





## Key-Programmable Control

Microcontrollers are everywhere: in household devices, in devices of entertainment electronics, in meters and even in unmanned space vehicles. Everywhere, they do things that a programme tells them. It is very exciting to generate simple control programmes yourself as well.

The first step always is choosing a microcontroller or processor that matches your desired task as precisely as possible. You can choose between any number of types from different companies. You can also choose your programming language. Assembler and C are offered most often; in many cases, so is Basic or another high language. Usually, programming requires elaborate software and a programming unit. In addition to the financial effort, the familiarisation time is quite considerable.

The microcontroller used is entirely different. You can programme it with only two pushbuttons. The *button-programmable control* (tastenprogrammierbare Steuerung; TPS) knows only relatively few commands that can be learned easily and that can be programmed into the controller with the buttons. The programme can be changed at any time and without any special aids.

The system is particularly suitable for compact applications in the areas measuring, controlling and regulating. Many tasks can already be completely solved with this system. Additionally, you can install the microcontroller into circuits of your own after successfully programming it. Basic knowledge in the electronics area is required for this.

At the same time, the system is suitable for training and the first steps in microcontroller programming. Success will occur faster than in other systems. The structures are, however, very similar to other programming languages, so that the later transfer is made easier.



# Table of Contents

Key-Programmable Control	3
Table of Contents	5
1 Introduction	7
2 Alternating Flash	10
3 Binary Counter and PWM Output	11
4 Analogue-Digital Converter	15
5 Random Generator	17
6 Impulse Length Measurement	19
7 Reading Programmes	21
8 Entering Programmes	23
9 Recovery of the Example Programmes	25
10 TPS Basic Commands	25
11 Computing with Variables	28
12 Jumps and Branches	30
13 Command Overview	32
14 Counting Loops	33
15 Comparisons	34
16 AND, OR and XOR	35
17 Subprogrammes	36
18 Twilight Switch	38
19 LED-Dimmer	38
20 Number lock	40



# 1 Introduction

The principle of the TPS controller is simple. You have four digital inputs E1 to E4 and four digital outputs A1 to A4. There are also two analogue inputs AD1 and AD2 and a quasi-analogue PWM-output. A reset input with a connected reset button resets a programme to the start. The controller is supplied with three AA cells with approx. 4.5 V and can work in a range of 2.2 V to 5.5 V.

Technical Data:

Microcontroller: HT46F47

Clock frequency: 2 MHz

Internal EEPROM: 128 Bytes

Power supply VCC: 2.2 V to 5.5 V

Current consumption: 1 mA at 4.5 V

4 output ports: resilient until 10 mA

1 PWM output: resilient until 10 mA

4 input ports: Quiescent condition 1

2 analogue inputs: 0 V ... VCC

2 key inputs: Quiescent condition 1

Components in the learning package:

Pinboard

Battery compartment 3 \* AA

Wire

HT46F47 with TPS-firmware

3 Pushbuttons

4 LEDs 5 mm, red

1 LED 5 mm, green

1 LDR

3 disc capacitors 100 nF

1 electrolyte capacitor 47  $\mu$ F

5 resistors 2.2 k $\Omega$

1 resistor 10 k $\Omega$

1 resistor 27 k $\Omega$

2 resistors 100 k $\Omega$

For programming, you need the two buttons S1 and S2 and a simple LED display of four LEDs at the outputs A1 to A4. There is a total of 14 simple commands with associated data or subcommands. Commands and data are each encoded as 4-bit binary numbers from 0000 to 1111 (decimal 0 to 15) and are directly visible at the display LEDs. The respective number is programmed to S1 by pushing buttons when programming. S2 switches between command and data and increases the address in the command line. The entire programme structure is so simple that you will know it by heart with a little practice.

Fig. 1: Basic circuit of the system

Fig. 2: Standard setup with pushbuttons

Some basic programmes are already present in the delivery condition of the TPS-controller (default) and can be started directly. Therefore, it is possible to take the controller into operation

step by step. First, familiarise yourself with the hardware functions and start your own programmes only after this.

In the first test, you will start small programmes that are already finished in the controller. The associated programme lists provide an initial impression of the options. They are only briefly explained. The precise explanation of the individual commands follows in the next chapter.

For the first test, set up only the basic configuration with the controller and the required additional elements on the pinboard. You need:

- Connection of the voltage supply at GND (minus) and VCC (plus)
- A blocking capacitor 100 nF between VCC and GND
- Reset resistor downstream of VCC and reset capacitor downstream of GND
- Oscillator resistor 100 kΩ downstream of VCC and capacitor downstream of GND

The microcontroller HT46F47 is operated with its internal RC-oscillator. The resistor at the Osc1 input specifies the cycle frequency. 100 kΩ sets a frequency of approx. 2 MHz. On demand, you can work with lower or faster speeds. The connected capacitor serves only to block and has no influence on the cycle frequency. The connection Osc2 remains free. On demand, you can connect an additional resistor against VCC here and disconnect impulses with a quarter of the cycle frequency.

Fig. 3: For LEDs at the outputs

Use the upper and lower supply rails on the pinboard as ground connection GND. Here, the black cable of the battery compartment, i.e. the minus pole, is connected. The plus line VCC is connect to the red connection cable of the battery compartment. Wrong polarity must be avoided

under all circumstances, since it may destroy the controller. Install a brief wire piece as tension relief. Once the voltage connection has been established, it should remain connected continually if possible. To deactivate it, take one of the batteries from the compartment.

Fig. 4: Minimum application with LEDs

Already use the reset button additionally and connect four LEDs to dropping resistors of 2.2 kΩ. They are needed for the first hardware tests. Observe the order. A1 is connected to the left LED and A4 to the right one. The binary display with the highest-value bit is on the left. This is helpful particularly during later programming.

## 2 Alternating Flash

Now insert three 1.5-V batteries or alternatively three NiMh rechargeable batteries into the battery compartment. This starts the first example programme with an alternating flash with the left and right LED. the flashing frequency is approx. 1 Hz. The programme listing shows the simple programme with only five lines. Alternatingly, LED 1 and LED 8 are switched on. Between them there are waiting commands with a waiting time of 0.5 s. Jumping back to the beginning ensures that flashing is continued forever. The individual commands are explained in more detail below. You can see how simple the programming is by this example. The firmware of the controller has an interpreter that recognises and executes the simple commands. Programmes therefore are much more compact than in other systems.

The example occupies the address range from 20h onwards (decimal 32). Several programmes in the upper address range can be started later from dedicated applications. The addresses can also be overwritten with your own programme code. On demand, the controller can also be reset to the basic condition, which resets the original example programmes.

