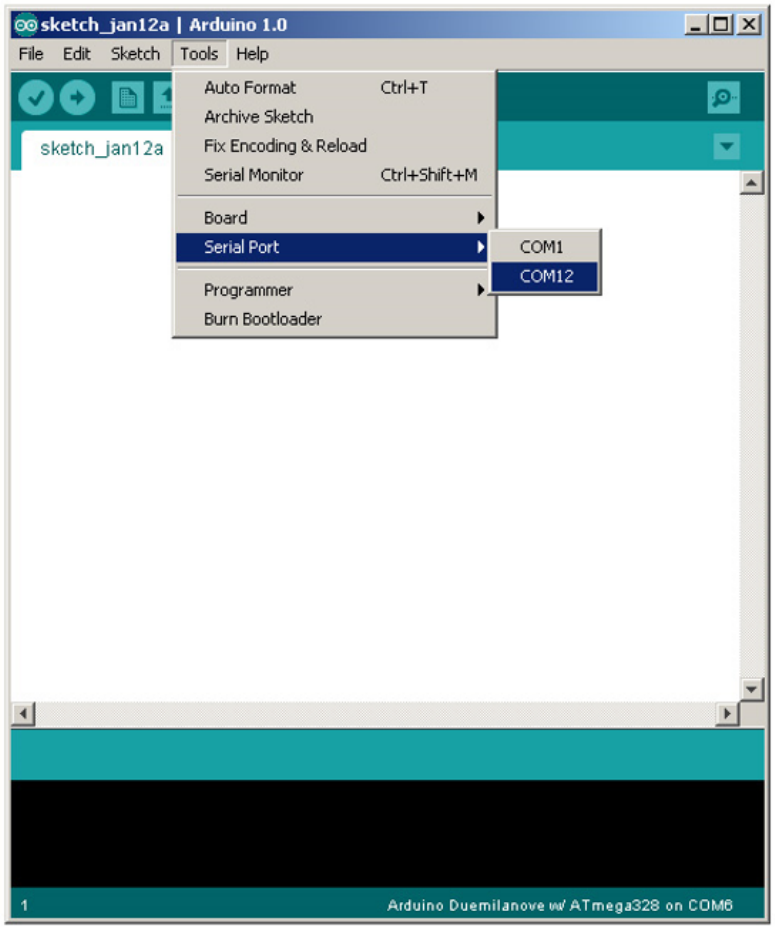
Try it: Connect the USB cable

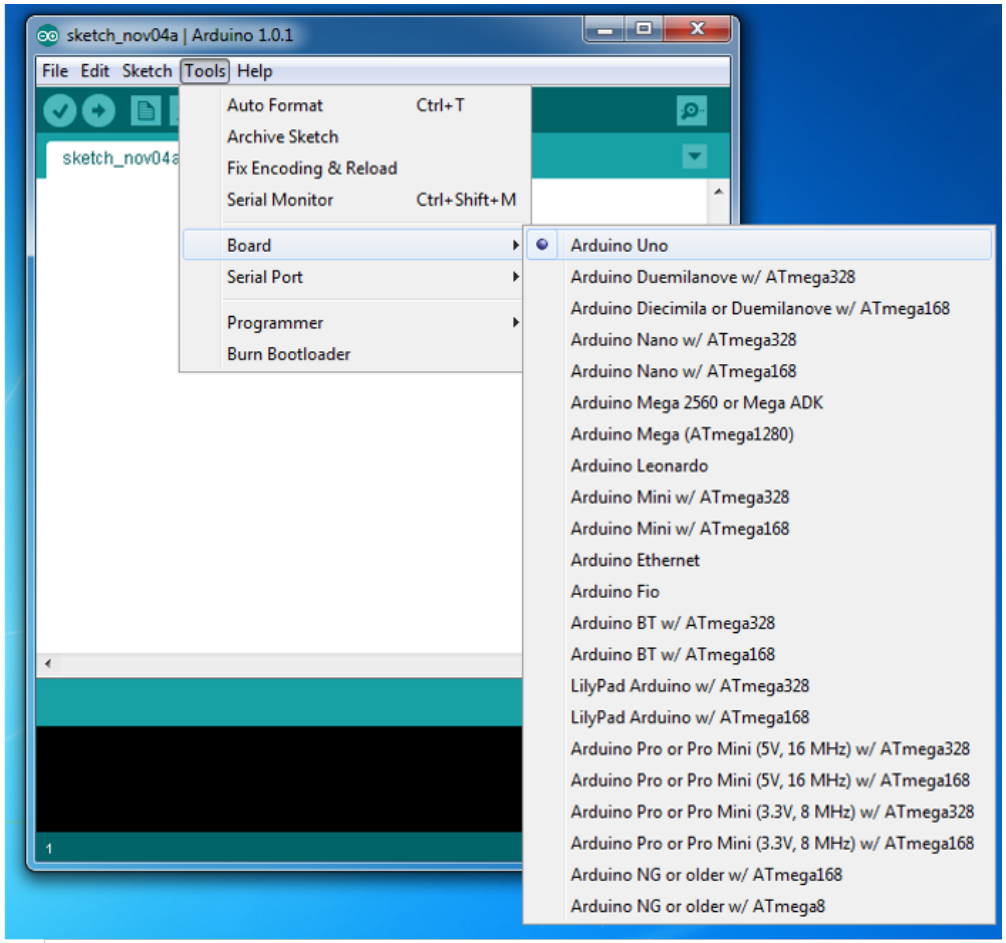


Arduino IDE


Blink | Arduino 0021
Menu bar
File Edit Sketch Tools Help
Button bar
Blink
Blink
Turns on an LED on for one second, then off for one second, repeatedl
This example code is in the public domain.
*/
Input / Edit Area
void setup() {
// initialize the digital pin as an output.
// Pin 13 has an LED connected on most Arduino boards:
pinMode(13, OUTPUT);
}
void loop() {
digitalWrite(13, HIGH); // set the LED on
delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
delay(1000); // wait for a second
Done compiling.
Status bar
Program Notification Area
Binary sketch size: 1512 bytes (of a 126976 byte maximum)
19

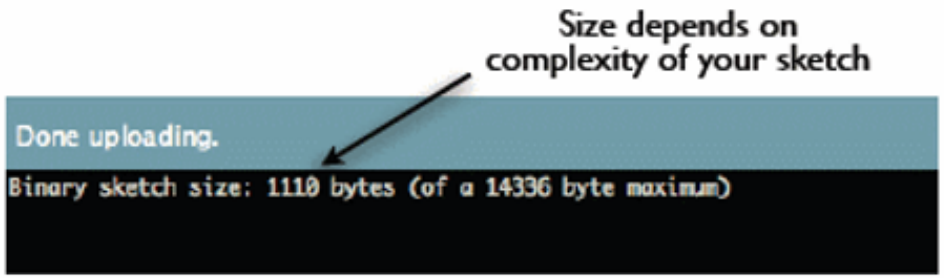
Select serial port and board





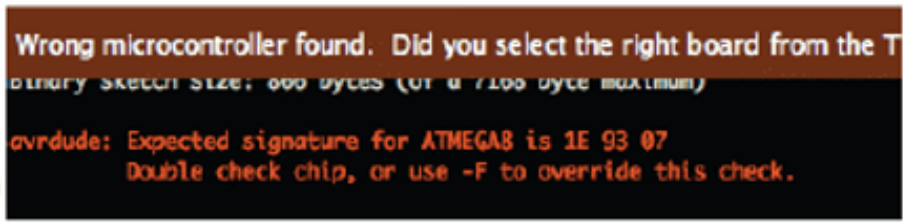
Status messages

Uploading worked



Wrong serial port selected

Wrong board selected



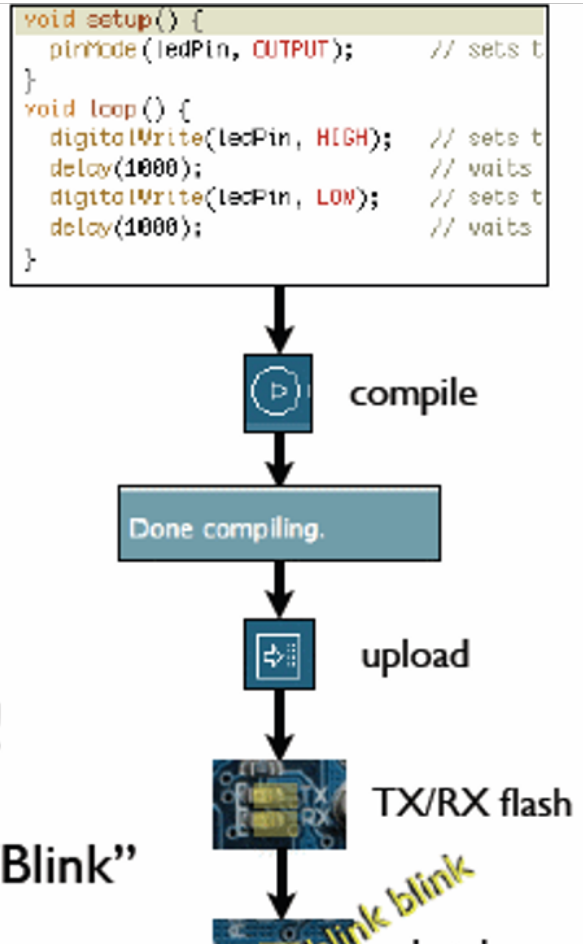
nerdy cryptic error messages

Using Arduino

- Write your sketch
- Press Compile button (to check for errors)
- Press Upload button to program Arduino board with your sketch

Try it out with the "Blink" sketch!

Load "File/Sketchbook/Examples/Digital/Blink"



First step: Blinking LED!

- Write a program to blink the built-in LED (Pin 13)

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Fading LED

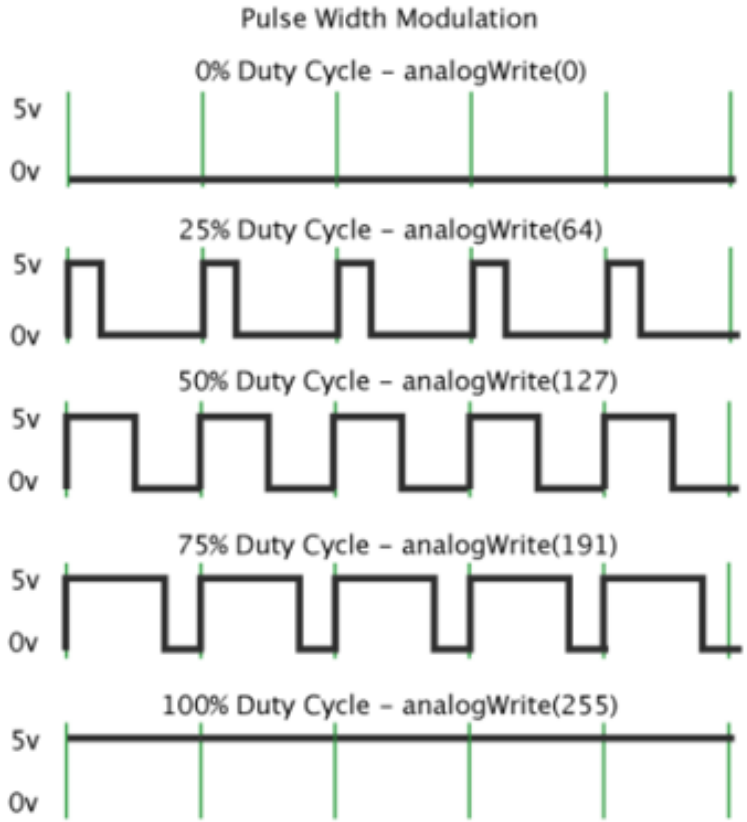
- Pulse Width Modulation (PWM)
  - A technique for getting analog results with digital means
  - Digital control is used to create a square wave, a signal switched between on and off
  - This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts)
    - By changing the portion of the time the signal spends on versus the time that the signal spends off

Fading LED

- Pulse Width Modulation (PWM)
  - This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts)
    - The duration of "on time" is called the duty cycle
    - To get varying analog values, you change, or modulate, that duty cycle
    - If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED

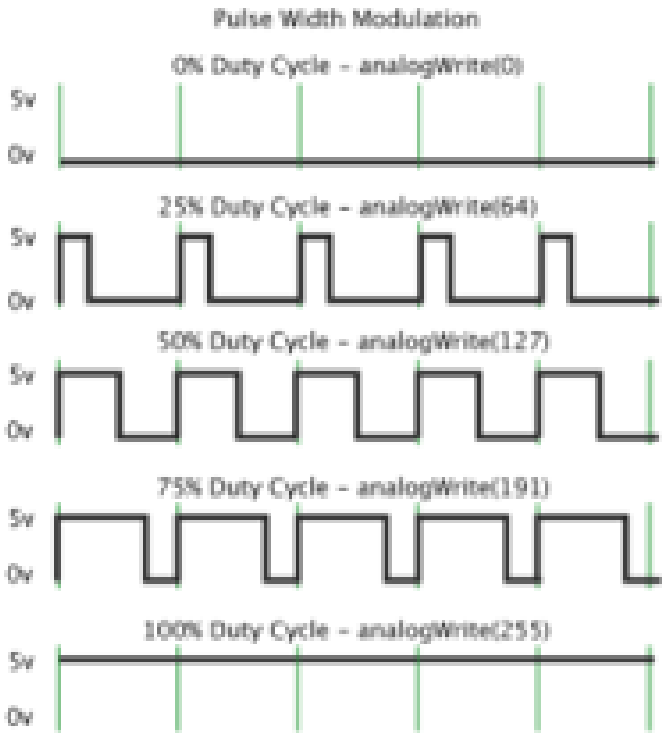
Fading LED

- In the graphic below, the green lines represent a regular time period



Fading LED

- This duration or period is the inverse of the PWM frequency
  - In other words, with Arduino's PWM frequency at about 500Hz
  - The green lines would measure 2 milliseconds each
  - A call to analogWrite() is on a scale of 0-255
    - analogWrite(255) requests a 100% duty cycle (always on)
    - analogWrite(127) is a 50% duty cycle (on half the time).



