STM32 for beginners

From the Mikrocontroller.net collection of articles, with contributions by various authors (see version history)

Always recurring here in the forum "Which microcontroller should I start with?". There is a large selection and just as many recommendations. This article is intended to help first whether a Cortex-M3 / M4 core is the right one for the start. The Cortex is not recommended for everyone , because the requirements and wishes that one would like to realize, as well as one's own abilities, are different.

The processors AVR , PIC , Arduino , MSP430 , LPC1xxx and STM32 are usually recommended. More rarely also 8051 and M16C. All of them have advantages and disadvantages.

STM32F417 on a do-it-yourself board

table of contents

Own skills and desires

This article assumes that you already have some knowledge of electronics. If not then you should first work through the article " Absolute Beginner ", as well as the other articles from the category " Basics ".
More articles that describe other processors in more detail because this article mainly focuses on STM32 : Decision Microcontroller and Microcontroller Comparison . The pages AVR , MSP430 , LPC1xxx and PIC show more details about this µC. Only a few tables are shown here for comparison purposes.

First of all, the boundary conditions with which you should first assess yourself:

capability	Cortex	AVR	Arduino	PIC			MSP430
Bit width, but unimportant for beginners	32-bit	8-bit		8-bit (PIC18)	16-bit (PIC24)	32-bit (PIC32)	16-bit
Newcomers, hardly any knowledge of electronics, never programmed	O	O [1] [2]	X	O			O
An SD card or graphic display	X	O		O [3]	X [3] [4]	X [3] [4]	X

is required							
The wish is a TCP / IP network	X	O		X [5]			O
The wish is camera and video / image processing	X	-		-	-	X	-
Would like to use the knowledge professionally	X	X	-	X			X
Energy-saving applications	- 300 nA sleep [6] 230 µA / MHz	- 100 nA sleep [7] [8] 340 µA / MHz	-	X 9 nA Sleep [9] [10] 35 µA / MHz		-	X 100 nA Sleep [11] [12] 100 µA / MHz
Multithreading, RTOS, schedulers	X	- [13] [14]	-	- [15] [16]	X [17]	X [17]	X [18]
Particularly large, memory-intensive programs e.g. graphics, fonts	up to 2MB Flash up to 1024kB SRAM [19]	up to 256kB Flash up to 16kB SRAM		up to 128kB Flash up to 4kB SRAM	up to 2MB Flash up to 98kB SRAM	up to 2MB Flash up to 512kB SRAM + 32MB DRAM (PIC32MZ DA)	up to 512KB Flash up to 66kB SRAM
A lot of PWM with complex timing	X	AT90 Spwm	-	-	X	X	O
German speaking community	O	X	X	X [20] [21]	-	-	-
Number of possible HW breakpoints	4 to 6	2		1 to 5 [22]	1 to 10 [22]	6 [22]	2

- X = yes
- O = Partial, restrictions
- - = not recommended

Just so as not to confuse, even if parts are "not recommended", that does not mean that the processor does not work, but rather that it takes more effort to realize it or has more restrictions. Only certain models of a microcontroller family have many properties. However, changing within a family is often easy (e.g. within STM32 or within PIC24).

The Cortex column shows the processor family from STM32F0xx to STM32F4xx with a Cortex-M0 or M3 / M4 core. The technical data is similar to that of other manufacturers who also use a Cortex-Mx core such as NXP (LPC1xxx), Freescale, Atmel, TI, Toshiba, etc. However, with the STM32 , ST offers a high degree of flexibility in terms of housing variations (comparable to NXP LPC1xxx) and is quite easy to obtain privately. Overview of all available STM32 µC from ST [19] .

The STM32Fxxx is in fact not the best in the "Power Saving" category, the newer STM32L0xx and STM32L4xx are significantly more economical, especially at high temperatures. An interesting alternative with a Cortex-Mx core:
"EFM32" from Silabs [23] only needs 0.9 µA in sleep mode.

The PIC column shows the properties of the 8-bit PIC18 (comparable to AVR), 16-bit PIC24 / dsPIC (comparable to MSP430) and 32-bit PIC32 (comparable to STM32). A change of the microcontroller within the families is possible without any problems in terms of code. [24] In terms of hardware technology, different models of the same family are also mostly pin-compatible [25] so that it is possible to switch between different models without changing the design. Switching between families is more complex due to the architecture, especially from 8-bit to 16/32-bit. [26]Due to the constant IDE (Mplab) and constant libraries when using a high-level language (C), however, especially between 16-bit and 32-bit possible without further problems [27] .
A PIC10 / 12/16 is not recommended for beginners, as they have many limitations due to the architecture that are more of a hindrance to learning.

