PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
Escuela de Ingeniería - Departamento de Ingeniería Eléctrica
**IEE2463 Sistemas Electrónicos Programables**

# 1 Lab 3: Basic Output - Inputs, GPIOs

## 1.1 Summary

This lab introduces basic output operations on MicroController Units (MCUs). You shall learn how to set digital outputs to a high or low state. Digital pins only know these two states voltage on or voltage off. They thereby represent the binary logic used in almost all modern electronic systems. Your task is to properly configure one port of the microcontroller as output and to connectfivesimpleLEDstoit.YoushallusetheseLEDstodisplayvariousbasicpatterns.

## 1.2 Objectives

By the end of the lesson, you shall have written a program which shows three different blink patterns on the five LEDs. Each pattern is to be repeated a few times before showing the next one.Thepatternsare:

- **Simple chaser light** - a light dot "walking up the LED line".It starts back at zero after reaching the end of the line. Repeat this pattern three times before showing the next one.
  10000 → 01000 → 00100 → 00010 → 00001 → 10000 → 01000 → 00010 → *etc*
- **Simple "knight rider" light effect** - a light dot that walks up the LEDs (as in the fist pattern), but then returns walking back down instead of starting over. Repeat this pattern three times before showing the next one.
  10000 → 01000 → 00100 → 00010 → 00001 → 00010 → 00100 → 01000 → *etc*
- **Simple blink pattern** - have every other LED on, and then toggle the pattern. Repeat this pattern ten times before starting again with pattern one.
  10101 → 01010 → 10101 → 01010 → 10101 → *etc*

You can choose the timing to make it look nice, but pick a value that makes the effect clearly visible. A few hundred millisecond per state are typically a good value. You will already have learned about basic control structures in the lecture. Use them to generate these effects. Do not just place a long list of the fixed patterns in the memory  loops can do this much nicer and with shortercode!
Note that you will have to temporarily assemble the circuit yourself with bare components (LEDs, resistors, wires) given to you at the beginning of the class (just plug it together no soldering required). Think about how to do this. Which pins from the microcontroller would you use?Why?

### 1.3 Further Information

The microcontroller used in this lab is an ATmega328P on a developer board called ATmega328P Xplained mini. It stems from the widespread AVR line of the manufacturer Atmel. The according avr toolchain consists of avrgcc with the avrlibc for writing and compiling code, the avr binutils for numerous smaller steps like adjusting the output file formats and avrdudeforloadingthefinalbinarycodeintothemicrocontroller.

**Also you can use MSP430F5529 from Texas Instruments.**

The actual microcontroller development board, five LEDs, five resistors and a variety of wires willbegiventoyouatthebeginningofthelabsession.
The pins on Atmega microcontrollers are organised in several ports. Each port has up to eight pins. Pin two on port B is typically denoted as PORTB.2 or PB2. Note that pin counting starts at zero. Almost each individual pin can be used as an GPIO, a General Purpose Input/Output. By default, all pins are inputs and off (no voltage applied). You will have to switch some pins to be an output instead and to be high (supply voltage and source current to the connected LEDs) or low(novoltagewithrespecttoelectricalground).
The registers for doing that always manage an entire port. DDRB is e.g. the Data Direction Register of port B. To configure PB2 as an output, the second bit in the register DDRB has to be set to the value 1 (as pin counting starts at zero, this is the third bit from the right). In this example, DDRB looks like 00000100. All other pins on port B are still inputs. If you want the lowest three pins to be outputs, DDRB looks like 00000111. Similarly, the high/low state of each pin is controlled by means of the register PORTB. Setting bit 2 in it will make the pin PB2 go high if it has been configured as an output before. Deleting bit 2 (setting it to zero) will make thepingolow,accordingly.

## 2 Task1: Write a program to generate the specified LED test patterns

### 2.1 Required preparation

- You need to bring your own device (e.g. laptop).
- Same as the last time: Read the hardware manual and understand the tool chain. Compiling and loading firmware onto these microcontrollers has been covered in a previous lab. This knowledge is now an expected prerequisite.

- you will need a plan on how to connect the LEDs and resistors to the microcontroller. You do not need to draw a schematic, but you should at least have it in you head. Your supervisor might ask you to draw it at any point, so better be prepared.
- Find out how to initialise some pins or ports of this microcontroller as an output.
- Find out how to set values (patterns) on these ports. How can you switch a single port high or low? (Voltage on or off)
- How can you change the value of one pin without affecting the others on the same port?