Fading LED

- Write a program to fade an LED on one of the PWM pin of Arduino

```
/*
  Fade

  This example shows how to fade an LED on pin 9 using the analogWrite()
  function.

  The analogWrite() function uses PWM, so if you want to change the pin you're
  using, be sure to use another PWM capable pin. On most Arduino, the PWM pins
  are identified with a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Fade
*/

int led = 9;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Arduino digital read and serial communication

- Write a program to read an digital voltage and show its value in the serial port

```
/*
  DigitalReadSerial

  Reads a digital input on pin 2, prints the result to the Serial Monitor

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/DigitalReadSerial
*/

// digital pin 2 has a pushbutton attached to it. Give it a name:
int pushButton = 2;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // read the input pin:
  int buttonState = digitalRead(pushButton);
  // print out the state of the button:
  Serial.println(buttonState);
  delay(1);        // delay in between reads for stability
}
```

Assignment for the next session: Arduino calculator

- Write a simple calculator program in Arduino. The calculator should read the equations from the serial port (USB) and should write the value again in serial port

Arduino calculator: Answer

```
/* Serial arduino calculator
in this project, you can make basic
arithmetic with the help of arduino,
almost like a very rustic calculator.
It accepts two numbers and a signal, and
makes the operation, witch can be of +, -, * or
/.
E.G. : send "2+3" (Without quotes and with no
space separing the info), and arduino answers 5.
Digit "7-3" and arduino te responde com 4.
Criado por João Paulo Rodrigues Poltronieri

This code is on public domain
*/

// first of all, create variables to store
// the information sent to arduino

long number1; // first number of the calculation,
// sent through the Serial monitor

// If you take a look, it's a long varible, so
// we're able to use big numbers

long number2; // second number sent through the
SM
```

```
char calSignal; // create a char variable to
store
// the calcuation signal.

long result; // result of the calculation

void setup() {
  Serial.begin(9600); // begins serial
communications
  Serial.println("Send me a calculation");
  Serial.println("E.G. : 2+3");
  Serial.println();
  // prints this to test serial communication,
and
  // prints a line space
}

void loop() {
  while(Serial.available() > 0) {
    // while there are dada being sent to
arduino,

    number1 = Serial.parseInt();
    // number1 will be the first number
```

Arduino calculator: Answer

```
// Note the use of "Serial.parseInt, so,
    // in case you use 23, it stores in
    // number1 the number 23

    // if we used Serial.read(), it would
    // only store 2

    calSignal = Serial.read(); // calSignal will
be the first
    // info after the first number

    number2 = Serial.parseInt(); // stores the
second
    // number in number2

    resolve(); // Custom function to solve the
calculations

    Serial.println("Result = ");
    Serial.println(result);
    // Prints the result of the calculation
    Serial.println(); // jumps a line
    Serial.println("Send me a calculation");
    Serial.println("E.G. : 2+3");
    Serial.println();
    Serial.read();
  }

void resolve() { // Custom function that
  // solves the calculations

  switch (calSignal) {
    // Here we use "switch...case" to save some
space on
    // the sketch. It's, basicaly, a function
that verifies
    // various "if" statements.

    // Here, it verifies what's the value held by
    // calSigna. Basicaly, it verifies the
"signal"
    // of the calculation

    case '+' : // if calSignal is '+'
    result = number1 + number2; // sums the
numbers
    // and makes result hold the value of the
calculation
    break; // break to exit the "case"
    case '-' : // if calSignal is '+'
    result = number1 - number2; // subtracts the
numbers
```

Arduino calculator: Answer