The " Arduino " is not an independent processor, but a finished board (with an AVR or STM32 processor, for example) and an " Arduino " programming environment, specially created for beginners. However, the learning effect is much less because the processor is programmed with the Arduino software and not the registers directly. On the other hand, the Arduino is better suited for someone who doesn't really want to learn, but just wants to control something / tinker with something quickly and thus achieve their goal without great µC knowledge. The Arduino is perfect for this target group.

1. [1] , Robot Network: AVR entry made easy
2. [2] , Tutorial for learning the assembly language of AVR single-chip processors
3. [3] , Microchip Memory Disk Drive File System for PIC18 PIC24 dsPIC PIC32
4. [4] , Microchip Graphics Library
5. [5] , Microchip TCP / IP stack
6. [6] , STM32 L1 series of ultra-low-power MCUs
7. [7] , Atmel picoPower Technology
8. [http: /www.futurlec.com/News/Atmel/PicoPower.shtml], Atmel Releases New picoPower AVR Microcontrollers
9. [8] , The Truth about Power Consumption in PIC® MCUs with XLP Technology vs. TI's MSP430
10. [9] , nanoWatt XLP eXtreme Low Power PIC® Microcontrollers
11. [10] , Texas Instruments: Ultra-Low Power Comparison: MSP430 vs. Microchip XLP
12. [11] , Texas Instruments: Meet MSP430
13. [12] , Atmel AVR freeRTOS port
14. [13] , Femto OS: RTOS for small MCUs like AVR
15. [14] , Microchip PICmicro (PIC18) freeRTOS port

16. [15] , Salvo RTOS
17. [16] , 3rd party RTOS selection guide
18. [17] , TI Wiki: MSP430 Real Time Operating Systems Overview
19. [18] , overview of all available STM32 µC from ST
20. [19] , pic-projekte.de
21. [20] , sprut.de
22. [21] , PIC hardware breakpoints (page 6)
23. [22] , EFM32, the power-saving one with Cortex-Mx core
24. [23] , Microchip: Introduction to the 16-bit PIC24F Microcontroller Family
25. [24] , Microchip: 2007 Product Selector Guide (page 114 ff.)
26. [25] , Microchip: PIC18F to PIC24F Migration: An Overview
27. [26] , article: Practical migration from 8- / 16- to 32-bit PIC

Unimportant boundary conditions

Arguments for or against a µC family are often wrongly expressed, which in practice are rather unimportant at **the beginning** (this can look different for specific applications).

- Power supply 3.3V / 5V (whereby e.g. Atmel XMega has no 5V tolerant inputs)
- 8 (e.g. AVR), 16 (e.g. MSP430) or 32 (e.g. STM32) bit
- Processor core Cortex-M0 / M3 / M4, MIPS, ARM7 / 9 / ..., AVR-RISC, PIC-RISC, 8051, ...
- Interrupt system with more or less features
- Programming language (Assembler, Basic, C, C ++, Pascal)
- Programming environment - is a matter of taste anyway
- Understand assembler (should only be understood theoretically, a program should be written in a high-level language)
- DIL housing - suitable for breadboard (STM32 processors are also available ready-soldered on a breadboard-compatible board)
- Programming adapter - as long as it can debug
- 90% of the time a small processor (AVR / PIC) is sufficient , on the other hand, STM32 models are not significantly more expensive.

costs

Rough estimate of what all this will cost. Only a few examples are shown here (individual prices for sources in Germany).

board	STM32	AVR	PIC18 / 24/32	MSP430
Demo board	Nucleo64 Boards [1] and Stm32 Discovery Boards [2]	Arduino Uno clone from 9 €	Microchip demo board [3] from 18 € (for 16 and 32 bit	MSP430 demo board [4] ~ 35 €

	from 15 € (incl. Debugger)		also with integrated debugger)	
Board-compatible board	Blue Pill Board [5] from 1.50 €, S64DIL-405 [6] with STM32F405 30 €	Arduino Nano clone from € 2	PIC micro stick	
Single chip (single unit prices)	1..15 € only SMD (TSSOP..LQFP..BGA)	€ 0.6..15 SMD + DIP	€ 0.5..15 SMD + DIP	
Programming adapter with debugger	ST-Link clone € 2.50 ST-Link original € 48 Segger J-LINK EDU [7] € 50	Atmel AVR Dragon 40 € Atmel ICE without housing from 99 €	PICkit 3 clone 8 € Microchip PICkit 3 original 90 €	

The demo board should be inexpensive in order to keep possible losses low if it is used incorrectly. If you like the processor, you can always upgrade to a more extensive board (e.g. with a display) later.

