Arbeitskreis Hardware

Prof. Dr. Michael Rohs, Dipl.-Inform. Sven Kratz
michael.rohs@ifi.lmu.de
MHCI Lab, LMU München
Goals

• Learn how to…
• Build embedded interactive systems
• Build interactive devices and objects
• Build actuated installations
  – Example: actuated projector
• Build mobile robots
Robotized Objects

http://www.youtube.com/watch?v=sYutehhGknI
Actuated Pico Projector

- LuminAR, MIT Media Lab
  - Actuated desk lamp projector

- Pinhanez: large steerable projector (on per room)
  - Large projector →
    cf. mainframe computer

- Steerable pico projector (one per desk)
  - Personal projector →
    cf. personal computer

http://direct.media.mit.edu/people/natan/current/luminar.html

Linder, Maes: LuminAR: Portable robotic augmented reality interface design and prototype. UIST 2010, Demo.
Actuated Pico Projector

http://www.youtube.com/watch?v=XV5V-dQW8CI
Actuated Pico Projector

- Illuminate objects of interest
  - on-tabletop
  - walls, shelf, door, etc.

- Guide user’s attention
  - on-tabletop
  - walls, shelf, door, etc.

- Tangible interaction with the projector itself
  - Physical input: move lamp
  - Virtual input: interact with projection
  - Physical output: actuation
  - Virtual output: projection

- Issues
  - Safety, mechanical stability, energy

Linder, Maes: LuminAR.
Organization

• **Objective:** Learn about embedded interactive systems
  – Just for fun, **no ECTS credits**!

• **Date:** Mondays 18-20
  – presentation and discussion of new topic
  – work on topic / project

• **Schedule overview**
  – 15 sessions

• **Hardware components provided**
  – May buy AVR programmer (15 EUR) and power supply (7 EUR)
  – May need to buy materials (e.g. used in laser cutter)
## Schedule (preliminary)

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.10.</td>
<td>Introduction to embedded interaction, microcontrollers, hardware &amp; software tools</td>
</tr>
<tr>
<td>24.10.</td>
<td>Soldering ISP adapter, AVR architecture, electronics basics, USB to serial chips</td>
</tr>
<tr>
<td>31.10.</td>
<td>LED displays, LED multiplexing, transistors, electronics basics</td>
</tr>
<tr>
<td>7.11.</td>
<td>AVR analog-digital-converter, sensors, op-amps</td>
</tr>
<tr>
<td>14.11.</td>
<td>PCB design &amp; fabrication, EAGLE, 3D printing, OpenSCAD</td>
</tr>
<tr>
<td>21.11.</td>
<td>Actuation (servo / stepper motors), I2C: interfacing to other chips</td>
</tr>
<tr>
<td>28.11.</td>
<td>storage on memory cards, capacitive sensors</td>
</tr>
<tr>
<td>5.12.</td>
<td>OpenSCAD tutorial</td>
</tr>
<tr>
<td>12.12.</td>
<td>Eagle tutorial</td>
</tr>
<tr>
<td>19.12.</td>
<td>Displays (character LCDs, graphics LCDs), audio (speakers, amplification, op-amps)</td>
</tr>
<tr>
<td>9.1.</td>
<td>Communication: fixed-frequency RF, ZigBee, Bluetooth</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>6.2.</td>
<td>Project</td>
</tr>
</tbody>
</table>
Books


Simple Hardware...
Actuation...
Mobile Robots

- Human-robot interaction hot topic
- Robot tasks
  - manual tasks
  - cleaning
  - communicate
  - observe
Rotary Encoder
Technologies and Tools

Milling, drilling, cutting PCB: Roland Modela

PCB Design: EAGLE

Printing casings: RepRap 3D printer

ATtiny, Atmega microcontroller

ATtiny, Atmega microcontroller

www.rolanddg.com/product/3d/3d/MDX-20_15/MDX-20_15.html

en.wikipedia.org/wiki/RepRap

www.reprap.org/wiki/Mendel
Tools

**Workshop:** Drilling, cutting, etc.

**Laser cutter:** Cutting, engraving
Embedded Systems

• Computer systems with dedicated functionality
  – Cf. general-purpose computer (PC)
  – Microcontrollers, digital signal processors, sensors, actuators

• Often not perceived as a “computer”
  – Users may not know that a computer system is inside

