Building Interactive Devices and Objects

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

• Servo Motors
• DC Motors
• Stepper Motors
• Motor Drivers
• PWM
• WLAN module
• Exercise 4
## Schedule

<table>
<thead>
<tr>
<th>#</th>
<th>Date</th>
<th>Topic</th>
<th>Group Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19.4.2012</td>
<td>Session 1: Introduction</td>
<td>Team building</td>
</tr>
<tr>
<td>2</td>
<td>26.4.2012</td>
<td>Session 2: Microcontrollers &amp; Electronics</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3.5.2012</td>
<td>Session 3: Sensors</td>
<td>Concept development</td>
</tr>
<tr>
<td>4</td>
<td>10.5.2012</td>
<td>CHI</td>
<td>Concept development</td>
</tr>
<tr>
<td>5</td>
<td>17.5.2012</td>
<td>Christi Himmelfahrt</td>
<td>Concept development</td>
</tr>
<tr>
<td>6</td>
<td>24.5.2012</td>
<td>Session 4: Actuators</td>
<td>Concept presentation, Hardware requ.</td>
</tr>
<tr>
<td>7</td>
<td>31.5.2012</td>
<td>Session 5: Physical Objects (Sven)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7.6.2012</td>
<td>Frohnleichnam</td>
<td>Project</td>
</tr>
<tr>
<td>9</td>
<td>14.6.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>10</td>
<td>21.6.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>11</td>
<td>28.6.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>12</td>
<td>5.7.2012</td>
<td></td>
<td>Project</td>
</tr>
<tr>
<td>13</td>
<td>12.7.2012</td>
<td></td>
<td>Evaluation</td>
</tr>
<tr>
<td>14</td>
<td>19.7.2012</td>
<td></td>
<td>Evaluation, Presentation</td>
</tr>
</tbody>
</table>
Sessions 4: Actuators, Concept Presentation, Hardware Req.

- Servo motors, DC motors, PWM

- Exercises
  1. Control servo motor, attach pointer to motor and move to predefined positions
  2. H bridge for DC motor

http://www.wimp.com/pointlessmachine/
Mobile Robots

• Human-robot interaction hot topic

• Robot tasks
  – manual tasks
  – cleaning
  – communicate
  – observe
Rotary Encoder
MIT LuminAR: Actuated Desk Projector

http://www.youtube.com/watch?v=XV5V-dQW8CI
Robotized Objects

http://www.youtube.com/watch?v=sYutehhGknI
SERVO MOTORS
Servomotors

- Precise angular position control
- Limited to ±90° rotation
- Can be modified to unlimited rotation and velocity control
- Used in RC models, robots, sensor positioning, etc.
Controlling Servo Motors

- Wiring: red, black, yellow cables
  - red = $V_{cc}$ (4.8-6V), black = GND, yellow = PWM signal

- PWM signal: 1.5ms is always neutral, min/max times and positions may vary
Operating Principle

• DC motor with a servo mechanism for precise control of angular position
• Motor + feedback device + control circuit

Motor speed depends on “error”
  – Fast if large difference between sensor and signal
  – Slow if small difference between sensor and signal
Controlling Servo Motors

• Motor can draw huge amounts of power
  – Use large Elko between red and black wires (≥100µF)
  – Separate power supplies / voltage regulators recommended

• High precision requirements for PWM signal
  – External quartz rather than internal RC oscillator (otherwise jitter)

• Simplest case: busy waiting (not recommended)
  // yellow wire of motor on PB3
  DDRB = 0b00001000; // port PB3 output, all others input
  PORTB = 0b11110111; // port PB3 low
  while (1) {
    PORTB |= 0b00001000; // port PB3 high
    _delay_us(1500); // 1.5 ms = neutral position, high precision required
    PORTB &= 0b11110111; // port PB3 low
    _delay_ms(18); // 1.5 + 18 = 20 ms, low precision between pulses
  }
External Clock: Quartz Crystal Oscillators

- More precise than internal oscillators
- Quartz 1..20 MHz
- Ceramic capacitors 12-22pF
  - Can be omitted
- Place quartz and capacitors close to AVR pins
- Change CLKSEL fuse bits
Unlimited Rotation and Velocity Control

