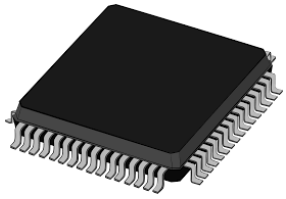
Computer Organization

Lab and Exercise 2



EITF70
EITA15

I/O Handling

Goals

- To understand what an I/O device is
- To be able to interface internal and external I/O devices
- To understand communication protocols
- To get an understanding of practical problems and how to solve them

Christoffer Cederberg
Jonathan Sönnerup
2019

Contents

Introduction

The purpose of this laboratory exercise is to learn how to write programs that interact with the different I/O devices available both inside and outside of the microcontroller.

Lab Equiment

During this laboratory exercise, several I/O devices will be used. The external device that will be interfaced is shown in Figure 1 and 2. In Figure 1 the I/O devices used are encapsulated in a red rectangle and labeled with a number In Figure 2 the corresponding blocks are colored black.

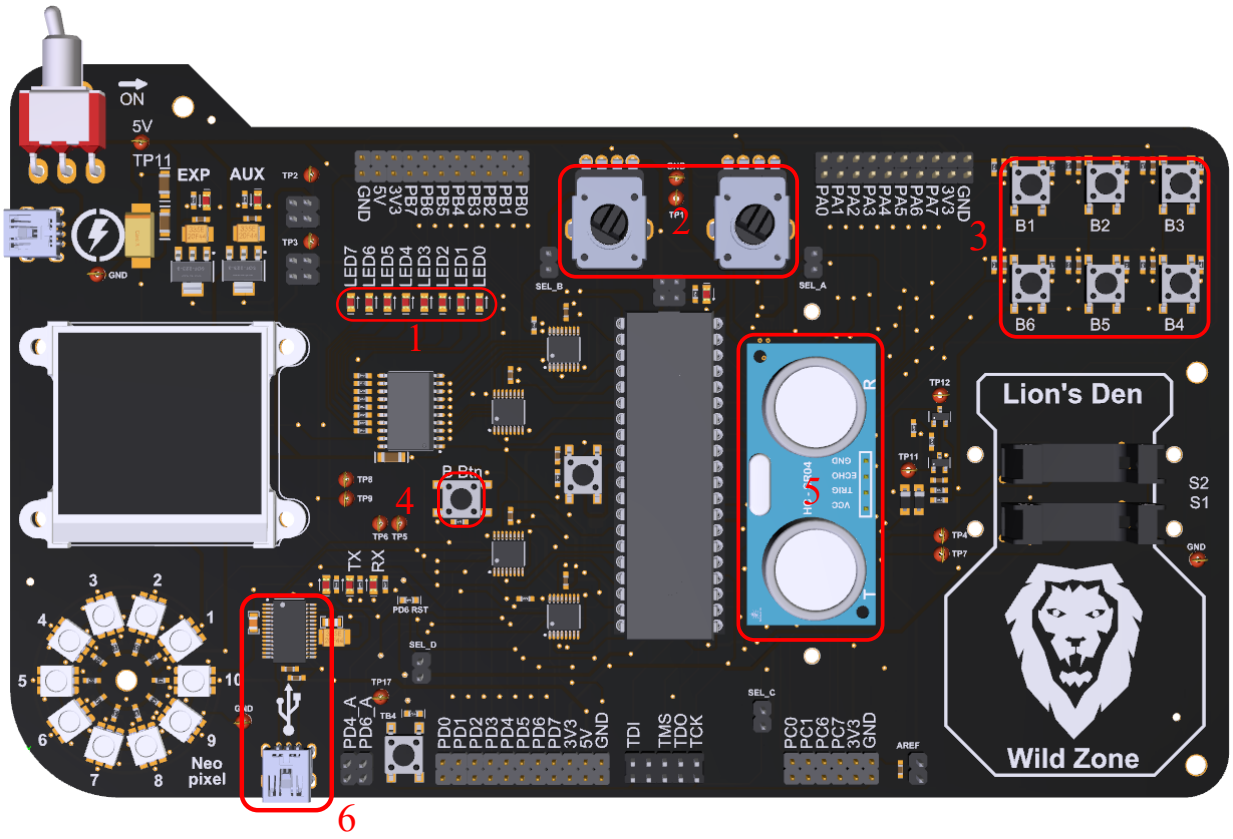


Figure 1: A picture of the circuit board with the used I/O devices marked.

1 - LED's Eight LEDs that are connected to Port B. By setting a pin high (3.3 V), the corresponding LED emits photons with the energy 2.8^{-19} J ;).

2 - Potentiometers Two potentiometers are connected to Port A on the microcontroller (PA0 and PA1). A potentiometer can be seen as a voltage divider where the resistance of the two resistors is changed when the button is turned.

3 - Buttons There are six buttons on the circuit broad. They are connected to Port A on the microcontroller. If a button is pressed, the corresponding pin will be set high (high = 1 in software or 3.3 V in the real world).

4 - Spare button This is a spare button, nothing fancy.

5 - Ultrasonic sensor This piece of hardware can be used to measure the distance to an object. This is done by measuring the time of flight of a burst of sound waves (from the sensor to an obstacle and back). This time is proportional to the distance to the object the sound reflects on.

6 - USB-UART interface This integrated circuit translates UART to USB (and vice versa) and enables the microcontroller to communicate with a PC. In order to send data back and forth to a PC, a USB cable needs to be connected between the USB mini connector on the circuit board and the PC.

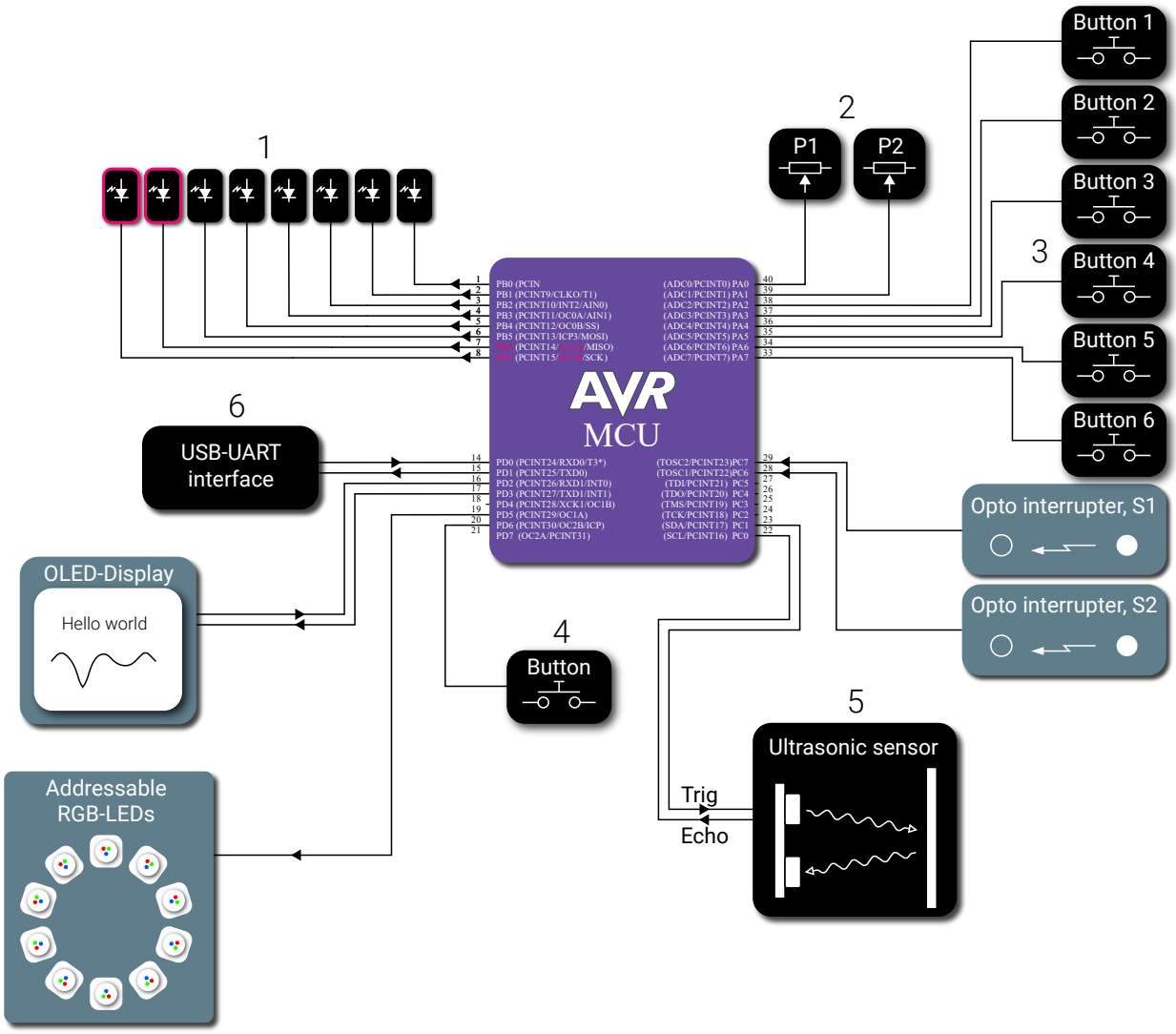


Figure 2: Block schematic of the circuit board with the used I/O blocks coloured black.

