# Embedded Control Laboratory

## Exercise 3 – Calibration and Control of the real BioP system with a PID-controller in embedded-C language

"Atmel® Studio 7 is the Integrated Development Environment (IDE) for developing and debugging embedded Atmel AVR® applications. The Atmel Studio 7 IDE gives you a seamless and easy-to-use environment to write, build, and debug your embedded-C and assembler code." (Atmel)
The following Atmel Studio is used to implement the BoiP system and FreeRTOS project.

### Step-by-step instructions to create a project in Atmel Studio 7, include all working files, and download the program into PCB:

1. Start ATMEL Studio (Figure 1)
2. Click “New Project”
    - Select “executable C project”
    - Define a project name (e.g. "vPID", This name is used in the following)
    - Select the location folder and press OK

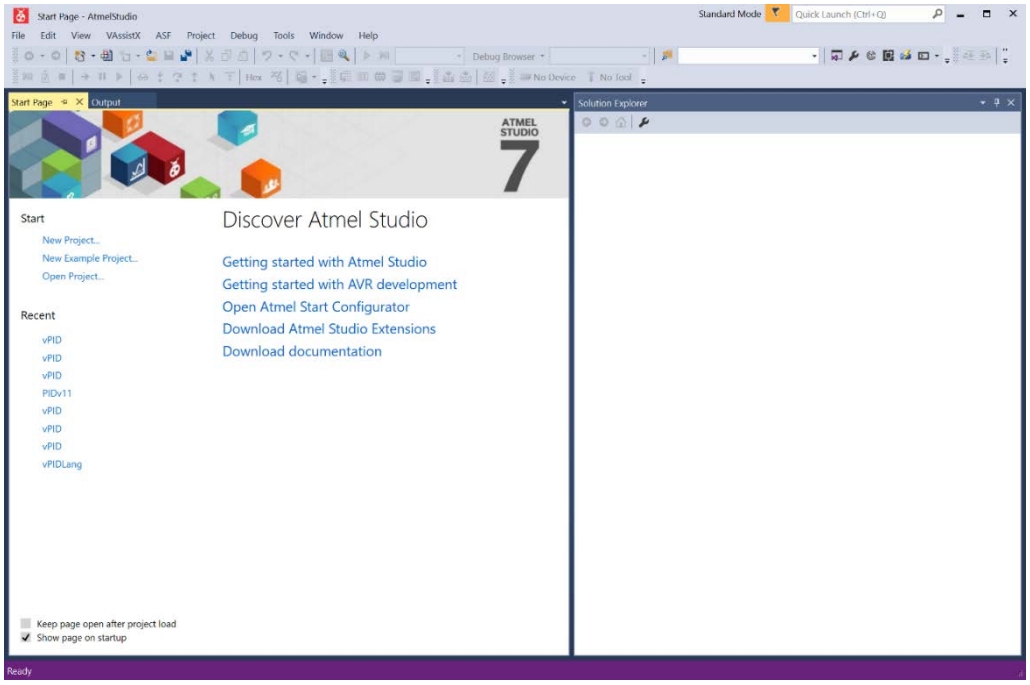


**Figure 1: Atmel Studio Start Page**

3. Choose device "ATmega128" and download the datasheet from the device info section on the right side. Then press OK.

4. Via the Atmel Studio Solution Explorer add all files (*.c and *.h) from the zip-file (provided in moodle) to the project vPID (“drag and drop” the files from the folder/zip-file into the project folder vPID in the solution explorer, see Figure 2).

5. Delete the main.c file from the project folder (via the Atmel Studio Solution Explorer). The file EmCoEx-Wippe.c contains a main() function and the initialization of microcontroller peripherals and the FreeRTOS kernel, so the main.c created by default is not required.

