HARDWARE ACADEMY

The Ultimate Guide to STM32 Microcontrollers

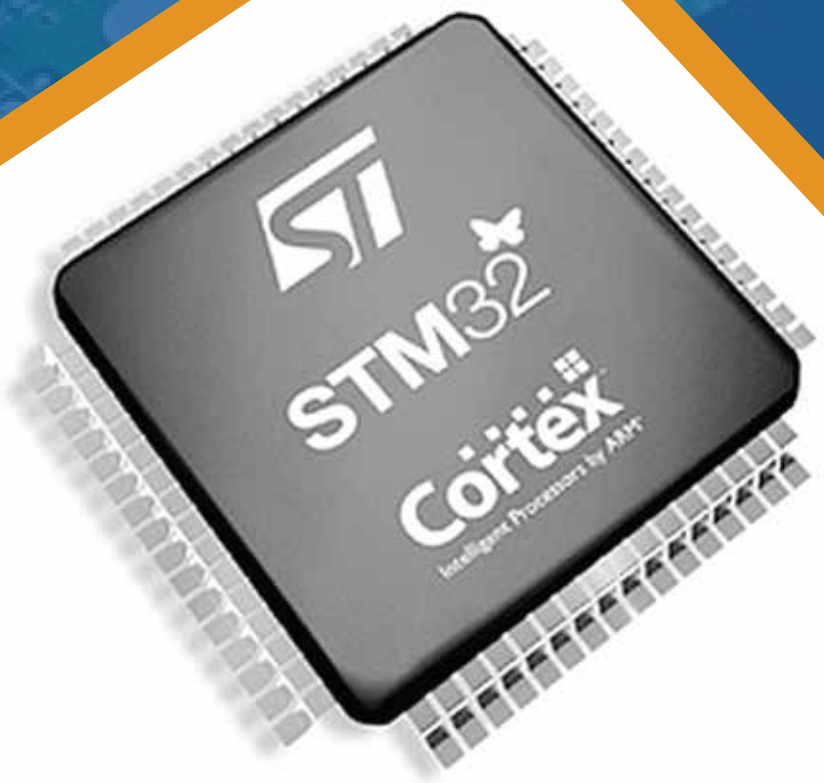


Table of Contents

Introduction 5

PART 1 – Programming the STM32 ..... **6**

Development Tools 7

Developing the first application 8

Building and flashing the code 9

Interrupt system 10

Extended Interrupts and Events Controller (EXTI) 11

External Interrupt and GPIO mapping 12

Programming Conclusion 14

PART 2 – Introduction to the STM32F030 ..... **15**

ARM 32-bit Cortex-M0 CPU, frequency up to 48 MHz 15

Memories 17

16 to 256 Kbytes of Flash memory 17

4 to 32 Kbytes of SRAM with HW parity 17

CRC calculation unit 17

Reset and power management 17

Digital & I/O supply: VDD = 2.4 V to 3.6 V 17

Analog supply: VDDA = VDD to 3.6 V 17

Power-on/Power down reset (POR/PDR) 18

Low power modes: Sleep, Stop, Standby 18

Clock Management 19

4 to 32 MHz crystal oscillator 20

32 kHz oscillator for RTC with calibration 20

Internal 8 MHz RC with x6 PLL option 20

Internal 40 kHz RC oscillator 20

Up to 55 fast I/Os 20

All mappable on external interrupt vectors 21

Up to 55 I/Os with 5V tolerant capability 21

5-channel DMA controller 21

One 12-bit, 1.0 µs ADC (up to 16 channels) 22

Conversion range: 0 to 3.6 V 22
Separate analog supply: 2.4 V to 3.6 V 22
Calendar RTC with alarm and periodic wakeup 22
11 timers 23
One 16-bit advanced-control timer for six-channel PWM output 23
Up to seven 16-bit timers 23
Independent and system watchdog timers 23
SysTick timer 23
Communication interfaces 23
Up to two I2C interfaces 23
Up to six USARTs 24
Up to two SPIs (18 Mbit/s) 24
Serial wire debug (SWD) 25
Packages 25
Conclusion 26

PART 3 – Introduction to the STM32F469 27
Arm 32-bit Cortex-M4 CPU with FPU 27
Adaptive Real-Time Accelerator (ART Accelerator) 28
180 MHz / 225 DMIPS 28
DSP Instructions 29
Memories 30
Flash and RAM Memories 30
External Memory Controller 30
Quad-SPI Interface 30
Graphics 31
Graphical Hardware Accelerator 31
LCD TFT controller 31
MIPI-DSI host controller 32
Analog-to-Digital Converter (ADC) 32
Debug mode 33
Advanced connectivity 33
Inter-IC Sound (I2S) 33
CAN Bus 33

USB 2.0 OTG 33
Ethernet 34
Parallel camera interface 34
Packages 35
Summary 35

PART 4 – Overview of the STM32H7 36
Hardware Development with the STM32H7 39
Application Development for the STM32H7 42
Conclusion 44

PART 5 – Custom STM32 Board Design 45
System / Preliminary Design 45
Block Diagram 45
Select Microcontroller 46
Schematic Circuit Design 48
Capacitors 50
Microcontroller pinout 50
Clock 51
Programming Connector 52
Power 52
Electrical Rules Check 53
Printed Circuit Board (PCB) Design 53
Component Placement 53
PCB Layer Stack 56
Routing 56
Verification 59
Generating Gerbers 60
Summary 60

Introduction

The STM32 family of microcontrollers from STMicroelectronics is based on the ARM Cortex-M 32-bit processor architecture. The STM32 series hassome of the most popular microcontrollers used in a wide variety of products. They also have an excellent support base from multiple microcontroller development forums.
STM32 microcontrollers offer a large number of serial and parallel communication peripherals which can be interfaced with all kinds of electronic components including sensors, displays, cameras, motors, etc. All STM32 variants come with internal Flash and RAM memories.

The range of performance available with the STM32 is quite expansive. Some of the most basic variants include the STM32F0 and STM32F1 sub-series that start with a clock frequency of only 24 MHz, and are available in packages with as low as 16 pins.

At the other performance extreme, the STM32H7 operates at up to 400 MHz, and is available in
packages with as many as 240 pins. The more advanced models are available with Floating Point Units (FPU) for applications with serious numerical processing requirements. These more advanced models blur the line between a microcontroller and a microprocessor.

Finally, the STM32L sub-series is designed specifically for low-power portable applications running from a small battery.

Development Tools

Development tools are required to develop the code, program the microcontroller and test/debug the code. The development tools include:

- Compiler
- Debugger
- In-Circuit Serial Programmer (ICSP)

There are several software development tools available for code development on STM32 microcontrollers. The software tools are available as Integrated Development Environments (IDE) which combines all of the necessary tools into an integrated environment.

Two common development packages include:

- Keil MDK ARM (uVison5 IDE) – The MDK ARM IDE is a very stable development environment which can be downloaded for free. It allows development of code up to a program size of 32KB. For developing larger programs a licensed version needs to be purchased here: http://www2.keil.com/mdk5/install
- CoIDE – A free tool chain which is based on a trimmed down version of the Eclipse IDEintegrated along withan embedded ARM version of thefree GCC compiler. It can be downloaded here: http://www.coocox.org/software/coide.php

There are also several other IDEs that are available for use with STM32 microcontrollers. However, this guidefocuses on developing and flashing a program using the very popular Keil MDK ARM uVision5 IDE.

Apart from the software tools, an In-Circuit Serial Programmer(ICSP) is required to program and test the code on the actual microcontroller. The ICSP is required to interface the microcontroller to the PC software tools via a USB port. The ARM Cortex-Mmicrocontrollers support two programming protocols: JTAG (named by the electronics industry association the Joint Test Action Group) and Serial Wire Debug (SWD).