Some of the I/O units need to be interfaced in a way that requires more finesse than what a simple, general-purpose I/O pin can achieve. For instance, the outputs of the potentiometers are analog signals. An I/O pin, when configured as an input, can only tell if a signal is above or below a certain threshold. For this reason, it lacks the ability to measure the actual amplitude of an analog signal. In case of the AVR microcontroller (and microcontrollers in general), each I/O pin can be connected (inside the microcontroller) to a couple of different sources. The name of the alternative sources can be seen in Figure 2. It is the text between the parentheses. When it comes to Port A, all of the I/O pins can be connected to the built-in analog-to-digital converter. See the name of the alternative source in Figure 2, `ADC0`, `ADC1` and `ADC2` and so on.

During the lab, two other internal I/O devices will be used, namely one of the two serial communication units and a timer. All of the used I/O devices, coloured black, can be seen in Figure 3.

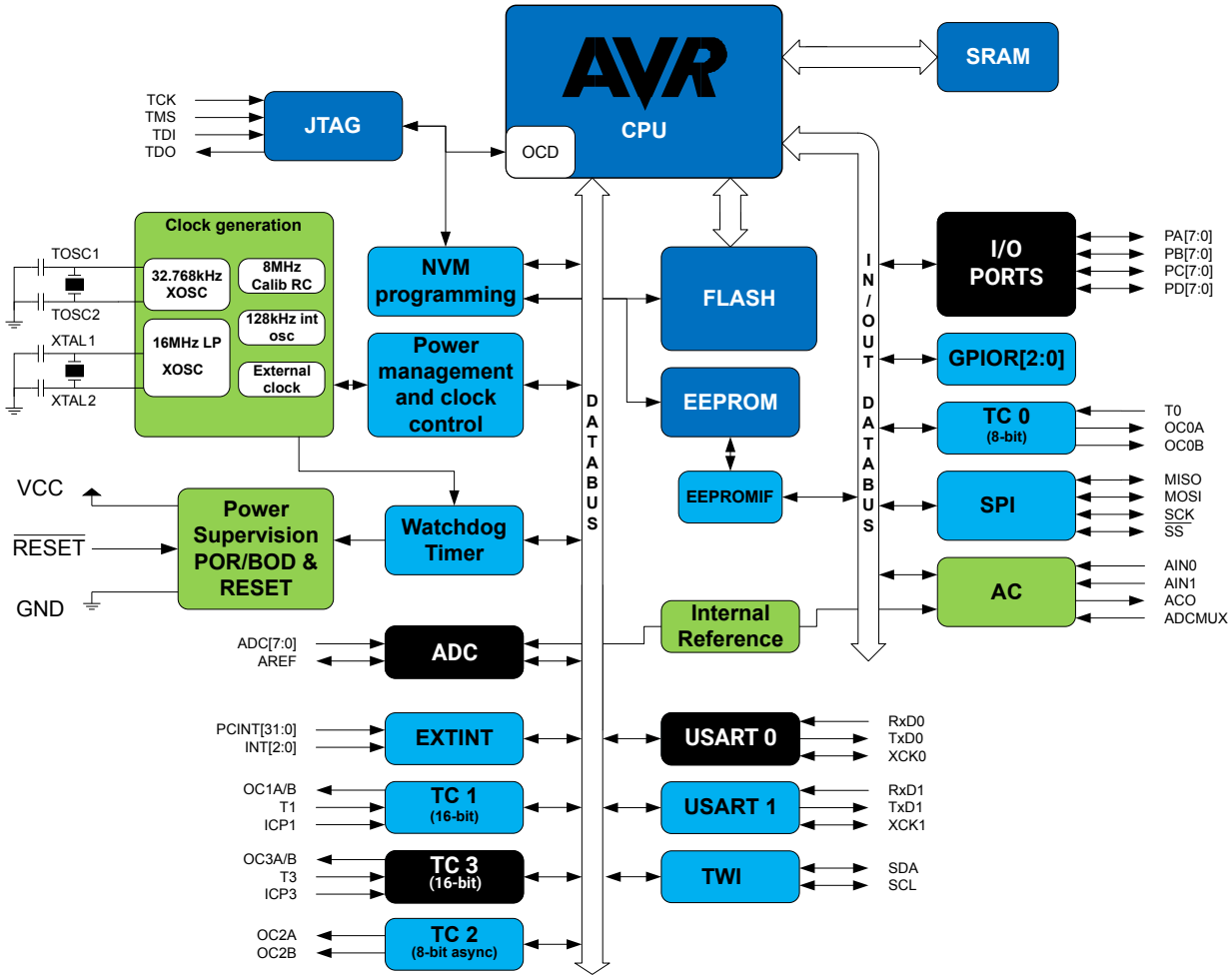


Figure 3: Block diagram for an ATmega1284P.

Chapter 1

I/O Ports

In the previous laboratory exercise, two out of four 8-bit, bi-directional I/O ports were used. The name of the ports are Port A, B, C, and D. With this piece of hardware it is possible to change the state (high or low) of an individual I/O pin. This was done when turning an LED on and off. The I/O ports can also be configured to be an input. This function was utilized when checking the state of a button.

Each port has three different registers. For Port A the name of these registers are `DDRA`, `PINA` and `PORTA`. Each port contains 8 pins and thus the corresponding registers have the same length. In Table1.1 below, their functionalities are described.

Table 1.1: I/O port registers and their corresponding functionality.

Register	**Description**
DDRx	Specifies the data direction (input or output)
PINx	Read the state of the port, if it is an **input**
PORTx	Read and write to the port, if it is an **output**

To set the first pin on Port B, `PB0`, as an output and the pin high, follow the steps below.

- Set the data direction register to `0x01` (hexadecimal) or `0b00000001` (binary).
- Set the pin `PB0` to high by writing `0x01` or `0b00000001` to the port register.

When writing to a I/O port register (or any other register, for that matter) it is important to use the logic operator, OR (|), instead of directly assigning a value. By using the "|" operator, a bitwise OR operation is performed with the content of the register and the value `0b00000001`. By doing so, it is ensured that only the specified bit is changed. This is good, since in the majority of cases there will be different types of devices connected to the same port.

As mentioned earlier, the microcontroller has four I/O ports. See Table 1.2 for their addresses.

Table 1.2: I/O registers and their corresponding addresses.

Address	**Register**
0x20	PINA
0x21	DDRA
0x22	PORTA
0x23	PINB
0x24	DDRB
0x25	PORTB
0x26	PINC
0x27	DDRC
0x28	PORTC
0x29	PIND
0x2A	DDRD
0x2B	PORTD

1.1 Exercises

Answers to the questions can be found in Appendix A.1.

C Programming

Assume you store the results in an 8 bit register.

1.1 What is 12 | 1?

1.2 What is 12 & 1?

1.3 What is $1 \ll 3$?

1.4 What is $3 \ll 1$?

1.5 What is $3 \gg 4$?

1.6 What is $11 \mid (1 \ll 2)$?

1.7 What is $\sim(1 \ll 2)$?

1.8 What is $255 \ \& \sim(1 \ll 5)$?

1.9 Using the pointer, `char *p`, how do you read the value on address 0x1337?

1.10 Using the pointer, `char *p`, how do you read the value from an input port on address 0x1337?

1.11 Using the pointer, `char *q`, how do you write the value 42 to address 0x0666?

1.12 You have declared a variable, `int a`. What is the address of variable `a`?

I/O Devices

1.13 Assume you have 8 LEDs connected to a port, Port G. How would you configure the data direction register (DDRG) for that port?

1.14 Assume the configuration as above, which register (PING and PORTG) would you read from / write to?

1.15 Assume that a pacemaker is connected to the H port, using the 4 most significant pins, sending an A/D converted value 4 bits at a time, how would you read the numerical value from it, the lowest value being 0? Remember that the port is configured as an input.

1.16 The medical company manufacturing the pacemaker realizes that 4 bits is not enough for transmitting data, and now wants to use 8 bits instead. However, we only have 4 available pins on the microcontroller. Assume that the pacemaker now first sends the lower half of an 8 bit number, then sends the upper half of that number (using the same pins as before). How do you, in two consecutive reads, read and store the full 8 bit value?

1.17 You are controlling a servo motor for an Amazon drone to open a hatch for releasing packages. The servo motor expects a pulse every 20 ms. The angle of the servo motor is proportional to the pulse width. Sending a pulse with a width of 2 ms fully opens the hatch, whereas a pulse width of 1 ms closes the hatch. A global variable `release` is set to 1 when you should open the hatch. You should send the pulse to open the hatch for as long as the variable is equal to 1, and send the pulse for closing the hatch when the variable returns to 0. The servo motor is connected to Pin 3 (counting from 0) on Port M. Implement a program controlling the hatch on the drone.

1.2 Lab Exercises

1.2.1 Oh, You Again...

As may be remembered from the first laboratory exercise a state machine, see Figure 1.1, was implemented. In this assignment the same state machine should be implemented, but this time usage of the provided functions in the course library are not allowed. Each port's corresponding register addresses should be used instead. The addresses for the port registers can be found in Table 1.2. There are no restrictions regarding which LEDs or buttons (B1 to B6) to choose.