• Examples
  – Wrist watches, mp3 players, digital cameras, GPS receivers, bike computers, heart rate monitors, cars (motor, ABS, ESP), traffic lights, microwave ovens, dishwashers, washing machines, door openers, weather stations, TV sets, remote controls, DVD players, factory automation systems, telephone switches, networked thermostats, implantable medical devices, toys
Technological Enablers

- **Processing & storage**
  - Cheap, fast, reliable, small, large capacity, energy efficient
  - Moore’s Law

- **Networking**
  - Cheap, fast, reliable, global, local, wireless, ad-hoc, low power

- **Displays**
  - Cheap, small, high quality, energy efficient, integrated

- **Sensors & actuators**
  - Cheap, small, accurate, invisible, many types
Ubiquitous Computing

- Computers embedded in everyday things
- Technology moves into the background
- Computers in the world, instead of world in the computer
- Mobile devices as always available mediators
- Entry point into the digital world

“The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”

Mark Weiser
Embedded & Tangible Interaction

• Challenges for human-computer interaction
  – How to interact with so many systems?
  – How to keep users from constant interruptions and distractions?
  – Device interaction happens in an everyday situation. How to take that into account?
  – What are novel forms of interaction?
  – Design opportunities?

• Interaction themes
  – Natural interfaces
  – Context-aware applications
  – Automatic capture and access
  – Continuous interaction
Microcontrollers

• Integrates processor, memory, I/O peripherals, and sensors on a single chip
  – Replaces many traditional hardware components in a single chip
  – Lower cost, fewer additional components, smaller circuit board
  – Very memory efficient (sleep modes)
  – Software flexibility through software

• Memory types
  – Flash: program
  – RAM: working memory (stack, heap)
  – EEPROM: non-volatile memory

• Interrupt-driven I/O
  – Sources: signal changes, timer overflow, ADC conversion done
  – Interrupts can wake microcontroller from low-power sleep state
Microcontrollers

Source: Gadre, Malhotra: tinyAVR projects
Microcontrollers

• I/O Pins
  – Used as input or output (controlled by software)
  – Serial communications (UART, I²C, SPI)
  – Signal generation (PWM, timers)
  – Analog input (ADC conversion)

• Development
  – In-circuit programming and debugging, field update of firmware
  – Programming in assembly language or C

• Selectable clock frequencies
  – Lower clock rate → less energy

• No floating point unit (typically)
Atmel AVR: ATtiny, ATmega

• 8-bit RISC chip, Harvard architecture

• ATtiny
  1–8 kB program memory
  6–32-pin package


• ATmega
  4–256 kB program memory
  28–100-pin package
  Extended instruction set
    • Multiply instructions
    • Handling larger program memories


• Large family of devices, specific features
Many types of AVRs: Choose depending on required features

<table>
<thead>
<tr>
<th>ATtiny13</th>
<th>ATtiny45</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 6 I/O pins, 1.8-5.5V operation</td>
<td>• 6 I/O pins, 1.8-5.5V operation</td>
</tr>
<tr>
<td>• 20 MPIS @ 20 MHz (clock rate selectable), internal oscillator</td>
<td>• 20 MPIS @ 20 MHz (clock rate selectable), internal oscillator</td>
</tr>
<tr>
<td>• 64B RAM, 64B EEPROM, 1kB Flash program memory</td>
<td>• 256B RAM, 256B EEPROM, 4kB Flash program memory</td>
</tr>
<tr>
<td>• 8-bit timer, 2 PWM channels, 10-bit ADC, analog comparator</td>
<td>• 2 8-bit timers, 4 PWM channels, 10-bit ADC, analog comparator, SPI, TWI, temperature sensor</td>
</tr>
<tr>
<td>• Price: €1.15</td>
<td>• Price: €2.05</td>
</tr>
</tbody>
</table>
Many types of AVRs: Choose depending on required features

**ATmega8**
- 23 I/O pins, 2.7-5.5V operation
- 16 MPIS @ 16 MHz (clock rate selectable), internal oscillator
- 1kB RAM, 512B EEPROM, 8kB Flash program memory
- 2 8-bit timers, 1 16-bit timer, 3 PWM channels, 10-bit ADC, analog cmp., SPI, TWI, USART