There are several ICSP programmers available that support theseprotocols, including:

- Keil U-Link 2
- Segger J-Link
- ST-Link

Developing the first application

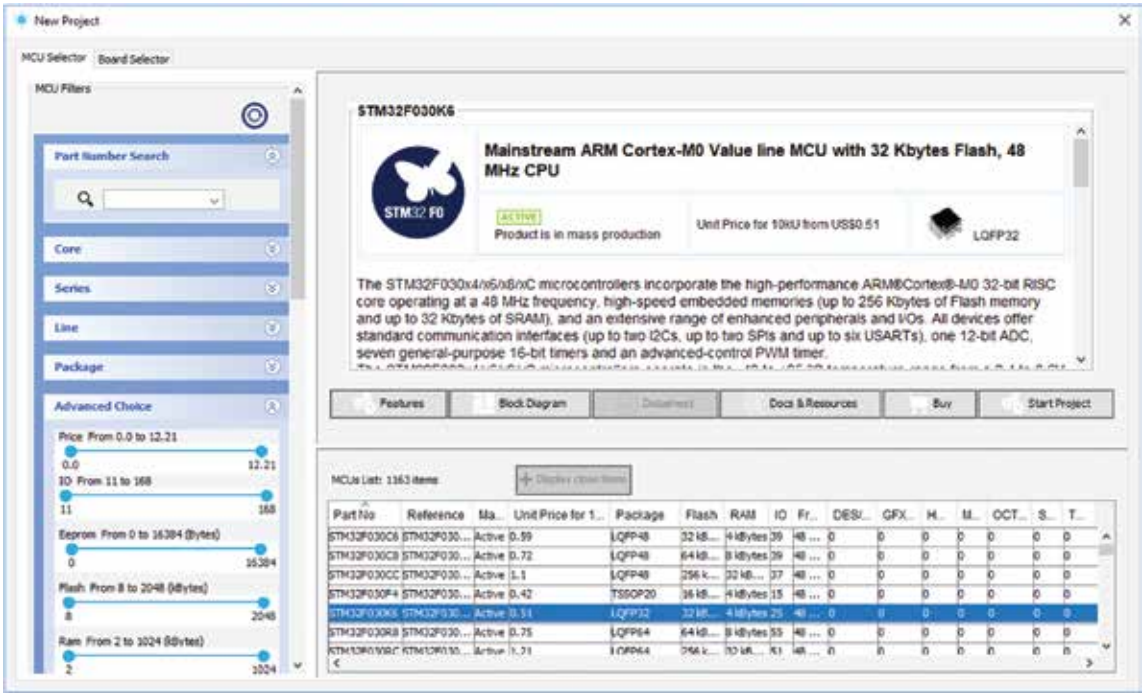
It's always easiest to start with a readily available basic code framework. Then, add the code that is required for the specific application and model of microcontroller.

Fortunately, STMicroelectronics provides a very useful graphical tool called STM32CubeMx that helps in creating a basic application project for any STM32 microcontroller of your choice. It also can be used to configure the peripherals on the multiplexed pins of the microcontroller.

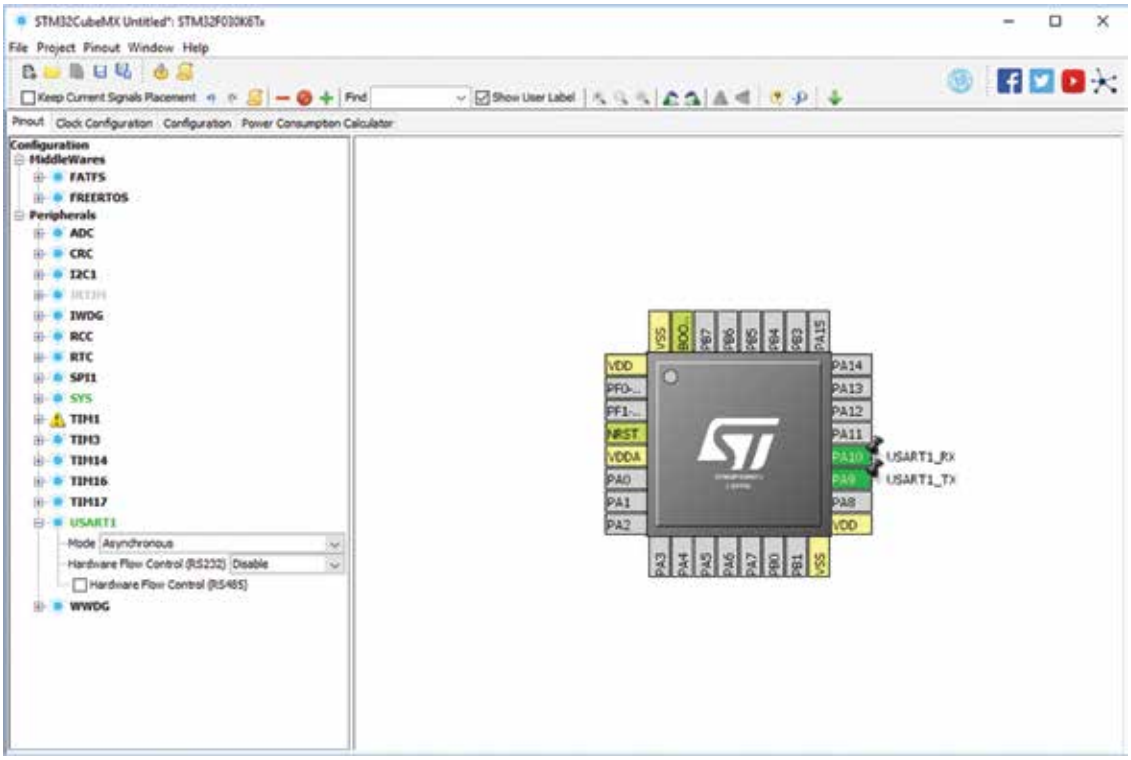
The STM32CubeMXtoolcan be downloaded from here. The STM32Cube comes with an extensive set of drivers for all types of peripherals and support for an optional FreeRTOS (a free Real-Time Operating System) pre-integrated with the code.

The following section describes in detail how to create a simple UART application for the STM32F030 microcontroller that echoes whatever is typed on a terminal window.

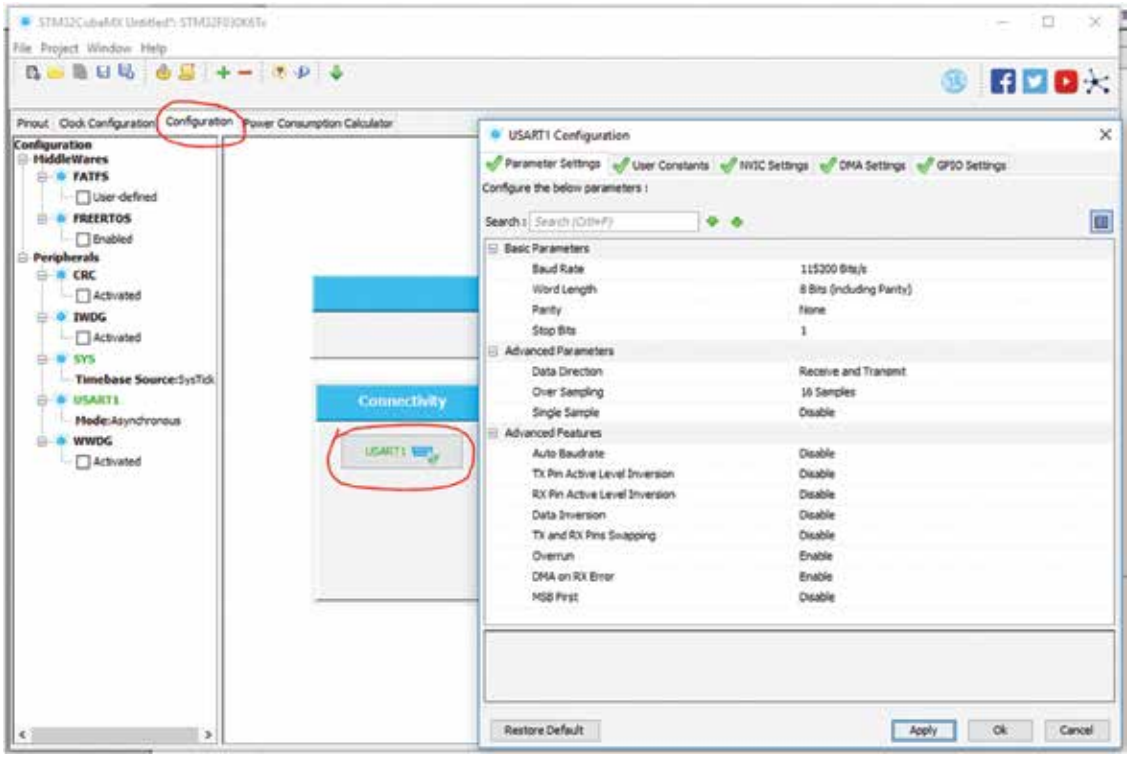
- Install the STM32CubeMX software.
- Run the application and select New Project. It will then open the MCU Selector window as shown below.
- Double click to select the microcontroller model being used. In this case we're using the STM32F030K6. It then takes you to the pinout page for the selected microcontroller.