Address	Command	Data	Comment
20	1	1	LED 1
21	2	8	Wait for 500 ms
22	1	8	LED 8
23	2	8	Wait for 500 ms
24	3	4	Jump -4

Listing 1: Alternating flash

If the desired result does not occur, check proper polarity of the LEDs first. It is also helpful to measure some voltages. Always use, e.g., a digital multimeter in the 20 V range and leave the minus connection at GND. All voltages are thus measured against GND:

VCC: 4.5 V  
 Reset: 4.5 V  
 Osc1: 1.5 V  
 E1 to E4: 4.5 V  
 A1: Alternating  
 A2, A3: 0V  
 A3: Alternating

### 3 Binary Counter and PWM Output

All digital inputs are pulled up with an internal resistor against VCC (pull-up resistor) and have a quiescent voltage at the amount of the operating voltage. However, you can put each of the inputs to GND with a wire or contact. When starting, the default programme reads the port condition and evaluates it. Individual connections can be put on GND so that a zero condition is read here. Depending on the result, different programmes are called.

Fig. 5: Use of the PWM-LED

Fig. 6: Starting the binary counter

Apply E1 to GND. This starts the second example programme after a reset. It counts up the output conditions binarily. The conditions 0000 (decimal 0) to 1111 (decimal 15) are continually

gone through. The programme uses variable A for a simple addition and for output to the digital outputs as well as the PWM-output. Commands 7 and 5 have subfunctions that are written as data.

Address	Command	Data	Comment
25	7	1	A = A + 1
26	5	4	Port = A
27	5	9	PWM = A
28	2	6	Wait for 100 ms
29	3	4	Jump -4

Listing 2: Binary counter with LEWD and PWM output

The counting programme can be used as a practice programme for reading binary counters that you must master for your own programming. Each of the four LEDs represents one bit. All in all, a 4-bit number can therefore be displayed. The LEDs are called 8, 4, 2 and 1 according to their value in the circuit diagram. Addition of the respective values results in the decimal figure. Written hexadecimally, the numbers 10 to 15 are represented by capital letters A to F.

8	4	2	1	Decimal	Hexadecimal
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	10	A
1	0	1	1	11	B
1	1	0	0	12	C
1	1	0	1	13	D
1	1	1	0	14	E
1	1	1	1	15	F

The programme can be used as a flashing encoder for different frequencies as well.

The next-higher output has half the frequency or twice the period duration each:

A1: 200 ms  
A2: 400 ms  
A3: 800 ms  
A4: 1,600 ms

Additionally, the rising numerical values are also output to the PWM output (pulse width modulation) . The PWM signal is a rectangular signal with a frequency of approx. 16 kHz. The pulse length is controlled so that the pulse-pause ratio determines the average activation duration and thus the brightness of the LED. The brightness of the connected LED is controlled in 15 levels between zero and full brightness.

A PWM signal can be smoothed to a direct voltage with an RC low-pass filter. The PWM-output thus becomes an analogue output. With this programme, you will receive a direct voltage that increases gradually from 0 V to 4.5 V. Track the voltage progress with a meter or an oscilloscope.

Fig. 7: Low-pass filter at the PWM output

Fig. 8: Smoothed PWM output voltage

## 4 Analogue-Digital Converter

A connection E2 to GND and pushing the reset button will start a small example programme for the analogue-digital converter (AD-converter). The analogue voltage at the analogue input AD1 is measured and converted to a digital numeric value. Because the TPS controller works with 4-bit values continually, the result of the analogue-digital conversion is a number from 0 to 15. The result 0 represents the input voltage 0, the result 15 for a voltage that corresponds to the operating voltage, i.e. e.g. 4.5 V. The AD-value is output as a binary number at the four LEDs and additionally handed over to the PWM-output. Connect a voltage divider from a fixed resistor and a light-dependent sensor resistor (LDR) to the analogue input AD1.

Fig. 9: Connection of the light sensor

Fig. 10: The LDR at the AD1-input

The example programme is very similar to the programme from the last section because of the output to the digital outputs and the PWM-output. The first line has the command to convert an analogue value, however.

Address	Command	Data	Comment
2A	6	9	A = AD1
2B	5	4	Port = A
2C	5	9	PWM = A
2D	2	6	Wait for 100 ms
2E	3	4	Jump -4

Listing 3: AD converter and PWM output

Test the programme with different sensor light exposure. The more light on the LDR, the lower the voltage at AD1. Vice versa, darkness leads to maximum AD-values and thus maximum brightness of the LED at the PWM output. Read the binary lines from the LED display and try to adjust the brightness, e.g. to precisely half the range. The digital value then is at 0111 or 1000. Flickering artificial lighting may cause the result to switch between two levels.

## 5 Random Generator

Use a jumper E3 against GND to start a basic programme for random generator. The button S1 is evaluated here. The associated input has an internal pull-up resistor that pulls the voltage to VCC-level. The button is connected against ground. Pushing the button pulls the input S1 to zero.

Fig. 11: Start of the random switch

Fig. 12: Jumper between E3 and GND

The programme uses a conditional jump command. When the S1 input condition is one, the following command is skipped. Pushing the button makes the condition zero, and thus increases

the variable A. This leads to quick counting up of the starting condition. When releasing, the last counter reading is retained. Due to the high counting speed, you have no influence on the result, which therefore is random.

Address	Command	Data	Comment
30	5	4	Port = A
31	C	E	S1 = 1?
32	7	1	A = A + 1
33	3	3	Jump -3

Listing 4: Random generator

Briefly push the button to reactivate a new random result. Test the random function by generating a statistic of the results. After a sufficient number of runs, it should be clear that all results are about equally frequent. The programme is also suitable as a game where you need to try to get, e.g., 1111.

At the same time, the programme is a counter with the maximum possible working speed because no waiting command is used. Therefore, you can use this example to examine the working speed of the TPS controller. While the button is pushed, the output A1 shows a rectangular signal with a frequency of approx. 133 Hz and a period duration of 7.5 ms. The port changes its condition after 3.75 ms each. The programme goes through four commands in the counting loop. About one millisecond is required per command. The last output A4 shows a frequency of 16.6 Hz, which is still visible as a flicker.

If time-critical tasks require higher working speeds, you can increase the cycle speed of the controller by reducing the resistance at Osc1. 100 kΩ provides a cycle rate of 2 MHz. Replace the resistor with a 27 kΩ one. This will lead to an almost four times higher cycle rate and a command time of approx. 0.25 ms. In the usual case, however, the controller should run at 100 kΩ at Osc1. Thus, a low current intake and safe work are also ensured at a low operating voltage down to 2.2 V.

## 6 Impulse Length Measurement

E4 at GND starts an example programme to measure an impulse length after a reset. Again, the condition at the input S1 is evaluated.