• Useful for robot wheels
• Servo needs to be modified by cutting off link to potentiometer

• Steps: remove mechanical stop on gear, cut/file off potentiometer axis, glue potentiometer to neutral position
STEPPER MOTORS
Stepper Motors

- Rotates fixed number of degrees per step
  - Typically 15° or 30°
  - No need for feedback device
- Lower maximum speed than DC motor
- High torque at low speeds
- Used in printers, plotters, sensor positioning
- Requires control circuit
- Different wiring schemes
  - Unipolar, bipolar, etc.
Smaller Steps: Hybrid Stepper Motor

- Rotor contains two disks of magnets

![Diagram of a hybrid stepper motor with north (N) and south (S) poles labeled.](image source: Stündle, Public Domain)

![Image of a hybrid stepper motor.](image source: Coyote83, cc-by-sa)
Bipolar and Unipolar Motors

**Bipolar Motor**

- Bipolar motor
- 4 connections

**Unipolar Motor**

- Unipolar motor
- 5 or 6 connections

Image source: Ulfbastel, Public Domain
Driving Stepper Motors (basic)

Port 1
- INPUT 1
- OUTPUT 1
- GND

Port 2
- INPUT 2
- OUTPUT 2
- 9..12V V_S

Port 3
- INPUT 3
- OUTPUT 3
- GND

Port 4
- INPUT 4
- OUTPUT 4
- GND

Enable 1
- 0V or 5V

Enable 2
- 0V or 5V

Source: ST Datasheet

bipolar motor
Driving Stepper Motors (advanced)

Source: ST Datasheet
H-BRIDGE
H-Bridge for Controlling DC Motors

- Let motor run in forward or reverse direction
  - S1, S4: forward
  - S2, S3: reverse
  - S1, S3: brake
  - S2, S4: brake

Image source: Cyril Buttay, cc-by-sa
H-Bridge for Motor Control

• Let motor run in forward or reverse direction
  – S1, S4: forward
  – S2, S3: reverse
  – S1, S3: brake
  – S2, S4: brake

• Protection diodes for reverse voltage

Image source: Biezl, Public Domain
L293D Motor Driver

- 600mA per motor
- Control speed via PWM signal

Source: ST Datasheet

Image source: http://oomlout.com/L293/L293-Guide.pdf
L293D Motor Driver

- 600mA per motor
- Control speed via PWM signal

Source: ST Datasheet
PULSE WIDTH MODULATION (PWM)
AVR ATtiny45 Architecture

- Timer / Counter 0

Source: Datasheet
AVR Timers

• Tasks: Generating periodic events, PWM
  – Pulse-width modulation (PWM) on I/O pins
  – Timers can generate interrupts

• Synchronous clock source: device clock
  – divided by prescaler, if necessary

• Asynchronous clock source: external clock

• Modes
  – normal: count to $2^8-1 = 255$, generate interrupt, continue at 0
  – clear-timer-on-compare: count to value
  – fast PWM: single slope, count to 255 or value, set/clear pin on match
  – phase-correct PWM: dual slope, 50% speed of fast PWM

![PWM with 25% duty cycle](image)
Fast PWM (single-slope operation)

- power regulation, rectification, and DAC applications
Phase-Correct PWM (dual-slope operation)

- motor control applications
RGB LEDs

- Red, green, and blue in one package
- Different forward voltages ($I_f = 20mA$):
  - $U_{f,\text{red}} = 2.0V$
  - $U_{f,\text{green}} = 2.2V$
  - $U_{f,\text{blue}} = 3.8V$
- Example (right): 30° angle, 2x blue
- Control brightness via PWM
INTERRUPTS
AVR ATtiny45 Architecture

- Interrupt Unit
AVR Interrupts

- Interrupt normal execution, jump to interrupt service routine (ISR), resume normal execution
- Interrupt vectors at start of program memory space (Flash)
  - intr. vectors = jump instructions to interrupt services routines
  - Lower address = higher priority
- Memory layout