## 2.2 Class work

You are supposed to connect the five LEDs to the microcontroller and to write a program that displays the LED patterns as specified in the objectives. The result and the source codes must beshowntoasupervisor.

## 2.3 CLI instruction cheat sheet, for AVR

```
avr-gcc -Wall -Wextra -Wpedantic --std=gnu99 -mmcu=atmega328p -Os -o LAB03_output.elf
    LAB03_output.c

avr-objcopy -j .text -j .data -O ihex LAB03_output.elf LAB03_output.hex

avrdude -vvvv -p atmega328p -F -c stk500v1 -P /dev/ttyACM0 -b57600 -D -V -U flash:w:
    LAB03_output.hex:i
```

## 2.4 CLI instruction cheat sheet, for MSP

```
msp430-gcc -Wall -Wextra --std=gnu99 -mmcu=msp430f5529 -Os -o LAB03_output.elf
    LAB03_output.c

//optional, one can also  flash the .elf file directly
msp430-objcopy --output-target=elf32-msp430 LAB03_output.elf LAB03_output.bin

mspdebug tilib -d /dev/ttyACM0 "prog LAB03_output.bin"
```

## 3 Task 2: Basic Input

### 3.1 Summary

This lab introduces basic input operations on MicroController Units (MCUs). You shall learn how to read the state of digital inputs and process these informations. Your task is to properly configure several outputs and inputs on the microcontroller and to connect buttons and LEDs to it. You shall use these LEDs to display a light dot (always one LED on) which can be moved by means of the buttons. Thereby you will experience an effect called bouncing. Debouncing your inputs is part of this experience.

