

LABORATORY

Microcontrollers

LAB 1

NAME	KELVIN MAKAKA
MATRICULATION NUMBER	26219
STUDY COURSE	MECHATRONIC SYSTEMS ENGINEERING
PRESENTATION DATE	18.11.2020
SUBMISSION DATE	26.11.2020

TASK 1.

- a) Flashing of the file program.hex was possible after replacing the PORT in the command from `/dev/ttyUSB_MySmartUSB` to `COM3`.
- b) `avrdude -P COM3 -p m88p -c avr911 -Uflash:w:main.hex:i`

```
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1>avrdude -p m88p -c avr911 -P COM3 -U flash:w:program.hex:i

Connecting to programmer: .
Found programmer: Id = "AVR ISP"; type = S
    Software Version = 2.5; Hardware Version = 2.0
Programmer supports auto addr increment.
Programmer supports buffered memory access with buffersize=512 bytes.

Programmer supports the following devices:
Device code: 0x01
Device code: 0x02
Device code: 0x03
Device code: 0x04
```

TASK 2.

- a) Flashing of all the programs was successful and the Keys functioned as expected.
 - b) *make flash1* – the program checks for the status of the two buttons connected to PIN B4 and B5 where if Button 1(PIN B4) is pressed PIN C1 is set to high which in turn lights up all LEDs. Pressing Button 2 (PIN C1) switches off all the LEDS.
 - c) *make flash2* - simulates a traffic light by a sequence of delays and switching the Leds on and off. This made possible by the delay function
 - d) *make flash3* – the program generates two different tones on the buzzer use of toggling functionality and the delay function which together generate different frequencies depending on how long the delay is.
 - e) *Make flash4* – plays a series of tones dictated by the array *uchNote* at different durations which is dictated by the array *uDuration*.
 - f) *Make flash5* – prints a string of moving characters on the LCD. Two functions are used to shift the strings and add spaces, *shift_runstr* and *init_runstr*
 - g) *make flash6* – the buzzer makes sounds corresponding to the code he generated frequency. The frequency is obtained from the potentiometer through the ADC
- ❖ I find *program "four"* and *"six"* impressive.

TASK 3.

- a) Compiling the *main.c* file created the object file *main.o*.

```
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\Templates\Origin\Experiment1\Task3>ls
init.c  init.h  main.c  makefile

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\Templates\Origin\Experiment1\Task3>
```

```
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>ls
init.c init.h main.c main.o makefile

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>
```

Task 4.

- a) The linking command uses the object file *main.o* as the input and outputs the *.elf* file *main.elf*

```
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>avr-gcc -g -mmcu=atmega88pa -o main.elf
main.o

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>ls
init.c init.h main.c main.elf main.o makefile

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>
```

Task 5.

- a) The *.elf* file was translated into a *.hex* file successfully.

```
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>avr-gcc -g -mmcu=atmega88pa -o main.elf main.o


C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>ls
init.c init.h main.c main.elf main.o makefile

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>avr-objcopy -j .text -j .data -O ihex main.elf main.hex

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>ls
init.c init.h main.c main.elf main.hex main.o makefile

C:\Users\Kelvin\OneDrive\HS3\MicroControllers\2021\Templates\Experiment1\Task3>
```

- b)



```
main.hex
1: 1000000019C020C01FC01EC01DC01CC01BC01AC00C
2: 1000100019C018C017C016C015C014C013C012C034
3: 1000200011C010C00FC00EC00DC00CC00BC00AC064
4: 1000300009C008C011241FBECFEFD4E0DEBFCDBF82
5: 1000400002D00EC0DDCF529853985A9A5B9A219AEB
6: 100050004A9904C04B9902C0299AFACF2998F8CF3F
7: 04006000F894FFCF42
8: 00000001FF
9:
```

Task 6.

- a) Editing the *main.c* makes it possible for the I/O initialisation to be done in a separate file (*init.c*). The header file (*init.h*) is also included in the directory to enable execution of the function call *init()*.
- b) Flashing the created hex file into the microcontroller was possible after inputting the correct PORT.