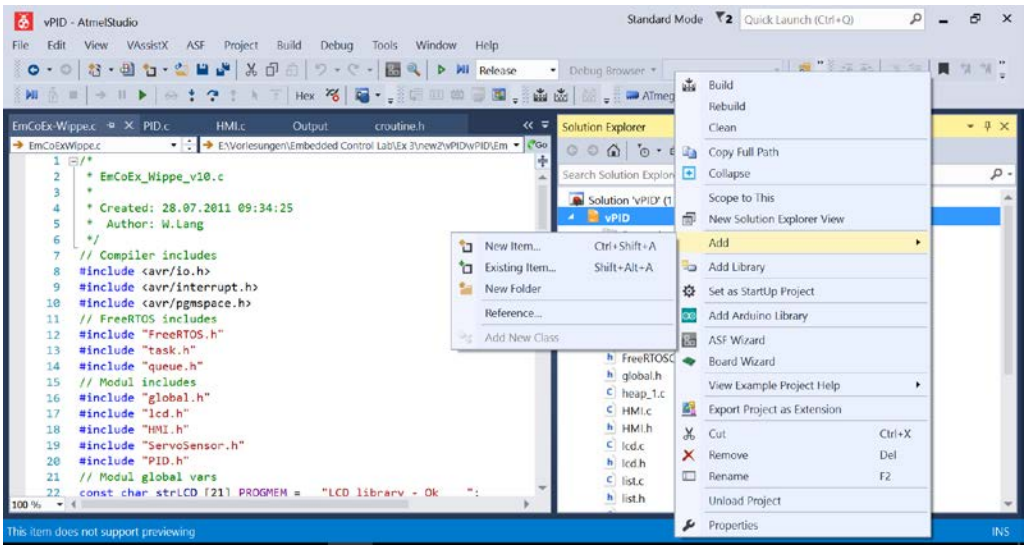


**Figure 2: Atmel Studio Project**

6. For optimization of codes without stack faults select "Release" instead of "Debug" as Solution Configuration.

7. Deactivate the feature “Default compiler char type is unsigned (-funsigned-char)”. To do this right click on the “vPID” folder in the Solution Explorer and click “Properties.” Navigate to Toolchain -> AVR/GNU C Compiler -> General. Uncheck the corresponding checkbox.

8. Add the symbol “GCC_MEGA_AVR” to the Defined symbols (-D) for the GCC compiler. Navigate to Toolchain -> AVR/GNU C Compiler -> Symbols. Click Add Item and enter “GCC_MEGA_AVR.”

9. Start the compiling by selecting Build -> Rebuild Solution. If any errors occur, carefully read the fault description and solve the faults before you recompile. The provided programs should not result in errors after compile. If not, errors may be caused by unsuccessful addition of code files. Try the whole procedure again. Warnings about unused variables can be ignored in this step.

10. IMPORTANT: Before plugging in the USB-ISP programmer always unplug first the power supply of the board. After downloading the program, unplug the programmer from the PCB and then connect the power supply to run the experiments.

11. Connect the Computer with the PCB via the USB-ISP programmer.

12. Download the built solution to the PCB. Navigate Tools -> Device programming. Choose Tool: AVRISP mkII (or STK500 depending on your programmer), Device: ATmega128, and Interface: ISP. Then press Apply.

13. Navigate to the Memories tab. Download the vPID.elf (or vPID.hex) file to the PCB by pressing “Program.”

14. IMPORTANT: DO NOT MAKE ANY CHANGES IN OTHER TABS (e.g. Fuses or Lock bits), since it may cause the device to become unprogrammable.

15. If the device is not listed, add new COM device in Tools -> Add Target -> Add Serial Port.

16. Use the buttons on the PCB and select “Start the PID control” to run the program.

## Introduction to servo motor control:

Figure 3 illustrates motor arm positions in relation to signal pulse lengths. In order to move the servo motor arm to the below mentioned positions, the following pulse lengths have to be applied:

- Left (L) position: pulse length 1 ms.
- Middle (M) position: pulse length 1.5 ms.
- Right (R) position: pulse length 2 ms.

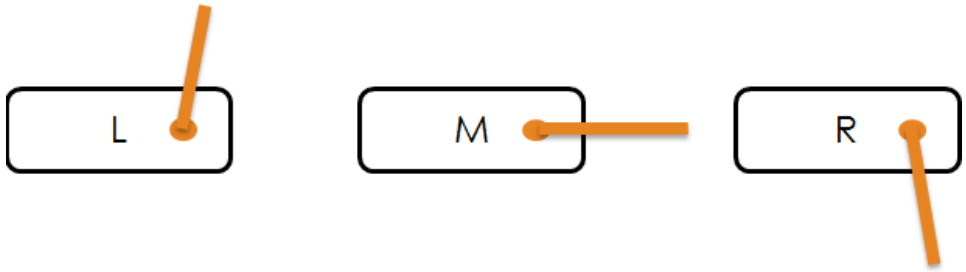


**Figure 3: Servo motor arm positions**

Pulses are repeated every 18 ms as shown in Figure 4.