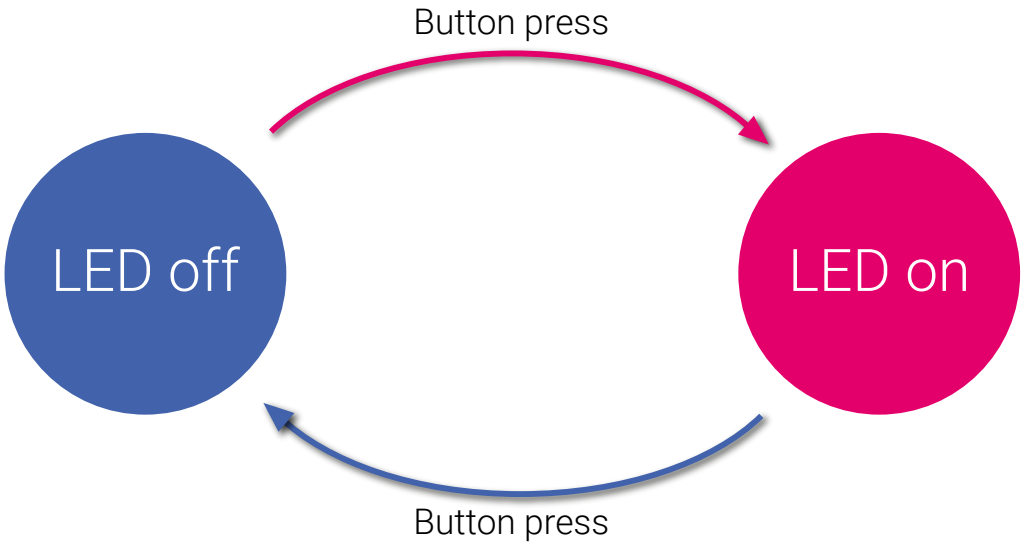


Figure 1.1: State diagram of the program.

Tasks:

- Create a new C project in Atmel Studio.

> **Home Assignment 1.1**
> The LEDs are connected to Port B. The address to the output register is as shown in Table 1.2. Using a pointer to the address, how do you turn LED number 3 (only) on?

> **Home Assignment 1.2**
> The buttons are connected to Port A. The address to the input register is as shown in Table 1.2. Using a pointer to the address, how do you read the value of (only) button 4?

Use the schematic in Figure 2 to see where the the button is connected.

- Implement the state machine with the provided register addresses.

You are now done with this part, show your work to a lab assistant! _____ ☑

1.2.2 Keep It Simple

To make life a bit more easy, Atmel has provided macros for all the memory-mapped I/O register addresses. Instead of writing the actual address and then type cast the number to a `volatile uint8_t` (or any other suitable data type) and then use it as a pointer, it is possible to just use the macro `PINA` instead.

Tasks:

- Create a new project in Atmel Studio.
- Copy and paste the code from the previous assignment and replace all the pointers with their corresponding macros, and verify that it works. Please refer to Table 1.2 if needed.

1.2.3 State Machine, once Moore

Here, we will investigate some of the physical properties of buttons.

Tasks:

- Now we will spice things up a bit. Change the currently used button to the spare button connected to PD7 (Port D). It is labeled with B_Btn and can be found on the left hand side of the MCU, see Figure 1. This button is of the same kind as button B1-B6. And remember to configure PD7 as an input only. There are other components connected to port D.

The correct macro for the pin labeled PD7 in the schematic and the datasheet, is `PORTD7`. Note that the macros are just numbers. `PORTD7` is simply equal to 7. To set the bit that corresponds to pin PD7, left shift a one `PORTD7` positions. (Do not forget that you should only affect the state of the specified pin.)

Lab Question 1.1
Is the application working as expected? Press the button *several* times. Do you notice anything strange? (*There is most likely nothing wrong with the code.*)

Home Assignment 1.3
When pressing a mechanical button, do you get a perfectly clean signal? If no, how do we handle such situations?

The origin of the problem comes from the mechanical properties of a button. When it is pressed the contact plate inside the button is bouncing up and down before settling, and thus a clean transition

from zero to one is not obtained. See Figure 1.2. It has been taken by an oscilloscope and shows the button voltage as a function of time. Each horizontal division is equal to 1 ms.

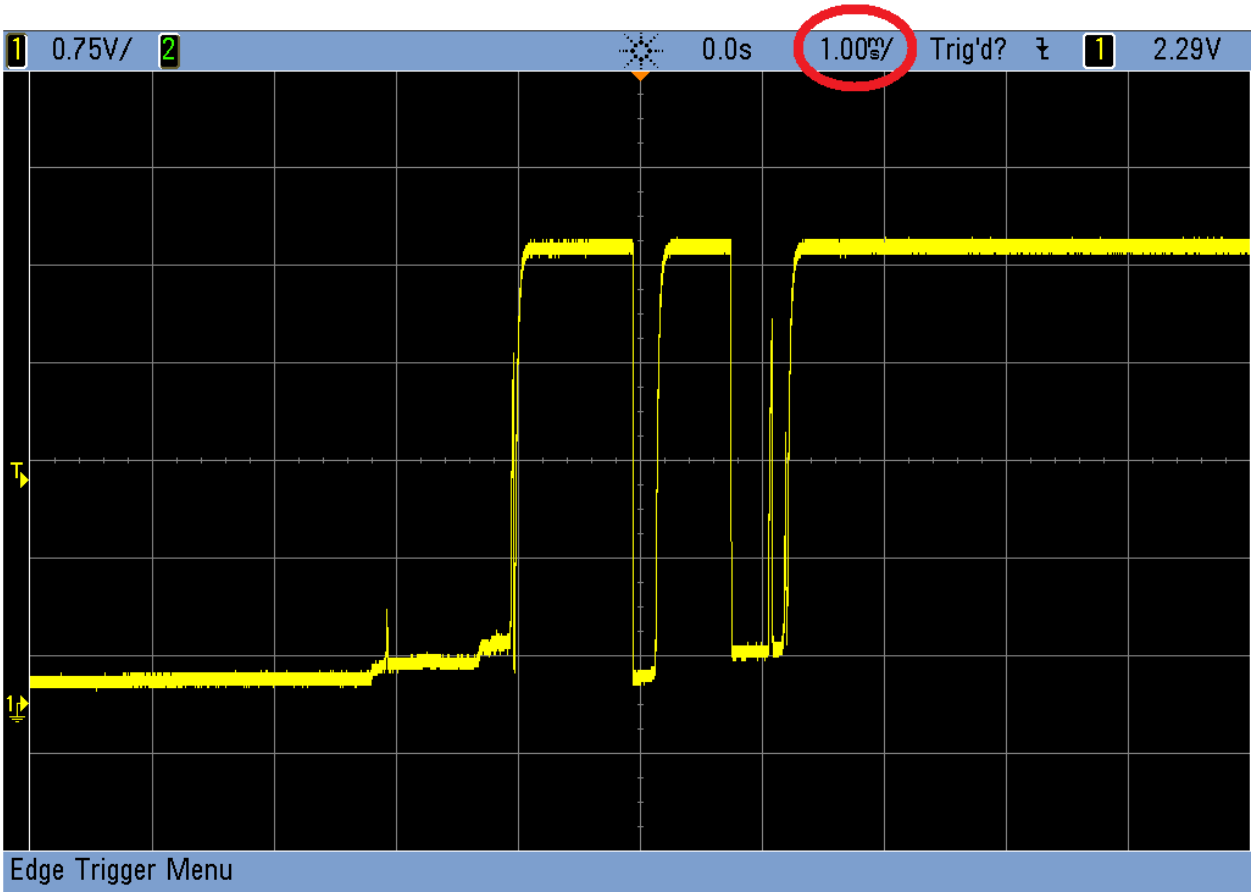


Figure 1.2: A picture of the button voltage taken by an oscilloscope.

Home Assignment 1.4
Create a function, `button_read_reliably`, that reads the button and excludes the bounces, shown in Figure 1.2.

- Test the code written you created in the previous home assignment.

You are now done with this part, show your work to a lab assistant! _____ ☑

Chapter 2

USART - A Serial Communication Protocol

A microcontroller is often used to gather some sort of information and then send it to a computer for analysis. Another typical application is that a computer send commands to a microcontroller, which then performs an action that corresponds to the received command. This can be done with the USART peripheral device inside the microcontroller. The abbreviation USART stands for Universal Synchronous Asynchronous Receiver and Transmitter. With this device, the data is sent bit by bit. The transfer rate, that is, bits/s, is referred to as the baud rate. Each data package consists of one start bit, a number of data bits, a parity bit (optional), and one or two stop bits. See Figure 2.1.

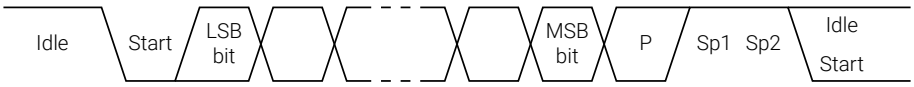


Figure 2.1: UART data package.

There are two versions of this type of communication, one is asynchronous and one is synchronous. The synchronous version requires that a clock signal is connected between the two devices. The more common is the asynchronous version, which is usually referred to as UART (Universal Asynchronous Receiver and Transmitter). This is the version that is going to be used during the laboratory exercises. Each UART device consists of a transmitter and a receiver. The receiver is often labeled Rx and the transmitter Tx. See Figure 2.2.

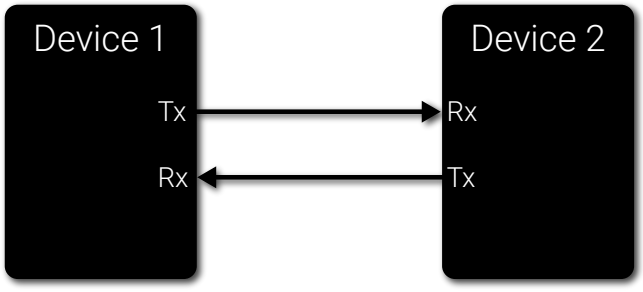


Figure 2.2: UART communication between two devices.

Before the communication starts, the devices needs to be configured, that is, the mode of operation, baud rate, the number of data bits, stop bits, and parity[1] mode has to be selected. This is done by setting bits in the status and control registers of the USART device that corresponds to these options. The registers of the USART device for the used microcontroller can be seen in Table 2.1. For a detailed version, please refer to page 257-265 in the data sheet.