```
// and makes result hold the value of the
calculation
    break; // break to exit the "case"
    case '*' : // if calSignal is '+'
    result = number1 * number2; // multiplies the
numbers
    // and makes result hold the value of the
calculation
    break; // break to exit the "case"
    case '/' : // se calSignal for '/'
    result = number1 / number2; // divides the
numbers
    // and makes result hold the value of the
calculation
    // PS: in case the division isn't exact, the
result
    // will be the nearest integrer
    break; // break to exit the "case"
    default : // If it's not any of these...
    Serial.println("Error in inputs");
    // Creates an "error"
    Serial.println();
    result = 0;
    break;
  }
}
```

Arduino cheat sheet to quickly write programs!

Structure
void **setup**() void **loop**()

Control Structures
if (x<5){ } **else** { }
switch (myvar) {
  case 1:
  break;
  case 2:
  break;
  default:
}
for (int i=0; i <= 255; i++){ }
while (x<5){ }
do { } **while** (x<5);
continue; //Go to next in do/for/while loop
return x; // Or 'return;' for voids.
goto // considered harmful :-)

Further Syntax
// (single line comment)
/* (multi-line comment) */
#define DOZEN 12 //Not baker's!
#include <avr/pgmspace.h>

General Operators
= (assignment operator)
+ (addition) - (subtraction)
* (multiplication) / (division)
% (modulo)
== (equal to) != (not equal to)
< (less than) > (greater than)
<= (less than or equal to)
>= (greater than or equal to)
&& (and) || (or) ! (not)

Pointer Access
& reference operator
* dereference operator

Bitwise Operators
& (bitwise and) | (bitwise or)
^ (bitwise xor) ~ (bitwise not)
<< (bitshift left) >> (bitshift right)

Compound Operators
++ (increment) -- (decrement)
+= (compound addition)
-= (compound subtraction)
*= (compound multiplication)
/= (compound division)
&= (compound bitwise and)
!= (compound bitwise or)

Constants
HIGH | LOW
INPUT | OUTPUT
true | false
143 // **Decimal** number
0173 // **Octal** number
0b11011111 //**Binary**
0x7B // **Hex** number
7**U** // Force unsigned
10**L** // Force long
15**UL** // Force long unsigned
10.0 // Forces floating point
2.4**e**5 // 240000

Data Types
void
boolean (0, 1, false, true)
char (e.g. 'a' -128 to 127)
unsigned char (0 to 255)
byte (0 to 255)
int (-32,768 to 32,767)
unsigned int (0 to 65535)
word (0 to 655word (0 to 65535)
long (-2,147,483,648 to 2,147,483,647)
unsigned long (0 to 4,294,967,295)
float (-3.4028235E+38 to 3.4028235E+38)
double (currently same as float)
sizeof(myint) // returns 2 bytes

Strings
char S1[15];
char S2[8]={'a','r','d','u','i','n','o'};
char S3[8]={'a','r','d','u','i','n','o','\0'};
//Included \0 null termination
char S4[ ] = "arduino";
char S5[8] = "arduino";
char S6[15] = "arduino";

Arrays
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};

Conversion
char() **byte**()
int() **word**()
long() **float**()

Qualifiers
static // persists between calls
volatile // use RAM (nice for ISR)
const // make read-only
PROGMEM // use flash

Digital I/O
pinMode(pin, [INPUT,OUTPUT])
digitalWrite(pin, value)
int **digitalRead**(pin)
//Write High to inputs to use pull-up res

Analog I/O
analogReference([DEFAULT, INTERNAL,EXTERNAL])
int **analogRead**(pin) //Call twice if switching pins from high Z source.
analogWrite(pin, value) // PWM

Advanced I/O
tone(pin, freqhz)
tone(pin, freqhz ,duration_ms)
noTone(pin)
shiftOut(dataPin, clockPin, [MSBFIRST,LSBFIRST], value)
unsigned long **pulseIn**(pin,[HIGH,LOW])

Time
unsigned long **millis**() // 50 days overflow.
unsigned long **micros**() // 70 min overflow
delay(ms)
delayMicroseconds(us)

Math
min(x, y) **max**(x, y) **abs**(x)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)
pow(base, exponent) **sqrt**(x)
sin(rad) **cos**(rad) **tan**(rad)

Random Numbers
randomSeed(seed) // Long or int
long **random**(max)
long **random**(min, max)

Bits and Bytes
lowByte()
highByte()
bitRead(x,bitn)
bitWrite(x,bitn,bit)
bitSet(x,bitn)
bitClear(x,bitn)
bit(bitn) //bitn: 0-LSB 7-MSB

External Interrupts
attachInterrupt(interrupt, function, [LOW,CHANGE,RISING,FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()

Libraries:

Serial.
  begin([300, 1200, 2400, 4800, 9600,14400, 19200, 28800, 38400, 57600,115200])
  end()
  int **available**()
  int **read**()
  flush()
  print()
  println()
  write()

EEPROM (#include <EEPROM.h>)
byte **read**(intAddr)
  write(intAddr,myByte)

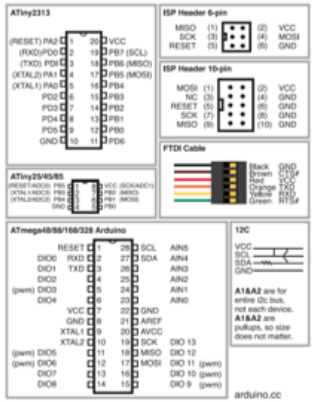
Servo (#include <Servo.h>)
  attach(pin , [min_uS, max_uS])
  write(angle) // 0-180
  writeMicroseconds(uS) //1000-2000,1500 is midpoint
  read() // 0-180
  attached() //Returns boolean
  detach()

SoftwareSerial(RxPin,TxPin)
  // #include<**SoftwareSerial**.h>
  begin(longSpeed) // up to 9600
  char **read**() // blocks till data
  print(myData) or **println**(myData)

Wire (#include <Wire.h>) // For **I2C**
  begin() // Join as master
  begin(addr) // Join as slave @ addr
  requestFrom(address, count)
  beginTransmission(addr) // Step 1
  send(mybyte) // Step 2
  send(char * mystring)
  send(byte * data, size)
  endTransmission() // Step 3
  byte **available**() // Num of bytes
  byte **receive**() //Return next byte
  onReceive(handler)
  onRequest(handler)

	ATMega168	ATMega328	ATMega1280
Flash (2k for bootloader)	16kB	32kB	128kB
SRAM	1kB	2kB	8kB
EEPROM	512B	1kB	4kB

	Duemilanove/ Nano/ Pro/ ProMini	Mega
# of IO	14 + 6 analog (Nano has 14 + 8)	54 + 16 analog
Serial Pins	0 - RX 1 - TX	0 - RX1 1 - TX1 19 - RX2 18 - TX2 17 - RX3 16 - TX3 15 - RX4 14 - TX4
Ext Interrupts	2 - (Int 0) 1 - (Int 1)	2,3,21,20,19,18 (IRQ0 - IFQ5)
PWM Pins	5,6 - Timer 0 9,10 - Timer 1 3,11 - Timer 2	0 - 13
SPI	10 - SS 11 - MOSI 12 - MISO 13 - SCK	53 - SS 51 - MOSI 50 - MISO 52 - SCK
I2C	Analog4 - SDA Analog5 - SCK	20 - SDA 21 - SCL



Introduction

- What is the main problem of current programming languages in dealing with the different problems?
  - Dealing with problems that have complex behaviors
    - Like: embedded monitor systems, embedded health care systems, and etc.
  - Solution
    - Automata-based programming (ABP)
    - The idea has come from automata theory and control theory

ABP as a programming style

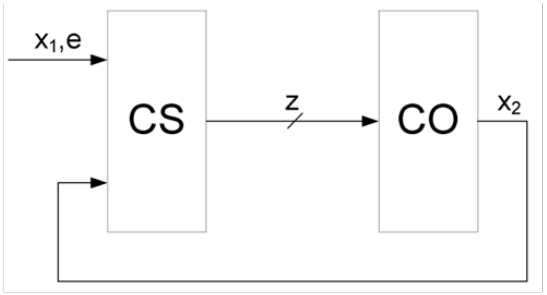
- Automata-based programming doesn't mean programming with the use of automata, but
  - The entire programming paradigm and programming technology
    - Aimed for designing systems with complex behavior
- Automata programming proposes to describe the behavior of programs
  - Using Automata
  - Which are later isomorphically converted into the code

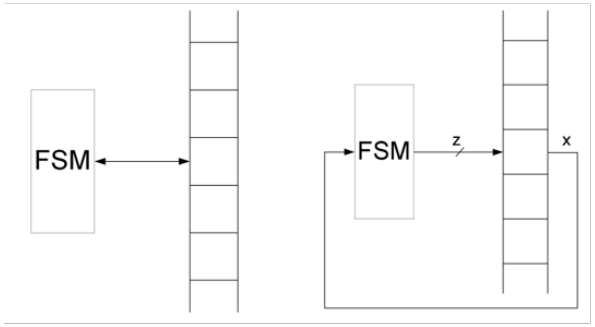
ABP as a programming style

- Programming using automata can't be seen as a programming paradigm
  - It still leaves unclear how to design and implement a program as a whole using automata!

ABP as a programming paradigm (up to/from here: session 18/19)

- Many systems with meaningful behavior are nothing but automated objects
- Automated control object
  - An aggregate of the control object (CO) and control system (CS) related with feedbacks

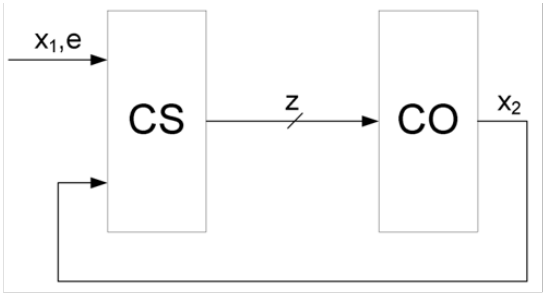


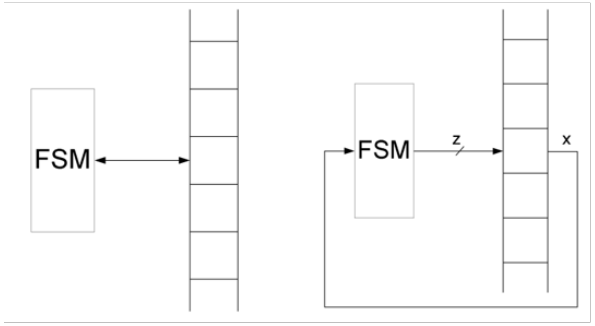


Turing Machine Automatic Turing Machine

ABP as a programming paradigm

- The task of designing automated control objects is discussed in every course of automated control theory
  - Surprisingly, it didn't affect software engineering practice
    - In spite of the fact that one of the major models in algorithms is the Turing machine





Turing Machine Automatic Turing Machine

ABP as a programming paradigm

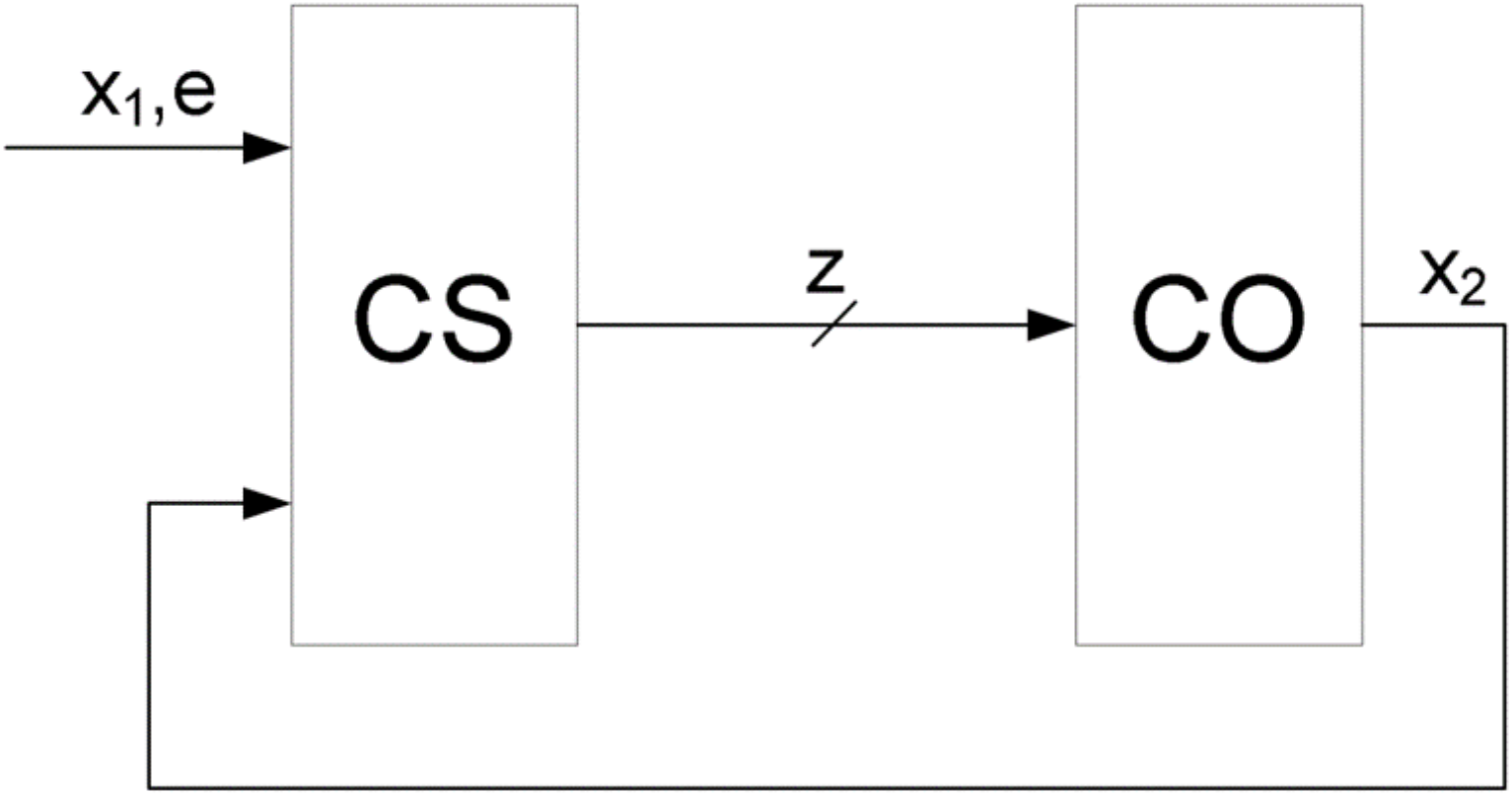
- The main feature of automata-based programming
  - Programs should be coded in a way similar to the automatization of the technological processes
- All listed components are shown on a relation diagram
  - Which can also be an interaction diagram
  - For each input and output action, the full name is shown as well as a short identifier, that is used in transition diagrams (state diagrams) and in the code
  - Short identifiers make transition diagrams compact and comprehensible

ABP as a programming paradigm

- All listed components are shown on a relation diagram
  - Control objects can be either real or virtual (implemented as programs) - In the first case, their logic is fixed, in the second case the logic of the control objects can and should be extracted into the automata in the control systems