A programming adapter with SWD or JTAG protocol is required to debug STM32 microcontrollers. Most ST demo boards already contain an SWD-capable ST-Link adapter.

The Segger J-LINK EDU is not the cheapest programming adapter / debugger, but it has a good reputation and can be used for practically all processors with an ARM core. The manufacturer advertises it as being particularly fast.

1. [27] Nucleo64
2. [28] Stm32 Discovery
3. [29] Microchip demo board
4. [30] MSP430 demo board
5. [31] Blue Pill Board
6. [32] S64DIL-405
7. [33] Segger J-LINK EDU

Programming environments

The programming environments include free, powerful software for all processor families

The CooCox IDE is now considered out of date. The installation instructions for CooCox describe how to get the LED of a Nagel new STM32F4DISCOVERY board to flash within an hour.

The company ST advertised the free IDE System Workbench until the end of 2017 . The TrueStudio has also been free since January 2018 after ST bought it.

The System Workbench and the True Studio are very similar. Both are based on Eclipse, OpenOCD and GCC. Both run on Linux and Windows. True Studio can take over projects from the System Workbench, but it doesn't work the other way around. Both development environments will continue to be supported by ST. Both have the disadvantage of only supporting STM32 controllers.

The following development environments are common in the professional environment: Keil , IAR .
The Segger Enbedded Studio can be used free of charge with restrictions.

documentation

Before deciding on a processor, you should at least scan its documentation and read its errata. Not that you want to create an application and you find out afterwards that exactly this part is so buggy that it cannot be used.

With the STM32 , the documentation has a lot of pages, much more than with an AVR or PIC, and many things are rather briefly described. The structure of the STM32 documentation is described here
.
The documentation for the PICs is structured differently, see the PIC article .

For the STM32 , AVR and PIC microcontrollers there are also many beginner help and tutorials, including German.

At the same time as the documentation, you should also look at the manufacturer's demo examples; this will make a lot of things easier to understand right away.

An overview of the functions can be found in the article: STM32

Working with the STM32

First of all, the differences between the individual processors are not SOOO big. All have inputs / outputs and in order to be able to control an LED by means of a port pin, the pin must first be parameterized for each processor, regardless of whether it is an STM32 or an AVR or an MSP430 .
But with an STM32 you have a few more functions, e.g. switchable pull-up or pull-down resistor and special set / reset registers, which other processors do not have, but which simplify programming. The main difference to the other processors is that the STM32 houses so many peripheral modules that you always have to activate them individually with a clock, because this saves a lot of electricity.

And let's be honest, the discussion about which µc is easier to configure is absolute nonsense. The way is always the same:

- 1. Look at the data sheet to see which registers are required for this function.
- 2. Determine the values that are entered in the registers.
- 3. Write values into the register. It made NO DIFFERENCE whether it was an AVR , 8051/2, ARM , ...

It's true, there are differences. The registers have different names, different addresses, different bit meanings, ... But everything is in the data sheet. And whether I have an AVR or an ARM in front of me for the first time. I need the steps outlined above in all cases.

The interrupt system is also prioritized with an STM32 . This can be used to determine which interrupt is processed with priority. The prioritization is also not rocket science.

The respective manufacturers supply extensive demo codes and libraries for all processors. Most of them are all written in C, so the C programming language should also be used. Especially if you are pursuing professional intentions.

And if you can use an STM32 , then switching to an LPC1xxx (NXP) or another manufacturer is no problem at all (*), because they also offer processors with a Cortex-M3 core and you can program them with the same programming environment. This means that you are not necessarily dependent on the manufacturer. (* each manufacturer installs its own peripherals, which have different functionalities.)

As a hobby hobbyist and professionally, almost all applications can be implemented with an STM32 . It has enough RAM and FLASH memory and also enough speed. Many housing variants to create small to larger devices. Many interfaces are described in the article STM32 .