Table 2.1: Registers of the USART 0 unit.

Register	Description
UDR0	USART I/O Data Register 0
UCSR0A	USART Control and Status Register 0 A
UCSR0B	USART Control and Status Register 0 B
UCSR0C	USART Control and Status Register 0 C
UBRR0L	USART Baud Rate 0 Register Low byte
UBRR0H	USART Baud Rate 0 Register High byte

2.1 Data register

Figure 2.3: UART data Register.

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDRn[2]. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR1 Register location. Reading the UDRn Register location will return the contents of the Receive Data Buffer Register (RXB).

2.2 USART control and status

Control and status register A - UCSR0A:
Only two bits in this register is important for the laboratory exercises, UDRE0 and RXC0. They can be used while transmitting and receiving data.

Control and status register B - UCSR0B:
The only important bits in this register are the bits that enable the USART transmitter and receiver.

Control and status register C - UCSR0C:
The default value of UMSEL0[1:0], UPM0[1:0], USBS0 and UCSZ0[2:0] initializes the USART unit as following:

[1]The parity is used as an error detection feature, but it will not be used during the lab exercise.
[2]When coding, the n in UDRn should be replaced with 0 since it is USART0 that will be used. This goes for all names containing an n.

- USART mode is asynchronous,
- parity mode is disabled,
- one stop bit and
- the frame size is 8-bits.

During the laboratory exercises there is no need to change this configuration.

2.3 USART Baud Rate Register `UBRR0H` and `UBRR0L`

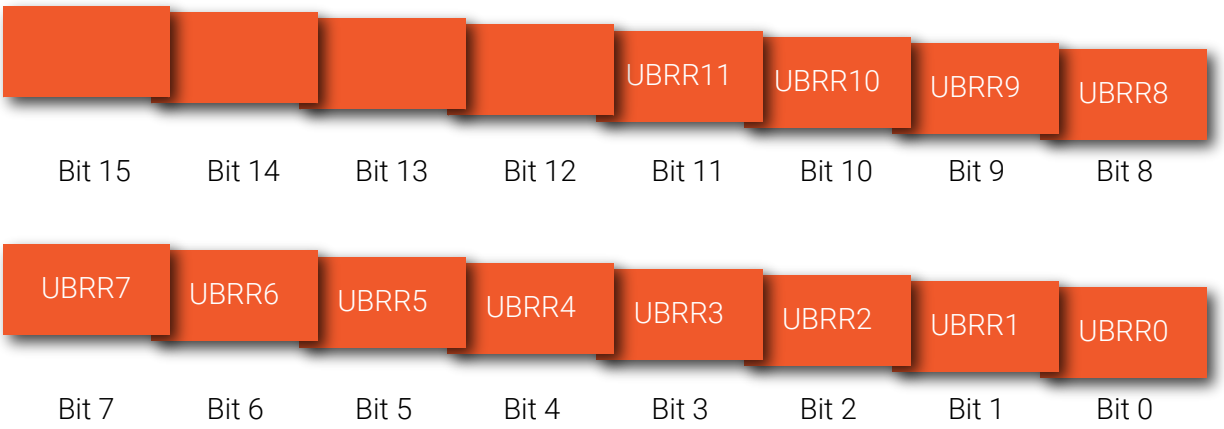


Figure 2.4: UART data Register.

Bits 11:0 – `UBRR011:0`: USART Baud Rate This is a 12-bit register which contains the USART baud rate. The `UBRR0H` contains the four most significant bits and the `UBRR0L` contains the eight least significant bits of the USART n baud rate. Ongoing transmissions by the Transmitter and Receiver will be corrupted if the baud rate is changed. Writing `UBRR0L` will trigger an immediate update of the baud rate prescaler.

Baud Rate [bps]	**fosc = 16.0000MHz and U2X = 0**	**Error rate**
2400	416	-0.1%
4800	207	0.2%
9600	103	0.2%
14.4k	68	0.6%
19.2k	51	0.2%
28.8k	34	-0.8%
38.4k	25	0.2%
57.6k	16	2.1%

To write a value to both `UBRR0L` and `UBRR0H` "simultanously" use the macro in the listing below.

```
UBRR0 = 103;
// alternatively
UBRR0H = (103 >> 8);
UBRR0L = 103;
```

Listing 2.1: Test.

2.4 Exercises

Answers to the questions can be found in Appendix A.2.

Communication

2.1 The maximum transfer speed in the Atmega is 2.5 Mbit/s. What is the fastest transfer speed of the SATA protocol used for SSD drives in a modern computer? What about USB 3.0?

2.2 Using UART with 1 start bit and 1 stop bit, sending data 8 bits, how much overhead do you get?

2.3 Using the UART above, transmitting bits at 1 Mbit/s (including start and stop bits), what is the effective transmission rate (excluding control bits)?

2.4 You recently bought a remote switch for controlling the lights and the TV in your apartment. Being an engineer, you find it a tad boring controlling it manually, hence you decide to use a microcontroller (MCU) to do it for you. You attach a WiFi circuit both to the MCU, and the switches. The WiFi circuits use a serial protocol (UART) for transmitting data. You have the following functions at your disposal. `set_baud(char val)` takes a `char` argument, `val`. The value of `val` is calculated as

$$\text{val} = \left\lfloor \frac{f_{osc}}{16 \cdot \text{baud rate}} - 1 \right\rfloor .$$

The function `set_control(char val)` takes an argument, `val`. Bit 0 determines the number of parity bits (0 - 1), bit 1 determines the number of stop bits (1 - 2), and bits 2 to 4 determine how many data bits (5 - 9) are used. 5 data bits are encoded as "000" and so on. Bits 5 - 7 are not used. The last function is `send_char(char c)` that sends a single character over WiFi using UART.

To control the lights and the TV, you send an identifier along with a value in the following format:

`L1:<on/off>`, for light 1
`L2:<on/off>`, for light 2
`TV:<channel>`, for the TV

For example, sending

"TV:5"

turns on the TV and changes it to channel 5. Send a zero (0) to turn the TV off.

There are two global variables, `hours` and `minutes` that you may use. You want the following functionality:

- At 06:30, turn on light 1 and the TV on channel 4
- At 07:15, turn off the TV, and turn on light 2
- At 07:45, turn off all lights
- At 12:00, turn on light 2
- At 17:30, turn on the TV on channel 6, and both lights
- At 22:00, turn off everything

The UART communicates at a baud rate of 9600 using 1 parity bit, 8 data bits, and 1 stop bit. The MCU is running at 16 MHz.

2.5 Lab Exercises

2.5.1 Communicating with the World - Part I

In this assignment the computer will be used to send a command that toggles a LED.

In order to receive data from the computer, the USART hardware inside the microcontroller needs to be initialized. On the computer side, a serial terminal has to be used to send and receive data. For this purpose a software called YAT (Yet Another Terminal) should be utilized. Instructions on how to use this software will be given later. There are two USART units in the Atmega1284, USART 0 and USART 1. In this assignment the USART 0 will be used. In the data sheet there are some useful tables when selecting the baud rate (page 255-256). Just remember that the CPU frequency is 16 MHz.

Tasks:

- Create a new C project in Atmel Studio.

During this exercise the data sheet may need to be consulted, specifically the pages 258-264. Do not worry, you will not need to read everything.

Home Assignment 2.1
How do you enable the transmitter and receiver for USART0?

Home Assignment 2.2
How do you set the data length to 8 bits and 1 stop bit for USART0?

Home Assignment 2.3
How do you set the baud rate for USART0? You are free to choose any applicable value.

- Implement a "`void usart0_init()`" function using the answers to the above home assignments.

Home Assignment 2.4
How do you read the received data from the USART0? Remember that you need to wait until there is data to read.

- Implement a "`uint8_t usart0_receive()`" function.

Home Assignment 2.5
How do you transmit data via USART0?

- Implement a "`void usart0_transmit(uint8_t)`" function.

Maybe there are some examples in the data sheet (page 244 and 247) for receiving and transmitting data. But remember, a correct explanation of all code is required to pass the laboratory exercises.

- Watch the following video to configure YAT properly: `https://youtu.be/qrL23q0g4VU`. Make sure that all the boxes are marked/unmarked according to the video.
- In YAT, enter your baud rate, the number of data bits, and how many stop bits that are going to be used.
- At this point, it is time to check if the transmit and receive functions work as intended. This can be done by sending back (echo) the data that is received (from the microcontroller's point of view). Write an application that echoes back the received data, using your implemented functions.
- Run the application on the microcontroller and send a character from YAT. You should see the character printed twice in YAT (why?).

Lab Question 2.1
What happens if the baudrate setting in YAT is changed?

- Modify your application to turn an LED on and off based on the character sent from YAT. Choose any characters of your liking.

Lab Question 2.2
Try sending a non-printable character, such as "\x00" or similar. Is it still working?

You are now done with this part, show your work to a lab assistant! _____ ☑

2.5.2 Communicating with the World - Part II

In this assignment, a simple protocol will be created. With this protocol it should be possible to turn on or off each LED on Port B individually. By sending a frame containing a header, a number of data bytes, and a trailer, the state of each LED should be fully determined. See Figure 2.5 below. The header and trailer is used for synchronizing the transmission of data (marking the start and the end of transmission).