ABP as a programming paradigm

- Paradigm of automata-based programming
  - Representation and implementation of programs as systems of automated control objects



Main thesis of ABP

- Main concept in the automata-based programming
  - State
- Different types of states in ABP
  - Control state
    - As in Turing machines, a few control states is enough to control many computational states
  - Computational state
    - Not covered here!

Main thesis of ABP

- Different types of automata
  - Abstract
    - Input and output actions are formed consequently
  - Structure
    - Actions formed simultaneously
    - ABP uses this type of automata
- Time is not used in automata
  - If needed
    - Delay elements are added as control objects
    - Time elapse events are received by the automata as input actions, Called “Timed automata”

Main thesis of ABP

- Designing automata
  - Manually
    - Should possess cognitive properties
      - Reached when automata are represented as transition graphs (transition diagram)
  - Automatically
    - Can be represented as a table
      - Less comprehensible

System specification

- The first and the most important step in the design flow
  - Requires human intelligence
  - Can we use natural language?
  - It is necessary to check specifications for
    - Completeness
    - Absence of contradictions
- It should be possible to derive implementations from the specification in a systematic way

System specification: Required features

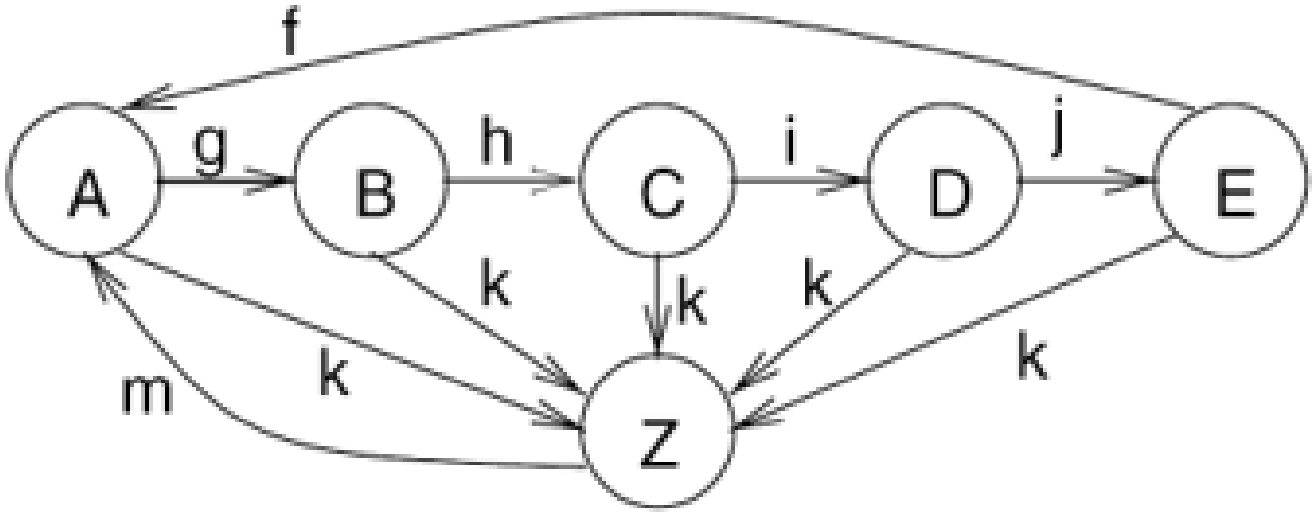
- Specification languages for ES should have the following features:
  - Hierarchy
    - Behavioral hierarchies : e.g., Super-states
    - Structural hierarchies : Like what is supported by VHDL and Verilog
  - Timing behavior
    - Delay
    - Cause and effect relationship

System specification: Required features

- Specification languages for ES should have the following features:
  - State-oriented behavior
    - Automata provide a good mechanism for modeling reactive systems
  - Event handling
    - The reactive nature of ES
    - Mechanisms for describing events must exist
      - External events (caused by environment)
      - Internal events (caused by components of the system)
  - Support for efficient implementation

System specification: Required features

- Specification languages for ES should have the following features:
  - Support for dependable system design
    - Unambiguous semantics
    - Facilitate formal verification
  - Exception-oriented behavior
    - It is not acceptable that exceptions have to be indicated for each and every state



System specification: Required features

- Specification languages for ES should have the following features:
  - Concurrency
    - Real-life systems are concurrent systems
    - It is necessary to be able to specify concurrency conveniently
  - Synchronization and communication
    - Concurrent actions have to be able to communicate and it must be possible to agree on the use of resources (e.g., mutual exclusion)

System specification: Required features

- Specification languages for ES should have the following features:
  - Presence of programming elements
    - Usual programming languages have proven to be a convenient means of expressing computations
    - Classical hardware description techniques (e.g., state diagrams) do not meet this requirement
  - Executability
    - Simulation (design verification)

System specification: Required features

- Specification languages for ES should have the following features:
  - Readability and flexibility
    - Readable by human
    - Small changes of the system small changes of the specification
  - Support for non-standard I/O-devices
    - To conveniently describe inputs and outputs for non-standard I/O-devices

System specification: Required features

- Specification languages for ES should have the following features:
  - Non-functional properties
    - A non-functional property (NFP) of a software system is a constraint on the manner in which the system implements and delivers its functionality
      - Reliability
      - Size
      - Power consumption

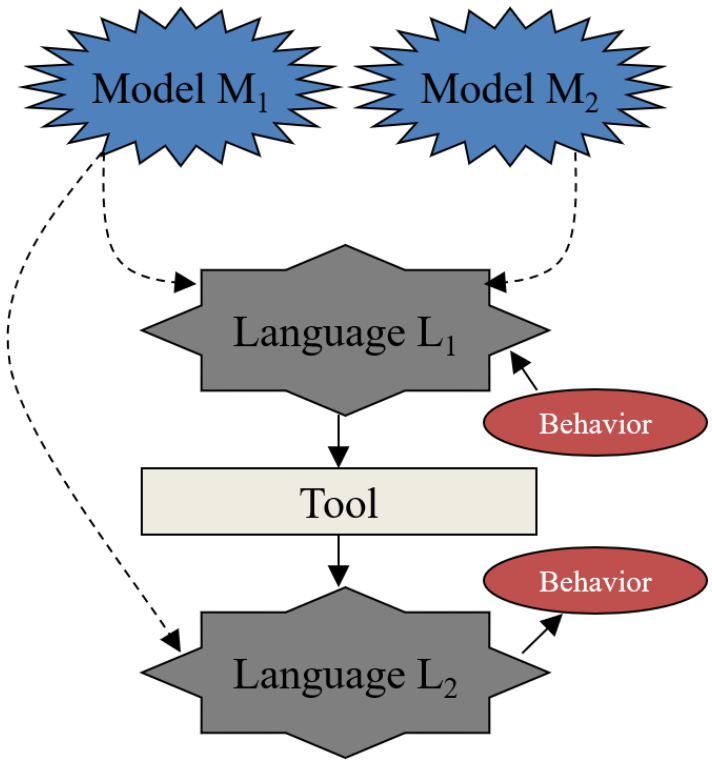
System specification: Required features

- Specification languages for ES should have the following features:
  - Appropriate model of computation (MOC)
    - Von Neumann paradigm is not suitable

There is no hope to develop a formal language capable of meeting all these requirements.

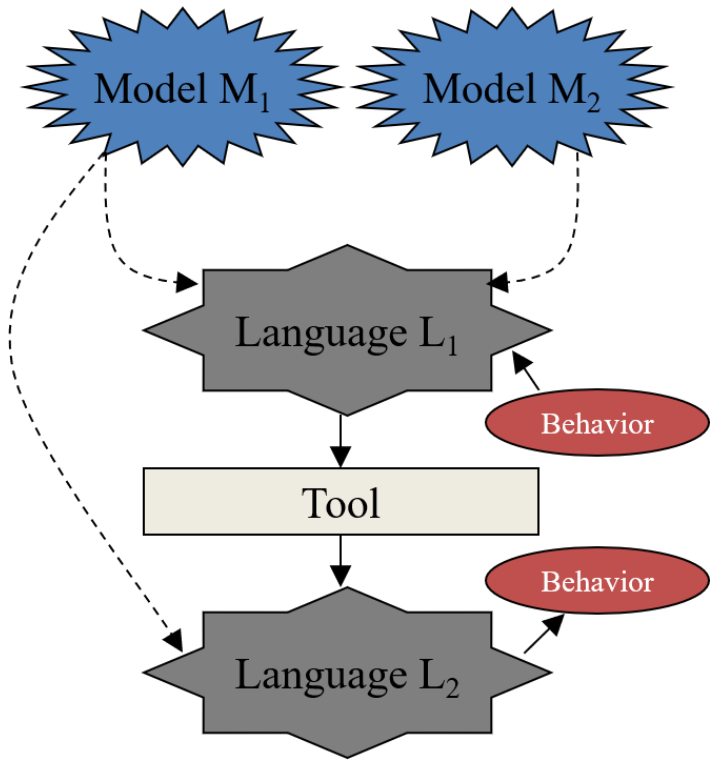
Models, languages, tools

- A model of computation (MOC) is a conceptual notions used to capture a system behavior, e.g.:
  - A set of objects
  - Composition
  - Execution semantics
- A languages defines the syntax to capture a models of computation



Models, languages, tools

- A tool is a "compiler" transforming a model captured in one language to a model captured in another language



Models of computation

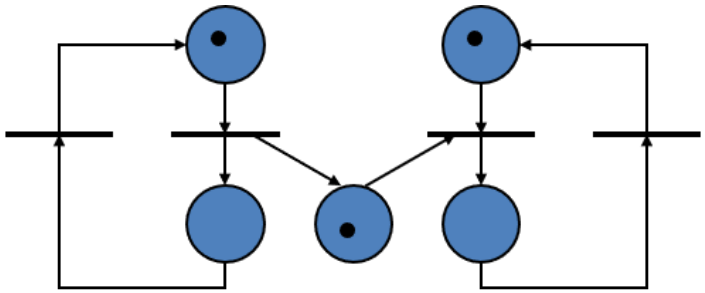
- Sequential model of computation
  - A single thread of execution
- Concurrent model of computation
  - Multiple threads of execution
  - Synchronization points
  - Communication mechanisms

Models of computation

- Object oriented (OO) model of computation
  - View the computation as a set of objects
    - Based, inherited, or composed
  - Polymorphism: a derived class can modify the behavior of its base class

Models of computation

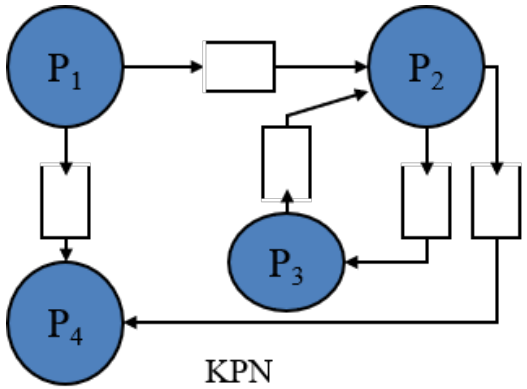
- FSM
  - A set of states and transitions
  - Mealy and Moore
- DFG
  - A set of computation nodes and flow paths
- Petri net
  - A set of places, transitions, edges, and tokens
  - Offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution



Petri net

Models of computation

- Kahn Process Network (KPN)
  - A distributed model of computation
  - A group of deterministic sequential processes are communicating through unbounded FIFO channels
  - The resulting process network exhibits deterministic behavior that does not depend on the various computation or communication delays
- Communicating Sequential Processes (CSP)
  - A formal model for describing patterns of interaction in concurrent systems



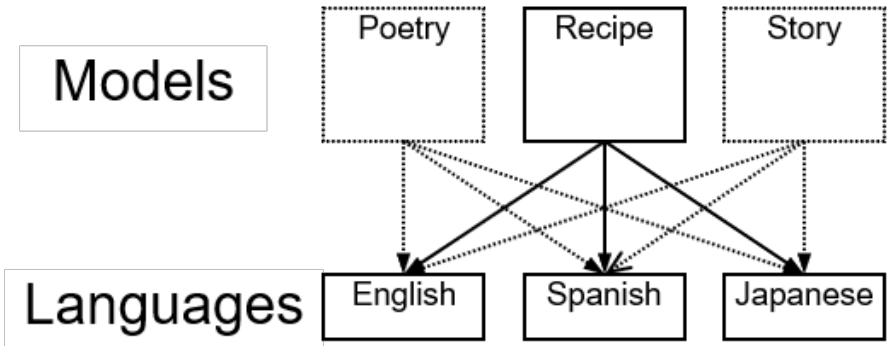
KPN

Models vs. languages

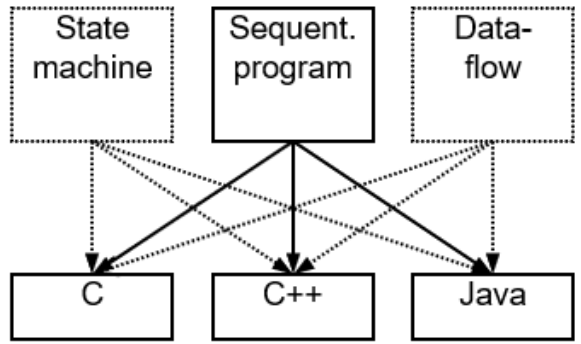
- MoCs describe system behavior
  - Conceptual notion, e.g., recipe, sequential program
- Languages capture models
  - Concrete form, e.g., English, C
- Variety of languages can capture one model
  - E.g., sequential program model –> C,C++, Java
- One language can capture variety of models
  - E.g., C++ –> sequential program model, object-oriented model, state machine model

Models vs. languages

- Certain languages better at capturing certain computation models



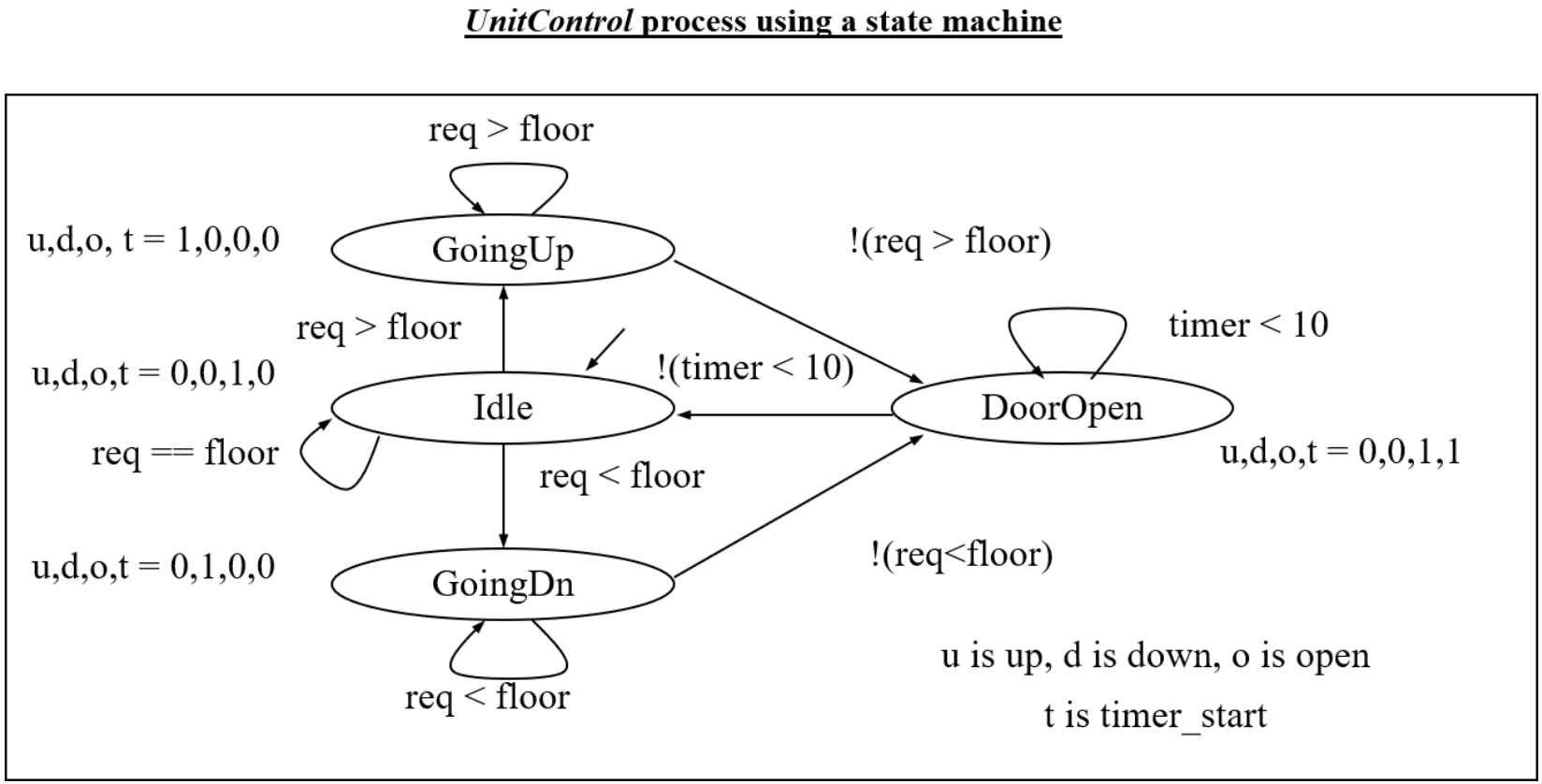
Recipes vs. English



Sequential programs vs. C

Finite-State Machine (FSM) model

- Elevator FSM



Formal definition (up to/from here: session 19/20)

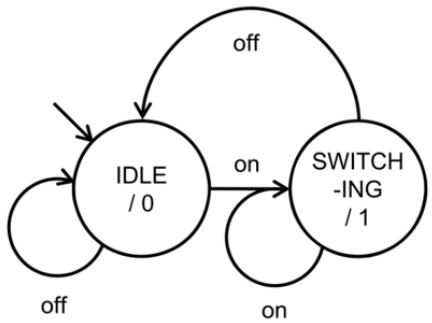
- An FSM is a 6-tuple F <S, I, O, F, H,S_0 >
  - S is a set of all states{ $s_0, s_1, ..., s_i$ }
  - I is a set of inputs{$i_0, i_1, ..., i_m$ }
  - O is a set of outputs{ $o_0, o_1, ..., 0_m$}
  - F is a next-state function (S x I –> S)
  - H is an output function (S –> O)
  - $s_0$is an initial state

Formal definition

- Moore-type
  - Associates outputs with states (as given above, H maps S –> O)
- Mealy-type
  - Associates outputs with transitions (H maps S x I –> O)
- Shorthand notations to simplify descriptions
  - Implicitly assign 0 to all unassigned outputs in a state
  - Implicitly AND every transition condition with clock edge (FSM is synchronous)

How to code FSMs in C

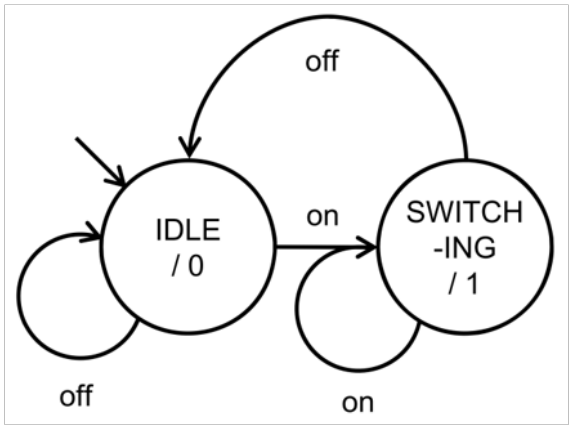
- State machines are the primary elements of ABPs
- State machine
  - A model of a machine that reacts to any change at inputs from outside
    - Generates outputs accordingly
  - The way state machine reacts to input changes depends on its state
- Example: Useless machine
  - Switches itself off
    - Every time someone switches it on!



How to code FSMs in C

- Typical way to translate the behavior of "useless machine" into a control flow

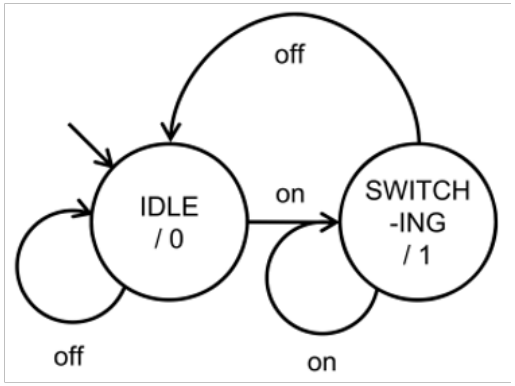
```
int get_switch_position(); /* Input function defined elsewhere */
void move_finger(); /* Output function defined elsewhere */