Fig. 13: E4 at GND

Fig. 14: Start of impulse length measurement

A time measurement runs in condition  $S1 = 0$ , i.e. with the button pushed. Another approx. 5 ms are added to the waiting time of 5 ms for execution of a total of five commands in the counting loop. Therefore, the time unit of the measurement is 10 ms.

Address	Command	Data	Comment
34	2	2	Wait for 5 ms
35	C	C	$S1 = 0?$
36	3	2	Jump -2
37	4	0	$A = 0$
38	2	2	Wait for 5 ms
39	7	1	$A = A + 1$
3A	5	4	Port = A
3B	C	E	$S1 = 1?$
3C	3	4	Jump -4
3D	3	9	Jump -9

Listing 5: Time measurement

Push the button  $S1$  as briefly as possible. You will receive, e.g. the result 1010, i.e. decimal 10. Since the time unit of the programme is 10 ms, the display means 100 ms. With a little training, you can reach short times down to 50 ms.

## 7 Reading Programmes

For programming, you need the buttons  $S1$  (data input, left) and  $S2$  (programming, right). The reset button is required as well. These buttons alone can be used to read programmes and enter any programmes. With a little practice, you can enter new programmes very quickly and modify present ones.

To enter programming mode, perform a reset with the programming button  $S2$  pushed, and release  $S2$  only about half a second after the reset. Now you can use the button  $S2$  to scroll through the present programme and view the commands and data. Each address requires two button pushes at  $S2$  for this. This switches between display of the command and the data. The current address is also briefly displayed.

Fig. 15: S1 and S2 for programming mode

Fig. 16: Three buttons and LED displays

- First push of a button S2
- Display address (lower four bits), 300 ms

- Display off, 300 ms
- Show command
- Second push of button S2
- Show data
- Third push of button S2
- Display next address, 300 ms
- etc.

If you want to view only a present programme with five steps, for example, without changing it, a total of ten pushes of S2 takes you to the end. Because the current address is displayed briefly, orientation is easy. You will always know if the display is currently showing a command or data. In the delivery condition, the following commands are found in the first five addresses. This is the start of the selection programme to start the individual example programmes.

Address	Command	Data	Comment
00	6	4	A = Din
01	5	1	B = A
02	4	E	A = 14
03	8	0	AdrHi = 0
04	C	3	A = B?

Listing 6: Programme code in the basic condition

A 4-bit command and the associated 4-bit data together form one byte, i.e. an 8-bit number. A half byte is also called a "Nibble". The higher-value nibble forms the command, the lower-value nibble the associated data. The EEPROM of the controller includes a total of 128 bytes. Therefore, a programme can hold up to 128 commands. This is enough for most applications, since the programme code is extremely compact. Many useful programmes can get by with fewer than ten commands.

Display the individual commands and data and compare the content of the memory. Then push the reset button again. The old programme will start unchanged.

## 8 Entering Programmes

The button S1 is used only if a command or its data are to be changed or entered anew. Generally, only numbers between 0 and 15 can be entered. With the first push of S1, 0 is set. Every following push of a button increases the number by 1. The current status is displayed binarily with the four LEDs. If you want to enter 4, push S1 five times: 0, 1, 2, 3, 4. The binary display then is 0100.

Once you have entered either the command, the data or both in this manner, the second push of S2 programmes this byte into the EEPROM. To clarify this, the LED display for 600 ms is switched off before the next address and the next command is displayed. This brief pause is to be intuitively understood as programming. You can imagine that the system saves energy for

display and uses it for programming the EEPROM. You know this from the car: When you activate the starter, the light and radio will go out for a brief moment.

You can change the already-present programme in a single location as well. Use S2 to scroll to the desired location and change the command or data with S1, to save them with S2.

For the first part, a programme with only two commands is entered. It switches on three LEDs and leads into an endless loop.

Address	Command	Data	Comment
00	1	7	A1-4 = 0111
01	3	0	Jump 0

Listing 7: Switching on LEDs

Instead of a detailed listing, you can choose an abbreviated form as well. The two bytes are summarised in hex numbers: 17h, 30h. Below, the hexadecimal way of writing is used. The programmes are therefore written in the short version without the hex mark: 17 30

You need to type the following for input:

S2 + Reset  
2 x S1  
S2  
8 x S1  
S2  
4 x S1  
S2  
1 x S1  
S2

If you have pushed the button S1 once too often by accident, you can still reach the right number. Go to above 15, which is followed by the value 0.

After complete input, the new programme is started again with the reset button. You can see that three LEDs are connected. Nothing else happens. The controller will no longer react to the conditions at the inputs E1 to E4 anymore, since the original programme has been partially overwritten. The example programmes cannot be started anymore either.

Since you have only changed the first two memory addresses, you can easily start the original programme again. For this, just enter the first two commands (64 51) anew according to the listing from the last section.

Test the original function of the example programmes. Best enter the new exercise programme again. You will quickly gain security in handling the programme input.

## 9 Recovery of the Example Programmes

If you want to restore the controller's original condition after some time, this can be done by entering two bytes FF. In fact, this corresponds to the condition of the unwritten EEPROM. The firmware of the TPS controller contains a start function that initially reviews the first two addresses to recognise an empty memory. If two FF-bytes are read here, the controller assumes that no programme has been entered yet. In this case, the EEPROM is automatically filled with the example software. This function actually serves to apply the controller with the example programme in the EEPROM at the first start. However, it can be used at any time to restore factory settings.

Address	Command	Data	Comment
00	F	F	-
01	F	F	-

Listing 8: Return to the factory condition

Start programming mode by resetting with the S2 button pushed. Enter the value F (decimal 15) four times then, where all LEDs A1 to A4 are on. Complete the last input with S2 as well.

Then push reset. The controller needs just a moment longer than usual to reprogramme all bytes from the example programmes. This restores the original condition. Test, e.g., the alternating flash from page 8 without the jumper at the inputs.

## 10 TPS Basic Commands

The key-programmable control knows a total of 14 commands (1–14). Many of these commands have a parameter in the form of a 4-bit number 0000 to 1111 (0–F), i.e. with a number range to 15 (decimal). Other commands know subfunctions that are indicated as a parameter. A command code may therefore contain up to 16 subcommands. For example, command 7 means "Compute A = ...". The parameter indicates which computation function is to be performed.

Below, the commands and data are written together in hexadecimal code as one byte. Command 1 together with parameter 4 thus becomes the command 14h. The hex sign is left out because all commands and addresses are generally written in hexadecimal code.

The first three commands are:

10–1F: Direct port output at A1–A4, 0–15, binary 0000 to 1111

20–2F: Waiting time 0–15

(1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000, 5,000, 10,000, 20,000, 30,000, 60,000 ms)

30–3F: Jump back 0–15

Command 1 is used for port output of a constant number. This permits outputting any bit pattern and e.g switching on several LEDs at the same time.