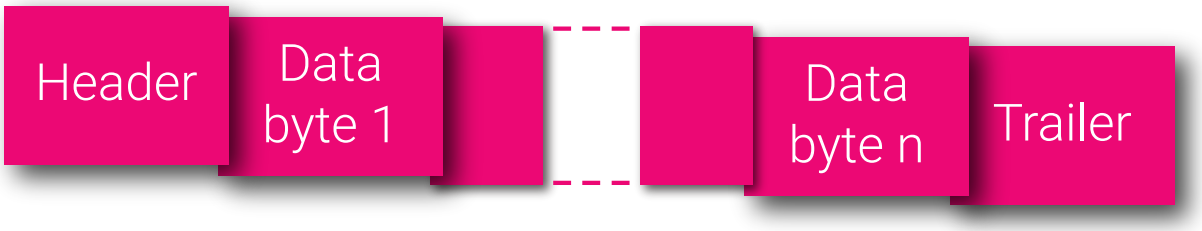


Figure 2.5: A frame from the LED protocol containing the header, n bytes of data, and the trailer.

Tasks:

Home Assignment 2.6
Develop a simple protocol that can be used to turn one or several LED's, connected to `PORTB`, on or off. There should be 8 bytes of data, that is, one byte per LED.

A good idea is to select the header and trailer in such a way that they will never have the same value as the payload, that is, the bytes containing the data.

- Create a new C project in Atmel Studio. Copy and paste the receive and transmit functions from the previous assignment.
- Write an application that utilizes the protocol from the home assignment. Remember to use Atmel Studio's debug capabilities, such as break points and watches (where it is possible to see variables values), if any problem arises.

Of course there are many different ways to solve the given problem. But one thing that needs to be done for all of them is that the received byte has to be be examined. If it is the header of the frame, then it is marking the start of a transmission and the next bytes are data. If the trailer is received, it is the end of transmission and no more data belonging to this frame is expected.

Lab Question 2.3
How could the protocol in this particular case be simplified?

Lab Question 2.4
Why was it not done in the simplified way from the beginning? (*Believe it or not, but the authors do not just want to torment you...*)

You are now done with this part, show your work to a lab assistant! _____ ☑

Chapter 3

Pulse Width Modulation

A pulse width-modulated (PWM) signal is a pulse train where the pulse width is modulated but the period is, in most cases constant. See Figure 3.1. The relationship between the active time and the period is referred to as the duty cycle, see Equation 3.1 below.

$$D = \frac{PW}{T}, \quad \text{where } D \text{ is the duty cycle, } PW \text{ is the pulse width and } T \text{ is the period} \tag{3.1}$$

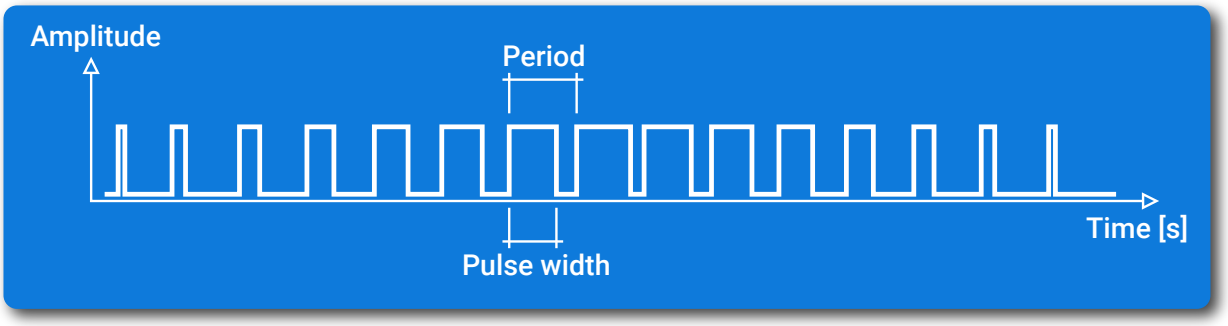


Figure 3.1: Pulse width modulation.

Such a signal can be used for a variety of things. For example it can be used to produce audio, controlling the rpm of a DC motor, or to adjust the intensity of an LED.

3.1 PWM with an AVR Timer/Counter

A timer/counter is a hardware peripheral that essentially counts pulses. The most common used input to a timer is the CPU clock signal. The counter value will be incremented once every clock cycle. The timer/counter unit that is used in the AVR family contains a prescaler (among other things). With the prescaler the input to the timer, that is, the clock signal, can be divided with a selectable factor ranging from 1 to 1024. During the laboratory exercises the CPU clock frequency is 16 MHz. This results in that the time between a counter increment can range from 62.5 ns to 64 μs. The count limit of the timer is given by its word length. The Atmega1284 has two 8-bit and two 16-bit timers and thus, it can count from 0 to 255 or from 0 to 65535.

The timer/counter units can, as the section title implies, be used to produce a pulse width modulated (PWM) signal that will be available on certain I/O pins. Each timer/counter unit can control at least two I/O pins individually. Each pin is called a channel. The channels are labeled `A`, `B`, and so on. The pins associated with the timer/counter PWM functionality are named `OC` (Output Compare) followed by a suffix that denotes which timer/counter unit and channel it is connected to. For timer/counter 3 Channel `A` is named `OC3A` and thus `OC3B` for Channel `B`.

The registers associated with timer/counter 3 can be seen in Table 3.1. Not all of them are need during the laboratory exercises.

Table 3.1: Timer 3 registers.

Register	Description
`TCCR3A`	Control Register A
`TCCR3B`	Control Register B
`TCCR3C`	Control Register C
`TCNT3`	Counter Value
`ICR3`	Input Capture Register 3
`OCR3A`	Output Compare Register A
`OCR3B`	Output Compare Register B
`TIMSK3`	Interrupt Mask Register
`TIFR3`	Interrupt Flag Register

For detailed information, please refer to page 187-198 in the data sheet.

3.2 Fast PWM Mode

There are several varieties of a PWM signal that can be generated with an AVR timer/counter. Here is a description on how to initialize timer/counter 3 to generate "fast PWM." For that purpose an output compare register, `OCR3A` or `OCR3B`, is used together with the counter value register. The desired behaviour is as follows. When the counter value register is zero, the selected `OCR3` pin is set to high. As the timer increments the value, it is constantly compared with the value of the output compare register. When they match, the `OCR3` pin is cleared (set to low). When the counter value is the same as the value in register ICR3 the timer/counter overflows and the output pin is set to high. After this the procedure starts again. See Figure 3.2 and 3.3.

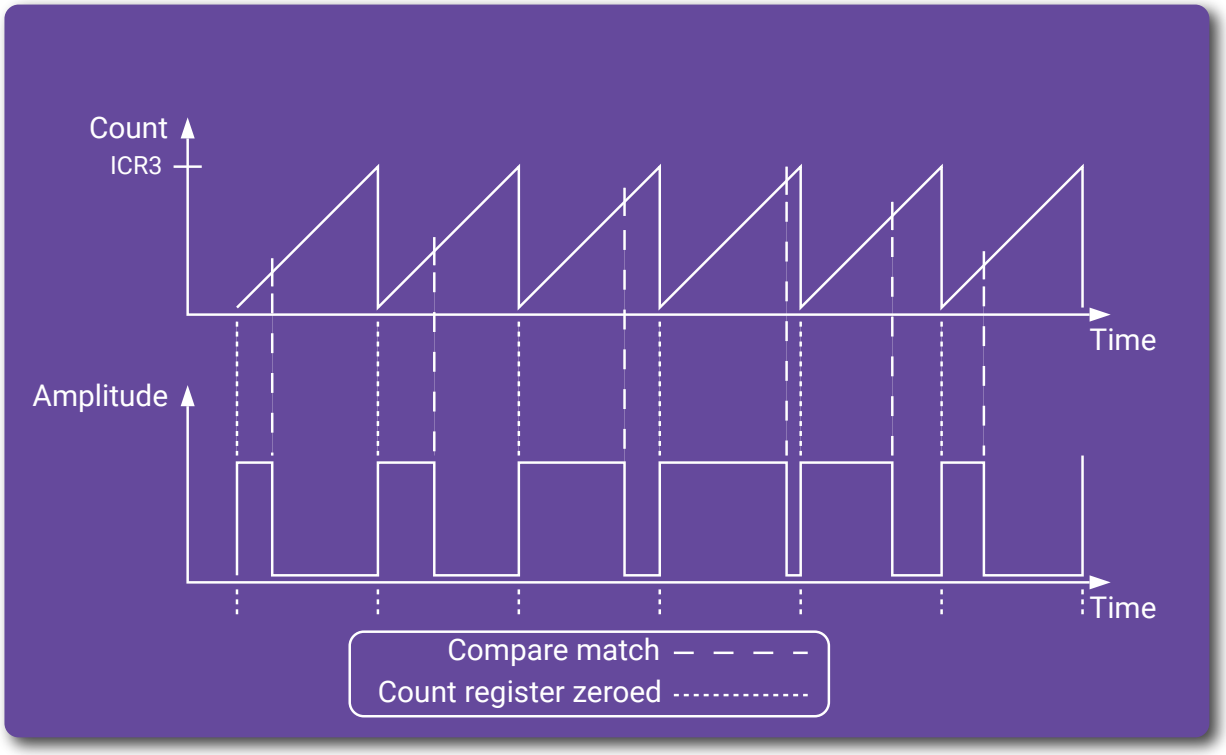


Figure 3.2: Pulse width modulation with an AVR timer/counter.