The STM32F030K6 is an ARM Cortex-M0 core with 32KB of Flash memory and 4KB of RAM memory. The example code enables the UART that uses the PA9 and PA10 pins for receiving and transmitting serial data as shown below with the green pins.



Configure the UART settings under the *Configuration Tab* and choose the UART settings as shown below. Enable the NVIC global interrupt option under the *NVIC Settings* tab.



Next, navigate to *Project-->Settings* in order to add the new project name and select the tool chainIDE to be used. For this example, set the project name to 'UARTEcho' and select the Keil-MDK5 IDE for the project development.

Finally, generate the project code by clicking *Project -> Generate Code.*

Building and flashing the code

Now open the generated MDK-ARM project file `UARTEcho\MDK-ARM\UartEcho.uprojx.`

This program so far just initializes the UART peripheral and stops in an infinite loop.

It's important to note that the STM32Cube generates `/* USER CODE BEGIN x */` and `/* USER CODE END x */` comment blocks to implement the user specific code. The user code must be written within these comment blocks. Whenever the code is re-generated with modified configurations the STMCube tool retains the user code within these user comment blocks.

Next, define a global variable to receive a byte from the UART in the main.c source file:

```
/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/
static uint8_t recv_data;
/* USER CODE END PV */
```

After all of the initialization code, enable the driver to receive 1 byte. The following function enables the RXNE interrupt bit.

```
/* USER CODE BEGIN 2 */
  HAL_UART_Receive_IT(&huart1, &recv_data, 1);
  /* USER CODE END 2 */
```

Now, add a callback function to handle the receive interrupt and transmit the received byte.

```
/* USER CODE BEGIN 0 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
     HAL_UART_Transmit(huart, huart->pRxBuffPtr, 1, 1000);
}
/* USER CODE END 0 */
```

Finally, we need to compile the code and flash (download) it to the microcontroller.

When the Keil MDK ARM IDE is installed, drivers for ST-LINK V2, J-Link and Ulink2 are available. The ST-Link debugger will be selected by default. Go to *Projects-->Optionsfor Target* and in the *Debugtab* select the ICSP programmer used.

Flash the code by selecting *Flash->Download.*

The microcontroller will now echo any data received over the UART. It can be connected to a PC by using a USB-to-Serial Converter. On the PC open the COM port with a terminal application using the settings of 115200-8-N-1. Now anything that is sent from the terminal will echo back through the microcontroller.

Interrupt system

The STM32 interrupt system is based on the ARM Cortex M core NVIC peripheral. The STM32 MCUs support multiple maskable interrupt channels apart from the 16 interrupt channels of the ARM core.

For example the STM32F0 MCU series support 32 maskable interrupts. The exception and the interrupt vector table for this family of MCUs is given in the table below.

Interrupt	Description	Vector Address
-	Reserved	0x00000000
Reset	Reset	0x00000004
NMI	Non maskable interrupt. The RCC clock security system (CSS) is linked to the NMI vector	0x00000008
HardFault	All class of faults	0x0000000C
SVCall	System service call via SWI Instruction	0x0000002C
PendSV	Pendable request for system service	0x00000038
SysTick	System tick timer	0x0000003C
WWDG	Window watchdog interrupt	0x00000040
PVD_VDDIO2	PVD and VDDIO2 supply comparator interrupt (combined with EXTI lines 16 and 31)	0x00000044
RTC	RTC interrupts (combined EXTI lines 17, 19 and 20)	0x00000048
Flash	Flash global interrupt	0x0000004C
RCC_CRS	RCC and CRS global interrupts	0x00000050
EXTI0_1	EXTI line[1:0] interrupts	0x00000054
EXTI2_3	EXTI line[3:2] interrupts	0x00000058
EXTI4_15	EXTI line[15:4] interrupts	0x0000005C
TSC	Touch sensing interrupt	0x00000060
DMA_CH1	DMA channel 1 interrupt	0x00000064
DMA_CH2_3 DMA2_CH1_2	DMA channels 2 and 3 interrupts DMA2 channel1 and 2 interrupts	0x00000068
DMA_CH4_5_6_7 DMA2_CH3_4_5	DMA channel 4,5,6 and 7 interrupts DMA2 channel 3, 4, and 5 interrupts	0x0000006C
ADC_COMP	ADC and COMP interrupts (Combined EXTI lines 21 and 22)	0x00000070
TIM1_BRK_UP_TRG_COM	TIM1 break, update, trigger and commutation interrupts	0x00000074

TIM1_CC	TIM1 capture compare interrupt	0x00000078
TIM2	TIM2 global interrupt	0x0000007C
TIM3	TIM3 global interrupt	0x00000080
TIM6_DAC	TIM6 global interrupt and DAC underrun interrupt	0x00000084
TIM7	TIM7 global interrupt	0x00000088
TIM14	TIM14 global interrupt	0x0000008C
TIM15	TIM15 global interrupt	0x00000090
TIM16	TIM16 global interrupt	0x00000094
TIM17	TIM17 global interrupt	0x00000098
I2C1	I2C1 global interrupt (combined with EXTI line 23)	0x0000009C
I2C2	I2C2 global interrupt	0x000000A0
SPI1	SPI1 global interrupt	0x000000A4
SPI2	SPI2 global interrupt	0x000000A8
USART1	USART1 global interrupt (combined with EXTI line 25)	0x000000AC
UART2	USART2 global interrupt (combined with EXTI line 26)	0x000000B0
USART3_4_5_6_7_ 8	USART3, USART4, USART5, USART6, USART7, USART8 global interrupts (combined with EXTI line 28)	0x000000B4
CEC_CAN	CEC and CAN global interrupts (combined with EXTI line 27	0x000000B8
USB	USB global interrupt (combined with EXTI line 18)	0x000000BC

Extended Interrupts and Events Controller (EXTI)

The STM32 MCUs have an Extended interrupts and Events controller which manages the external and internal asynchronous events/interrupts and generates the event request to the CPU/Interrupt Controller and a wake-up request to the Power Manager.

Each of the one or more EXTI lines are mapped to one of the NVIC interrupt vectors.

For the external interrupt lines, to generate an interrupt, the interrupt line should be configured and enabled. This is done by programming the two trigger registers with the desired edge detection and by enabling the interrupt request by writing a ‘1’ to the corresponding bit in the interrupt mask register.