The waiting command 2 uses a parameter that contains the time in milliseconds and a graduation of 1-2-5. In spite of the low number scope from 0 to 15, this permits execution of delay times between one millisecond and one minute. You would have to wait even longer if you were to programme execution of the waiting command several times, e.g. in a counting loop.

The return command 3 is particularly simple and is sufficient for any tasks where a process should be repeated endlessly. The jump step is limited to the area up to 15. Since the jump width is relative to the current address, programme parts can be moved to any other address with this jump back.

The alternating flash programme only needs these three commands. It is written into the address area from 00 here in a slightly modified form. The output bit samples and waiting times are changed as well.

Address	Command	Data	Comment
00	1	1	A1-4 = 0001
01	2	7	Wait for 200 ms
02	1	4	A1-4 = 0100
03	2	7	Wait for 200 ms
04	3	4	Jump -4

Listing 9: Flashing programme

In short hexadecimal form, the programme now looks as follows:

11 27 14 27 34

Based on these first three commands, many simple programmes can be written. Analyse and test the three following programmes. The objective should be to be able to intuitively use these commands. Simple programme courses like this can even be programmed by heart and entered directly with a little practise. One example of this is a simple running light with four output patterns:

Address	Command	Data	Comment
00	1	1	LEDs 0001
01	2	8	Wait for 500 ms
02	1	2	LEDs 0010
03	2	8	Wait for 500 ms
04	1	4	LEDs 0100
05	2	8	Wait for 500 ms
06	1	8	LEDs 1000
07	2	8	Wait for 500 ms

Address	Command	Data	Comment
08	3	8	Jump -8

11 28 12 28 14 28 18 28 38

Listing 10: Running light 1

Expand the programme with two more output patterns, so that the light point runs back and forth. Experiment with other output patterns and delay times as well.

Address	Command	Data	Comment
00	1	1	LEDs 0001
01	2	8	Wait for 500 ms
02	1	2	LEDs 0010
03	2	8	Wait for 500 ms
04	1	4	LEDs 0100
05	2	8	Wait for 500 ms
06	1	8	LEDs 1000
07	2	8	Wait for 500 ms
08	1	4	LEDs 0100
09	2	8	Wait for 500 ms
0A	1	2	LEDs 0010
0B	2	8	Wait for 500 ms
0C	3	C	Jump -12

11 28 12 28 14 28 18 28 14 28 12 28 3C

Listing 11: Running light 2, back and forth

A time switch may contain a delay of up to one minute with a waiting command. At the end, there is a jump back with the jump width 0, i.e. an endless loop without content, which serves as the end of the programme. Restart is triggered with the reset button. Expand the programme into a three-minute kitchen timer. You can display the remaining time by the number of lit LEDs as a light bar.

Address	Command	Data	Comment
00	1	F	LEDs 1111
01	2	F	Wait for 1 min
02	1	0	LEDs 0000
03	3	0	End

1F 2F 10 30

Listing 12: Time switch for one minute

## 11 Computing with Variables

Until now, the parameters have used only consistent number values in the parameters of the individual commands. This is sensible when a programme is to run the same way every time. More complex programmes, however, work with variable data. E.g. a calculation such as  $A = A + B$  can be executed. Depending on the content of the variables A and B, something different will result each time. The result could control the LEDs at the outputs as follows, for example.

The control has the four variables A, B, C and D. The most important variable is A. It is also called the *accumulator* or briefly *Accu*. A is involved in all computing operations and receives the computation result. A is also used for data transport. B is mostly needed for computing operations. C and D can be used as interim memories and are later needed as counters for counting loops.

There also are two analogue inputs (AD1 and AD2) and a PWM-output. The processed data are limited to four bit and only accessible via variable A ( $A = AD1$ ,  $PWM = A$ ). Accumulator A can also be loaded directly with a number (commands 40–4F). To fill B, C or D, first load A and then assign the content to the other variable (commands 51–53). A and B permit some computing steps (commands 71–7A).

The commands 40–4F assign A a new value. The command group 51–5A transfers the content of A to a target such as another variable or the PWM output. In this group, there also are commands that set a single bit of the output port.

The other data direction is present in the command group 61–6A, where data from a source are read into A. The command group 71–7A finally performs a few computing operations with the result generally appearing in A. The output port Dout comprises the four outputs A1 to A4, which can be tripped either together or as individual bits Dout.0 to Dout.3. The inputs E1 to E4 are triggered equally as input port Din.

40–4F:  $A = 0-15$

51–5A: Target 1–9 = A

51:  $B = A$

52:  $C = A$

53:  $D = A$

54:  $Dout = A$

55:  $Dout.0 = A.0$

56:  $Dout.1 = A.0$

57:  $Dout.2 = A.0$

58:  $Dout.3 = A.0$

59:  $PWM = A$

61–6A:  $A = \text{Source } 1-10$

61:  $A = B$

62:  $A = C$

63:  $A = D$

64:  $A = Din$

65: A = Din.0  
 66: A = Din.1  
 67: A = Din.2  
 68: A = Din.3  
 69: A = AD1  
 6A: A = AD2

71 –7A: A = Expression 1–10

71: A = A + 1  
 72: A = A – 1  
 73: A = A + B  
 74: A = A – B  
 75: A = A \* B  
 76: A = A / B  
 77: A = A And B  
 78: A = A Or B  
 79: A = A Xor B  
 7A: A = Not A

One example of using the variable A is found in the programme examples in chapter 3. The programme was set at the address Zero here and slightly expanded. Additionally, there is a defined start with the value 0 in the variable A. Address 01 contains a computing command, here increase by 1. The content of the variables A is then handed over to the PWM-output and the output port.

Address	Command	Data	Comment
00	4	0	A = 0
01	7	1	A = A + 1
02	5	4	Port = A
03	5	9	PWM = A
04	2	6	Wait for 100 ms
05	3	4	Jump –4

40 71 54 59 26 34

Listing 13: Increase by 1

Another example has already been shown in chapter 4. The data come from the analogue input AD1 and are transferred to the output port and the PWM output. The modified programme contains an additional computing step, i.e. inversion of the content of the variable A. This way turns the value 0000 into the new value 1111, i.e. 0 becomes 15 and vice versa. The rising input voltage thus leads to a reducing PWM-output.

Address	Command	Data	Comment
00	6	9	A = AD1
01	5	4	Port = A
02	7	A	A = Not A
03	5	9	PWM = A
04	2	6	Wait for 100 ms
05	3	5	Jump -5

69 54 7A 59 26 35

Listing 14: Invert

## 12 Jumps and Branches

Up to now, there was only a simple jump back (command 3) that went back up to 15 addresses. Now we add an absolute jump. Since the jump target can only be indicated with 4 bit, there is an additional command that specifies the high nibble of the address. This gives you an address space of 0–255. This is more than you need, since the EEPROM of the controller only holds 128 bytes, i.e. the area 00 to 7F (decimal 0 to 127). The memory is thus virtually divided into eight pages, pages 0 to 7. The page of the jump target must be specified before an absolute jump.