Task 7.

➤ **Connections**

Button 1 – PD2

Button 2 – PD3

redLED – PB0

yellowLED – PB1

greenLED – PB2

- a) Editing of the make file was successful with changes made to the PORT selection.
- b) Flashing the hex file into the micro controller was successful.
- c) The LED behaviour was successfully inverted by using the NOR operator “!”.
- d) To represent AND function, the logical operator “&&” is used.
- e) To represent XOR Function , logical operation "!A != !B" is used.

Task 8.

- a) Using and including the file *led.h* to control the led behaviour in main.c made the program was successful, hence main.c is easier to understand.
- b) Instead of switching the LED's using long code each time in the main.c file, it is replaced with a simple line of code for example *ledRed(1)* and *ledRed(0)* which are functions executed in the led.c file.
- c) The make file is edited to include led.c and led.h

Task 9.

- a) Button 1 connected to PB0
- b) Red Led connected to PB1
- c) Delay function introduced to ensure smooth running of the button press.
- d) Switching of the behaviour of the LED is done by switching

Task 10.

- a) Debounce function introduced for the button at PB0 which only runs when the button is pressed.
- b) Debounce time of 1000ms
- c) *uint8_t buttonWasPressed* is the state holder for the button, either pressed or not pressed.
- d) Once button is pressed the debounce function is run which then the result is handed to the second if statement which toggles the LED and updates the *buttonWasPressed* state holder.

Task 11.

- a) If statement checks if the button is pressed, then toggles the LED in an infinite loop
- b) The else statement keeps the LED on if the button is not pressed.

Task 12.

➤ **Connections**

PB3 as Output (LED green)

PB2 as Output (LED yellow)

PB1 as Output (LED red)

PC3 as Input (Poti)

- a) `ADMUX |= (1 << MUX0)|(1 << MUX1); // Selects Channel ADC3`

- b) Writing one to ADSC the *Auto trigger* of the ADC is enable thus free running mode is initiated.
- c) ADC3 has 1023 possible values
- d) The reference voltage 5V corresponds to 1023 ADC value
- e) 3.32v : 682 ADC value
- f) 1.66v : 341 ADC value

$$\frac{\text{Resolution of the ADC}}{\text{System Voltage}} = \frac{\text{ADC Reading}}{\text{Analog Voltage Measured}}$$

Task 13.

- a) Defined F_CPU to enable use of delays.

b) Photosensor.

- When the photosensor is fully covered from light (by finger) only the red LED lights up.
- When a shadow is casted on the photosensor only the yellow LED lights up.
- When the photosensor is subjected to intense light the only the red LED lights up.

c) Finger

- Touching the wire connected to the ADC without grounding results in all the LEDs lighting up however on grounding by example putting feet on the ground only the green LED lights up.
- Disconnecting the cable from the potentiometer results in a floating input to the ADC. Touching this endpoint gives unpredictable results or random results that's why all the LEDs light up.
- However, when grounded only the green LED lights up.
- The behaviour observed is a result of the property of **±2 LSB absolute accuracy** of the ADC causing the values after conversion to fluctuate either under or over by 2 levels due to the floating input.

Task 14.

- Connections
 - PC3 connected to Pot.1
 - PC4 connected to Pot.2
- a) A function `uint16_t readADC` is created to read the ADC where it starts a new conversion using ADSC in ADCSRA with each iteration. (single conversion mode).
 - b) `ADMUX &= 0xF0;` Clear the older channel that was read by setting ADC registers 0-2 to 0's `//11110000//`
 - c) `loop_until_bit_is_clear(ADCSRA, ADSC);` is a GCC convenience macros that waits for wait for conversion to finish for the single-conversion mode.
 - d) ADCW is the value holder for what is read from the ADC.

- e) Pota and Potb are assigned the ADC value read from the pins connected to the respective potentiometer.