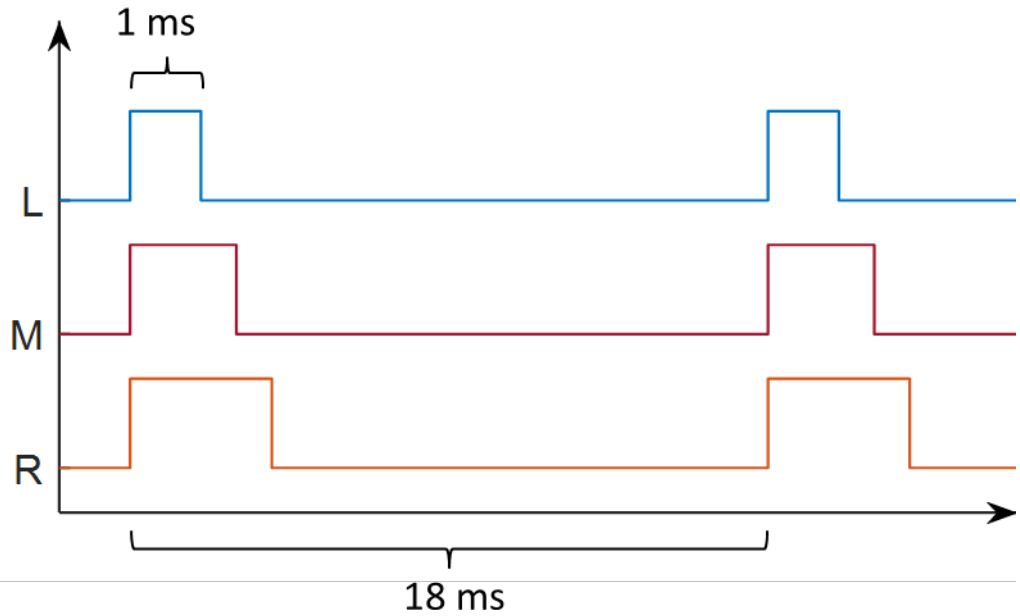


**Figure 4: Signal period and pulse lengths**

# To-Do list for Exercise 3:

1.) Follow the above given instructions and create a project:
   - Build the current project code and download it to the PCB successfully. The default program already contains a P-controller for testing purposes. Warnings about unused variables can be ignored in this step.

2.) Calibration of the model:
   - In the SensorServo.c file:
      - Find out how OCR3A is used to generate the pulses of different lengths with the help of TCCR3B and the system clock frequency. The ATmega128 data sheet (the complete version with 386 pages) provides explanations of how these variables work.

      - In order to maneuver the servo motor arm so the plane becomes horizontal, a pulse length of 1.5 ms should be generated in terms of operating clock ticks, which should be assigned to the variable "MIDPOS." The operating clock tick rate can be derived from the system clock frequency and the applied prescaler (TCCR3B). Calculate and implement the theoretical value of the variable MIDPOS which should corresponds to the plane in the horizontal position (Figure 3).

      - Due to the assembly of the real model, the inclined plane is not exactly horizontal when the pulse length of 1.5 ms is given. Thus, the variable MIDPOS has to be adapted manually from the theoretical value, until the plane is completely horizontal (Perform measurements with calipers). Find the actual MIDPOS value for your model after calibration.

      - Find and implement the correct value of the variable DELTA according to MIDPOS. DELTA represents the gain necessary to move the arm from middle-position to the right or left position.

3.) Welcome message:
   - Modify the welcome message (see below) and display it on the LCD of your PCB
      - "GroupXX Ex 3 20YY", where XX is your group ID and YY the current year

4.) PID implementation:
   - Implement your solution for the discrete PID controller in embedded-C (in PID.c file)
      - Test your code on the real model.
      - Find appropriate PID parameters to control the system. PID coefficients may be altered by giving inputs via buttons on PCB.
      - Goal: The ball has to settle at the reference position within 10 seconds.

5.) Program structure understanding: Tasks and queues
   - Use a graphical representation (e.g. UML diagrams) to describe the following:

- Identify and explain the tasks and timings in the program.
- Identify and explain the queues in the program.
- Identify and explain the exchanged data between different tasks and queues.

## Submission files:

- The report should contain the following with detailed explanations:
  - Process of calibration (including theoretical MIDPOS, actual MIDPOS, and DELTA).
  - Discrete PID controller introduction, implementation, and its parameter tuning.
  - Tasks, queues, and exchanged data.
  - Answers to the questions

- The zipped file for submission should include the below mentioned files:
  - Report in pdf
  - Working program with tuned PID parameters

- The zipped file should be named as follows:
  group<your group ID>_lab<number>.zip/rar
  (e.g. group1_lab1.zip).

  and be sent by E-mail before the deadline to: **ec_lab@eti.uni-siegen.de**
- Please use the file "Lab report cover sheet example" as cover sheet for your report

- The file "Guideline for the lap report" summarizes some important information that may help you prepare a good lab report