External Interrupt and GPIO mapping

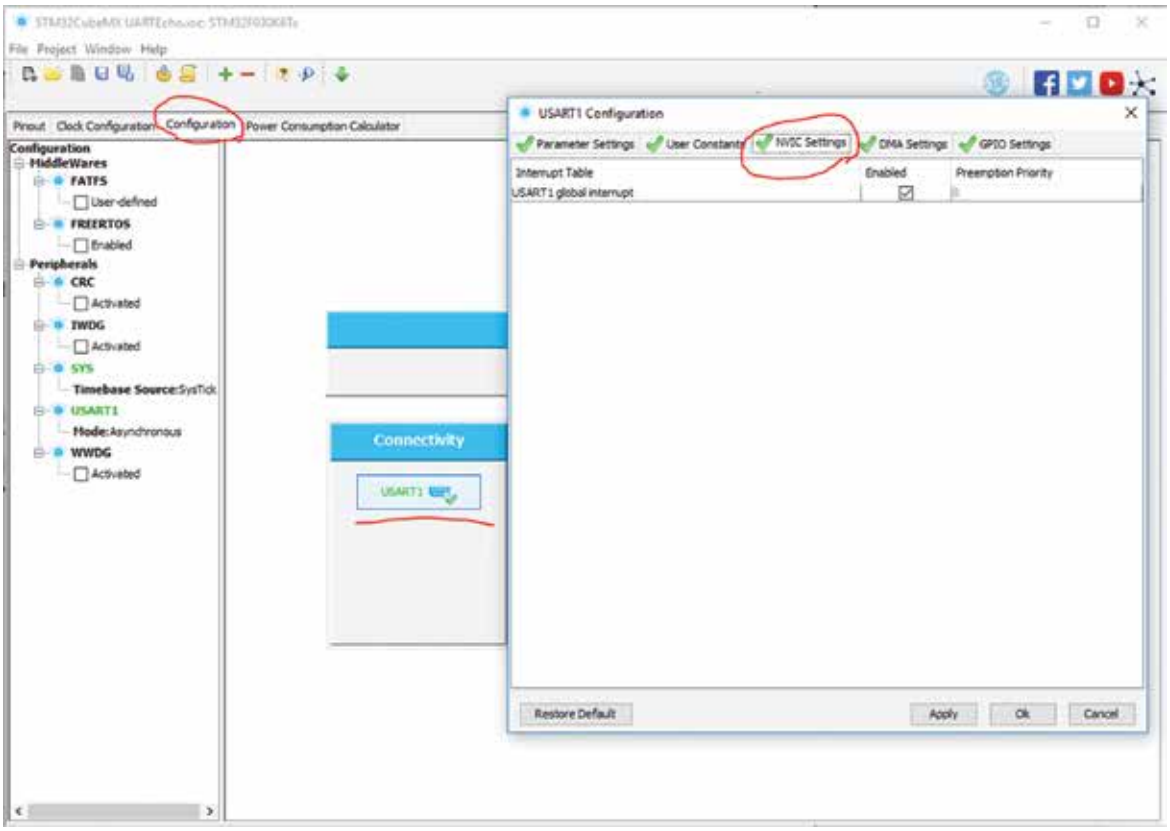
Each of the GPIO available on the system can be configured to generate an interrupt. But each of the EXTI interrupt lines is mapped to multiple GPIO pins. For example, PIO0 on all the available GPIO ports (A,B,C, etc.) will be mapped to the EXTI0 line. PIO1 for all ports will be mapped to the EXTI1 line and so on.

Some of the EXTI lines are combined to a single NVIC vector. For example, EXTI4_15 is mapped to a single vector address so there will be a single interrupt routine for all the interrupts from PIO4 to PIO15. But the source of the interrupt can be identified by reading the interrupt pending register.

One important thing to consider while designing a system using the STM32 MCUs is the selection of the GPIO pins for the interrupts. The MCU may have more than 16 GPIOs available on the device but there are only 16 external interrupt lines available.

For instance, the EXTI_0 can be mapped to either PA0 or PB0 but not both. So while choosing the pins for external interrupts they should be chosen such that they can be uniquely mapped to one of the EXTI lines.

The following section describes how to configure an interrupt using the STM32 Cube.



Select the Configuration Tab and choose the hardware module for which the interrupt has to be configured. The module configuration window opens.

Then select the NVIC settings tab and enable the global interrupt.

The code to enable the interrupt for the module will be generated in the `stm32f0xx_hal_msp.c` in the `HAL_<module>_MSPInit(...)` function.

```
/* USART1 interrupt Init */
    HAL_NVIC_SetPriority(USART1_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(USART1_IRQn);
```

The code generated by the STM32 Cube will have the IRQ_Handler implementation of all the interrupts. When the interrupt is enabled the code will be included into the application.

Usually the generated code already handles the IRQ and clears the flag which generated the interrupt. It then calls an application callback that corresponds to the event that generated the interrupt for the module.

The STM32 HAL (Hardware Abstraction Layer) implements a callback for each of the event types within each module as part of the driver. In this example the Rx Transfer Complete callback should be copied from the `stm32f0xx_hal_UART.c` file.

The callback functions within the driver will be implemented with a__weak [linker attribute](). The user needs to implement a copy of the necessary callback function by removing the__weakattribute in one of the application files and then writing the specific handling required within that function.

```
/**
  * @brief Rx Transfer completed callback.
  * @param huart UART handle.
  * @retval None
  */
__weak void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(huart);

  /* NOTE : This function should not be modified, when the callback is needed,
            the HAL_UART_RxCpltCallback can be implemented in the user file.
   */
}
```

Programming Conclusion

There are several other methods for writing an application but the STM32Cube discussed is an easy and intuitive method to get started.

This tool simplifies the initialization of the microcontroller peripherals. It also improves the maintainability of the code especially when there are hardware revisions which require remapping of the signals to different pins.

Another advantage of using the STM32Cube tool is that it generates a report of the user configuration for the microcontroller. In this report it details the clock tree, pin mapping and hardware module configuration which are all very useful.

There are also several other code libraries and example programs available for all the STM32 variants. Support for several IDEs is also included.

PART 2 –Introduction to the STM32F030

In this section I'll be reviewing the datasheet for the entry-level STM32F030.

The datasheet for the STM32F030 is ninety-one pages long with way too much information to ever cover without being overwhelmed. Don't worry, I won't subject you to that. Instead, we're going to only review the front page, which lists most of the microcontroller's key features.

In order to follow along I highly suggest you download the STM32F030 datasheet yourself and refer to the first page. I'll be reviewing this page on a line-by-line basis. Be sure you also watch this video where I walk you through this datasheet.

ARM32-bit Cortex-M0 CPU, frequency up to 48 MHz

The first line on the datasheet specifies the CPU core, which is a 32-bit ARM Cortex-M0 architecture processor that can operate with a clock speed up to 48 MHz.

ARM is a separate company that designs processor architectures but doesn't manufacture their own chips. Instead, they license out their designs to multiple semiconductor manufacturers like ST, TI, Nordic, NXP, Silicon Labs, and many others.