Two counting loops with the variables C and D also perform absolute jumps, with the page of the address having to be specified before here as well.

The conditional jumps work as skip commands. When the respective condition is true, an address is skipped. There, e.g. a jump command or computing command might be written. The conditions can be comparisons between A and B or direct bit queries of the input port.

There also is a subprogramme call and the associated return command. While several subprogrammes are permitted, a subprogramme must not call another subprogramme, because the interpreter only remembers the return address at a time.

80–8F: Adr-high = 0–15

90–0F: Direct jump to Adr-high, Adr-low (0–15)

A0–AF: Counting loop C-times Adr-high, Adr-low (0–15)

B0–BF: Counting loop D-times Adr-high, Adr-low (0–15)

C1–CF: Conditional jump: if (condition 1–15) then skip

C1: if A > B then Adr = Adr + 1

C2: if A > B then Adr = Adr + 1

C3: if A = B then Adr = Adr + 1

C4: if Din.0 = 1 then Adr = Adr + 1

C5: if Din.1 = 1 then Adr = Adr + 1

C6: if Din.2 = 1 then Adr = Adr + 1  
 C7: if Din.3 = 1 then Adr = Adr + 1  
 C8: if Din.0 = 0 then Adr = Adr + 1  
 C9: if Din.1 = 0 then Adr = Adr + 1  
 CA: if Din.2 = 0 then Adr = Adr + 1  
 CB: if Din.3 = 0 then Adr = Adr + 1  
 CC: if S1 = 0 then Adr = Adr + 1  
 CD: if S2 = 0 then Adr = Adr + 1  
 CE: if S1 = 1 then Adr = Adr + 1  
 CF: if S2 = 1 then Adr = Adr + 1

D0–DF: Subprogramme call Adr-high, Adr-low (0-15)

E0–EF: Return from subprogramme

One example of using conditional jump commands is found in the example programme in chapter 6. Here, it has been slightly modified and put in address 0. Since the upper part of the address (Adr-hi) is in the quiescent condition 0, and the controller starts on page 0, the command 80 does not need to be used here. The length of a push of a button is measured and displayed again. All waiting commands have been removed from the programme so that it now works with a higher resolution.

Address	Command	Data	Comment
00	C	C	S1 = 0?
01	3	1	Jump -1
02	4	0	A = 0
03	7	1	A = A + 1
04	5	4	Port = A
05	C	E	S1 = 1?
06	3	3	Jump -3
07	3	7	Jump -7

CC 31 40 71 54 CE 33 37

Listing 15: Reactions to button S1

The jump command CC in address 00 evaluates the condition at buttons S1. In the quiescent condition, S1 = 1. The condition therefore is not true and the command in address 01 is not skipped. There is a relative jump command to the start. The programme repeats the commands in addresses 00 and 01 until the button is pushed. The the condition becomes true and address 01 is skipped. This starts the actual measuring process. The accumulator is deleted and then increased by 1 continually and output to the LEDs. Another conditional jump command CE is placed in address 05. Here, the condition for skipping of a command is S1 = 1. Since the button is still pushed at first, the condition is not true. The command in 06 therefore is performed and causes a return to 03. Only when the button is released will the programme get to address 07 and thus a return to the start.

Enter the programme and test it. The reaction time is no much faster. The time unit is at approx. 5 ms.

The original example programme is still in the memory from address 34h onwards, since only the lower addresses have been overwritten. Write a little programme that contains only a jump to this address. You first need to indicate page 3. The following absolute jump with the specified address 4 then actually targets the address 34.

Address	Command	Data	Comment
00	8	3	Page 3
01	9	4	Address = 34

83 94

Listing 16: Absolute jump to the time measuring programme

The original example programme is thus called again. Test this also for other examples. For a complete overview of all possible programmes, see the Appendix.

### 13 Command Overview

All commands at a glance – this simplifies work with the controller. The following table contains the entire command stock in a compact form.

	1	2	3	4	5	6	7	8	9	A	B	C	D	E
	Port=	Wait	Jump	A=	... = A	A = ...	A = ...	Page	Jump	C*	D*	Skip if ...	Call	Ret
0	0	1 ms	0	0				0	0	0	0		0	
1	1	2 ms	1	1	B = A	A = B	A = A + 1	1	1	1	1	A > B	1	
2	2	5 ms	2	2	C = A	A = C	A = A - 1	2	2	2	2	A < B	2	
3	3	10 ms	3	3	D = A	A = D	A = A + B	3	3	3	3	A = B	3	
4	4	20 ms	4	4	Dout = A	A = Din	A = A - B	4	4	4	4	Din.0 = 1	4	
5	5	50 ms	5	5	Dout.0 = A.0	A = Din.0	A = A * B	5	5	5	5	Din.1 = 1	5	
6	6	100 ms	6	6	Dout.1 = A.0	A = Din.1	A = A / B	6	6	6	6	Din.2 = 1	6	
7	7	200 ms	7	7	Dout.2 = A.0	A = Din.2	A = A And B	7	7	7	7	Din.3 = 1	7	
8	8	500 ms	8	8	Dout.3 = A.0	A = Din.3	A = A Or B		8	8	8	Din.0 = 0	8	

9	9	1 s	9	9	PWM = A	A = AD1	A = A Xor B		9	9	9	Din.1 = 0	9	
---	---	-----	---	---	------------	------------	----------------	--	---	---	---	-----------	---	--

	1	2	3	4	5	6	7	8	9	A	B	C	D	E
A	10	2 s	10	10		A = AD2	A = Not A		A	A	A	Din.2 = 0	A	
B	11	5 s	11	11					B	B	B	Din.3 = 0	B	
C	12	10 s	12	12					C	C	C	S1 = 0	C	
D	13	20 s	13	13					D	D	D	S2 = 0	D	
E	14	30 s	14	14					E	E	E	S1 = 1	E	
F	15	60 s	15	15					F	F	F	S2 = 1	F	

## 14 Counting Loops

A process is to be performed, e.g. five times. For this, a count loop is formed. A jump command is performed five times in this case, and then no longer. The count variable is called C. The count value 5 must be loaded in A first, and from there in C. The command A2 performs an absolute count to 02 and at the same time reduces the content of the variable C by 1. When C has reached the value 0, the jump is no longer performed. The absolute jump address refers to the indicated page. For a programme on page 0, the page command 80 may also be left out. However, it is necessary when jumping to any other page.