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>rjmp RESET ; Reset Handler</td>
<td>0x000A RESET: ldi r16, low(RAMEND); Start</td>
</tr>
<tr>
<td>0x0001</td>
<td>rjmp EXT_INT0 ; IRQ0 Handler</td>
<td>0x000B out SPL,r16 ; Stack Pointer to RAM end</td>
</tr>
<tr>
<td>0x0002</td>
<td>rjmp PCINT0 ; PCINT0 Handler</td>
<td>0x000C sei ; Enable interrupts</td>
</tr>
<tr>
<td>0x0003</td>
<td>rjmp TIM0_OVF ; Timer0 Overflow Handler</td>
<td>0x000D &lt;instr&gt; xxx</td>
</tr>
<tr>
<td>0x0004</td>
<td>rjmp EE_RDY ; EEPROM Ready Handler</td>
<td></td>
</tr>
<tr>
<td>0x0005</td>
<td>rjmp ANA_COMP ; Analog Comparator Handler</td>
<td></td>
</tr>
<tr>
<td>0x0006</td>
<td>rjmp TIM0_COMPA ; Timer0 CompareA Handler</td>
<td></td>
</tr>
<tr>
<td>0x0007</td>
<td>rjmp TIM0_COMPB ; Timer0 CompareB Handler</td>
<td></td>
</tr>
<tr>
<td>0x0008</td>
<td>rjmp WATCHDOG ; Watchdog Interrupt Handler</td>
<td></td>
</tr>
<tr>
<td>0x0009</td>
<td>rjmp ADC ; ADC Conversion Handler</td>
<td></td>
</tr>
</tbody>
</table>
AVR Interrupts

• Timers, ADC, analog comparator, etc. generate interrupts
  – No need for busy waiting
  – Multitasking

• Interrupts must be enabled
  – Global interrupt enabling, disabling: sei(), cli()
  – Various registers enable/disable interrupts
    (e.g. timer overflow, timer compare, ADC ready, etc.)
  – Flag register shows interrupt states

• External interrupts
  – INT0 pin or PCINT5..0 pins
  – Even if configured as outputs (software interrupt)
  – Pin change interrupts: trigger if PCINT5..0 pin toggles
  – Level interrupt: triggers as long as INT0 pin low
CONTROLLING SERVO MOTORS WITH TIMERS AND INTERRUPTS
Controlling Servo Motors

• Timer-generated PWM signal
  – Problem, long gaps (20-30ms) between signals (1-2ms)
  – For 8-bit timers (e.g. ATtiny45) this results in very low resolution:
    20ms = 256 counts ⇔ 1ms = 13 counts = -90°,
    2ms = 26 counts = +90° ⇔ resolution = 180°/14 counts = 13°

• Solution: 16-bit timers
  – For 16-bit timers (e.g. ATmega8) resolution is better:
    20ms = 65536 counts ⇔ 1ms = 3277 counts = -90°,
    2ms = 6554 counts = +90° ⇔ resolution = 180°/3278 counts = 0.05°

• Solution: Combine PWM with timer interrupts
  – Use shorter timer period to optimally use 1-2ms
  – Deactivate signal generation (but not timer) during gaps
  – Tradeoff between interrupt rate and angular resolution
Timer-generated PWM + Interrupts

GTCCR = (1 << TSM) | (1 << PSR0); // halt timer, reset prescaler
DDRB |= 0b00000001; // port PB0 (OC0A) output
PORTB & 0b11111110; // port PB0 (OC0A) low
TCCR0A = (2 << COM0A0) | (0 << COM0B0) | (3 << WGM00); // Clear OC0A on Compare Match, set OC0A at BOTTOM (non-inverting mode); Fast PWM, TOP = 0xFF
TCCR0B = (0 << WGM02) | (4 << CS00); // prescaler: clkIO/256
TCNT0 = 0; // reset counter
OCR0A = 94; // should be 93.75 for 1.5ms
TIMSK = (1 << OCIE0A); // Timer0 Output Compare Match A Interrupt Enable
sei(); // enable interrupts
GTCCR = (0 << TSM) | (0 << PSR0); // start timer

while (1) { …}
Timer-generated PWM + Interrupts

• Code

\[ \text{TCCR0A} = (2 \ll \text{COM0A0}) | (0 \ll \text{COM0B0}) | (3 \ll \text{WGM00}); \]