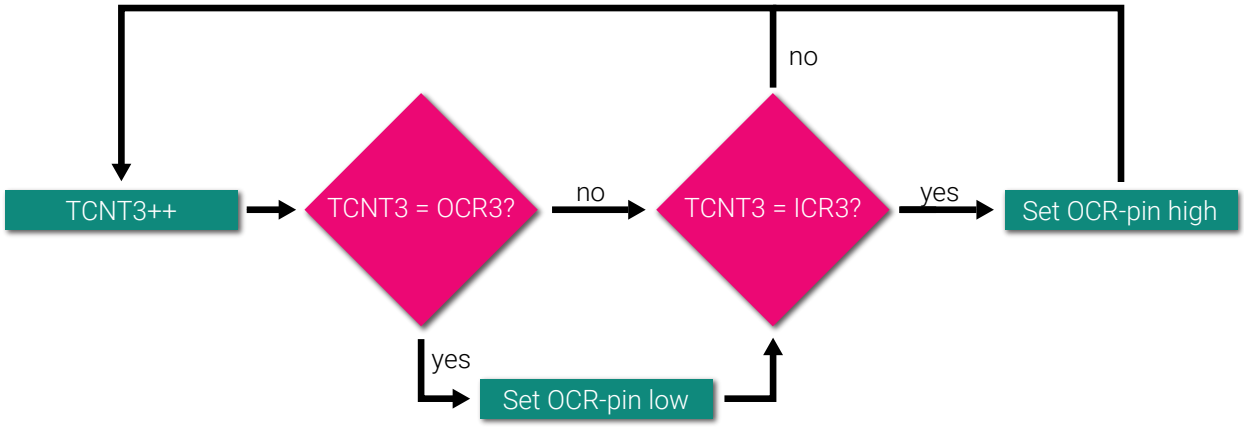


Figure 3.3: Flowchart describing how the timer/counter creates a "fast PWM" signal.

By following the steps below the timer/counter unit will be initialized as described. The timer/counter registers can be seen in Table 3.2.

- In control register A, `TCCR3A`, the Compare Output Mode for Channel A is selected by bit `COM3A1` and or `COM3A2`. For Channel B the corresponding bits are `COM3B1` and `COM3B2`. The bits should be set so that the `OC3A` or `OC3B` pin gets cleared on a compare match. For more details, please refer to Table 17-9 for B on page 187-189 in the data sheet. Both channels can be found on Port B. For this reason

it does not matter which one you choose, since all I/O pins are connected to LEDs. The two I/O pins that are connected to Channel A and B are highlighted with red in Figure 2.

- Select Fast PWM with ICR3 as top by setting the Waveform Generation Mode Bits `WGM30`, `WGM31`, `WGM32` and `WGM33` accordingly. See Table 17-11 on page 188-189 in the data sheet. Having ICR3 as top means that at this value the timer will restart, that is, clearing the counter register and start from zero again. See Figure 3.2. The Waveform Generation Mode Bits can be found in control register A and B.
- Configure the data direction register so that the `OC3A` or `OC3B` becomes an output.
- Set the prescaler bits, `CS30, CS31` and `CS32` in the timer/counter control register `TCCR3B` so that an appropriate division factor is used. See Table 17-12 on page 190-191 in the datasheet.

Table 3.2: Timer 3 registers.

Register	**Description**
`TCCR3A`	Control Register A
`TCCR3B`	Control Register B
`TCCR3C`	Control Register C
`TCNT3`	Counter Value
`ICR3`	Input Capture Register 3
`OCR3A`	Output Compare Register A
`OCR3B`	Output Compare Register B
`TIMSK3`	Interrupt Mask Register
`TIFR3`	Interrupt Flag Register

3.3 Exercises

Answers to the questions can be found in Appendix A.3.

Signals

3.1 You have a continuous stream of pulses. The pulses are high (5V) for 20 ms, and low (0V) for 60 ms.

(a) What is the period time of the signal?

(b) From above, what is the frequency of the signal?

(c) From above, what is the duty cycle?

3.2 The RMS value is defined as the amount of AC (alternating current) power that produces the same effect as DC (direct current) power. For an AC signal, $u(t)$, it can be calculated as,

$$u_{RMS} = \sqrt{\frac{1}{T}\int_0^T u(t)^2 dt},$$

where T is the period. Use the above formula to calculate the RMS value for the pulse signal from the above exercises.

3.3 What is the RMS value for a general square wave with amplitude V_p and duty cycle D? Use the integral formula above.

3.4 Suppose you have a heater at home that you want to control using PWM. You measured the voltage to the heater when it has a comfortable temperature. The measured value is 10V DC. You have a relay capable of handling 25V, which you can control using PWM. What do you need to set the duty cycle to in order to get the same power to the heater, thus the same temperature?

3.4 Lab Exercises

3.4.1 Controlling the Intensity of a LED Using Pulse Width Modulation

In previous assignments, the LED's intensity has been either 0% or 100%. Since LED's are non-linear devices (the relation between the current and the emitted light is not linear), a simple way of controlling the intensity is to periodically turn the LED on and off (the light intensity is proportional to the amount of photons per unit area per unit time). This is a perfect occasion to use a PWM signal. If the period of the signal is short *enough* the human brain will not perceive it as individual blinks. This is similar to still pictures being displayed in a sequence fast *enough* to create a motion picture, that is, a movie clip. To produce a PWM signal, Timer 3 of the microcontroller can used.

Tasks:

- Create a new project in Atmel Studio.

  Home Assignment 3.1
  How do you configure Timer 3 to generate a Fast PWM signal? Look at the `TCCR3A` register in the data sheet.

  Home Assignment 3.2
  How do you configure Timer 3 to use a prescaler value of 1024? Look at the description of `TCCR3B` register in the data sheet.

- Implement a "`void timer3_init()`" function using your answers to the above home assignments.

  Home Assignment 3.3
  How can Pin PB6 in Port B be configured to be an output? This pin is connected to the timer's output compare logic which generates the PWM signal.

- Implement a "`void set_pulse(uint16_t)`" function that sets the output compare register, `OCR3A`, to a value between 0-65535 (0xFFFF) passed in as an argument. You should also implement a "`void set_period(uint16_t)`" which sets the the input capture register, `ICR3`, to the value specified by the input argument. The value to the output compare register, `OCR3A`, should be an input parameter to the function.

  Lab Question 3.1
  What is the period of the generated clock signal from the prescaler? (The frequency of the clock signal to the timer/counter unit is 16 MHz.)

  Lab Question 3.2
  Write an application that initializes timer/counter 3 to generate a PWM signal. The period should be set to 0xFFFF. Set the pulse time to 0xFF.

Lab Question 3.3
What is the PWM period when the timer is configured as above?

Lab Question 3.4
What is the duty cycle for the generated PWM signal?

Now it is time to investigate how short the PWM period time needs to be in order to perceive the light emitted by the LED as continuous. Throughout this task a duty cycle of 50% should be kept.

- Change the period of the PWM signal until the light from the LED looks continuous (the output compare register has to be changed as well).

Lab Question 3.5
How short is the required period time?

When the period time is found, it is possible to modulate the light intensity by changing the duty cycle.

- Add code to the application that changes the duty cycle from 0% to 100% and from 100% to 0%, and then repeats. The duty cycle should be changed at a rate so that the change of the intensity is observable with the human eye.

Lab Question 3.6
The way the 16-bit timer/counter was used here is not good. Why is that?

Lab Question 3.7
What is the effective resolution of the PWM signal?

You are now done with this part, show your work to a lab assistant! _____ ☑

Chapter 4

Volts to Bits - Analog-to-Digital Conversion

An analog-to-digital converter (or ADC) is, as the name implies, used to convert a analog signal[1] to a digital value that represents its magnitude. A key property of an ADC is the resolution. The resolution determines how many quantization levels the magnitude of the analog signal can be encoded in. A digital value from ADC with the resolution of 10-bits can range from 0 to 1023 ($2^{10} - 1$). The resolution can also be directly translated to a voltage. With a 10-bit resolution and a reference voltage[2] at 3.3 V each bit equals 3.22 mV ($\frac{3.3V}{2^{10}}$). A conversion is often repeated, at equidistant time steps, to form a discrete-time and discrete-amplitude digital signal, see Figure 4.1. The rate at which the signal is converted, or sampled at, is referred to as the sampling rate or sampling frequency.

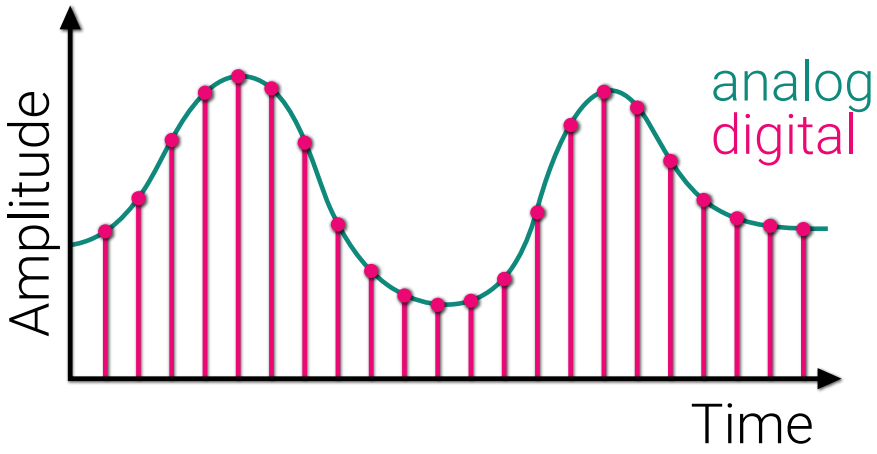


Figure 4.1: An illustration of an analog signal and its digital counterpart.

The reference voltage of the ADC inside the AVR can be set to a variety of things. A common source is AV_{CC}. This is the power supply to the analog parts of the microcontroller. In this case, it is the same as V_{CC}, which is the power supply to the digital part and is equal to 3.3 V. The separation between the two

[1] A continuous-time and continuous-amplitude signal. An example of an analog signal could be audio signal picked up by a microphone or the output voltage from a potentiometer.