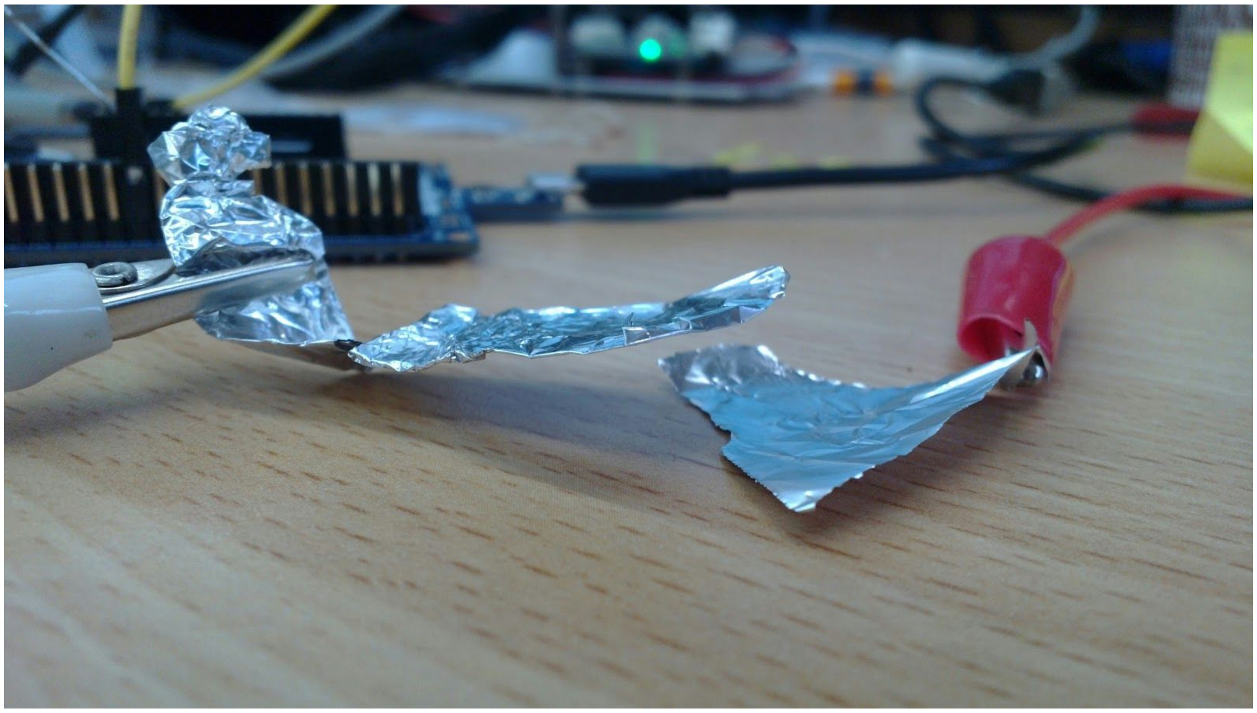
### 3.2 Objectives

By the end of the lesson, you shall have written a program which controls a row of five LEDs. At any given time, one out of these LEDs is on, the others are off.
By pressing the button SW0 on the developer board, you change the position of this LED to the next higher pin on this port. When switching again from the highest position, your program shall wrap around the LED position and start again at zero. Note that this button comes with a hardware debounce circuit on the PCB.
Connect a button to a free input pin on your microcontroller. Program it as the counterpart of SW0 when pressed, it shall move the LED dot one bit position down. When already at the lowest position, it shall wrap around back to the highest one. Prevent your program from switching through several LEDs as long as you keep one of those two buttons pressed. This button might bounce, so consider debouncing it in software instead of hardware (hint: delay functions).
Take some prototyping wires and alligator clips and build a very simple button from two pieces of aluminium foil. Connect this simple button to another (free) input pin of your microcontroller and give it the same function as SW0. You will experience very strong bouncing in this case. Understand what bouncing really means and how it affects your application, then implement countermeasures in software (no hardware debounce!).
Note that you will have to temporarily assemble the circuit yourself with bare components (LEDs, resistors, wires, buttons, metal foil) given to you at the beginning of the class (just plug it together no soldering required). Think about how to do this. Which pins from the microcontroller would you use? Why? Pay attention to where SW0 is connected to!

### 3.3 Further Information

The microcontroller used in this lab is an ATmega328P or a MSP430F5529. **Importan information:** you have to use a different microcontroller from the task 1.

The actual microcontroller development board, five LEDs, buttons, a variety of resistors and wires will be given to you at the beginning of the lab session. Metal foil is available for everyone to fetch a piece. You can fold two pieces of foil like this to make a very bouncy button:



The pins on Atmega microcontrollers are organised in several ports. Despite the information from the last lab, each port has a register PIN, e.g. PINB for port B. Reading this register gives the value present on any input pin of the according port. For pins configured as output, the set output value is read back. Note that each pin can be configured as input or output individually. It might be required to use bit masks and bitwise operations to separate the values one is interested in from the others value read from the same port register.
Also note that ATmega microcontrollers possess internal pullup resistors. They are typically a few 100 kOhm each and can be enabled or disabled for each pin which is configured as an input. Alternatively, external pullup resistors can be used.

# 4 Task1: Write a program to move a light dot trhough a row of LEDs by means of buttons

## 4.1 Required preparation

- Recall the last task. You will need output operations to display any reaction to your input. Controlling output pins is now required knowledge.
- You will need a plan on how to connect the LEDs and resistors to the microcontroller. The same way as the last time might work. But you will also need a plan on how to connect the additional buttons, and pay attention to SW0 (schematic is in the hardware documentation of the developer board). You do not need to draw a schematic for that, but you should at least have it in your head. Your supervisor might ask you to draw and explain it at any point, so better be prepared.
- Find out how to initialise some pins or ports of this microcontroller as an input.
- Find out how to read values (patterns) on these ports.
- Understand bit masks. If you read the input register of an entire port (e.g. PINB), how can you get the value of a single pin out of it?
- You can mix input and output pins on the same port. Does your code from the last lab need adaptations in order to not interfere with inputs?
- What is bouncing? What are software and hardware debouncing methods? Which one is applicable in which situations?

## 4.2 Class work

You are supposed to connect the LEDs and buttons to the microcontroller. Write a program that displays and moves the LED dot as specified in the objectives. Investigate into the bouncing problem and come up with a feasible solution to debounce your additional buttons. The result and the source codes must be shown to a supervisor.

### 4.3 CLI instruction cheat sheet, for AVR

```
avr-gcc -Wall -Wextra -Wpedantic --std=gnu99 -mmcu=atmega328p -Os -o LAB03_input.elf
    LAB03_input.c

avr-objcopy -j .text -j .data -O ihex LAB03_input.elf LAB03_input.hex

avrdude -vvvv -p atmega328p -F -c stk500v1 -P /dev/ttyACM0 -b57600 -D -V -U flash:w:
    LAB03_input.hex:i
```

### 4.4 CLI instruction cheat sheet, for MSP

```
msp430-gcc -Wall -Wextra --std=gnu99 -mmcu=msp430f5529 -Os -o LAB03_input.elf
    LAB03_input.c

//optional, one can also  flash the .elf file directly
msp430-objcopy --output-target=elf32-msp430 LAB03_input.elf LAB03_input.bin

mspdebug tilib -d /dev/ttyACM0 "prog LAB03_input.bin"
```