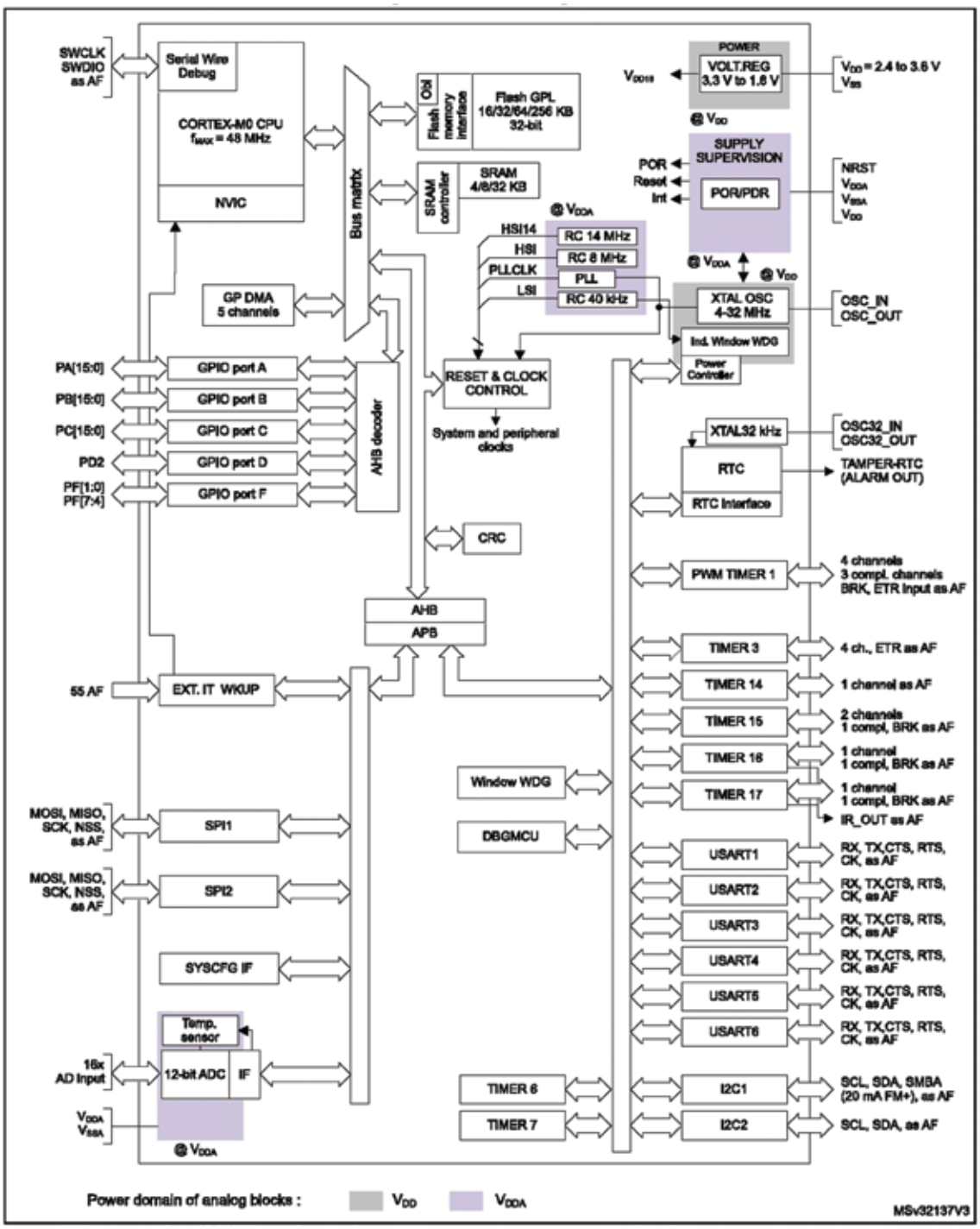
These manufacturers then incorporate ARM processor core designs into their own microcontroller chips.

There are various versions of the ARM Cortex-M line of processor cores. M0 is the lowest performance model, with M7 being the highest performance core in the line.

Cortex microcontrollers are by far the most common microcontroller used in commercial products. In fact, they are used in literally billions of products around the world. The STM32 series is one of the most popular implementations of the Cortex microcontroller architectures, and is my personal favorite line of microcontrollers.

8-bit microcontrollers like the AVR line embedded in Arduinos are mainly used for development boards aimed at makers and DIY'ers. They aren't as commonly used for commercial products. ARM cores dominate the commercial market and provide better performance for similar cost compared to most 8-bit microcontrollers.

That being said, if you are already familiar with these 8-bit microcontrollers, have developed significant code already, and your product is relatively simple, then they may be a good choice for your product.



Complete block diagram STM32F030

Memories

16 to 256 Kbytes of Flash memory

The STM32F030 microcontroller can support anywhere from 16 KB to 256 KB of flash memory depending on the exact model number. This flash memory is mainly used for storing the firmware program.

4 to 32 Kbytes of SRAM with HW parity

There's also from 4 KB to 32 KB of SRAM for temporary storage. The SRAM includes hardware parity which is a data check to make sure the data being read has not been corrupted since originally being written.

CRC calculation unit

CRC stands for Cycle Redundancy Check. This isn't anything you likely need to worry that much about, it's just another way of ensuring that digital data isn't corrupt.

Reset and power management

Digital & I/O supply: VDD = 2.4 V to 3.6 V

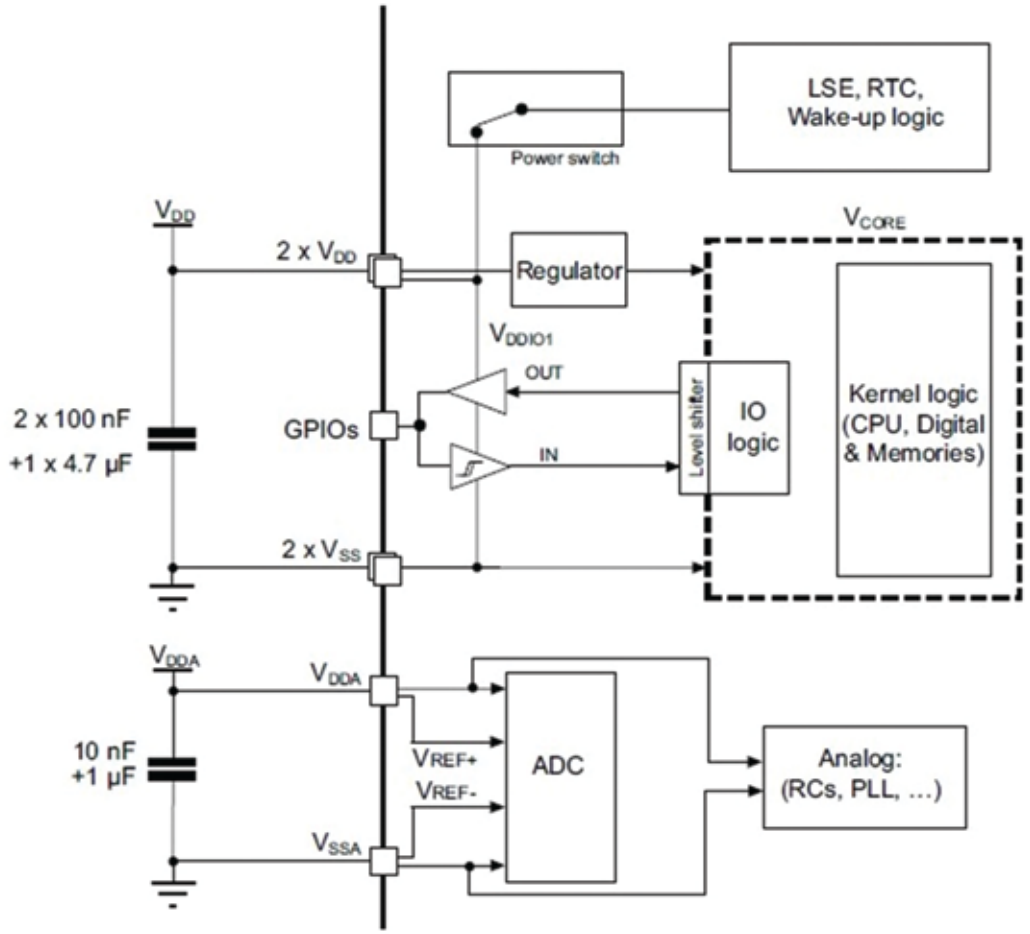
The digital supply voltage, referred to as VDD, can be anywhere from 2.4 V to 3.6 V. This supply is used to power the internal digital circuits (after being sub-regulated down to 1.8V by an internal voltage regulator).

It is also used as the supply voltage for the I/O drivers. This means that all of the GPIO pins will operate at this supply. If you want to interface this microcontroller via a GPIO pin to another chip with a 5V supply voltage, then you would need to use a level shifter.

Analog supply: VDDA = VDD to 3.6 V

A separate analog supply for the Analog-to-Digital Converter (ADC) can be anywhere from a low of VDD (digital I/O supply) up to 3.6 V. The analog supply must be equal to or higher than the digital supply.

For example, if the digital supply is 3V, then the analog supply voltage can be anywhere from 3V up to 3.6V.



Power supply scheme for STM32F030 (from datasheet)

Power-on/Power down reset (POR/PDR)

A Power-On Reset (POR), as the names implies, resets the microcontroller when first powered up. This is so the microcontroller always starts in a known state when first turned on.

It does this by monitoring the digital supply. As long as the VDD voltage is below 2V the microcontroller remains in reset mode. Once the supply voltage rises above this threshold the microcontroller becomes active.