**ATmega328P**
- 23 I/O pins, 1.8-5.5V operation
- 20 MPIS @ 20 MHz (clock rate selectable), internal oscillator
- 2kB RAM, 1kB EEPROM, 4kB Flash program memory
- 2 8-bit timers, 1 16-bit timer, 6 PWM channels, 10-bit ADC, analog cmp., SPI, TWI, USART, temperature sensor

**Price:** €2.60

**ATmega328P**

**Price:** €3.30
Pinout ATtiny13

- Multiplexed pin functions, software configurable
  - Example: Flash/EEPROM programming via SPI:
    MOSI = master out, slave in (from programmer to ATtiny)
    MISO = master in, slave out (from ATtiny to programmer)
    SCK = serial clock
  - Example: ADC1 = ADC input channel 1
  - Example: PCINT3 = pin change interrupt 3

DIP = DIL = Dual In-line Package
SOIC = Small-Outline Integrated Circuit

Source: Atmel data sheet
Pinout ATmega8

- (RESET) PC6
- (RXD) PD0
- (TXD) PD1
- (INT0) PD2
- (INT1) PD3
- (XCK/T0) PD4
- VCC
- GND
- (XTAL1/TOSC1) PB6
- (XTAL2/TOSC2) PB7
- (T1) PD5
- (AIN0) PD6
- (AIN1) PD7
- (ICP1) PB0
- 28 PC5 (ADC5/SCL)
- 27 PC4 (ADC4/SDA)
- 26 PC3 (ADC3)
- 25 PC2 (ADC2)
- 24 PC1 (ADC1)
- 23 PC0 (ADC0)
- 22 GND
- 21 AREF
- 20 AVCC
- 19 PB5 (SCK)
- 18 PB4 (MISO)
- 17 PB3 (MOSI/OC2)
- 16 PB2 (SS/OC1B)
- 15 PB1 (OC1A)
AVR Memory Layout

Program Memory

internal Flash
word size = 16 bits

0x0000
end

Data Memory

32 registers
0x0000
64 I/O registers
0x0020
internal SRAM
word size = 8 bits
0x0060

Data Storage

internal EEPROM
word size = 8 bits
AVR I/O Ports

- I/O pin either input or output
  - Individually software-controlled

- Pin as output
  - States: low, high
  - Can drive 40mA (→ LED)

- Pin as input
  - Internal pull-up resistor (enabled/disabled in software)
  - High resistance state (high-Z) if pull-up disabled
Accessing the I/O Ports

• Three memory addresses for each I/O port
  – Data Direction Register: DDRx
    • 1 = output
    • 0 = input
  – Data Register: PORTx
    • if input: 1 = pull-up enabled, 0 = pull-up disabled
    • if output: 1 = PIN driven high, 0 = PIN driven low
  – Port Input Pins: PINx
    • read: PIN state (independent of DDRx)
    • write 1: toggles PORTx
AVR I/O Ports: Pin Control Example

<table>
<thead>
<tr>
<th>PIN</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>in/out</td>
<td>out</td>
<td>out</td>
<td>out</td>
<td>out</td>
<td>in</td>
<td>in</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>value</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>pullup</td>
<td>hi-z</td>
<td>hi-z</td>
<td>hi-z</td>
</tr>
</tbody>
</table>

Assembly

```assembly
ldi r16, (1<<PB4) | (1<<PB1) | (1<<PB0)
ldi r17, (1<<DDB3) | (1<<DDB2) |
     (1<<DDB1) | (1<<DDB0)
out PORTB,r16
out DDRB,r17
nop      // synchronization
in r16,PINB
```

C

```c
unsigned char i;
PORTB = (1<<PB4) | (1<<PB1) | (1<<PB0);
DDRB = (1<<DDB3) | (1<<DDB2) |
     (1<<DDB1) | (1<<DDB0);
__no_operation();  // synchronization
i = PINB;
```
“μC Hello World”: Blinking an LED

```c
#define F_CPU 1200000
#include <avr/io.h>
#include <util/delay.h>

int main()
{
    DDRB = 0b010000;
    while (1) {
        PORTB = 0b010000;
        _delay_ms(500);
        PORTB = 0b000000;
        _delay_ms(500);
    }
    return 0;
}
```
Downloading the Program to the µC

- Serial programming via Serial Peripheral Interface (SPI)
  - MISO, MOSI, SCK

![Diagram of ATTiny13 microcontroller and programmer connections]
Memory Programming