Address	Command	Data	Comment
00	4	5	A = 5
01	5	2	C = A
02	1	5	Port = 0101
03	2	8	500 ms
04	1	A	Port = 1010
05	2	8	500 ms
06	8	0	Page 0
07	A	2	C-times 02
08	3	0	End

45 52 15 28 1A 28 80 A2 30

Listing 17: A count loop

Test the programme. The LEDs show the patterns 0101 and 1010 at each pass. However, this programme part is obviously not gone through five times but precisely six times. The jump command in address 07 is performed precisely five times, but to get to this point for the first time, a flashing process is performed. Therefore, the programme will flash six times in total.

Change the count variable to value 4 and test the programme again. Now the LEDs will flash precisely five times.

You can also use the count loop so that you will not jump back, but forwards. This time, the process is actually performed five times when C has been loaded with the value 5 in the beginning. The skipped address 04 contains a relative jump to itself and thus an endless loop that serves as the programme end.

Address	Command	Data	Comment
00	4	5	A = 5
01	5	2	C = A
02	8	0	AdrHi = 0
03	A	5	C-times 05
04	3	0	End
05	1	5	Port = 0101
06	2	8	Wait for 500 ms
07	1	A	Port = 1010
08	2	8	Wait for 500 ms
09	3	6	Jump -6

45 52 80 A5 30 15 28 1A 28 36

Listing 18: Five flashes

## 15 Comparisons

Two number values are to be compared. Depending on the result of the comparison, a jump is performed. The two number values must be in A and B. The following example loads B with the number 5. A receives its result from the analogue input AD1. Here, e.g. a light sensor can be connected as in chapter 4. The programme should now perform the following action continually:

```
If AD1 > 5
then: all LEDs on
else: all LEDs off
```

All in all, you will receive a twilight switch. Since LDR is connected against GND, more brightness causes a lower voltage at AD1. The LEDs will go out once a certain brightness is exceeded and a certain voltage therefore undercut. The limit is at 6, since the measuring result must be above 5.

Address	Command	Data	Comment
00	4	5	A = 5
01	5	1	B = A

02	8	0	AdrHi = 0
----	---	---	-----------

Address	Command	Data	Comment
03	6	9	A = AD1
04	C	1	Skip if A>B
05	9	8	Adr 08
06	1	F	LEDs 1111
07	3	4	Adr 03
08	1	0	LEDs 0000
09	3	6	Adr 03

45 51 80 69 C1 98 1F 34 10 36

Listing 19: Simple twilight switch

Test the programme by shading the light sensor with your hand more or less. You will find that the basic function is met. However, there is usually an unpleasant side effect. Exactly at the threshold between On and Off, the LEDs will flash uncontrolledly. Especially in artificial light, the brightness will fluctuate around a certain average quickly. This flickering is correctly evaluated by the programme, but the result is not what you would expect of a twilight switch. Chapter 18 shows an improved twilight switch.

## 16 AND, OR and XOR

Two binary conditions can be linked into a new condition. One example is the AND function: When bit 1 has condition 1 AND bit 2 has condition 1, the output condition is also 1. Binary numbers with several bits can also be linked in this manner. The link "10 AND 3 = 2" becomes understandable when writing it in binary numbers:

```
1010 AND
0011 =
0010
```

The following programme links the input conditions to the constant number 3. The AND function practically causes the two lower bits to be masked (filtered out). In the quiescent condition, the input port has condition 1111. The AND link with 0011 then delivers the condition 0011 at the LED. If you connect one of the inputs E1 or E2 to GND, however, the 0 condition is also visible at the other outputs. Changes to E3 and E4 have no effects.

Address	Command	Data	Comment
00	6	4	A = Din
01	5	1	B = A
02	4	3	A = 3

Address	Command	Data	Comment
03	7	7	A = A And B
04	5	4	Port = A
05	3	5	Jump -5

64 51 43 77 54 35

Listing 20:Application of the AND function

Change the programme and test other logical functions as well. The OR function (78) can be used to generally set specific input conditions to 1: 64 51 43 78 54 35

1010 OR

0011 =

1011

Use the XOR function (exclusive-or, 79) to invert individual bits: 64 51 43 79 54 35

1010 XOR

0011 =

1001

## 17 Subprogrammes

When parts of a programme are to be reused, write them into a subprogramme. This often saves memory space, and sometimes also a lot of typing. The following example shows use of a subprogramme that is called in two places in the main programme. The subprogramme only contains one instruction ( $A = A-1$ ) and the return jump command here. This does not save memory, but the example is only meant to demonstrate the CALL and RET commands.

Main programme:

Address	Command	Data	Comment
00	8	0	AdrHi = 0
01	D	8	Call 08
02	5	4	Output
03	2	9	Waiting 1 s
04	D	8	Call 08
05	5	4	Output
06	2	8	Waiting 0.5 s
07	3	7	Jump -7

Subprogramme:

Address	Command	Data	Comment
08	7	2	A = A-1
09	E	0	Ret

80 D8 54 29 D8 54 28 37 72 E0

Listing 21: Subprogramme calls

The result of the programme is a downwards counting binary counter with uneven time delays. Also test other commands in the subprogramme.

There are several useful subprogrammes for general use among the example programmes in the delivery condition. They are listed completely in the Appendix. Only the entrance address needs to be known for using them:

50: Subprogramme: Long sound

52: Subprogramme: Short sound

53: Subprogramme: Any sound, length in A

60: Subprogramme: Wait for pushed button S1

68: Subprogramme: Wait for pushed button S2

70: Subprogramme: Number input with S1 and S2

The subprogramme from address 60 onwards is only used to set up a counter that is controlled via button S1. The counter reading starts at 0. The main programme is relatively short because the complex task of the button queries has been outsourced into the subprogramme.

Address	Command	Data	Comment
00	4	0	A = 0
01	5	4	Output
02	7	1	A = A+1
03	8	6	Page 6
04	D	0	Call 60, button S1
05	3	4	Jump -4

40 54 71 86 D0 34

Listing 22: Counter controlled via S1

Test the programme. If you push S1 ten times, the result should be 1010. Change the programme so that the subprogramme is used from address 68. Now the counter reacts to S2.

## 18 Twilight Switch

A twilight switch is to switch on the lamp when the ambient brightness drops below a certain limit. When the light grows brighter, the lamp is to go out again. It should be ensured that the light does not flicker on the threshold between light and dark. This is possible with a hysteresis, i.e. a certain distance between the activation and deactivation brightness. The programme presented here works with the following rules:

- If the voltage at AD1 is not above 5, the lamp is switched off.
- If the voltage at AD1 is not below 9, the lamp is switched on.

This provides a middle area in which the output condition cannot change. This gap prevents flickering of the LEDs.