[2] The highest allowed voltage that can be converted.

power supplies is done to reduce noise on the ADC value. Almost all digital circuits are considered to be noisy.

See Table 4.1 for the ADC status, control and data registers.

Table 4.1: ADC registers.

Register	Description
ADMUX	ADC Multiplexer Selection Register
ADCSRA	ADC Control and Status Register A
ADCL	ADC Data Register Low Byte
ADCH	ADC Data Register High Byte
ADCSRB	ADC Control and Status Register B
DIDR0	Digital Input Disable Register 0

For more information regarding the associated registers, please refer to page 330-339 in the data sheet.

4.1 ADC Configuration

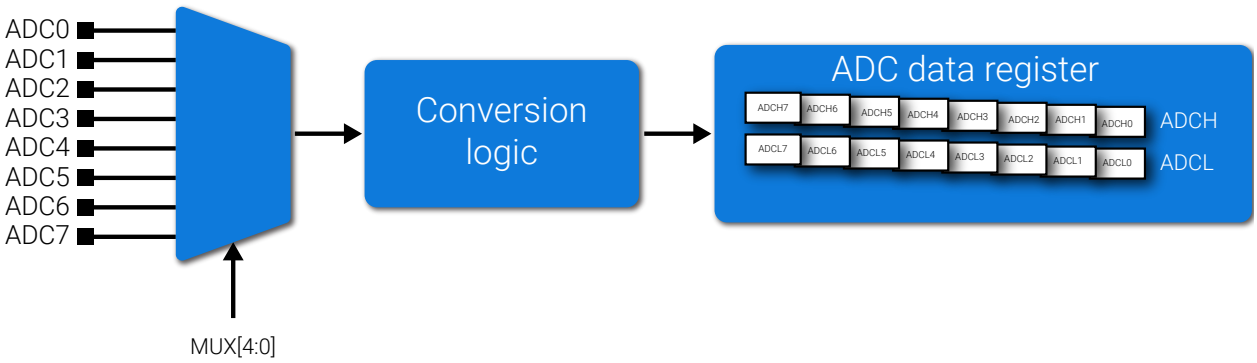


Figure 4.2: Pulse width modulation with an AVR timer/counter.

By completing the bullet points below the ADC will be configured.

- Enable the ADC by setting the enable bit in the ADC Control and Status Register A, `ADCSRA`.
- Furthermore the ADC prescaler should be set to produce a signal (which drives the ADC unit) that has a frequency between 50kHz - 200KHz. This will ensure that the ADC unit performs the conversion with good reliability. See Table 25-5 on page 334 and 335
- In the ADC Multiplexer Selection Register, `ADMUX`, set the `MUX`-bits to select the desired channel. See Table 25-4 on page 3320.
- In the same register, `ADMUX`, choose the AV_{CC} as the reference voltage. This is done with the `REFS1` and `REFS0` bits.

- With the `ADLAR` bit in `ADMUX` register it is possible to choose how the converted ADC value will be stored in the result register, `ADCH` and `ADCL`. For details see page 336-337. By setting this bit to one, the most significant bit is placed at index 1 and the next most significant at 0, in the high byte of ADC result register `ADCH`. The remaining part is placed in the low byte of the adc result register. The way the data is aligned is called right adjusted. See Figure 4.3.

Figure 4.3: The ADC value is stored in the result registers.

- To start a conversion, set the `ADSC` bit to one. This bit is found in the ADC Control and Status Register A, `ADCSRA`. When this is done, the ADC unit performs a conversion. This will take several CPU clock cycles. The result should not be read before the conversion has been completed. The `ADSC` bit (the same bit that starts a conversion) can be used to check if the conversion has completed. Use a loop that breaks if the `ADSC` bit is set to one. When the loop breaks, it is safe to read the data from the ADC Data Register, `ADC`, and another conversion can be started.

Table 4.2: ADC registers.

Register	Description
ADMUX	ADC Multiplexer Selection Register
ADCSRA	ADC Control and Status Register A
ADCL	ADC Data Register Low Byte
ADCH	ADC Data Register High Byte
ADCSRB	ADC Control and Status Register B
DIDR0	Digital Input Disable Register 0

4.2 Exercises

Answers to the questions can be found in Appendix A.4.

4.1 Assume you have signal with 5V amplitude. Assume you AD convert the signal, using a 5V reference and 10 bit resolution.

(a) How many millivolts (mV) per bit do we get?

(b) What is the largest conversion error (in mV)?

4.2 Given the following signal,

$$v(t) = 5\sin(\frac{15}{11}\pi t),$$

at time $t = 2$, what is the AD converted value (0 - 1023) with 10 bit resolution?

4.3 The Amazon drone from before seems to be crashing form time to time, destroying expensive piece of equipment. To mitigate the damages, you installed an accelerometer, ADXL335, along with a parachute in order to detect when the drone is about to crash to save it from breaking.

The accelerometer has an x, y and a z channel to measure movement in 3D. You have a function to get the value from an 8-bit ADC (3 V reference), `uint8_t read_adc(uint8_t channel)`, where the channel value is 0 to 2 for the x to z channels. The accelerometer is powered with 3 V, with a "zero g bias level" (no acceleration) of 1.5 V on all channels. Then, for every additional g force (up to 3g), the voltage increases with ±300 mV. That is, for every channel, the voltage ranges from $[1.5 - 0.9, 1.5 + 0.9] = [0.6, 2.4]$.

(a) How many distinct values can we measure with the ADC?

(b) What voltage does the maximum ADC value correspond to?

(c) What is the voltage range, per channel, from the accelerometer if we remove the constant offset?

(d) Experiments show that a resultant vector with a magnitude of 0.346 (no offset) results in a crash. Recall the Pythagorean theorem in 3 dimensions,

$$\sqrt{x^2 + y^2 + z^2} = d,$$

implement a program that calls the function `release_parachute()`, if magnitude exceeds 0.346. Note that the square-root function is expensive in a microcontroller and should be avoided if possible.

4.3 Lab Exercises

4.3.1 Setting the LED Intensity from an Input

In the previous exercise, the intensity was hard coded. It was not possible to change the intensity after the application was transferred to the microcontroller. In this exercise, an input device will be used to change the intensity while the microcontroller executes the application code. The input device that will be used is a potentiometer. There are two potentiometers on the circuit board, see Figure 1, and the box labeled with 2. With a potentiometer it is possible to change the output voltage. The voltage is an analog signal and needs to be converted to a digital value. This can be achieved with the microcontroller's analog-to-digital converter (ADC). The result should then be used to set the duty cycle of the PWM signal which drives the LED. The timer/counter 3 will still be used to generate the PWM signal.

Tasks:

- Create a new C project in Atmel Studio.
- Copy the `timer_init()` "`void set_pulse(uint16_t)`" and"`void set_period(uint16_t)`" functions from the previous assignment.

  Home Assignment 4.1
  How do you enable the ADC?

  Home Assignment 4.2
  In order to get reliable conversion, the frequency of the ADC should lie between 50kHz and 200kHz. How do you configure the prescaler to meet this requirement?

  Home Assignment 4.3
  The left and right potentiometers are connected to Channel 0 and 1, respectively. How do you configure the ADC to select one of these two?

  Home Assignment 4.4
  How do you configure the ADC to have AV_{CC} as the reference voltage?

- Implement a "`void adc_init()`" function using the answers to the above home assignments.

  Home Assignment 4.5
  How do you start a conversion and read the converted signal from the ADC? Remember that you need to wait until the conversion is done.

- Implement a "`uint16_t adc_read()`" function using the answer to the above home assignment.
- Set the timer/counter prescaler to generate a signal with a frequency that allows the period to be 65535 counts but the light from the LED is still perceived as continuous. Test different duty cycles to

validate that the frequency is suitable.

> **Lab Question 4.1**
> Which prescaling factor was selected and what is the frequency of the generated signal?

- Use the result from the ADC to set the pulse time of the PWM signal that drives the LED.

> **Lab Question 4.2**
> When turning the potentiometer, you do not get the full intensity range. Why is that and how could this problem be solved?

- Do the necessary changes to the your code?

> **Lab Question 4.3**
> What is the effective resolution of the PWM signal after the scaling?

You are now done with this part, show your work to a lab assistant! _____ ☑

Chapter 5

Ultrasonic Sensor - Measuring Distances

The ultrasonic sensor is a device used to measure the distance to an obstacle. It has four pins, Vcc, GND, Trig, and Echo. By supplying a pulse that last for 10 μs the sensor generates a set of sound pulses at 40 kHz. After the sound pulses have been transmitted, the Echo pin goes high. If an obstacle is present (in the path of the sound pulses), the sound reflects back to the sensor. When the sensor has detected the reflected sound pulses the Echo pin goes low. Thus the pulse width of the signal from the Echo pin is equal to the time of flight of the sound pulses, see Figure 5.1. **Note** that the sensor requires at least 60 ms between each measurement to work properly.