• Tasks
  – Download/upload program code to/from Flash memory
  – Download/upload data to/from internal EEPROM
  – Configuring the microcontroller (“fuse bits”)

• Programming options
  – Serial programming
    • In-system programming (ISP)
    • High-voltage serial programming (HVSP, only 8-pin controllers)
  – High-voltage parallel programming
    • If RESET pin used as I/O pin: high-voltage programming
  – debugWire on-chip debug system
    • Uses RESET pin for debugging and Flash/EEPROM programming
AVR Configuration via "Fuse Bits"

• AVRFuses tool
  – http://www.vonnieda.org/software/avrfuses

• Online fuse calculator
  – http://www.engbedded.com/fusecalc/

• ATtiny13 datasheet, 17.2 Fuse Bytes
  – ATtiny13 has two fuse bytes
  – Default: high byte = 0b11111111, low byte = 0b01101010

<table>
<thead>
<tr>
<th>Fuse Bit</th>
<th>Bit No</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>–</td>
<td>7</td>
<td>–</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>–</td>
<td>6</td>
<td>–</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>–</td>
<td>5</td>
<td>–</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>SELFPRGEN(1)</td>
<td>4</td>
<td>Self Programming Enable</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>DWEN(2)</td>
<td>3</td>
<td>debugWire Enable</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>BODELEVEL1(3)</td>
<td>2</td>
<td>Brown-out Detector trigger level</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>BODELEVEL0(3)</td>
<td>1</td>
<td>Brown-out Detector trigger level</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>RSTDISBL(4)</td>
<td>0</td>
<td>External Reset disable</td>
<td>1 (unprogrammed)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuse Bit</th>
<th>Bit No</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIEN(1)</td>
<td>7</td>
<td>Enable Serial Programming and Data Downloading</td>
<td>0 (programmed) (SPI prog. enabled)</td>
</tr>
<tr>
<td>EESAVE</td>
<td>6</td>
<td>Preserve EEPROM memory through Chip Erase</td>
<td>1 (unprogrammed) (memory not preserved)</td>
</tr>
<tr>
<td>WDTON(2)</td>
<td>5</td>
<td>Watchdog Timer always on</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>CKDIV8(3)</td>
<td>4</td>
<td>Divide clock by 8</td>
<td>0 (programmed)</td>
</tr>
<tr>
<td>SUT1(4)</td>
<td>3</td>
<td>Select start-up time</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>SUT0(4)</td>
<td>2</td>
<td>Select start-up time</td>
<td>0 (programmed)</td>
</tr>
<tr>
<td>CKSEL1(5)</td>
<td>1</td>
<td>Select Clock source</td>
<td>1 (unprogrammed)</td>
</tr>
<tr>
<td>CKSEL0(5)</td>
<td>0</td>
<td>Select Clock source</td>
<td>0 (programmed)</td>
</tr>
</tbody>
</table>
AVR Configuration via “Fuse Bits”

Caution: Wrong fuse bit settings may render chip unusable!

Tool: AVRFuses (www.vonnieda.org/AVRFuses/)
Configuring AVRFuses for the Programmer and USB Port

**mySmartUSB light:**
- Path to avrdude: `/ck-AVR/bin/avrdude`
- Programmer: `stk500v2`
- Port: `/dev/cu.SLAB_USBtoUART`
- Baud Rate: `[Default]`

**USBasp:**
- Path to avrdude: `/usr/local/CrossPack`
- Programmer: `usbasp`
- Port: `usb`
- Baud Rate: `[Default]`

For more information:
- [http://www.fischl.de/usbasp/](http://www.fischl.de/usbasp/)
USB Drivers for “mySmartUSB light”

- USB chip CP2102 from Silicon Laboratories
- Windows
  http://shop.myavr.ch/index.php?sp=article.sp.php&artID=200006
- Mac OS X, Linux
AVR ISP Connector

- Image of small PCB with one row of connectors

MISO 1  
SCK 3  
RST 5  
2 VCC  
4 MOSI  
6 GND  

not recommended:

better solution: solder small PCB with 6x1 pins
• Fuses show factory configuration of ATtiny13

• Brown-out detection
  – reset when Vcc below level

• Reset disabled
  – use reset pin as I/O pin: dangerous!