For a start, there are various tutorials on the Internet, some in German. An overview is here on the STM32 page . (Example: STM32 tutorial in German from Diller Technologies)
It must be said that these are mostly based on the standard ST-Libs. On the one hand, these libs simplify the control / use of the peripheral function, and on the other hand, you have to get to know them first. This is simplified because ALL ST demo code examples are also based on these Lib's and thus the change within the STM32 is significantly simplified.
STM32F4xx Library from ST: "STSW-STM32065 STM32F4 DSP and standard peripherals library" including documentation and demo projects for all CPU functions.

The only disadvantage of the STM32 : You have to read a little more, because a peripheral has a lot more functionality (if you need the module at all). Otherwise, I don't really know a reason why you should n't use an STM32 as a newcomer .

Programming languages

- As with many other controllers, the STM32 mainly uses C and C ++. But also Pascal or Basic is possible without any problems (e.g. from mikroe) You should already know the programming language C, if you have no idea about it then you should first write a console program using a tutorial on a PC so that you get to know it to some extent. There are also articles here in the forum. If the aim is (later) application in industry, C (or C ++) should definitely be chosen due to its widespread use; most of the available libraries are also written in C.
- Assembler should only be understood roughly, details of how an instruction works is unimportant. Even if you want to know the cycles for a function call, the STM32 (Cortex-M3)

offers a cycle counter that can be read out; However, due to the more complex pipeline and cache effects, the runtimes cannot be precisely foreseen. If cycle-precise timing is required, timers should be used - the STM32 has enough of that.

The 32-bit address space, which can address RAM, Flash, I / O registers in a standardized way (in contrast to AVR, for example) is ideal for use by high-level languages; In the case of pointers, there is no need to manage additional information about the type of memory in which they point. The address clearly states this (there is only **one** address 42, and not 2 as for example with the AVR (Flash, RAM & I / O)) and the hardware automatically addresses the correct memory. The options for offset addressing, the barrel shifter, division unit, the FPU (with STM32F4), interrupt model, etc. also favor the generation of more efficient codes. In addition, the STM32 simply have more "raw performance", ie more Flash / RAM memory and higher clock frequencies.

Conclusion : Programming in high-level languages is **more convenient than with 8-bit** , because you just have to worry less about whether the compiler can now efficiently implement a program (you can of course still do it to "squeeze out" even more performance if you want).

Further boundary conditions for the decision

The important point that is often forgotten: the availability. If someone recommends a Renesas M16C or Fujitsu, then you search first and find only a few that you can buy privately.

Or even the support on the Internet - is sparse with some exotic ones. This forum is the best help for AVR , PIC, MSP430 , LPC1xxx and STM32 .

Other processors are already very old or no longer modern because their peripherals sometimes offer quite limited functionality. e.g. many 8051 types or PIC16 (or dsPIC30). They may be good for mini applications, but who would like to deal with a small µC especially for a small application when they already know an STM32 with which they can do everything (and have already written good functions)?

Hence my summary: if you haven't just fallen on your head, you can confidently start with an STM32 . An STM32F4DISCOVERY board costs only approx. 20 EUR - and you don't have to invest more for the first start - if it is too complex there is a lot of help here in the forum. In any case, these 20 EUR are really not a big investment.
If you have professional intentions, you should definitely deal with a second, different processor at a later point in time in order to gain the necessary experience.