• Explanation

  – TCCR0A = Timer/Counter 0, Control Register A

  – COM0A1..0 = 2: Clear OC0A pin on compare match, set OC0A pin at counter = 0 (non-inverting mode)

  – WGM01..0 = 3: Fast PWM, TOP = 0xFF
Timer-generated PWM + Interrupts

• Code

\[ \text{TCCR0B} = (0 \ll \text{WGM02}) | (4 \ll \text{CS00}); \]

• Explanation

– \text{TCCR0B} = \text{Timer/Counter 0, Control Register B}

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOC0A</td>
<td>FOC0B</td>
<td>-</td>
<td>-</td>
<td>WGM02</td>
<td>CS02</td>
<td>CS01</td>
<td>CS00</td>
</tr>
</tbody>
</table>

– \text{CS02..0} = 4: divide clock by 256, at 16 MHz clock
– 16 MHz / 256 = 62.5 kHz per counter step
  = 16 \mu\text{s} per counter step
– one cycle (256 counter steps) = 4.096 ms = 244.14 Hz

Source: AVR Datasheet
Timer-generated PWM + Interrupts

GTCCR = (1 << TSM) | (1 << PSR0); // halt timer, reset prescaler

DDRB |= 0b00000001; // port PB0 (OC0A) output

PORTB &= 0b11111110; // port PB0 (OC0A) low

TCCR0A = (2 << COM0A0) | (0 << COM0B0) | (3 << WGM00); // Clear OC0A on Compare Match, set OC0A at BOTTOM (non-inverting mode); Fast PWM, TOP = 0xFF

TCCR0B = (0 << WGM02) | (4 << CS00); // prescaler: clkIO/256

TCNT0 = 0; // reset counter

OCR0A = 94; // should be 93.75 for 1.5ms

TIMSK = (1 << OCIE0A); // Timer0 Output Compare Match A Interrupt Enable

sei(); // enable interrupts

GTCCR = (0 << TSM) | (0 << PSR0); // start timer

while (1) { …}
#include <avr/interrupt.h>

int interruptCount = 0;

ISR(TIMER0_COMPA_vect) // interrupts occur at a frequency of 244.14Hz
{
    interruptCount++;  
    if (interruptCount == 1) { // switch off OC0A output
        // Normal port operation, OC0A/OC0B disconnected; Fast PWM
        TCCR0A = (0 << COM0A0) | (0 << COM0B0) | (3 << WGM00);
    } else if (interruptCount >= 5) { // produce OC0A output
        // Clear OC0A on Compare Match, set OC0A at BOTTOM; Fast PWM
        TCCR0A = (2 << COM0A0) | (0 << COM0B0) | (3 << WGM00);
        interruptCount = 0;
    }
    // set OCR0A: 63 = -90°, …, 94 = 0°, …, 125 = +90° (2.9° resolution)
}
WLAN MODULE
Roving RN-XV WLAN Modules

- Simple communication via WLAN
  - Roving RN-171 WiFi chip
  - UDP, TCP, HTTP, FTP
  - rovingnetworks.com/products/RN_XV

- Serial I/O to WLAN module

- Requires 3.3V power supply

- Connections
  - Pin 1: 3.3V power supply (use 3.3V voltage regulator)
  - Pin 2: TX (connect to RX of ATmega8, direct)
  - Pin 3: RX (connect to TX of ATmega8, via voltage divider!)
  - Pin GND: connect to common ground
Roving RN-XV WLAN Modules

• Pin 3 of RN-XV (RX): connect to TX of ATmega8
  – via voltage divider!
  – TX of ATmega8 uses +5V
  – RX of RN-XV expects +3.3V

• Example: $R_1 = 3000 \, \Omega$, $R_2 = 1500 \, \Omega$
  – $U_1 = 5V \times \frac{R_1}{R_1 + R_2} = 3.33V$
Roving RN-XV Commands

• Start terminal: screen /dev/tty.usbserial-A100XZ 9600
• Enter command mode: $$$
• Initialize
  factory RESET
  reboot
  set wlan ssid MYSSID
  set wlan pass 12345678
  set wlan join 1
  save
  reboot

• Get information
  ver, get ip, get adhoc, get com, get dns, etc.

Details:
rovingnetworks.com/products/RN_XV