• Start-up time
  – delay until conditions are stable
AVR Clock Options

• Clock frequency can be chosen
  – Application requirements, power consumption
  – Clock prescaler register (divide clock by factor)
  – Component clocks can be disabled to reduce power consumption

• Clock source can be chosen
  – Internal resistor capacitor (RC) oscillator
    • Convenient, but not precise (temperature, operating voltage)
    • ATtiny13: 4.8MHz, 9.6MHz (at 3V and 25°C), 128kHz (low power)
  – External crystal oscillator
    • Highly precise, requires external quartz

• Clock source distributed to modules
  – $\text{CLK}_{\text{CPU}}$, $\text{CLK}_{\text{I/O}}$, $\text{CLK}_{\text{flash}}$, $\text{CLK}_{\text{ADC}}$
  – $\text{CLK}_{\text{ADC}}$ allows switching off other clocks during ADC conversion
AVR Development Toolchain & IDEs

- Free AVR toolchain
  - GNU C compiler: avr-gcc (gcc.gnu.org)
  - C library: avr-libc
  - Down-/Uploader: avrdude (www.nongnu.org/avr-libc/)

- CrossPack for Mac OS X
  - avr-gcc on Mac OS X, Xcode can be used (but not required)
  - oder: “sudo port install avr-gcc” (mit MacPorts)

- WinAVR for Windows
  - IDE for avr-gcc on Windows

- Atmel AVR Studio
  - http://www.atmel.com
AVR Development Toolchain & IDEs

- Eclipse and avr-eclipse
AVR-GCC Toolchain Overview

- User’s input files
- GCC
- GNU Binutils
- AVR Libc
- GDB / AVaRICE / Simulavr
- AVRDUDE

AVR Libc

- Free Software toolchain for Atmel AVR microcontrollers
  - avr-binutils
  - avr-gcc
  - avr-libc

- AVR Libc
  - C library for use with GCC on Atmel AVR microcontrollers

- AVR Libc Home Page
  - http://www.nongnu.org/avr-libc/
CrossPack: Creating a Project (Mac OS X)

```
bash$ avr-gcc-select 3
Current default compiler: gcc 3
bash$ avr-project BlinkLED
Using template: /usr/local/CrossPack-AVR-20100115/etc/templates/TemplateProject
bash$ cd BlinkLED/
bash$ ls -l
  total 0
  drwxr-xr-x  4 michaelrohs staff 136 Apr  2 22:44 BlinkLED.xcodeproj
  drwxr-xr-x  4 michaelrohs staff 136 Apr  2 22:44 firmware
bash$ cd firmware/
bash$ ls -l
  total 24
    -rw-r--r--  1 michaelrohs staff  4139 Apr  2 22:44 Makefile
    -rw-r--r--  1 michaelrohs staff   348 Apr  2 22:44 main.c
```
Generated Project in XCode

```c
/* Name: main.c
 * Author: <insert your name here>
 * Copyright: <insert your copyright message here>
 * License: <insert your license reference here>
 */

#include <avr/io.h>

int main(void)
{
    /* insert your hardware initialization here */
    for(;;){
        /* insert your main loop code here */
    }
    /* insert your hardware cleanup code here */
    return 0; /* never reached */
}
```
• Adapt Makefile as required
  – DEVICE, CLOCK, FUSES
  – PROGRAMMER
  – OBJECTS
“Fuses Calculator”

- http://www.engbedded.com/fusecalc/
  - configure fuses
  - click “Apply feature settings”
  - use “AVRDUDE arguments” (bottom of page)

- For example (for AVRtiny45):
  -U lfuse:w:0xe2:m -U hfuse:w:0xdf:m -U efuse:w:0xff:m

- May want to use datasheet to verify settings
Building within XCode
Flashing AVR from within XCode

- Duplicate existing “firmware” target
- Rename to “install”
- Change Info | Arguments to “flash”

→ store custom template in
  ~/.CrossPack-AVR/templates/TemplateProject
Run external build tool

avrdude -c USBasp -p attiny13 -U flash:w:main.hex:i
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################# | 100% 0.01s
avrdude: Device signature = 0x1e9007
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "main.hex"
avrdude: writing flash (132 bytes):

Writing | ################################# | 100% 1.13s
avrdude: 132 bytes of flash written
avrdude: verifying flash memory against main.hex:
avrdude: load data flash data from input file main.hex:
avrdude: input file main.hex contains 132 bytes
avrdude: reading on-chip flash data:
Reading | ################################# | 100% 0.71s
avrdude: verifying ...
avrdude: 132 bytes of flash verified
avrdude done. Thank you.