0–5: LEDs off

6–8: LEDs unchanged

9–15: LEDs on

Address	Command	Data	Comment
00	1	0	LEDs 0000
01	4	5	A = 5
02	5	1	B = A
03	6	9	A = AD1
04	C	1	Skip if A>B
05	1	0	LEDs 0000
06	4	9	A = 9
07	5	1	B = A
08	6	9	A = AD1
09	C	2	Skip if A<B
0A	1	F	LEDs 1111
0B	3	A	Jump -10

10 45 51 69 C1 10 49 51 69 C2 1F 3A

Listing 23: Twilight switch with hysteresis

## 19 LED-Dimmer

The target of this programming example is a controllable LED lamp. The brightness of an LED at the PWM output is to be adjustable via buttons. You can briefly push a button to get to the next brightness stage, or you can keep pushing to change the brightness continually.

In the core of the programme, the skip commands that are already known are used. If the respective button is not pushed, the associated command to increase or reduce the accumulator content is skipped. The problem is that this can usually lead to an overrun from 15 to 0 or from 0 to 15. Preventing this overrun requires somewhat more effort. For this, you have to query whether the lower end (0) or the upper end (15) has already been reached. Since the accumulator is always involved in a comparison, its content must be put in interim memory. This is used by setting variable C.

Address	Command	Data	Comment
00	8	0	AdrHi = 0
01	5	9	PWM = A
02	2	7	200 ms
03	5	2	C = A
04	4	F	A = 15
05	5	1	B = A
06	6	2	A = C
07	C	2	Skip if A<B
08	9	B	Jump 0B
09	C	F	Skip if S2 = 1
0A	7	1	A = A + 1
0B	5	2	C = A
0C	4	0	A = 0
0D	5	1	B = A
0E	6	2	A = C
0F	C	1	Skip if A>B
10	9	0	Jump 00
11	C	E	Skip if S1 = 1
12	7	2	A = A - 1
15	9	0	Jump 00

80 59 27 52 4F 51 62 C2 9B CF 71 52 40 51 62 C1 90 CE 72 90

Listing 24: Brightness control

## 20 Number lock

The number lock presented here switches on the PWM output if the user enters the correct number sequence. The number input should be precisely according to the pattern of programming via buttons S1 and S2. The following programme demonstrates input of a single number via the button S1. As when programming, the first push of the button leads to result 0000. Any subsequent push of S1 increases output by 1. Pushing S2 ends the input. In this case, the programme will end in an endless loop.

Address	Command	Data	Comment
00	C	C	S1 = 0?
01	3	1	Jump -1
02	4	0	A = 0
03	5	4	Dout = A
04	2	3	10 ms
05	C	E	S1 = 1?
06	3	2	Adr 04
07	C	F	S2 = 1?
08	3	0	End
09	C	C	S1 = 0?
0A	3	3	Adr 07
0B	7	1	A = A + 1
0C	2	3	10 ms
0D	C	C	S1 = 1?
0E	3	1	Adr 0D
0F	3	C	Adr 03

CC 31 40 54 23 CE 32 CF 30 CC 33 71 23 CC 31 3C

Listing 25: Input of a number

The number input is also available as a finished subprogramme from address 70 onwards. Instead of the endless loop in line 08, there is a RET command here. The subprogramme is left with the result of the number input in A.

The following number lock calls the number input thrice and compares the results to pre-defined numbers. In this example, the correct input is 3, 5, 2. Then the PWM output is fully activated with value 15. Any wrong input, however, leads to an endless loop that can only be left by a reset.

The PWM output is treated like a normal digital port in this example. This is necessary because all four outputs A1 to A4 are needed for number input. After each complete input, the four LEDs are deleted to give the observer as little information about the secret combination as possible.

Address	Command	Data	Comment
00	8	7	Page 7
01	4	3	A = 3
02	5	1	B = A
03	D	0	call 70
04	C	3	Skip if A=B
05	3	0	End
06	1	0	LEDs off
07	4	5	A = 5
08	5	1	B = A
09	D	0	call 70
0A	C	3	Skip if A=B
0B	3	0	End
0C	1	0	LEDs off
0D	4	2	A = 2
0E	5	1	B = A
0F	D	0	call 70
10	C	3	Skip if A=B
11	3	0	End
12	1	0	LEDs off
13	4	F	A = 15
14	5	9	PWM=A
15	3	0	End

87 43 51 D0 C3 30 10 45 51 D0 C3 30 10 42 51 D0 C3 30 10 4F 59 30

Listing 26: The number lock

## 21 Appendix

Listing of the example programmes

Address	Command	Data	Comment
00	6	4	A = Din
01	5	1	B = A

02	4	E	A = 1110
03	8	0	Page 0
04	C	3	A = B?
05	9	8	Jump 08
06	8	2	Page 2
07	9	5	Jump 25, "Count up"
08	4	D	A = 1101
09	8	0	Page 0
0A	C	3	A = B ?
0B	9	E	Jump 0E
0C	8	2	Page 2
0D	9	A	Jump 2A, "AD/PWM"
0E	4	B	A = 1011
0F	8	1	Page 1

64 51 4E 80 C3 98 82 95 4D 80 C3 9E 82 9A 4B 81

Page 0: Selection and start of the example programmes

Address	Command	Data	Comment
10	C	3	A = B?
11	9	4	Jump 14
12	8	3	Page 3
13	9	0	Jump 30, "Random"
14	4	7	A = 0111
15	8	1	Page 1
16	C	3	A = B?
17	9	A	Jump 1A
18	8	3	Page 3

Address	Command	Data	Comment
19	9	4	Jump 34, "Stop watch S1"
1A	4	3	A = 0011
1B	8	2	Page 2
1C	C	3	A = B?
1D	9	0	Jump 20 "Alternating flash"
1E	8	4	Page 4
1F	9	0	Jump 40, "Stop watch S1/S2"

Address	Command	Data	Comment
20	1	1	0001 "Alternating flash"
21	2	8	Wait for 500 ms
22	1	8	1000
23	2	8	Wait for 500 ms
24	3	4	Jump -4
25	7	1	A = A + 1 "Count up"
26	5	4	Port = A
27	5	9	PWM = A
28	2	6	Wait for 100 ms
29	3	4	Jump -4
2A	6	9	A = AD1 "AD/PWM"
2B	5	4	Port = A
2C	5	9	PWM = A
2D	2	6	Wait for 100 ms
2E	3	4	Jump -4
2F	F	F	-

11 28 18 28 34 71 54 59 26 34 69 54 59 26 34 FF

Address	Command	Data	Comment
30	5	4	Port = A "Random"
31	C	E	S1 = 1?
32	7	1	A = A + 1
33	3	3	Jump -3
34	2	2	Wait 5 ms "Stop watch S1"
35	C	C	S1 = 0?
36	3	2	Jump - 2
37	4	0	A = 0
38	2	2	Wait for 5 ms
39	7	1	A = A + 1
3A	5	4	Port = A
2B	C	E	S1 = 1?