The Power-Down Reset (PDR) circuit works in a similar fashion except it is monitoring both the digital and analog supplies for when they fall below the minimum voltage threshold.

These two circuits prevent the microcontroller from activating when powered from a supply voltage too low to guarantee proper functionality.

Low power modes: Sleep, Stop, Standby

As with many microcontrollers, the STM32F030 offers various low power modes of operation called sleep, stop and standby.

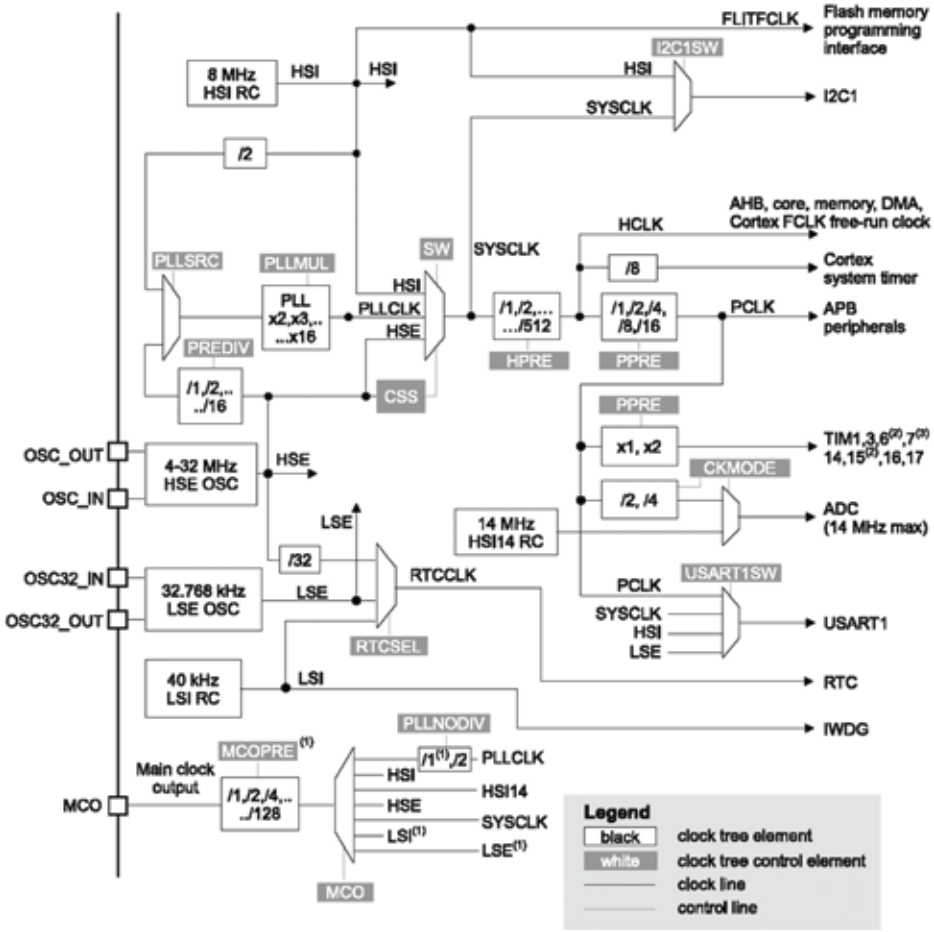
When in sleep mode all of the peripherals continue to operate. Only the CPU itself is stopped when in sleep mode. By using interrupts the peripherals can wake up the CPU whenneeded. Of the three low power modes, sleep mode will consume the most power.

When in stop mode all of the clocks are disabled, but the contents of SRAM and all of the registers is retained. As with the sleep mode, the microcontroller can be awoken from stop mode when an interrupt is detected. Stop mode is the intermediate low power consumption mode.

Standby mode offers the lowest power consumption of these three modes. However, when in standby mode all clocks are stopped, the internal 1.8V regulator is switched off, and the SRAM contents are lost.

Clock Management

Every digital processor requires a clock signal for timing purposes. There are several clock options with this microcontroller.



Clock tree for STM32F030 (from datasheet)

4 to 32 MHz crystal oscillator

You can use an external crystal oscillator with a frequency between 4 MHz and 32 MHz. This will provide the most accurate clock source.

32 kHz oscillator for RTC with calibration

For the Real-Time-Clock (RTC) an external 32.768 kHz crystal should be used. This frequency is commonly used for real-time clock applications because it is a power of 2, such that 2^15 = 32768. This makes it easy to internally divide the clock signal down to a very precise 1 Hz signal (one cycle per second).

The RTC includes an internal calibration circuit to keep the oscillator clock signal exactlyprecise for timing purposes.

Internal 8 MHz RC with x6 PLL option

Other possible clocks include an internal 8 MHz RC oscillator. An RC oscillator means an oscillator based on the timing characteristics of a resistor (R) and a capacitor (C).

The advantage of an RC clock over a crystal clock is that an RC oscillator can be easily embedded inside the microcontroller chip, whereas a crystal oscillator requires an extra external component.

The downside of an RC oscillator is their frequency accuracy isn't all that great. If you need precise timing, you'll want to use a crystal oscillator.

The 8MHz RC clock has a 6-X PLL option. PLL is the abbreviation for Phase-Locked Loop, which is a complex subject beyond the scope of this guide. But essentially, this is just a 6-X frequency multiplier. So it takes the 8MHz RC oscillator frequency, multiplies it by 6x, and outputs a 48 MHz clock signal.

Internal 40 kHz RC oscillator

Finally, there's also an independent 40 kHz RC oscillator built-in which is used for the watchdog function which I explain further below.

Up to 55 fast I/Os

This STM32 supports up to 55 different GPIO pins. The exact number depends on the model number and package. The higher pin count packages will almost always support a higher number of I/O pins

All mappable on external interrupt vectors

This means that any of the I/O pins can be used as an interrupt. An interrupt is just a signal that can come from an external peripheral which alerts the processor that it needs to do something.

You can have what are called interrupt service routines so that whenever one of these interrupt pins is triggered by an external device, the microcontroller will then execute the appropriate routine associated with that interrupt pin.

Up to 55 I/Os with 5V tolerant capability

All of the GPIO pins on the STM32F030 are specified as being 5V-tolerant even though the maximum I/O supply voltage is only 3.6V. This means a GPIO programmed as an input pin can safely be fed a 5V digital signal.

Be warned though this doesn't mean you can output a 3.6V signal and feed it to an input pin on a device with 5V I/O. The I/O pins being 5V tolerant only helps on pins programmed as inputs.

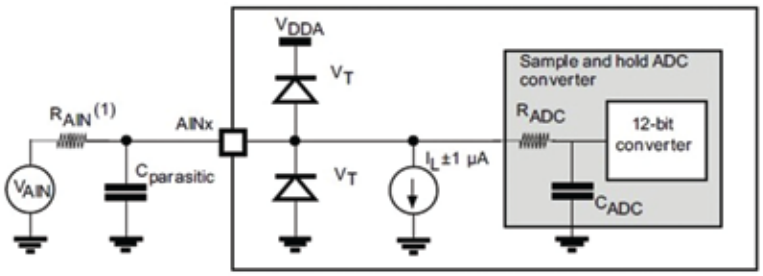
5-channel DMA controller

A Direct Memory Access (DMA) controller allows external memory or peripherals to communicate directly with the internal memory without the central processing unit (CPU) having to be involved in that communication.

Using a DMA frees up the CPU to keep on processing without having to deal with shuffling data around. A DMA allows you to get data in and out of the microcontroller, without having to interrupt the core processor.

One 12-bit, 1.0 µs ADC (up to 16 channels)

An Analog-to-Digital Converter (ADC) converts an analog voltage to a digital reading that the microcontroller can then process. This STM32 microcontroller has a 12-bit ADC with 16 channels. This means the analog voltage will be converted to a 12-bit digital code.



Typical ADC connection diagram

Conversion range: 0 to 3.6 V

The maximum number represented with a 12-bit code is 2^12 = 4,096. The analog input range can be between 0 and 3.6V. To calculate the smallest increment in analog voltage that can be measured with this 12-bit ADC you take voltage range and divide by 2^12 = 4,096.