Build succeeded 3.4.2011 0:29
No issues
AVR-Eclipse
AVRDUDE

Configuration: Debug [ Active ]

Programmer configuration

Please select a Programmer Configuration to enable avrdude functions

JTAG ICE BitClock

Specify the bit clock period in microseconds for the JTAG interface or the ISP clock (JTAG ICE only).
Set this to > 1.0 for target MCUs running with less than 4MHz on a JTAG ICE.
Leave the field empty to use the preset bit clock period of the selected Programmer.

JTAG ICE bitclock __________ μs

BitBang Programmer Bit State Change Delay

Specify the delay in microseconds for each bit change on bitbang-type programmers.
Set this when the the host system is very fast, or the target runs off a slow clock
Leave the field empty to run the ISP connection at max speed.

Bit state change delay __________ μs

AVRDUDE command line preview

Copy Project Settings Restore Defaults Apply Cancel OK
### Properties for LCD12232

**Environment**

**Configuration:** Debug [ Active ]

**Environment variables to set**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
<th>Origin</th>
</tr>
</thead>
<tbody>
<tr>
<td>AVRDUDEACTIONOPTIONS</td>
<td>-Uflash:w:${BuildArtifactFileName}.hex:a</td>
<td>BUILD</td>
</tr>
<tr>
<td>AVRDUDEOPTIONS</td>
<td>-pm8</td>
<td>BUILD</td>
</tr>
<tr>
<td>AVRDUDEPATH</td>
<td>/usr/local/CrossPack-AVR/bin/</td>
<td>BUILD</td>
</tr>
<tr>
<td>AVRTARGETCPU</td>
<td>1000000</td>
<td>BUILD</td>
</tr>
<tr>
<td>AVRTARGETMCU</td>
<td>atmega8</td>
<td>BUILD</td>
</tr>
<tr>
<td>BUILDARTIFACT</td>
<td>LCD12232.elf</td>
<td>BUILD</td>
</tr>
<tr>
<td>CWD</td>
<td>/Users/michaelrohs/dev/android/EclipseWorkspace/LCD12232/Debug</td>
<td>BUILD</td>
</tr>
<tr>
<td>PATH</td>
<td>/usr/local/CrossPack-AVR/bin:/usr/bin:/bin:/usr/sbin:/sbin</td>
<td>BUILD</td>
</tr>
<tr>
<td>PWD</td>
<td>/Users/michaelrohs/dev/android/EclipseWorkspace/LCD12232/Debug</td>
<td>BUILD</td>
</tr>
</tbody>
</table>

- Append variables to native environment
- Replace native environment with specified one

**OK**
Properties for LCD12232

Paths and Symbols

Configuration: Debug [ Active ]

Languages
GNU C
s,S,asm

Include directories

Add directory path

Directory:
/usr/local/CrossPack-AVR/avr/include

Add to all configurations
Add to all languages
Is a workspace path

Variables...
Workspace...
File system...

OK Cancel

Restore Defaults Apply

OK Cancel
AVR Target Hardware Properties

Define the AVR target properties

MCU Type: ATmega8

MCU Frequency (Hz): 16000000
AVR-Eclipse
AVR-Eclipse

• Building and uploading the program
USBasp programmer
http://www.fischl.de/usbasp/
with selectable SCK rate and option to power circuit

breadboard with ATtiny13, LED and 1kOhm resistor
Breadboard

- Quick prototyping
  - Changing/adding components is easy
- Can get confusing soon ("spaghetti wires")
Using the Command Line

bash$ ls
Makefile  main.c
bash$ make
avr-gcc -Wall -Os -DF_CPU=9600000 -mmcu=attiny13 -c main.c -o main.o
avr-gcc -Wall -Os -DF_CPU=9600000 -mmcu=attiny13 -o main.elf main.o
rm -f main.hex
avr-objcopy -j .text -j .data -O ihex main.elf main.hex
bash$ make flash
avrdude -c USBasp -p attiny13 -U flash:w:main.hex:i

avrdude: AVR device initialized and ready to accept instructions
...
avrdude: writing flash (132 bytes):
Writing | #%%%%%%%%%%%%%%%%%%%%%%%%%%%%%| 100% 1.13s
...
avrdude: 132 bytes of flash verified
avrdude: safemode: Fuses OK
avrdude done. Thank you.