3C	3	4	Jump -4
3D	3	9	Jump -9
3E	F	F	-
3F	F	F	-

54 CE 71 33 22 CC 32 40 22 71 54 CE 34 39 FF FF

Page 3: Example programmes: Random, stop watch S1

Address	Command	Data	Comment
40	8	6	Jump 6, "Stop watch Start/Stop"
41	D	0	Call "Waiting S1"
42	4	0	A = 0
43	7	1	A = A + 1
44	5	4	Port = A
45	2	3	Wait for 10 ms
46	C	D	S2 = 0?
47	3	4	Jump -4
48	D	8	Call "Waiting S2"
49	4	0	A = 0
4A	5	4	Port = A
4B	3	B	Jump -11

Address	Command	Data	Comment
4C	F	F	-
4D	F	F	-
4E	F	F	-
4F	F	F	-

86 D0 40 71 54 23 CD 34 D8 40 54 3B FF FF FF FF

Page 4: Example programme stop watch start/stop

Address	Command	Data	Comment
50	4	F	A = 15 "Sound long"
51	9	3	Adr 03
52	4	5	A = 5 "Sound short"
53	5	3	D = A "Sound variable"
54	1	9	A4 = 1
55	1	1	A4 = 0

56	2	1	2 ms
57	1	9	A4 = 1
58	1	1	A4 = 0
59	2	1	2 ms
5A	1	9	A4 = 1
5B	1	1	A4 = 0
5C	2	0	1 ms
5D	B	4	D-times 04
5E	1	0	Dout 0
5F	E	0	Return

4F 93 45 53 19 11 21 19 11 21 19 11 20 B4 10 E0

Page 5: Subprogramme sound output

Address	Command	Data	Comment
60	2	3	Wait 10 ms "Wait S1"
61	C	E	S1 = 1?
62	3	2	Jump -2
63	2	3	Wait for 10 ms
64	C	C	S1 = 0?
65	3	1	Jump -1
66	E	0	Return
67	F	F	-
68	2	3	Wait 10 ms "Wait S2"
69	C	F	S2 = 1?
6A	3	2	Jump -2
6B	2	3	Wait for 10 ms
6C	C	D	S2 = 0?
6D	3	1	Jump -1
6E	E	0	Return
6F	F	F	-

23 CE 32 23 CC 31 E0 FF 23 CF 32 23 CD 31 E0 FF

Page 6: Subprogrammes Wait S1 and Wait S2

Address	Command	Data	Comment
70	C	C	S1 = 0? »Button input"
71	3	1	Jump -1

72	4	0	A = 0
73	5	4	Port = A
74	2	3	Wait for 10 ms
75	C	E	S1 = 1?
76	3	2	Jump -2
77	C	F	S2 = 1?
78	E	0	Return
79	C	C	S1 = 0?
7A	3	3	Jump -3
7B	7	1	A = A + 1

Address	Command	Data	Comment
7C	2	3	Wait for 10 ms
7D	C	C	S1 = 1?
7E	3	1	Jump - 1
7F	3	C	Jump -12

CC 31 40 54 23 CE 32 CF 30 CC 33 71 23 CC 31 3C

Page 7: Subprogramme button input

Command table

	1	2	3	4	5	6	7	8	9	A	B	C	D	E
	Port =	Wait	Jump -	A =	... = A	A = ...	A = ...	Page	Jump	C*	D*	Skip if ...	Call	Ret
0	0	1 ms	0	0				0	0	0	0		0	
1	1	2 ms	1	1	B = A	A = B	A = A+1	1	1	1	1	A>B	1	
2	2	5 ms	2	2	C = A	A = C	A = A-1	2	2	2	2	A<B	2	
3	3	10 ms	3	3	D = A	A = D	A = A+B	3	3	3	3	A = B	3	
4	4	20 ms	4	4	Dout = A	A = Din	A = A-B	4	4	4	4	Din.0 = 1	4	
5	5	50 ms	5	5	Dout.0 = A.0	A = Din.0	A = A*B	5	5	5	5	Din.1 = 1	5	
6	6	100 ms	6	6	Dout.1 = A.0	A = Din.1	A = A/B	6	6	6	6	Din.2 = 1	6	
7	7	200 ms	7	7	Dout.2 = A.0	A = Din.2	A = A And B	7	7	7	7	Din.3 = 1	7	
8	8	500 ms	8	8	Dout.3 = A.0	A = Din.3	A = A Or B		8	8	8	Din.0 = 0	8	

9	9	1 s	9	9	PWM = A	A = AD1	A = A Xor B		9	9	9	Din.1 = 0	9	
A	10	2 s	10	10		A = AD2	A = Not A		A	A	A	Din.2 = 0	A	
B	11	5 s	11	11					B	B	B	Din.3 = 0	B	
C	12	10 s	12	12					C	C	C	S1 = 0	C	
D	13	20 s	13	13					D	D	D	S2 = 0	D	
E	14	30 s	14	14					E	E	E	S1 = 1	E	
F	15	60 s	15	15					F	F	F	S2 = 1	F	

**Imprint**

© 2012 Franzis Verlag GmbH, 85540 Haar

[www.elo-web.de](http://www.elo-web.de)

Author: Burkhard Kainka

ISBN 978-3-645-10104-2

Produced at the order of Conrad Electronic SE, Klaus-Conrad-Str. 1, 92240 Hirschau

All rights reserved, even including photomechanical reproduction and storage on electrical media. Generation and distribution of copies on paper, data carriers or online, in particular as PDF, is only permissible with the express consent of the publisher and will be prosecuted under criminal law.

Most product designations of hard- and software, company names and company logos named in this work are usually also registered trademarks and should be observed as such. The publisher essentially follows the manufacturers' spelling for the product designations.

All circuits and programmes presented in this book were developed, tried and tested with the greatest care.

Nevertheless, errors in the book and software cannot be excluded completely. Publisher and author are liable according to the statutory provisions in case of wilful intent or gross negligence. Apart from this, publisher and author shall only be liable according to the Product Liability Act for violation of life, body or health or due to culpable violation of essential contractual obligations. Damages claims for violation of any essential contractual obligations shall be limited to the foreseeable damage typical for the contract except in case of mandatory liability under the product liability act.

Electronic devices must not be disposed of in the domestic waste!  
At the end of its service life, dispose of the product according to the relevant statutory provisions. Collection points have been set up for return. You may dispose of your electrical devices there free of charge. Your community will inform you of where such collection points are located.

This product complies with the relevant CE directives if used according to the included instructions. The description is part of the product and must be included when you pass on the product.