Institute for Cognitive Systems
Technische Universität München
Prof. Gordon Cheng

Humanoid Sensors and Actuators - Tutorial 1

Microcontroller: Introduction (111 points)

In this tutorial we learn:

- How to program the AVR ATmega32 MCU
- How to use the general IO pins
- How to use the UART
- How to write assembler programs

Setup

If you want to use your own PC for this tutorial please follow the instructions of:
`http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Setup/setup.txt`

1. Microcontroller Circuit (5 points)

- Open the schematic
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/AVR-MCU-basic-circuit.pdf`
- Check if the circuit on your breadboard is correct.
- **ATTENTION**
  - Keep your breadboard ordered and avoid short circuits!
  - When you modify something on the breadboard: **DISCONNECT** the USB cable first!
  - The circuit is powered by the USB and a short circuit might damage the programmer board or the USB controller in the PC!
  - When you are sure that everything is fine connect the USB cable.
  - If the power LED doesn't light up after connecting the USB cable, then disconnect as fast as possible. You most probably have a short circuit.

Report (5 points)

R1.1: What are decoupling capacitors? **(1 points)**
R1.2: What properties of capacitors are important for good decoupling capacitors? Name at least two and elaborate. **(2 points)**
R1.3: Where would you place decoupling capacitors in a PCB layout and why? **(2 points)**

2. Programming

- Download the blinker program
  - Open `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/`
  - Download the file '`blink.hex`'
- Run the following command to flash the blinker program on the MCU

```
sudo avrdude -c avrispmkII -P usb B10 -p atmega32 -Uflash:w:blink.hex
```

3. Using the GPIO peripheral block (56 points)

a) Setting pin levels in assembler (16 points)

- Download the template project for assembler programs

  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/Atmega32TemplateAsm.tar.gz`
- Navigate into the project folder '`Atmega32TemplateAsm`' and create a '`build`' folder
- Open the project in qtcreator with
  1) Go to: `File – Open File or Project...`
  2) Navigate to template project folder and open the file `CMakeLists.txt`
  3) Configure the Project correctly and select the previously created build folder as default. Press the button `Browse` and navigate to the '`build`' folder
  4) Then press the button `Configure Project`

T3.1: Write a small assembler program which lets the LED on PORTC pin PC0 blink. The LED should stay one second on and one second off. Please use the given code fragment to generate a delay of one second. **(4 points)**

T3.2: Your program of **T3.1** avoids changing the bits of other pins. **(2 points)**

```
// wait for one sec
    ldi     r18, 0x3F
    ldi     r24, 0x0D
    ldi     r25, 0x03
1:  subi    r18, 0x01
    sbci    r24, 0x00
    sbci    r25, 0x00
    brne    1b                  ; local label backward
    rjmp    1f                  ; local label forward
1:  nop
```

- You can build the program directly in the qtcreator
- To flash the program onto the MCU you need to go into the build folder and type

`make prog_flash_<app-name>`

- Note: the command auto-completes

T3.3: Write a small assembler program which lets 3 LEDs turn on/off consecutively with 1 sec delay:

1) LED1 on, LED2 off, LED3 off
2) LED1 off, LED2 on, LED3 off
3) LED1 off, LED2 off, LED3 on. **(4 points)**

T3.4: Your program of **T3.3** avoids changing the bits of other pins. **(2 points)**
T3.5: Find the most efficient set of assembler instructions such that each step of **T3.3** can be completed in only 4 CPU cycles. **(4 points)**

b) Setting pin levels in C (6 points)

- Download the template project for C programs
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/Atmega32Template.tar.gz`
- Use the library delay function `_delay_ms(1000);` to create the one second delays. For this you need to include the header '`#include <util/delay.h>`'

T3.6: Implement **T3.1** using C code. **(2 points)**
T3.7: Your program of **T3.6** avoids changing the bits of other pins. **(1 point)**
T3.8: Implement **T3.3** using C code. **(2 points)**
T3.9: Your program of **T3.8** avoids changing the bits of other pins. **(1 point)**

c) Reading pin levels (8 points)

T3.10: Write an assembler program that mirrors the input on pin PC4 to pin PC3. Delay the input by one second using the delay code block from above. Use a 1k resistor to pull the pin PC4 high/low. **(4 points)**
T3.11: Your program of **T3.10** avoids changing the bits of other pins. **(1 point)**
T3.12: Implement **T3.10** using C code. **(2 points)**
T3.13: Your program of **T3.12** avoids changing the bits of other pins. **(1 point)**

d) Report (26 points)

R3.1: How can you ensure that the logic levels of 4 pins are changed at exactly the same time instance? Provide an example in assembly code where you set two pins high and two pins low at the same time. **(4 points)**

R3.2: Can you change the logic level of 2 pins of 2 different IO Ports at the same time? Justify your answer. **(2 points)**