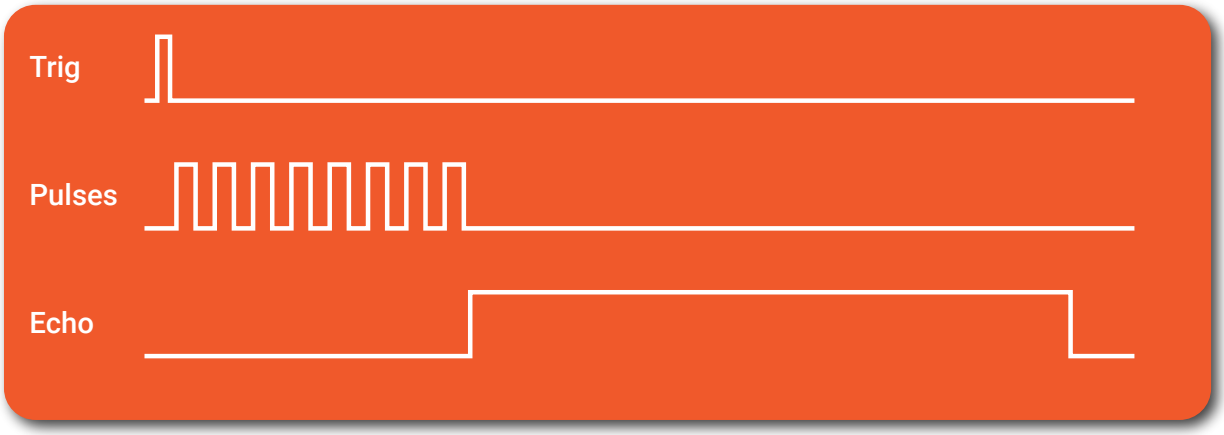


Figure 5.1: Signals to and from ultrasonic sensor.

5.1 Time measurement using the AVR Timer/Counter 3

In order to measure time with the timer/counter, the prescaler bits (`CS30, CS31` and `CS32`) need to be set. They are found in the timer/counter control register `TCCR3B`. These bits control how fast the timer/counter will increment the counter value register, that is, `TCNT3`. For time measurement, this is all that has to be done to start the timer, that is, the counter value register will be incremented at every pulse from the

prescaler. Before a time measurement is done, the counter value register needs to be set to a known value (preferably zero). To stop the timer from incrementing the counter value register, the prescaler bits should be set to zero. This action disconnects the clock signal from the timer prescaler.

5.2 Exercises

Answers to the questions can be found in Appendix A.5.

5.1 Assume a 16-bit timer running at 1 MHz,

(a) What is the timer value after 64 ms?

(b) What is the timer value after 67 ms? Why is this? Can we solve the problem?

5.3 Lab Exercises

5.3.1 Setting the LED Intensity with an Ultrasonic Sensor

Previously, we controlled the intensity of an LED with a potentiometer. While this is useful in real life, it is not really that exciting. Being engineering students, we want control the LED intensity using something way cooler. Therefore, in this assignment, the input device is a ultrasonic sensor (US) which measures the distance to an obstacle using sound waves. The ultrasonic sensor does not output an analog signal, hence the ADC will not be used. However, the sensor returns a digital pulse with a width proportional to the distance. In order to calculate the distance, the time of the pulse needs to be measured, and a timer does the trick. The distance value will then be used to set the pulse width of the PWM signal connected to the LED. See Section 5.1 on how to configure the timer/counter.

Tasks:

- Create a new C project in Atmel Studio.
- Assume the maximum distance to measure is 0.5m, a timer needs to be configured to be able to count the time corresponding to 0.5m.

  Home Assignment 5.1
  How long does it take for the sound to travel to an obstacle at 0.5m (and back!)? Assume that the sound travels 340m/s.

- Use timer/counter 3 and set the prescaler to a value such that the maximum time (from the question above) will fit.
- In order to measure the distance, a trig pulse is sent to the ultrasonic sensor for a duration of 10 µs. Next, wait for the output from the US to go high. When it does, the timer should be started and should continue until the input goes low again. The timer has now counted the number of clock cycles it took for the ultrasonic signal to make a round trip.

  Home Assignment 5.2
  Assume the timer value `TCNT1` is 13337, how far away is the obstacle?

- Add code to the application that creates a trig pulse, starts the timer when the echo pin goes high, and stops it as soon as it goes low. (*Hint: Use a loop that breaks when the Echo pin goes low.*)

  Lab Question 5.1
  Holding an obstacle 10 cm above the sensor, what is the value of the counter? Use the debugger in Atmel Studio and check.

- Use the measured time value to drive the LED. The LED should increase as the distance to the obstacle decreases.

Lab Question 5.2
What happens if the the distance to the obstacle becomes too large?

The timer/counter unit can is started by setting the prescaler bits to something else than all zeroes. To stop the timer, set all the prescaler bits to zero.

That's it folks. The lab is over. You done did it!

You are now done with this part, show your work to a lab assistant! _____ ☑

Appendix A

Answers to Exercise Questions

A.1 Exercise 1

Click here to fast travel back to Section 1.1, price: 1 bit.

1.1 13

1.2 0

1.3 8

1.4 6

1.5 0

1.6 15

1.7 251

1.8 223

1.9 `char val = *((char *) 0x1337)`

1.10 `char val = *((volatile char *) 0x1337)`

1.11 `*((char *) 0x0666) = 42`

1.12 `int *addr = &a`

1.13 All outputs, `DDRG |= 0xFF`

1.14 Read **and** write from/to PORTG, since it is an output.

1.15 `char val = (PINH & 0xF0) >> 4`

1.16 `char val = ((PINH & 0xF0) >> 4) | (PINH & 0xF0)`

1.17 See code in Listing A.1.

```c
#define F_CPU 16000000UL
#include <util/delay.h>

uint8_t release;

int main()
{
    DDRM |= (1 << 3); // pin 3 as output

    while (1) {
        if (release) {
            // send pulse to keep hatch open
            PORTM |= (1 << 3);  // pin high
            _delay_ms(2);
            PORTM &= ~(1 << 3); // pin low
            _delay_ms(18);
        } else {
            // send pulse to keep hatch closed
            PORTM |= (1 << 3);
            _delay_ms(1);
            PORTM &= ~(1 << 3);
            _delay_ms(19);
        }
    }
}
```

Listing A.1: Program for controlling the drone hatch.

A.2 Exercise 2

Click here to fast travel back to Section 2.4, price: 1 bit.

2.1 SATA 3.2 has a maximum speed of 16 Gbit/s, while USB 3.1 reaches 10 Gbit/s.

2.2 20% overhead

2.3 800 kbit/s or 100 kB/s

2.4 See code in Listing A.2.

```c
#define F_CPU 16000000UL
#include <util/delay.h>

void send_message(char *);

uint8_t hours;
uint8_t minutes;

int main()
{
    set_baud(103);
    set_control(0x0D);

    while (1) {
        if ((hours == 6) && (minutes == 30)) {
            send_message("L1:on");
            send_message("TV:4");
        } else if (hours == 7) {
            if (minutes == 15) {
                send_message("TV:off");
                send_message("L2:on");
            } else if (minutes == 45) {
                send_message("L1:off");
                send_message("L2:off");
            }
        } else if ((hours == 12) && (minutes == 00)) {
            send_message("L2:on");
        } else if ((hours == 17) && (minutes == 30)) {
            send_message("TV:6");
            send_message("L1:on");
            // light 2 is already on
        } else if ((hours == 22) && (minutes == 0)) {
            send_message("TV:off");
            send_message("L1:off");
            send_message("L2:off");
        }
    }
}

void send_message(char *msg)
{
    // loop until end of string
    while (*msg != '\0') {
        // send a char and move on to the next
        send_char(*msg++);
    }
}
```

Listing A.2: Program for controlling lights and the TV.

A.3 Exercise 3

Click here to fast travel back to Section 3.3, price: 1 bit.

3.1 (a) T = 80 ms

(b) f = 12.5 Hz

(c) $D = \frac{20}{80} = 0.25$

3.2 The function is not continuous, hence we must separate the integral as

$$u_{RMS} = \sqrt{\frac{1}{0.08}(\int_0^{0.02} 5^2 dt + \int_{0.02}^{0.08} 0^2 dt)} = \sqrt{\frac{1}{0.08}\int_0^{0.02} 25 dt}$$
$$= \sqrt{\frac{1}{0.08}\left[25t\right]_0^{0.02}} = \sqrt{\frac{1}{0.08}\left[25 \cdot 0.02 - 25 \cdot 0\right]}$$
$$= \sqrt{\frac{25 \cdot 0.02}{0.08}} = \sqrt{6.25} = 2.5V$$

3.3 In the general case, the RMS value for a square wave is calculated to be

$$u_{RMS} = V_p\sqrt{D}.$$

3.4 Using the general formula, we get

$$u_{RMS} = V_p\sqrt{D},$$

where $V_p = 25\text{V}$ and $u_{RMS} = 10\text{V}$. Solve for D, and we get $D = (\frac{10}{25})^2 = 0.16$. The duty cycle should be around 16%.

A.4 Exercise 4

Click here to fast travel back to Section 4.2, price: 1 bit.

4.1 (a) $\frac{5\text{V}}{2^{10}} \approx 4.88\text{mV}$

(b) The largest error is half the value per bit, 2.44 mV.

4.2 The voltage at time $t = 2$ is

$$v(2) = 5\sin(\frac{15}{11}\pi \cdot 2) = 3.778\text{V},$$

resulting in a value of

$$\frac{3.778}{0.00488} \approx 774.$$

4.3 Answer

(a) $2^8 = 256$

(b) $\frac{3\text{V}}{256} \cdot 255 = 2.988\text{V}$

(c) Removing the 1.5 V offset, we get $[-0.9, 0.9]$

A.5 Exercise 5

Click here to fast travel back to Section 5.2, price: 1 bit.

5.1 (a) Each clock cycle takes 1 μs. 64 ms is 64000 clock cycles.

(b) We are counting mod 2^{16}, hence the timer value is $67000 \mod 2^{16} = 1464$. We solve this by another counter increasing every timer overflow.