int main() {
   while( 1 ) { /* Infinite loop, as the controller is permanently on */
        while( get_switch_position() == 0 ) {
            move_finger( 0 );
        } /* while */
        while( get_switch_position() == 1 ) {
            move_finger( 1 );
        } /* while */
   } /* while */
} /* main */
```



How to code FSMs in C

- It's much better you code into a state-based control-flow
  - With states of the machine explicitly stored in a variable!

```
int get_switch_position(); /* Input function defined elsewhere */
void move_finger(); /* Output function defined elsewhere */
enum { IDLE, SWITCHING } state;
int main() {
    state = IDLE; /* Initially, the device's switch is off */
    while( 1 ) { /* Infinite loop, as the controller is permanently on */
           switch( state ) {
                 case IDLE:
                      move_finger( 0 );
                 if( get_switch_position() == 1 ) { state = SWITCHING; }
                   break;
                 case SWITCHING:
                 move_finger( 1 );
                 if( get_switch_position() == 0 ) { state = IDLE; }
                 break;
          } /* switch */
       } /* while */
} /* main */
```



How to code FSMs in C

- To translate a FSM graph into a C program
  - Enumerate all states
    - enum { S0, S1, S2, Sstop } state;
  - Set the state variable to the initial state
    - state = S0;
  - Set the condition of the while loop properly
    - while(state != Sstop)

How to code FSMs in C

- Rules to translate a FSM graph into a C program
  - Include a case for every state symbol in the enumeration, but for the stop state
    - case S0:
      break;
      case S1:
      break;
      case S2:
      break;

How to code FSMs in C

- Rules to translate a FSM graph into a C program
  - For every state case, you have to
    - Perform all actions within the state, i.e. computing outputs, and
      - Out (0);
    - Include an if condition to test each outgoing arc condition
      - if(C(1, 0)) state = S0;
      - if(C(1, END)) state = Sstop;
      - if(C(1, 2)) state = S2;
      - functions C() and Out() have to be replaced by corresponding condition evaluation and output updating procedures

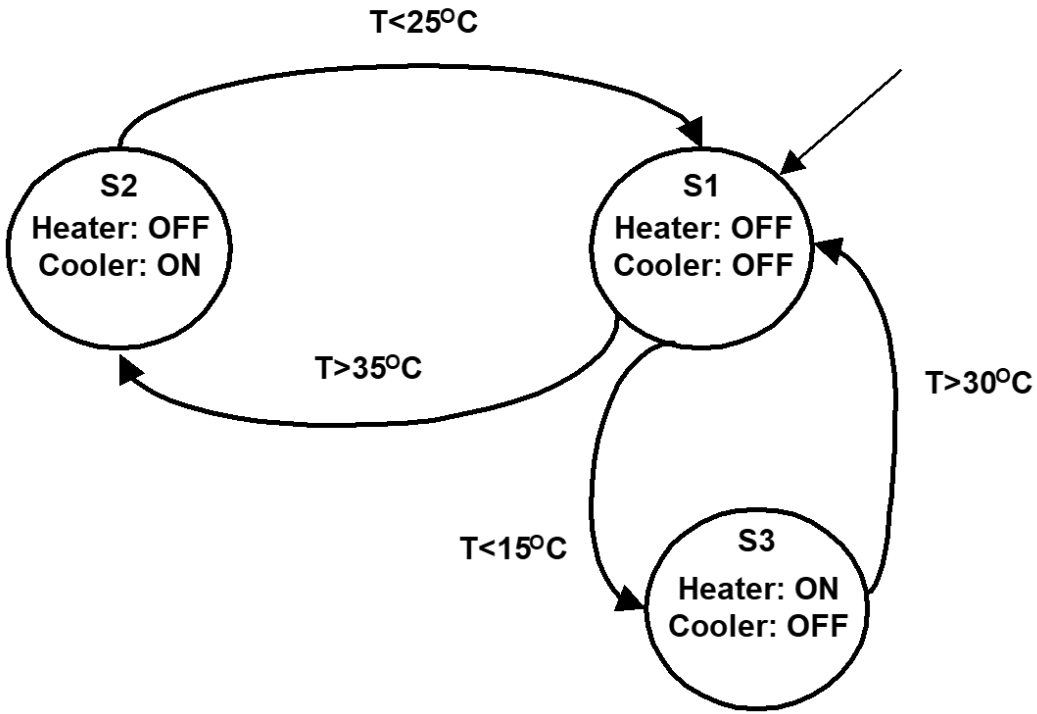
As you see:

It is quite a straightforward procedure!

In-class assignment: Question

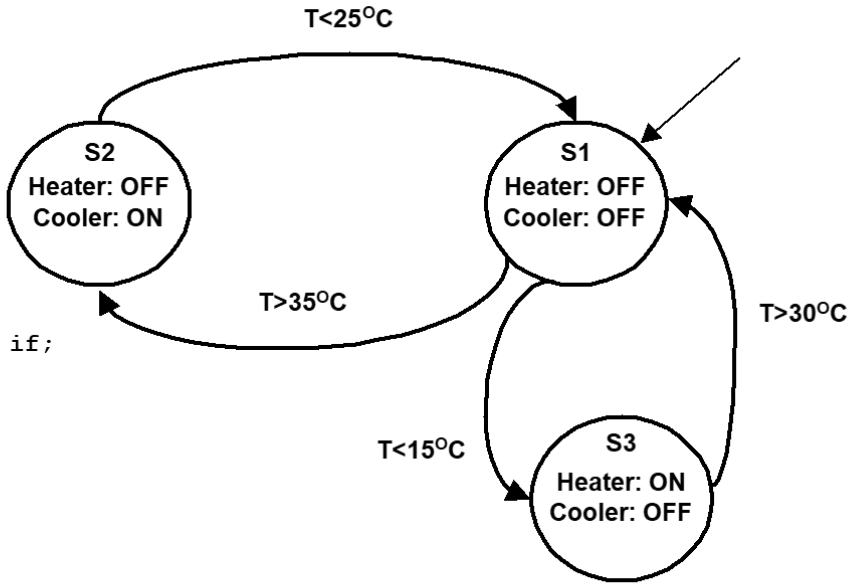
- Design an air conditioning system that keeps the environment temperature between 15 to 35 degree of centigrade
  - Consider a heater and a cooler modules

In-class assignment: Answer



In-class assignment: Answer

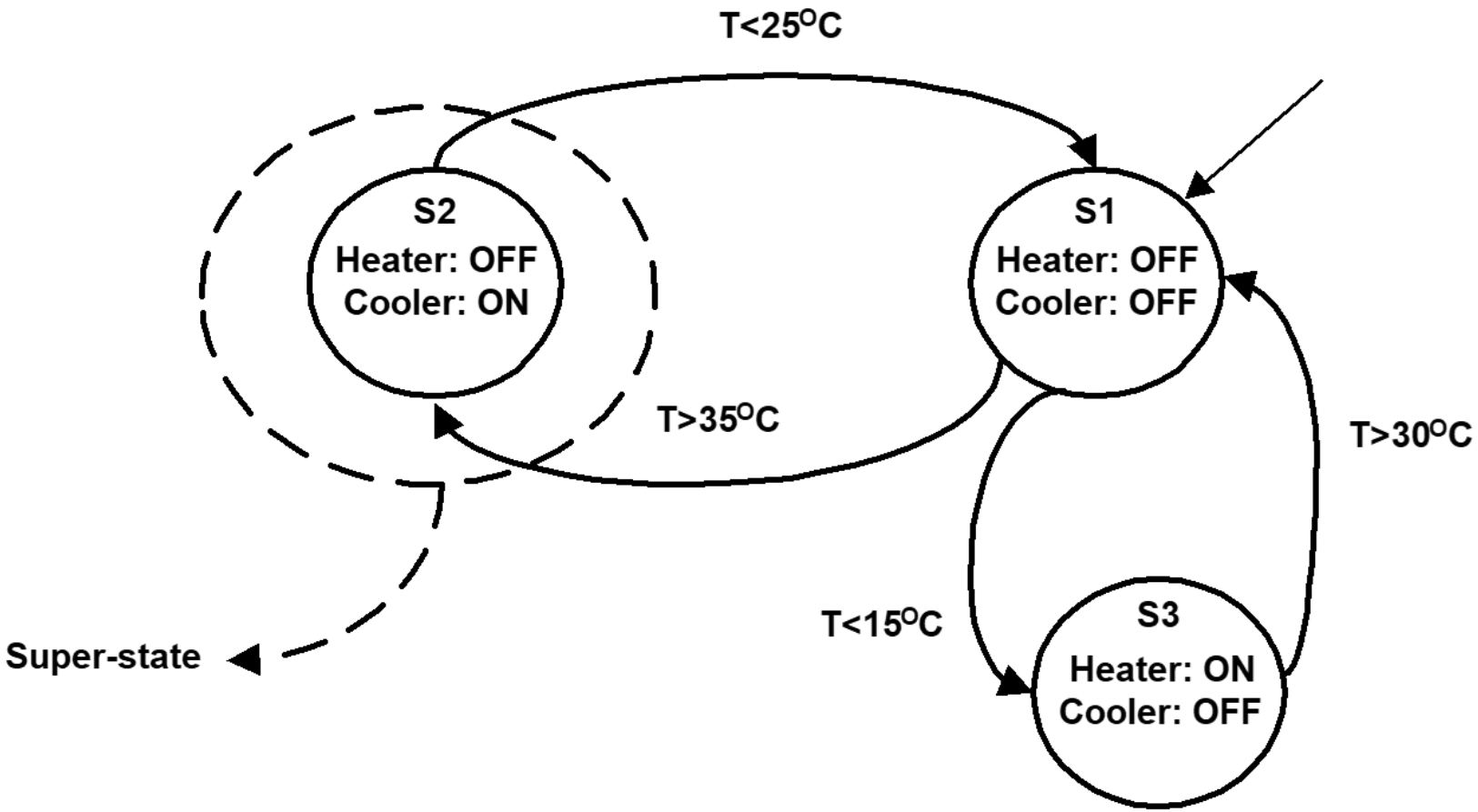
```
State PS=S1,NS;
Event e;