This means the smallest voltage change that can be measured by this ADC is 3.6 V / 4,096 = 0.88 mV when using a 3.6 V analog supply. Smaller increments are possible with lower voltage analog supplies.

Separate analog supply: 2.4 V to 3.6 V

A separate supply is almost used for powering an ADC because of the low noise requirement. You don't want the digital noise to couple into the ADC because it can cause significant error in measurements.

A very clean, stable supply typically filtered through an LC (inductor-capacitor) low-pass filter should be used for the analog supply.

Calendar RTC with alarm and periodic wakeup

A calendar RTC (Real-Time Clock) feature allows the microcontroller to measure not only time, but also to keep track of the current date and time. The STM32F030's calendar RTC also includes an alarm and periodic wakeup functionality.

11 timers

Most microcontrollers, if not all, include various timers which are used for lots of different applications.

One 16-bit advanced-control timer for six-channel PWM output

One common use of a microcontroller is to generate PWM signals. PWM stands for Pulse-Width Modulation. A PWM signal is generated with a specific on-time and off-time ratio. This ratio can be used for all kinds of purposes such as controlling a motor.

The STM32F0 has one 16-bit timer specifically for use with a six-channel PWM output.

Up to seven 16-bit timers

It also includes up to seven (number depends on exact model) other 16-bit timers for general purpose uses.

Independent and system watchdog timers

As with most microcontrollers the STM32F0 also includes what is known as a watchdog timer.

A watchdog timer will reset a microcontroller if it gets locked up. A watchdog routine must be regularly updated by the processor. If this doesn't occur, then the watchdog routines knows that the processor has locked up and a reset is automatically initiated.

SysTick timer

Then last up is SysTick timer. This is a system timer that generates regular interrupts. This is mainly used if you're going to run a Real-Time Operating System (RTOS) on the microcontroller.

Most applications do not need to run an operating system for microcontrollers, but if you need an OS for your application then you will use this timer.

Communication interfaces

A microcontroller that can't communicate with the world outside its own chip isn't of much use. Various interfaces are critical to allow the microcontroller to communicate with other chips, sensors, displays, cameras, etc.

Up to two I2C interfaces

I2C is a very common, two-pin serial interface. There's only a clock pin (called SCL) and a data pin (called SDA). This makes it ideal for connecting multiple devices on the same PCB with very little routing space required.

I2C is also an addressable bus protocol so you can connect many different peripherals using only those same two lines. Each device will have its own unique address. You can put multiple devices on these same two wires, and then you can address each one separately. Each device will know when it is the one being talked to.

It's not a super high-speed serial communication protocol, but it's really handy to use for any on-board low-speed communication. It’s especially common to use this protocol for communicating with various types of sensors.

The STM32F030 supports speeds up to 1 Mbps on both I2C interfaces. It also supports SMBus/PMBus which are two-wire protocols derived from I2C but that are specifically used for communicating with power sources.

Up to six USARTs

USART stands for Universal Synchronous Asynchronous Receiver Transmitter. Whew, that’s a mouthful.

Perhaps you’ve instead heard of a UART which offers only asynchronous (notice the ‘S’ is gone for synchronous) communication. Synchronous communication means there's a clock signal shared between the two devices communicating for timing purposes.

There are STM32F030 variants that provide up to six USART interfaces which can support both synchronous and asynchronous communication.

Up to two SPIs (18 Mbit/s)

SPI is a much faster interface which supports up to 18 Mbps, whereas I2C only supports 1 Mbps, making SPI roughly 18 times faster. Both I2C and SPI are synchronous protocols with a shared clock signal.

SPI can be used for more data-intensive peripherals, such as interfacing with a display, whereas I2C is a better choice for low-data peripherals such as sensors. A downside of SPI compared to I2C is that it requires at least three wires versus only two for I2C.

With I2C, data flows in both directions along a single data line. Whereas SPI has two data lines, one for each direction of data flow.

Because SPI isn’t an addressable bus protocol, a 4th signal line called Slave Select (SS) is commonly needed to ensure that only one device on the SPI lines is active at one time.

If you have peripherals A and B connected to the same SPI port, and you wish to communicate with device B, then you must disable device A and enable device B. It’s not nearly as elegant a solution as provided by I2C.

Serial wire debug (SWD)

Serial Wire Debug (SWD) is the interface you will use to connect to the microcontroller for programming and debug purposes. Like I2C, SWD is a two-wire synchronous communication protocol.

Unlike programming an Arduino, most microcontrollers require a special programming device to program your firmware into the flash memory.

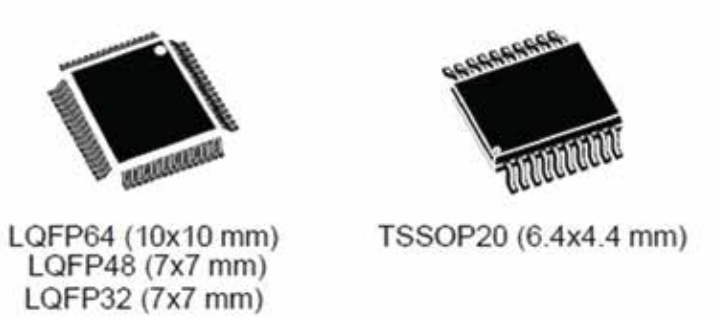
The most common choice for the STM32 series is the ST-LINK which supports SWD, JTAG, and SWIM programming interfaces.

The ST-LINK will connect to your computer via USB, but will then interface with the microcontroller using SWD.

Packages

The STM32F030 is available in a QFP package or a TSSOP package. The TSSOP package is only available in a 20-pin version so it will have limited features and GPIO pins available.

For the QFP package there are three pin count versions: 32 pins, 48 pins, and 64 pins. The QFP-32 and QFP-48 measure 7 mm x 7 mm. The QFP-64 measures 10 mm x 10 mm.



Available packages for STM32F030

Both of these packages are known as leaded packages. This means the pins actually come out the side of the package. Once they're soldered onto the board, you can still access the pins.

This is in contrast to leadless packages, such as QFN or BGA packages, for instance. For these packages the pins are actually underneath the plastic, so once they're soldered down, you can't access the pins.

In most cases I recommend using a leaded package, unless you need to squeeze every fraction of a millimeter out of the design, or if a certain chip only comes in a leadless package.

First of all they make it easier to debug any hardware issues because you can easily access all the pins without adding test points.

They also are cheaper to solder onto the printed circuit board which can lower your prototype and manufacturing cost.

Conclusion

The STM32F030 is good entry level Cortex-M0 microcontroller with a wide selection of peripherals. But there are hundreds of models in the STM32 lineup so you need to select the model with the performance and feature set that fits best with your product's requirements.

Based on what you've learned from this STM32F030 datasheet review, you should now have the knowledge to select the best microcontroller for your project.

PART 3 – Introduction to the STM32F469

In the previous section I reviewed an entry-level STM32 microcontroller. In this section I'm going to be reviewing a significantly more advanced version from the STM32 family.

The microcontroller I'll be discussing is the STM32F4, or more specifically the STM32F469, which is based on the higher performance ARM Cortex-M4 processor architecture.

This microcontroller is extremely fast and offers a huge assortment of features and peripherals. In fact, one of the features offered is normally only available on more expensive microprocessors. No other microcontroller on the market offers this advanced feature (keep reading to learn more it).

The datasheet for the STM32F469 is a whopping 216 pages long and there is a lot of technology stuffed into this chip. Needless to say we won't be reviewing the entire datasheet, and instead we'll just focus on the front page which details all of the key features.

I'm going to be focusing mostly on the features that differentiate this microcontroller from the STM32F030 that we looked at previously. So I'll just quickly pass over some of the more standard features that are similar or identical to the STM32F030.

Arm32-bit Cortex-M4 CPU with FPU

The STM32F469 is based on an ARM Cortex-M4 processor core. The STM32F030 that we looked at previously was based on a Cortex-M0 processor.