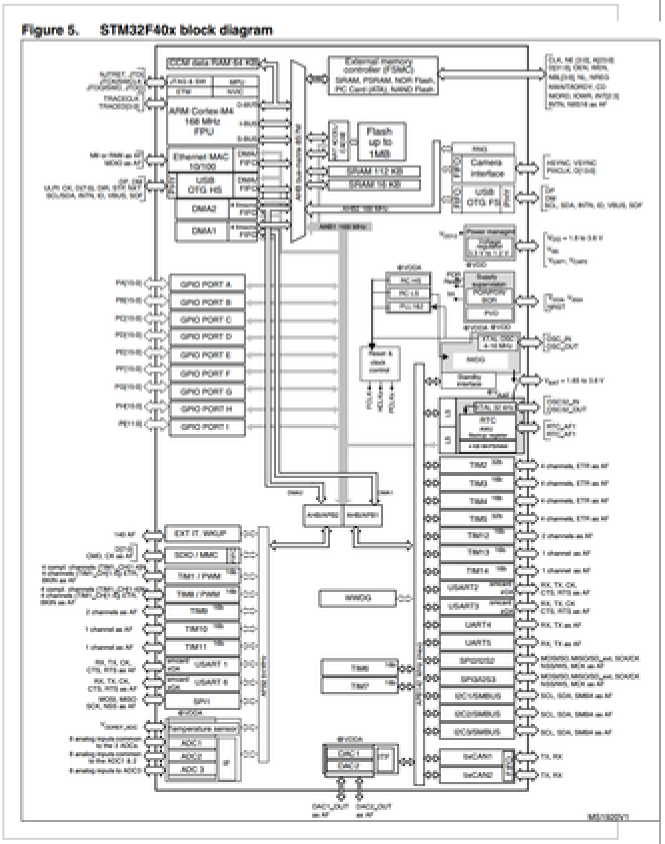
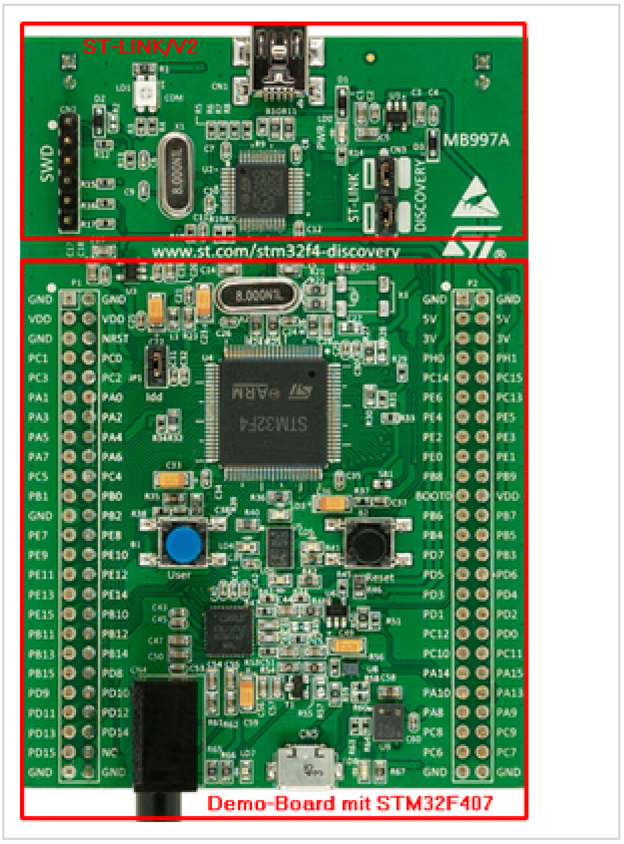
A few forum posts:

- "Start small!" is the motto
- I only use an 8-bitter for "nostalgia reasons"

Overview of CPU functionalities

This section briefly shows which options the CPUs have with the entry-level boards recommended above

STM32F407 - from the STM32F4DISCOVERY board

	
Block diagram of the STM32F407	STM32F4DISCOVERY demo board

There are many peripheral modules implemented in the µC. Each individual peripheral module (e.g. Ethernet, USB, CAN, timer, AD converter, etc.) can be activated by enabling the clock. As long as the clock has not been activated, the module behaves as if it were not there and does not affect processing by the processor in any way. Thus, the power consumption can be reduced.

A complete list of the individual modules can be found in the article: STM32 and on the ST homepage

The internal bus structure may be confusing at first, but you don't need to pay attention to it. If the CPU now wants to access UART4, the data go through the buses "AHB1"> "ABH / APB1-Bridge"> "APB1". The µC does this all by itself. This subdivision is technically necessary because the peripheral buses are not operated with the same processor cycle (as for example with [[MSP430], see diagram below), the "bridge" manages the handling on its own and automatically generates wait commands for the CPU.

The "AHB bus matrix" is also mostly of no interest to the user. ST has thus created a system so that the CPU, DMAs and display controllers can access the various RAM areas at the same time in order to process more data in parallel.

Likewise, you don't need to worry about the processor clock at the beginning, if you don't initialize anything, the STM32F4xx runs with the internal RC oscillator of 16MHz and the peripheral buses also run at the same speed. Only if you later write more extensive applications for which 16MHz is no longer sufficient, you can activate the PLL and flexibly increase the clock rate up to 168MHz. (See demo project of STM32 CooCox installation .)

LPC1x ie Cortex -M0 & -M3 family from NXP

Details can be found in the lead article LPC1xxx and the articles linked therein.