while(1){
  case(PS){
    S1:
      Turn_off(Heater);
      Turn_off(Cooler);
      e=Wait_for_event();
      if(e=='T<15') NS=S3;
      else if(e=='T>35') NS=S2; end if;
    S2:
      Turn_off(Heater);
      Turn_on(Cooler);
      e=Wait_for_event();
      if(e=='T<25') NS=S1; end if;
    S3:
      Turn_on(Heater);
      Turn_off(Cooler);
      e=Wait_for_event();
      if(e=='T>30') NS=S1; end if;
  }
  PS=NS;
}
```



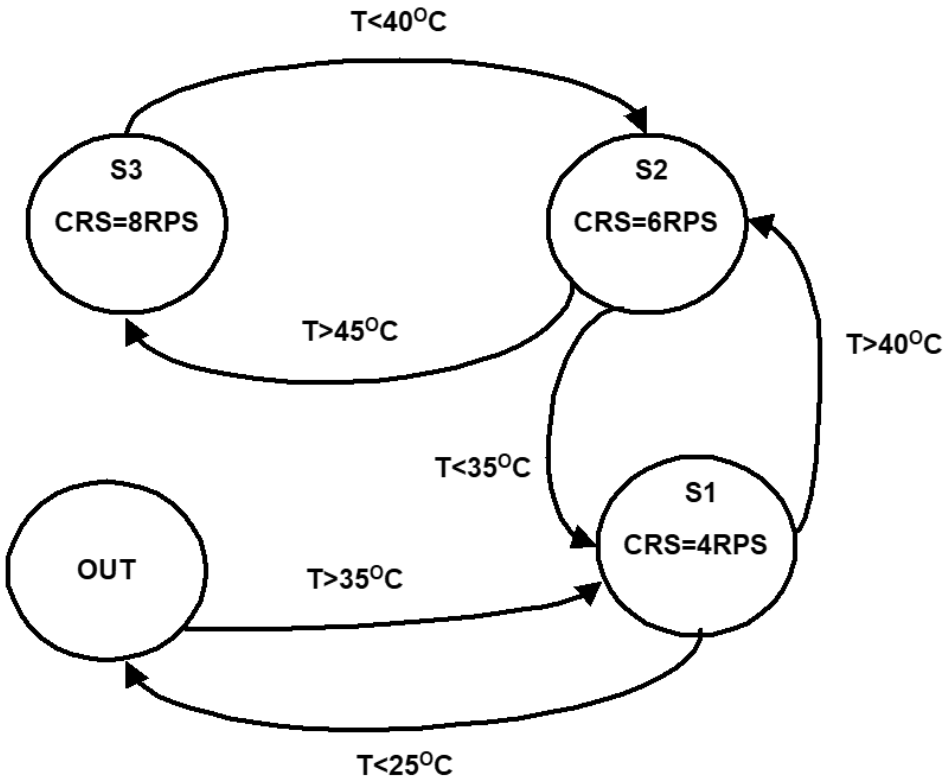
In-class assignment: Question

- Design an air conditioning system that keeps the environment temperature between 15 to 35 degree of centigrade
  - Consider a heater and a cooler modules
  - Assume cooler has three fan speed

In-class assignment: Answer



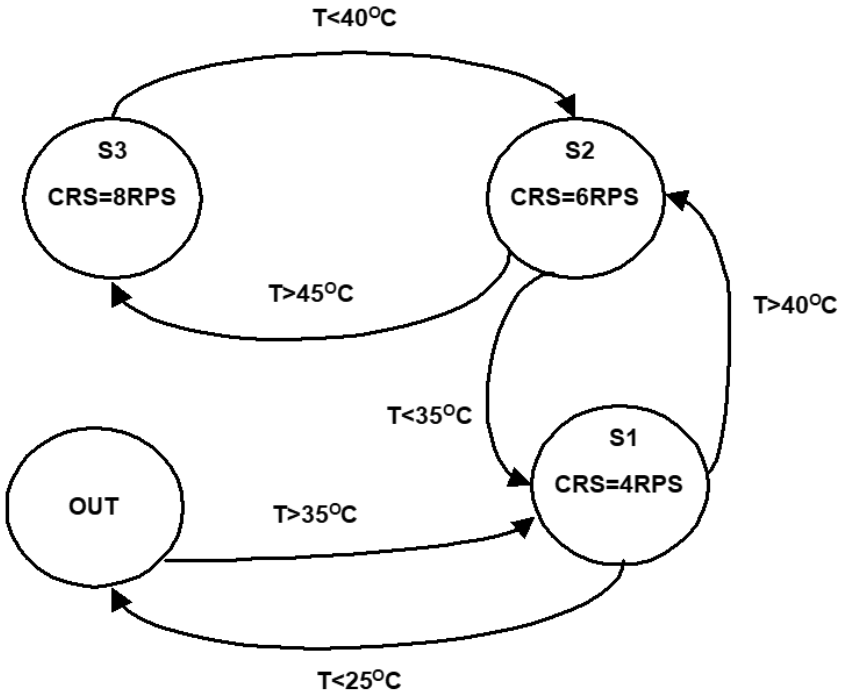
In-class assignment: Answer



CRS: Cooler Rotational Speed

In-class assignment: Answer

```
State PS=S1,NS;
State S2_PS,S2_NS;
Event e;

while(1){
  case(PS){
    S1: ...
    S2:
      Turn_off(Heater);
      Turn_on(Cooler);
      S2_PS=S1;
      while(S2_PS != OUT){
        case(S2_PS){
          S1:
            CRS(4);
            e=Wait_for_event();
            if(e=='T<25') S2_NS=OUT;
            else if(e=='T>40') S2_NS=S2; end if;
          S2:
            CRS(6);
            e=Wait_for_event();
            if(e=='T<35') S2_NS=S1;
            else if(e=='T>45') S2_NS=S3; end if;
          S3:
            CRS(8);
            e=Wait_for_event();
            if(e=='T<40') S2_NS=S2; end if;
        }
        S2_PS=S2_NS;
      }
      if(e=='T<25') NS=S1; end if;
    S3: ...
  }
  PS=NS;
}
```



CRS: Cooler Rotational Speed

In-class assignment: Question

- Write a program in C that reads a text from standard input stream, line by line, and prints the first word of each line

In-class assignment: Answer

- It is clear we need first to read and skip the leading spaces, if any, then read characters of the first word and print them until the word ends, and then read and skip all the remaining characters until the end-of-line character is encountered
- Upon reaching the end of line character (regardless of the stage), we restart the algorithm from the beginning, and upon encountering the end of file condition (regardless of the stage), we terminate the program

In-class Assignment: Answer

- Traditional (imperative) program in C

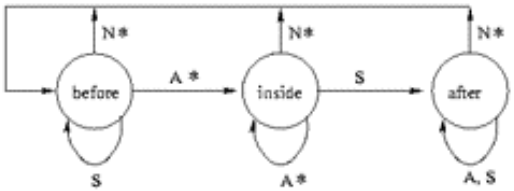
```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    int c;
    do {
        do {
            c = getchar();
        } while(c == ' ');
        while(c != EOF && !isspace(c) && c != '\n') {
            putchar(c);
            c = getchar();
        }
        putchar('\n');
        while(c != EOF && c != '\n')
            c = getchar();
    } while(c != EOF);
    return 0;
}
```

In-class assignment: Answer

- Automata-based style program
  - Thinking in terms of finite state machines
  - Note that line parsing has three stages:
    - Skipping the leading spaces (Before)
    - Printing the word (Inside)
    - Skipping the trailing characters (After)

In-class assignment: Answer

- Automata-based style program
  - N: The end of line character
  - S: Space
  - A: Any other characters