R3.3: Explain each line of assembly code of the 1 second busy wait code fragment. Describe what each assembly instruction does and how many CPU cycles it takes. **(6 points: 1 point for lines 1-3, 1 point for line 4, 1 point for line 5-6, 1 point for line 7, 1 point for line 8, 1 point for line 9)**

R3.4: Explain and calculate why the code fragment takes exactly 1 second when the CPU is running at 1 MHz. **(4 points)**

R3.5: What are the main disadvantages of busy waiting? **(1 point)**
R3.6: What limits the accuracy of busy waiting? **(1 point)**

R3.7: Why does the MCU implement two address spaces for data and registers and two sets of assembly instructions (**IN**/**OUT** and **LD**/**ST**) to access these address spaces? **(1 point)**
R3.8: Provide an example where you once use **IN**/**OUT** and then **LD**/**ST** to access a peripheral register. **(1 point)**
R3.9: How many CPU cycles are needed to load the address and access the register for each case in **R3.8**? **(2 points)**

R3.10: Given a peripheral register address (e.g. 0x15), how can you create a variable in C that allows you to read/write this register only using its address value without using the device support headers? (Hint: you will need to use type conversions.) **(4 points)**

4. Using the UART peripheral block (20 points)

a) Test the UART connection

- Download the template project for the PC program using the FtdiUart Library
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/FtdiUartTemplate.tar.gz`
- Build the project (similar to 3a)
- Run the executable with
  `./build/test`
- Adjust the serial number in the main.cpp to such that it fits with the serial number of your FTDI device

- Download the echo program for the MCU
  - Download the file '`echo.hex`'
  - Proceed as in Task 2
- The MCU should now echo the messages you send from the PC program

b) Write your own echo program for the MCU (8 points)

- Download the template project for C programs

  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/Atmega32Template.tar.gz`

T4.1: Consult the data sheet `atmega32.pdf` and program your own echo program considering the following:

- The CPU frequency is 1 MHz
- You can set the baudrate without setting the `U2X` bit
- Configure: 1 start bit, 8 data bits, no parity bit, one stop bit (8N1)
- Calculate the value for the `UBRR` register for a baudrate of 62500 Baud
- You can use busy waits. **(8 points)**

c) Send string messages from the MCU to the PC (4 points)

T4.2: Create a new project using the template for C programs. Use your results from **T4.1** and send the message "Hello world!" to the PC. **(3 points)**

T4.3: Send the message with a frequency of 1 Hz. **(1 point)**

d) Report (8 points)

R4.2: Explain and describe how you calculated your UBBR value for generating the desired baudrate. **(2 points)**

R4.3: Explain why you cannot use any arbitrary baudrate when using the ATMega32 with the setup of this tutorial. **(2 points)**

R4.4: Give one example baudrate which is not possible and explain why. **(2 points)**

R4.5: What can be done if a desired baudrate cannot be achieved? **(2 points)**

5. Adding 512 bit numbers in Assembler (30 points)

a) Implementation (14 points)

- Download the template project for the MCU
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/Atmega32Uart512BitAdderTemplate.tar.gz`
- Download the template project for the PC
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T1/Material/FtdiUart512BitAdder.tar.gz`
- Add your assembly code for a 512 bit unsigned adder into the '`add512.S`' assembler file

T5.1: You are not clobbering registers that are reserved by the C function calling convention and you use the stack for registers that are not free to use. **(4 points)**
T5.2: The arguments a and b are fetched correctly from the SRAM. **(2 points)**
T5.3: The carry bit of the addition is propagated correctly. **(2 points)**
T5.4: The for-loop is correctly implemented in assembly. **(2 points)**
T5.5: The number of iterations of the for-loop is correct. **(2 points)**
T5.6: The result is properly stored in the SRAM location of argument c. **(2 points)**

b) Report (16 points)

R5.1: How (2 points) and in which cases (2 points) do you need to save the CPU status register SREG? **(4 points)**
R5.2: Do you need to save this register when implementing the 512 bit adder? If so, please explain why. **(4 points)**

R5.3: Why is it important to understand the assembly code when creating programs for a platform? Explain at least one important case. **(2 points)**
R5.4: In which cases do you have to use assembly code and when is it preferable to use assembly code? **(2 points)**
R5.5: What are the drawbacks of using assembly code in projects? Name at least two. **(2 points)**
R5.6: What are the advantages of C code for implementing solutions for more general tasks? Name at least two. **(2 points)**

R5.7 (Bonus): How would you implement an unsigned 512bit multiplier using the divide and conquer principle? Elaborate and explain the derived computation in a symbolic way. **(8 points)**
R5.8 (Bonus): How many 8bit multiplications would you need for **R5.8**? Explain. **(4 points)**