Programming tips and tricks

Interrupt controller from the Cortex-M3 / M4

The "Nested Vectored Interrupt Controller" (NVIC) is a function of the ARMv7M architecture and can be used via the CMSIS.

With the STM32, each interrupt has a priority of 4 bits (with ARMv7M, depending on the implementation, up to 8 bits are possible). These 4 bits can be divided into a "pre-emption priority" and a "subpriority".

Example 2 bit to 2 bit subdivision:

```
NVIC_PriorityGroupConfig ( NVIC_PriorityGroup_2 );  // 2 bits for pre-emption priority
```

This means that "pre-emption priority" can have the value 0..3 and "subpriority" can also have the value 0..3. The smaller the number, the more significant the interrupt.

Example:
ISR1 Pre 2 / Sub 1
ISR2 Pre 1 / Sub 1
ISR3 Pre 1 / Sub 2

If ISR1 comes now, it will be called. If an ISR2 arrives during this time, it may interrupt the ISR1 because it has a higher "pre-emption priority" value.

If ISR2 is already active, ISR3 must not interrupt it, and neither is ISR1.

If ISR2 and ISR3 come at the same time, ISR2 is processed first, then ISR3, since both have the same "pre-emption priority", the decision is made on the basis of the "subpriority".

Example of interrupt configuration for USART1 using CMSIS functions:

```
NVIC_InitTypeDef  NVIC_InitSt ;
NVIC_InitSt . NVIC_IRQChannel  =  USART1_IRQn ;
NVIC_InitSt . NVIC_IRQChannelPreemptionPriority  =  3 ;  // << Depth
priority NVIC_InitSt . NVIC_IRQChannelSubPriority  =  1 ;
NVIC_InitSt . NVIC_IRQChannelCmd  =  ENABLE ;
NVIC_Init ( & NVIC_InitSt );
```

Cycle time calculation and monitoring

In the case of time-critical applications, the question arises again and again of how many processor cycles the function now uses. Is the interrupt really not too long and how much reserve is there? The ARMv7M core has implemented a clock counter in the DWT unit that can be used with the CMSIS:

```
#include  <stdint.h> // For the standard types uint32_t etc.
#include  <stm32f4xx.h> // Use the header file of the CMSIS for the respective
family here.

// Activate DWT unit
inline  void  DWT_Enable ()  {
  CoreDebug -> DEMCR  |=  CoreDebug_DEMCR_TRCENA_Msk ;
}
//
Clock counter - measuring the number of commands from the processor: inline  void
DWT_CycCounterEn  ()  {
  DWT -> CTRL  =  0x40000001 ;
}
inline  void  DWT_CycCounterDis  ()  {
  DWT -> CTRL  =  0x40000000 ;
}
inline  uint32_t  DWT_CycCounterRead  ()  {
  return  DWT -> CYCCNT ;
}
inline  void  DWT_CycCounterClear  ()  {
  DWT -> CYCCNT  =  0 ;
}

int  main  ()  {
  // ... program code ...

  // use
  systick counter DWT_Enable ();  // Activate DWT unit
  DWT_CycCounterEn  ();  // activate counter
  DWT_CycCounterClear  ();  // clear counter
  // ... program
  processing ... uint32_t  iZ  =  DWT_CycCounterRead  ();  // Read out the counter
  // ... further program code ...
}
```

IZ now shows how many machine cycles the processor required for processing and can be converted using the following formula:

Time in µSec = iZ / CPU clock in MHz

FAQ - Beginner Questions

- What do I need for programming / debugging? - All STM32 microcontrollers can be programmed and debugged via the SWD interface. The JTAG interface is also suitable, but requires a little more cables. Suitable adapters can be found, for example, under the name ST-Link or J-Link. All Discovery Boards and Nucleo Boards from ST already contain an ST-Link adapter. With the Nucleo-64 boards, it can even be separated and used individually. All newer STM32 have an internal bootloader so that it can be loaded via SPI, UART or USB. If debugging by print output is enough and a USB interface should be available anyway, you can do without extra programming hardware and only need a jumper on the boot pin.
- I would like to develop my own circuit board, but the 20-pin JTAG connector is too big for me, is there an alternative? - Yes, "The 10-pin JTAG connector from mmvisual"

Left

- STM32 main article, link collection there
- Instructions: STM32 CooCox installation
- Instructions: STM32 Eclipse installation
- Instructions: STM32 instructions and a small book to get you started
- Discussion: Dead time calculator for STM32

Categories :

- POOR
- STM32
- Microcontroller