with mySmartUSB:
avrdude -p attiny13 -c stk500v2
  -P /dev/cu.SLAB_USBtoUART
  -U flash:w:main.hex:i
Assembly Language

- ATtiny have relatively simple instruction sets and are reasonably simple to program
  - ATtiny13: 120 instructions


- make

- V-USB
Development Process

Great idea → Research → Develop idea, create todo list → Get hardware components

- Software development
  - Testing
  - Testing, fabrication

- Hardware development
  - PCB design
  - Testing, fabrication

Source: Gadre, Malhotra: tinyAVR projects
Reading Data Sheets

• Extremely important to read carefully
  – Easy to find online

• Example: 7805 +5V voltage regulator
  – Operate according to “electrical characteristics” ↔ “max. ratings”

4 Electrical characteristics

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Test conditions</th>
<th>Min.</th>
<th>Typ.</th>
<th>Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_O$</td>
<td>Output voltage</td>
<td>$T_J = 25^\circ C$</td>
<td>4.8</td>
<td>5</td>
<td>5.2</td>
<td>V</td>
</tr>
<tr>
<td>$V_O$</td>
<td>Output voltage</td>
<td>$I_O = 5mA$ to $1A$, $P_O \leq$15W, $V_I = 8$ to $20V$</td>
<td>4.65</td>
<td>5</td>
<td>5.35</td>
<td>V</td>
</tr>
</tbody>
</table>

– “Application Circuits” show typical usage

\[ V_I = 10..35V \quad V_O = 5V \text{ (for 7805)} \]
### Features
- High Performance, Low Power AVR® 8-Bit Microcontroller
  - Advanced RISC Architecture
  - 120 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
- High Endurance Non-volatile Memory segments
  - 1K Bytes of In-System Self-programmable Flash program memory
  - 64 Bytes EEPROM
  - 64 Bytes Internal SRAM
  - Write/Erase cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C (see page 6)
  - Programming Lock for Self-Programming Flash & EEPROM Data Security
- Peripheral Features
  - One 8-bit Timer/Counter with Prescaler and Two PWM Channels
  - 4-channel, 10-bit ADC with Internal Voltage Reference
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - debugWIRE On-chip Debug System
  - In-System Programmable via SPI Port
  - External and Internal Interrupt Sources
  - Low Power Idle, ADC Noise Reduction, and Power-down Modes
  - Enhanced Power-on Reset Circuit
  - Programmable Brown-out Detection Circuit
  - Internal Calibrated Oscillator
- I/O and Packages
  - 8-pin PDIP/SOIC: Six Programmable I/O Lines
  - 20-pad MLF: Six Programmable I/O Lines
- Operating Voltage:
  - 1.8 - 5.5V for ATtiny13V
  - 2.7 - 5.5V for ATtiny13
- Speed Grade
  - ATtiny13V: 0 - 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V
  - ATtiny13: 0 - 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- Industrial Temperature Range
- Low Power Consumption
  - Active Mode:
    - 1 MHz: 1.8V: 240 µA
  - Power-down Mode:
    - < 0.1 µA at 1.8V

---

### Table of Contents
1. Pin Configurations
2. Overview
3. General Information
4. CPU Core
5. Memories
6. System Clock and Clock Options
7. Power Management and Sleep Modes
8. System Control and Reset
9. Interrupts
10. I/O Ports
11. 8-bit Timer/Counter0 with PWM
12. Timer/Counter Prescaler
13. Analog Comparator
14. Analog to Digital Converter
15. debugWIRE On-chip Debug System
16. Self-Programming the Flash
17. Memory Programming
18. Electrical Characteristics
19. Typical Characteristics
20. Register Summary
21. Instruction Set Summary
22. Ordering Information
23. Packaging Information
24. Errata
25. Datasheet Revision History
“Homework”

• Install AVR GCC
  – Use Eclipse or Xcode (as described above)

• Have a look into ATtiny13 datasheet
  – Especially first chapters on architecture
http://xkcd.com/730/
Creative Commons Attribution-NonCommercial 2.5 License.