```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    enum states {
        before, inside, after
    } state;
    int c;
    state = before;
    while((c = getchar()) != EOF) {
        switch(state) {
            case before:
                if(c != ' ') {
                    putchar(c);
                    if(c != '\n')
                        state = inside;
                }
                break;
            case inside:
                if(!isspace(c))
                    putchar(c);
                else {
                    putchar('\n');
                    if(c == '\n')
                        state = before;
                    else
                        state = after;
                }
                break;
            case after:
                if(c == '\n')
                    state = before;
        }
    }
    return 0;
}
```

Introduction

- Statecharts is a language were introduced by David Harel in 1987
  - D. Harel, "Statecharts: A visual formalism for complex systems", Science of Computer Programming 8, 1987, pp. 231-274.
- Statecharts are useful for describing large, complex, reactive systems
  - A reactive system is one which must continuously react to external and internal events
- They are a graphic notation ("visual")

Definition based on FSM

- + Depth (also known as abstraction)
- + Orthogonality (also known as concurrency)
- + Broadcast communication

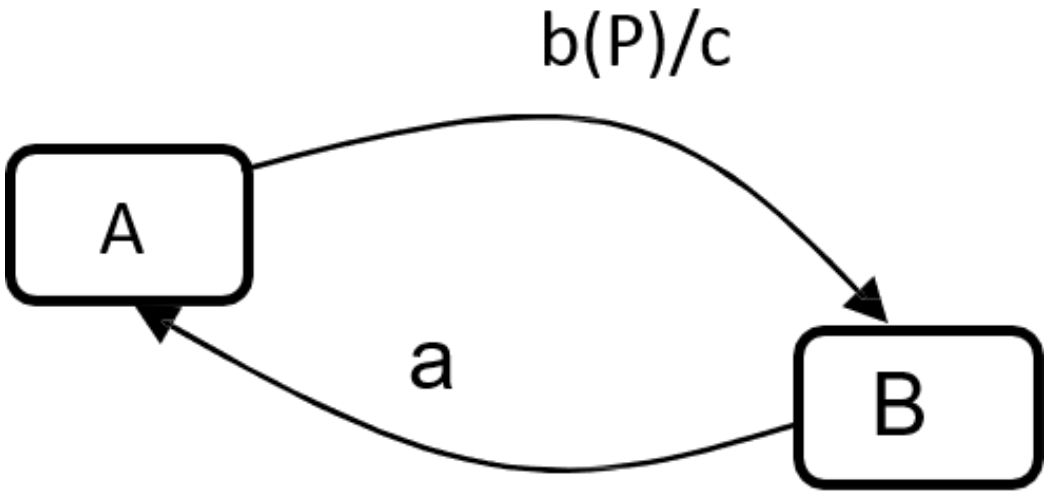
The Statechart notation is a kind of extended FSM with abstraction, concurrency, and communication

State diagrams

- Composed of states, transitions
- Transitions from one state to another happen when the event that is labeled on the arc (if any) occurs and the condition (if any) is true
- An output can be associated with the transition

Example

- State changes from A to B when event b occurs and the condition P is true; the output is c
  - c is global (can be seen everywhere in the Statechart model)
  - c can be used as an input on a transition
    - This supports communication in the model

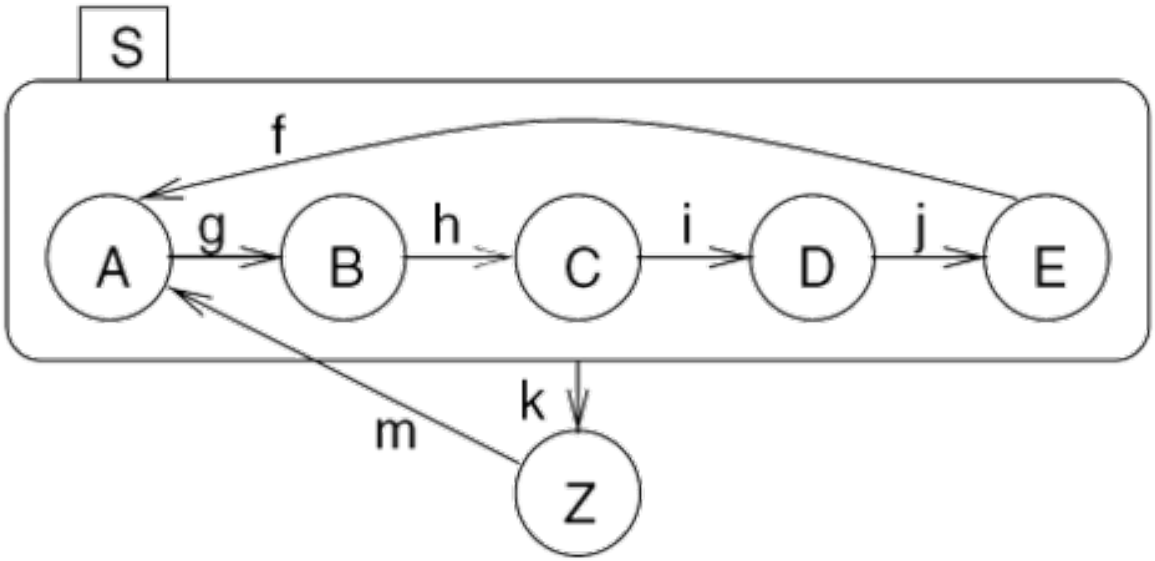


Depth (hierarchy)

- Statecharts extend this with:
  - Refinement, clustering
  - AND, OR decomposition of states
    - Actually XOR, not OR

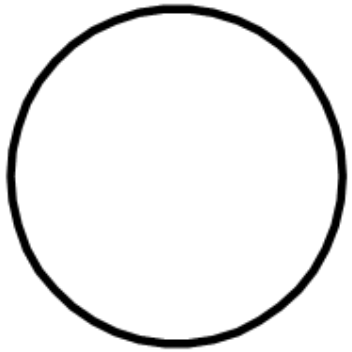
Modeling of hierarchy (depth)

- States comprising other states are called super-states
- States included in super-states are called sub-states of the super-states
- Each state which is not composed of other states is called a basic state



Super states vs. basic states

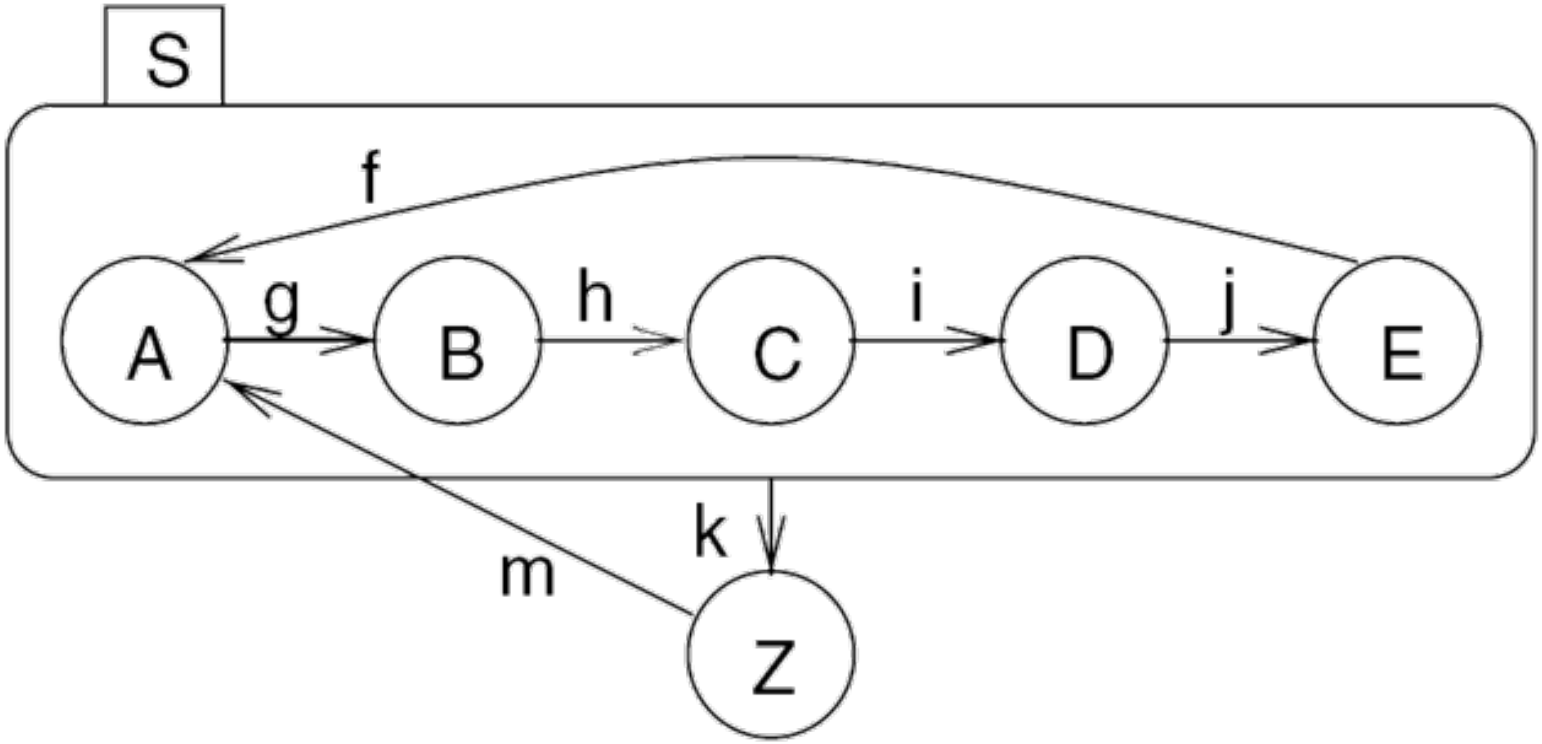
- Basic state



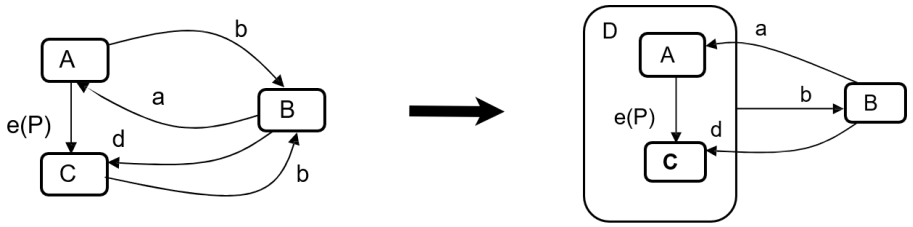
- Super state

OR-super-states

- The FSM can only be in one of the sub-states of super-state S at any time

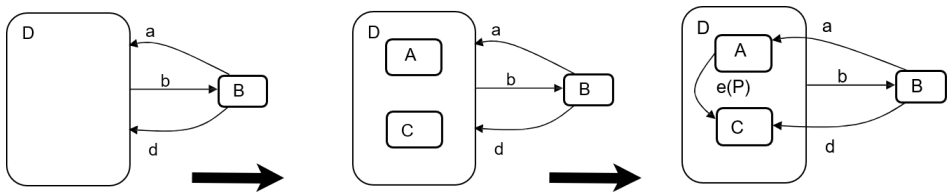


Example (bottom-up clustering)



- D is called a super state
- The semantics (aka meaning) of super state D is:
  - A xor C
- The arc labeled b is a common property to the super state D

Example (top-down refinement)



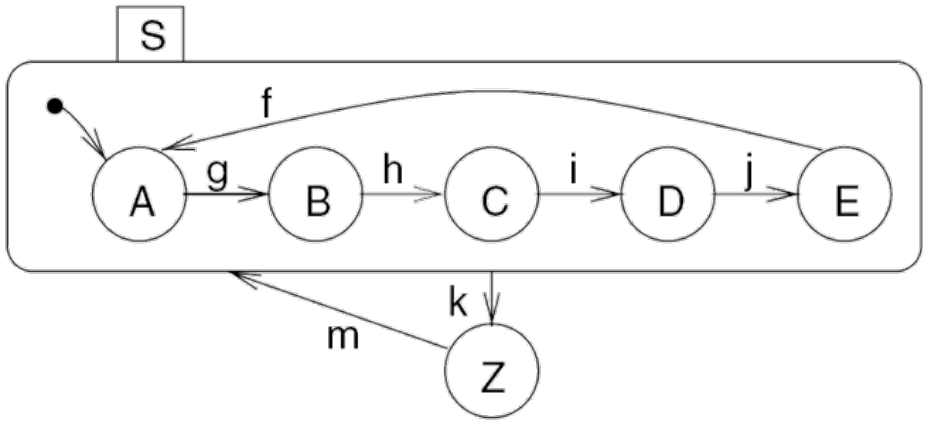
- The events a,d are underspecified (which one goes where?)
  - Needs to be fixed
- The transition from A to C also needs to be specified for the example

Design modularity (up to/from here: session 20/21)

- There are two mechanisms to hide the internal structure of super-states from the environment
  - Default state mechanism
  - History mechanism

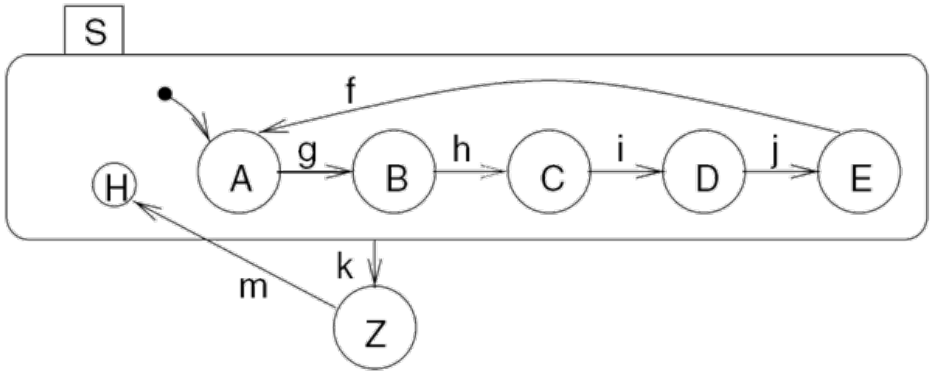
Default state mechanism

- Note that the filled circle does not constitute a state itself



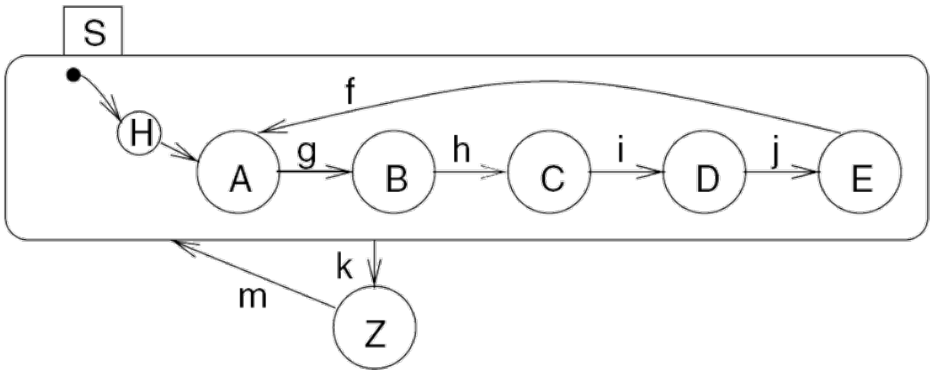
History mechanism

- It is possible to return to the last sub-state that was active before the super-state was left
- The filled circle defines the next state for the very initial transition into the super-state



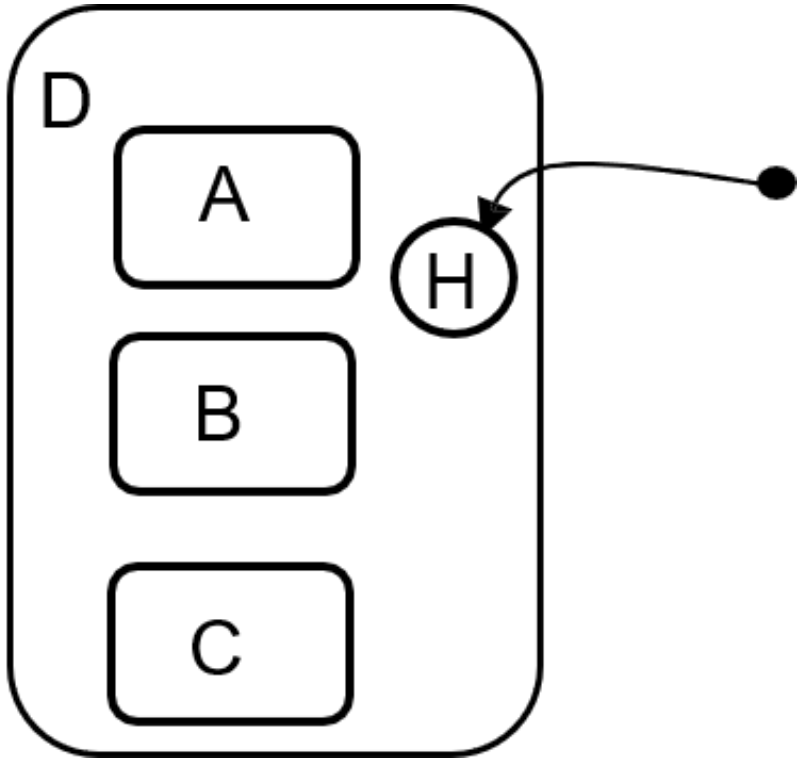
History mechanism

- Another notation (Equivalent to the previous one)



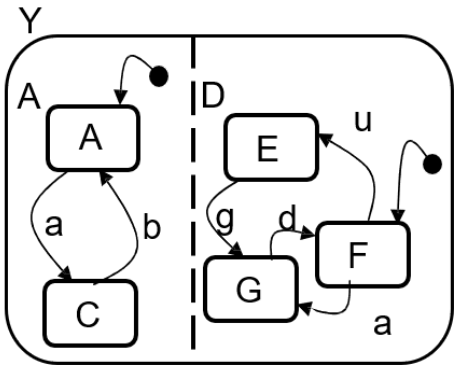
Example

- A timer that continues to count down as the state is entered and exited
- The timer does not get reset when the state is entered



Orthogonality (concurrency)

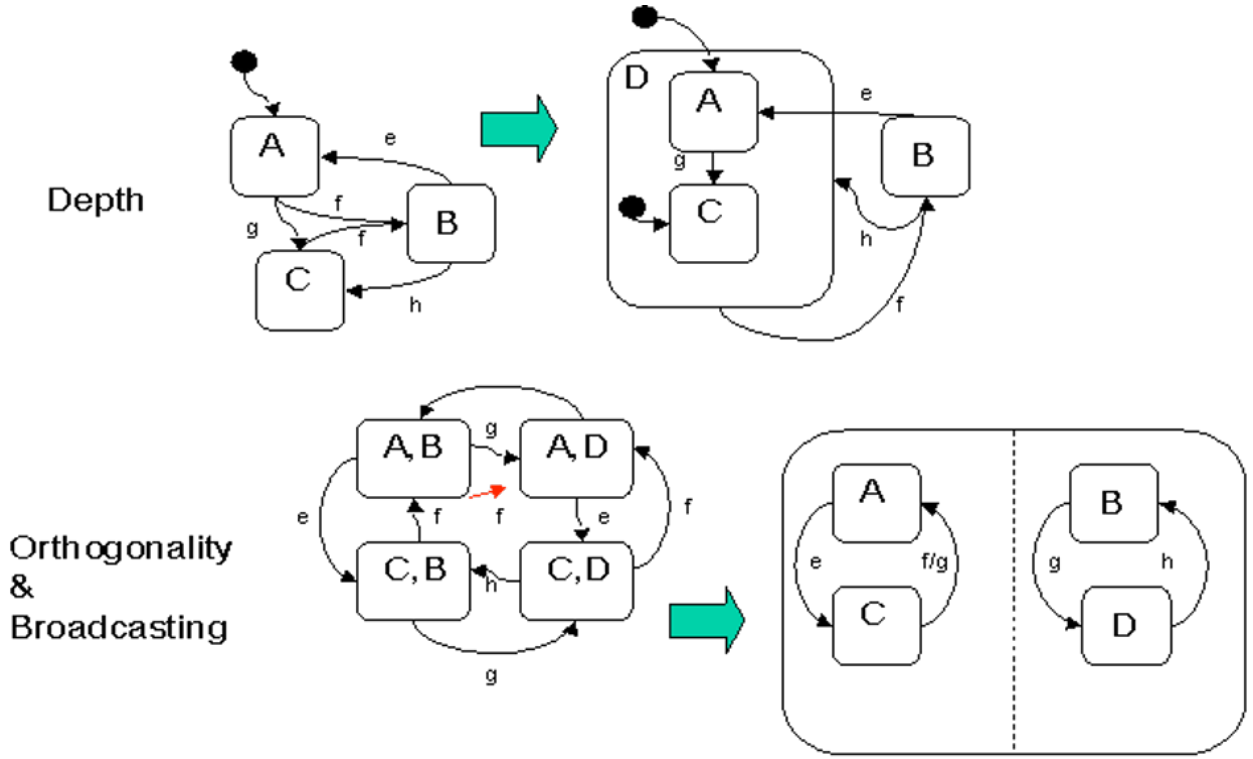
- AND
- Y is the orthogonal product of A and D



- From the default states:
  - If event ‘a’ occurs, then the diagram moves from states A,F into state C,G at the same time (A is synchronized with D)
  - If event ‘u’ occurs, then only D is affected and the diagram moves from A,F into state A, E (A is independent from D)

Broadcast communication

- An event is seen everywhere in the diagram at the same time



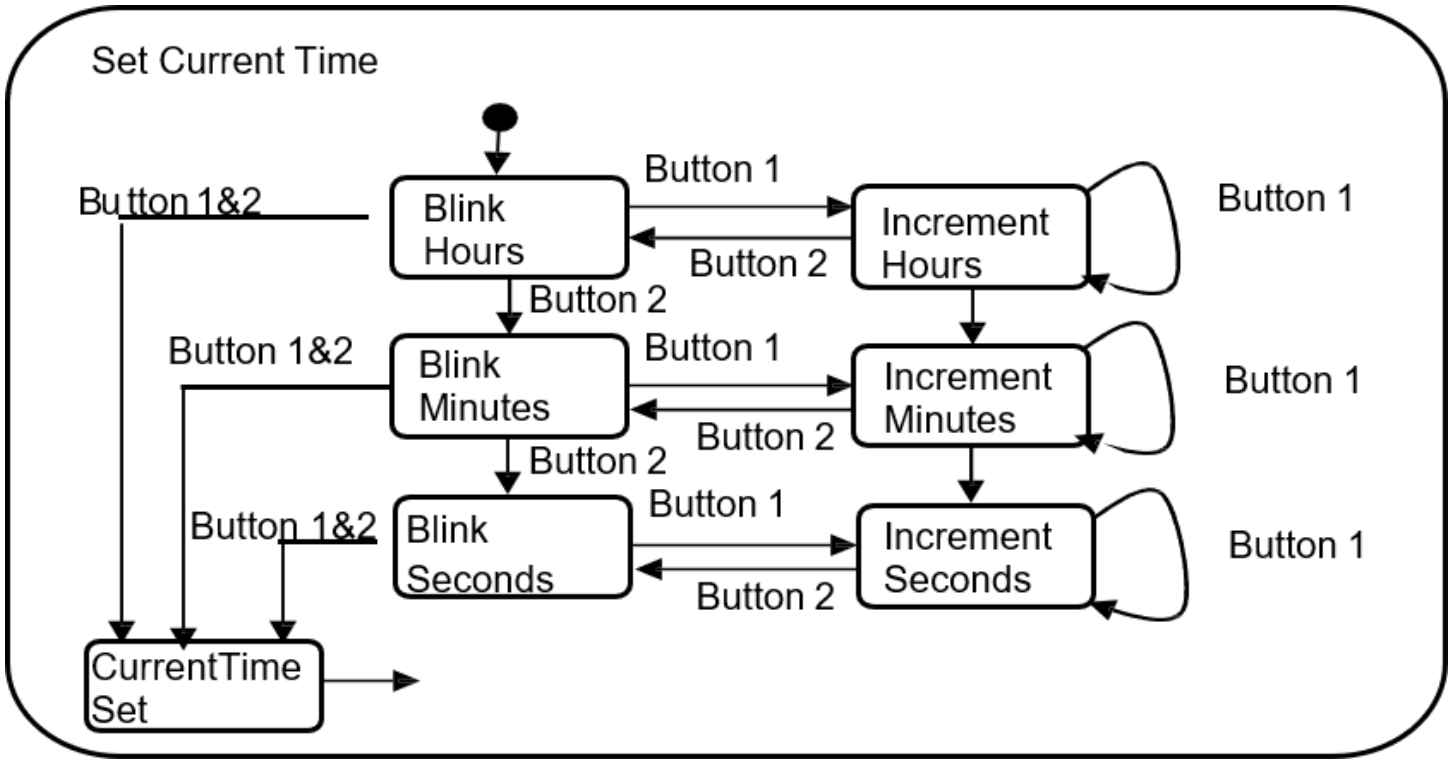
In-class assignment: Question

- Specify the behavior of an alarm clock using statecharts
  - Assumptions
    - The alarm clock has 2 buttons
    - Three facilities should be considered

      - Setting the current time
      - Setting the alarm time
      - Displaying the current time

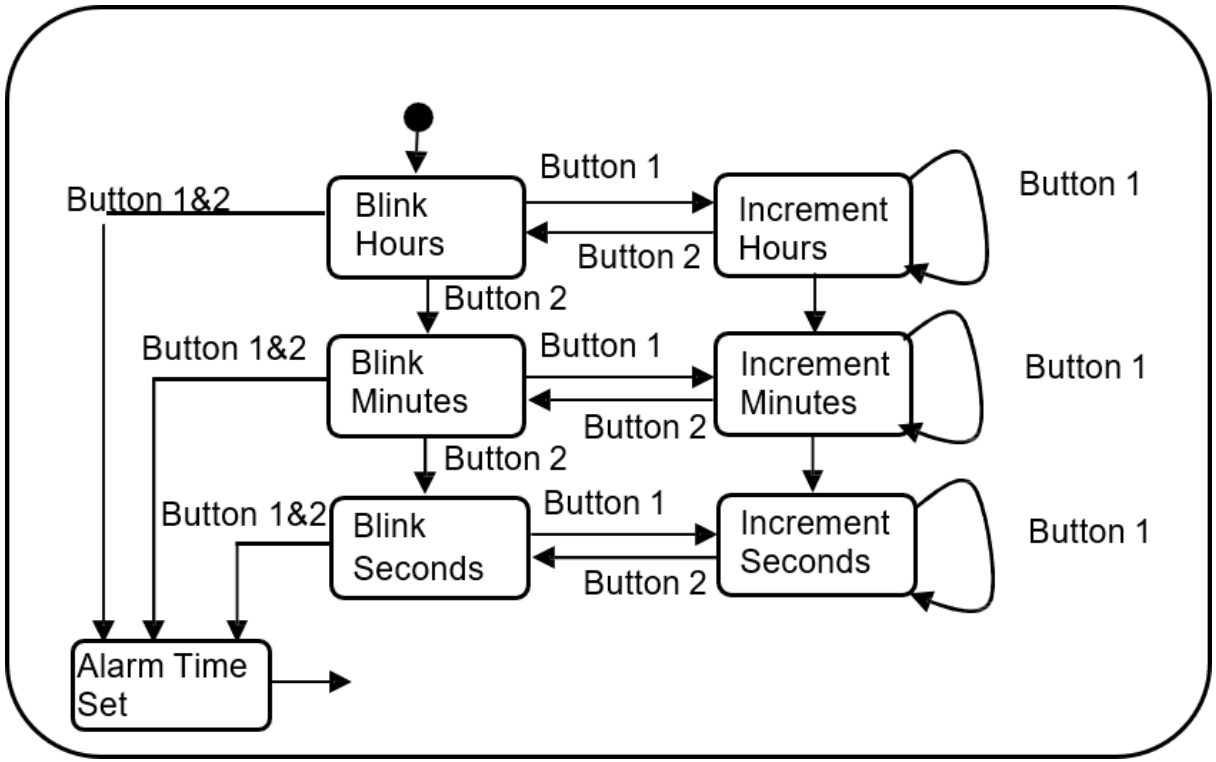
In-class assignment: Answer

- We can start with setting the current time
  - Need to set hours, minutes, seconds
  - Need to decide which buttons (one, both together) do what



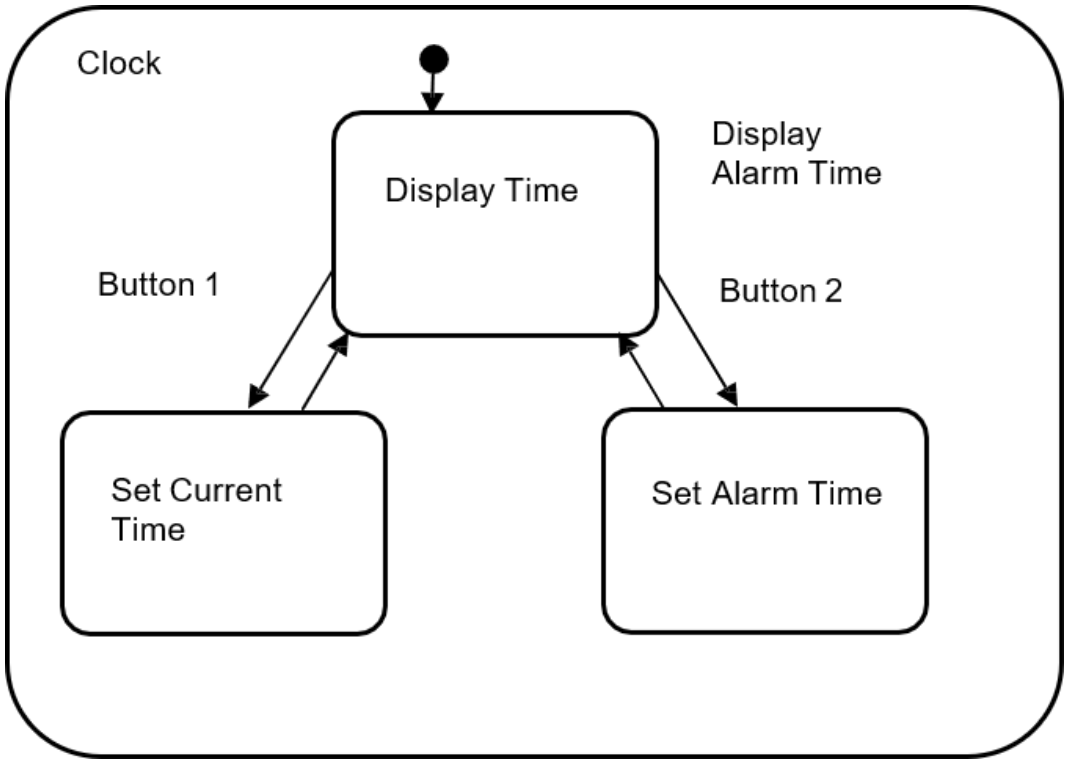
In-class assignment: Answer

- Now, consider setting the alarm time



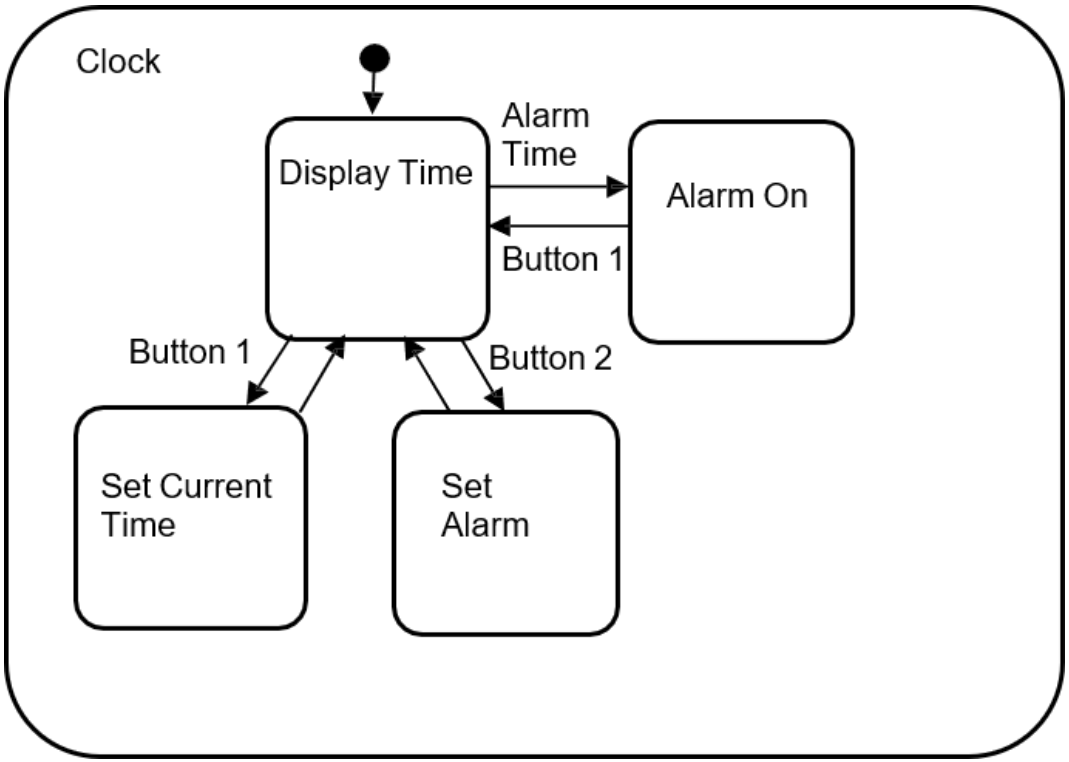
In-class assignment: Answer

- Now, consider how these super states relate to one another?
  - xor



In-class assignment: Answer

- Now, extend the statechart to describe the alarm going off



In-class assignment: Answer

- Is there a radio?
- Is there a snooze button?
- Is there a battery backup?
- ....

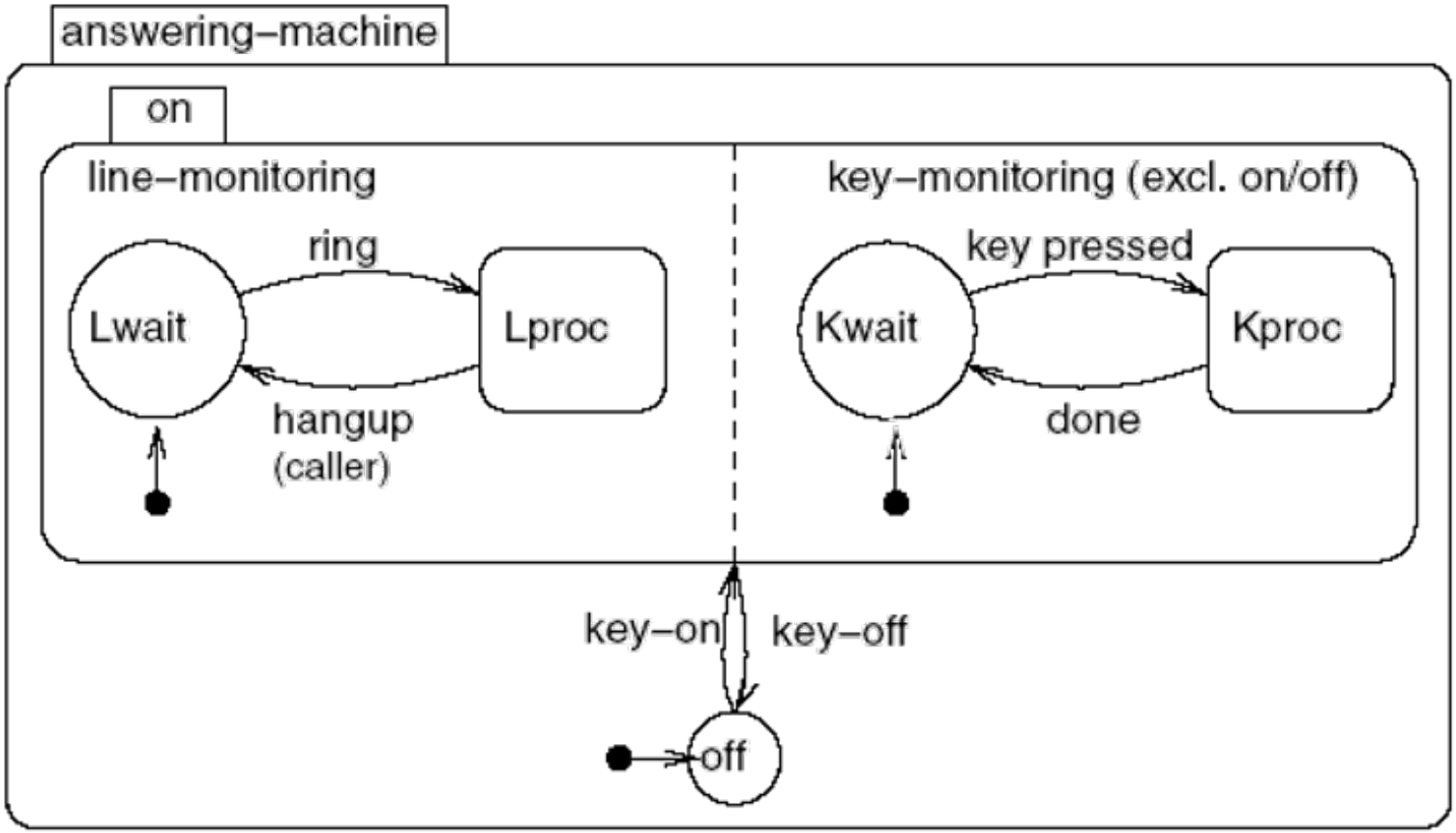
In-class assignment: Validation

- Now, validate the behavior of the statechart (i.e., does the statechart specify the system the way we want it to work)
- Question: What happens if the alarm goes off for 5 minutes? Does display maintain the time?
  - Maintaining the time needs to occur concurrently with:
    - The alarm going off
    - Setting the alarm time

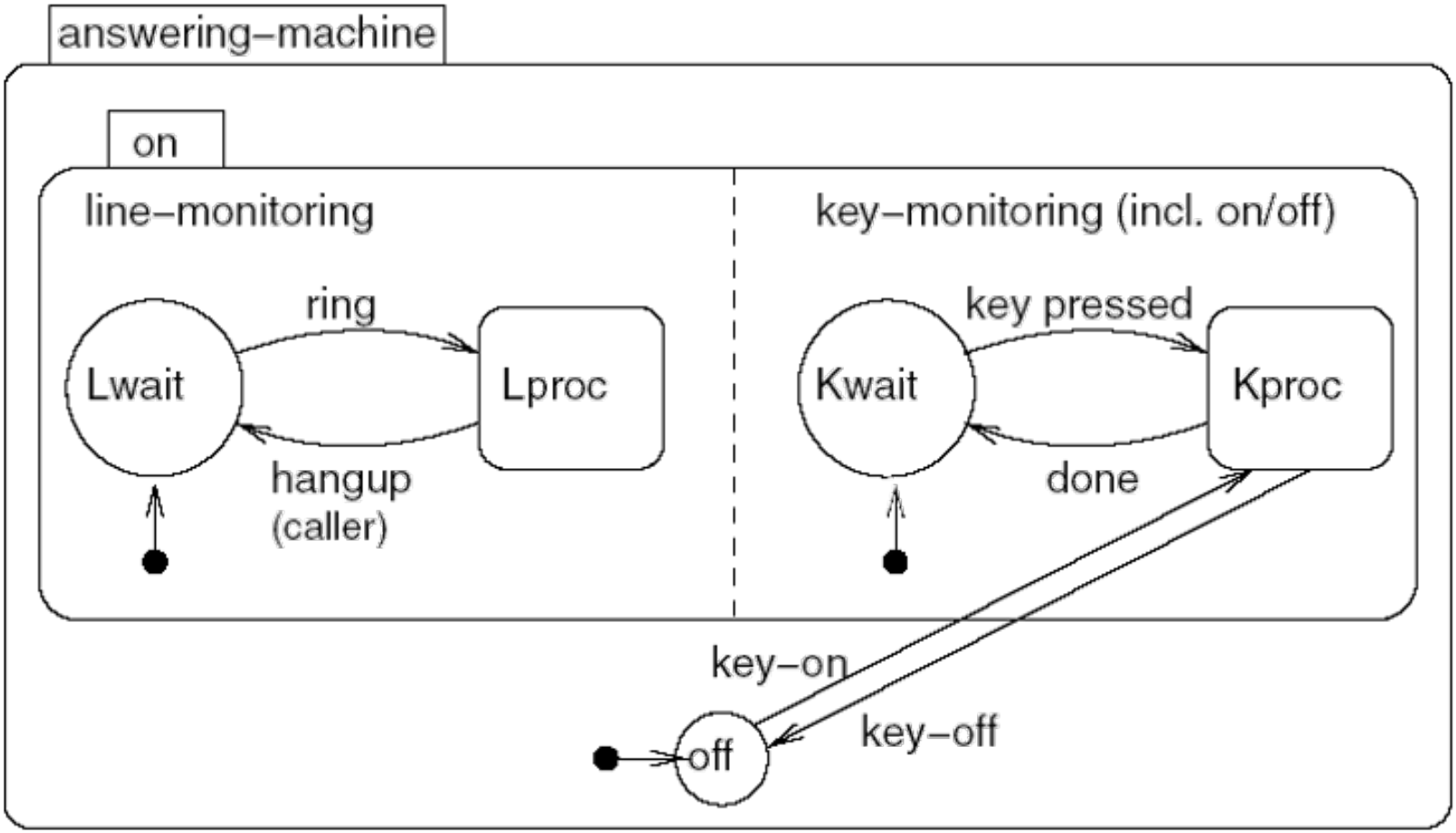
In-class assignment: Validation

- Question: Is it possible to display and maintain the current time concurrently? Next step is to fix the statechart
  - After it is fixed, need to re-validate the statechart

Example: Answering machine

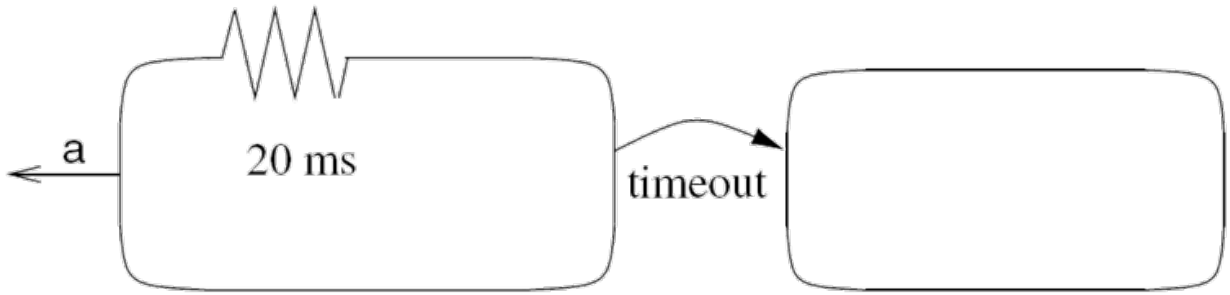


Example: Answering machine

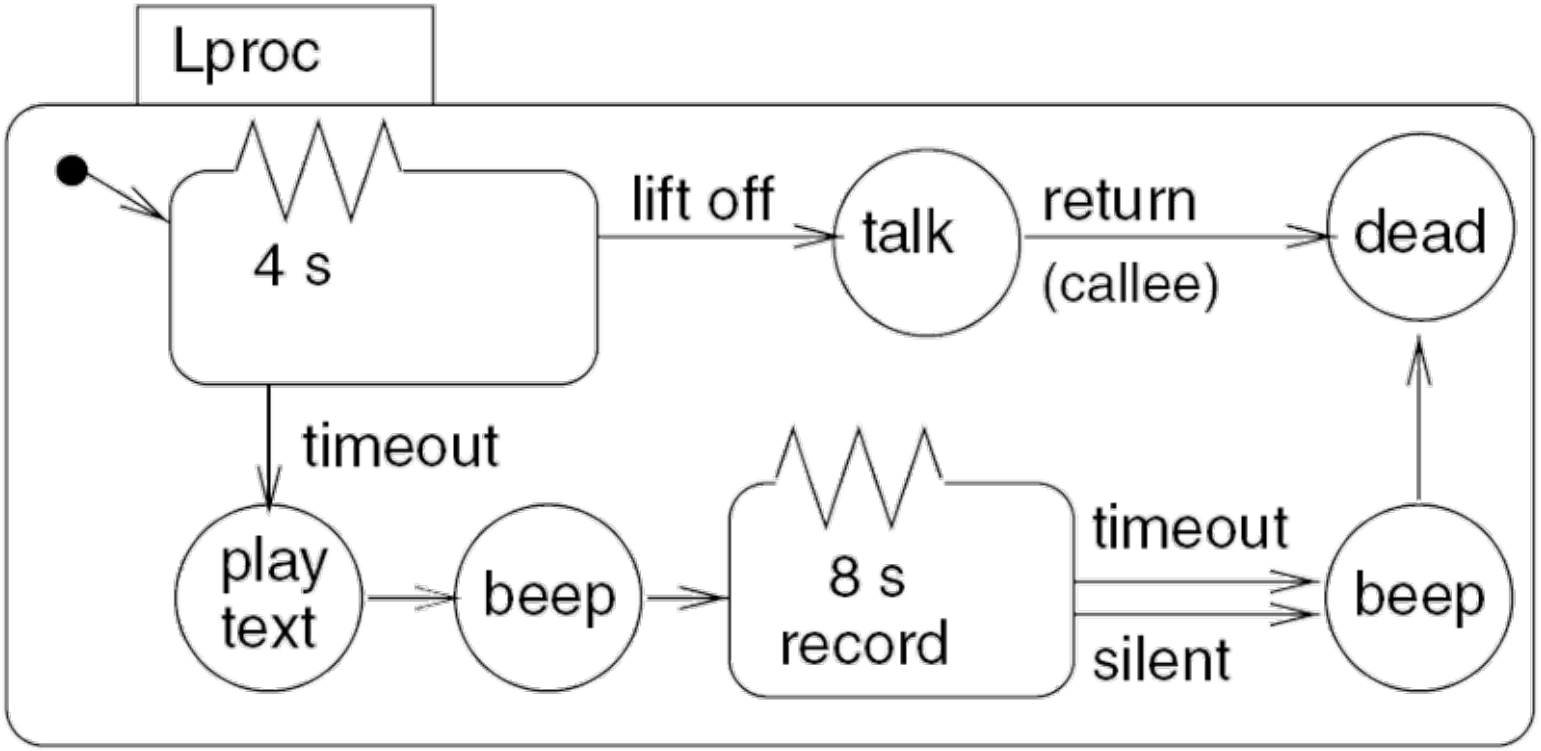


Timers

- Timing
- Real-time systems



Example


Lproc
4 s
lift off
talk
return
(callee)
dead
timeout
play
text
beep
8 s
record
timeout
silent
beep


The End