The number after the 'F' in an STM32 part number matches the 'M' number for the Cortex core. All STM32F0 controllers use Cortex-M0 processor cores, all STM32F1 controllers use an M1 core, and so on. The most advanced core is the Cortex-M7 which is available in STM32F7 and STM32H7 microcontrollers.

The Cortex-M4 is a much more advanced core than the M0. For one thing an Cortex-M4 gets more done for each tick of the clock.

For example, if you compare an M0 processor against an M4 processor with the exact same clock speed, the M4 will perform about 50% better than an M0 (based on performance benchmarks).

The other key feature that separates Cortex-M4 processors from less advanced ones is the addition of a Floating Point Unit (FPU).

An FPU is a separate processor core, sometimes called a math coprocessor, that is specifically designed for performing mathematical functions on floating-point numbers.

Not all versions of the Cortex-M4 include an FPU. Officially, Cortex-M4F is the proper designation for M4 cores with an FPU. But this isn't strictly followed, and many vendors refer to a M4 processor with an FPU simply as Cortex-M4.

Adaptive Real-Time Accelerator (ART Accelerator)

Microcontrollers, unlike microprocessors, usually run firmware code directly from their on-chip Flash memory. But Flash memory is slow. This bottleneck becomes a more serious issue the faster the processor.

The simple solution is to have the processor execute wait states when running from Flash memory. But that is far from ideal since much of the performance increase is lost because of these wait states.

One potential solution is to move the firmware code being executed into faster RAM memory. But RAM memory is less dense than Flash, so on-chip RAM is usually much more limited than Flash memory.

ST's much more elegant solution is called an Adaptive Real-Time Accelerator (ART Accelerator).

The details of how this memory accelerator works are beyond the scope of this guide, but suffice it to say that it significantly improves processor performance when executing code from Flash memory.

180 MHz / 225 DMIPS

The STM32F4 has a performance benchmark rating of 225 Dhrystone MIPS (Million Instructions Per Second), or DMIPS.

Although the M4 core is about 50% faster per clock tick than the M0, the biggest performance boost comes from the fact that an M4 core can run at a much higher clock frequency.

For example, the maximum clock frequency for the STM32F0 is 48 MHz, but the maximum clock for the STM32F4 is 180 MHz. This means at the maximum clock speed the STM32F4 is about 1.5 * (180 / 48) = 5.6 times faster than the STM32F0.

STM32F7 microcontrollers with the even more advanced Cortex-M7 core take this performance even further. Tick for tick an STM32F7 is about 70% faster than an STMF4. The STM32F7 also supports a slightly faster clock of 216 MHz. This means an STM32F7 is roughly about 2x (1.7 * 216 / 180) the performance of the STM32F4.

Finally, the STM32H7 takes this performance to the extreme. It too is based on the Cortex-M7 processor core, but ST's advanced semiconductor processor allows them to crank up the maximum clock speed to a whopping 400 MHz!

For a microcontroller that is insanely fast. The STM32H7 definitely blurs the line between a microcontroller and a microprocessor.

This speed comes at a price though. They cost more, they consume more power, and they significantly complicate development, so only use them when absolutely required.

Once the digital signal processing is completed, the information is then converted back into analog format by a Digital-to-Analog Converter (DAC).

There are separate high-performance DSP processors available as well that are specifically design for processing digital signals. My former employer Texas Instruments, along with Analog Devices mostly dominate the DSP chip market.

The STM32F4 doesn't include a separate DSP processor, but it does offer specialized instructions for performing DSP operations on the Cortex-M4 core.

DSP Instructions

The STM32F4 provides a set of special instructions for performing Digital Signal Processor (DSP) operations. A DSP is for processing continuous real-world analog signals.

Analog signals (such as an audio signal) are first converted to digital format via an Analog-to-Digital Converter (ADC). The analog information (audio in this case) can then be processed as digital data by a DSP.

Once the digital signal processing is completed, the information is then converted back into analog format by a Digital-to-Analog Converter (DAC).

There are separate high-performance DSP processors available as well that are specifically design for processing digital signals. My former employer Texas Instruments, along with Analog Devices mostly dominate the DSP chip market.

The STM32F4 doesn't include a separate DSP processor, but it does offer specialized instructions for performing DSP operations on the Cortex-M4 core.

Memories

Flash and RAM Memories

The STM32F469 microcontroller supports up to 2 MB of Flash memory. This is the upper limit on Flash memory available built-in on any STM32 microcontroller.

What separates the Flash memory on the STM32F469 from less advanced microcontrollers is the fact that it is split into two separate banks. This allows you to simultaneously read from one bank while writing to the second bank.

The STM32F469 also includes 384 KB of system RAM with 64 KB being what is called Core Coupled Memory (CCM). CCM is RAM memory that is tightly coupled (placed nearby) with the CPU core allowing the fastest access time.

CCM RAM is usually only necessary for computation intensive tasks and real-time processing.

External Memory Controller

Another feature that really sets this microcontroller apart from most other choices is that it has an external memory controller built-in. This allows you to add external high-speed RAM memory that will interface with the microcontroller via a super-fast 32-bit data bus.

The ability to interface with external RAM memory is normally the realm of high-performance microprocessors, not microcontrollers. Microcontrollers tend to have all the RAM built in, which is one of the many reasons designing with a microcontroller is a much simpler design process.

But there are cases when the ability to add additional fast, off-chip RAM will be a big advantage. For example, to serve as a frame buffer in video applications.

Although it can add significant design complexity, if you do need a large amount of high-speed RAM memory then the STM32F469 cans support it.

The external memory controller supports various types of RAM memory including SRAM (Static RAM), PSRAM (Pseudo-Static RAM), and SDRAM (Synchronous Dynamic RAM). It also supports NOR/NAND based Flash memory.

Quad-SPI Interface

SPI is a very common serial communication interface that you may be familiar with already. It's commonly used for communicating between chips on a PCB that need to pass data at a moderately high

speed. It's much faster than the two-wire I2C protocol.

SPI is also a common choice for interfacing a microcontroller to a Flash memory chip. Normally, SPI uses two unidirectional data lines called Master-In-Slave-Out (MISO) and Master-Out-Slave-In (MOSI). In order to increase the data transfer rate the quad-SPI interface instead uses four bidirectional data lines.

For example, if reading data from a Flash memory via a standard SPI interface, data is transferred on only a single data line (MISO). However, when using a quad-SPI interface this same data is transferred over four data lines, thus increasing the data throughput speeds by 4x.

This additional speed is necessary for some applications that need to be able to access Flash memory at the maximum transfer speeds.

If high-speed throughput with Flash memory is absolutely critical for your application you may also consider using Flash memory that connects to the 32-bit data bus offered by the STM32F469's external memory controller.

Graphics

The graphics capabilities of the STM32F469 really set it apart from not only the STM32F030 that I reviewed recently but also from many other high performance microcontrollers.

In fact, the STM32F479 offers one major graphics feature that is available on no other microcontroller on the market.

Graphical Hardware Accelerator

A graphics hardware accelerator is includedwhich ST calls the Chrom-ART Accelerator. This hardware accelerator is for performing image manipulation tasks. It offloads this work from the core processor, leaving it free to perform other tasks.

LCD TFT controller

The LCD-TFT display controller outputs 24-bit parallel RGB data (8-bit for each Red, Green, and Blue). This allows it to be directly interfaced to a wide variety of LCD-TFT displays. It supports resolutions up to 1024 x 768.

A TFT (Thin-Film-Transistor) LCD display is an active matrix display unlike simple passive LCD displays that offer a limited amount of segments. LCD-TFT is the same technology commonly used for computer displays and television sets.

MIPI-DSI host controller

Finally we're to the feature that I've been eluding to as being available in no other microcontroller on the market. Drum roll, please... That feature is called MIPI-DSI. It is a feature which is normally only found on much more expensive microprocessors, not on microcontrollers.

The MIPI Alliance is an organization that develops interface specifications for the mobile industry. It was founded in 2003 by ARM, ST Microelectronics, Texas Instruments, Intel, Nokia and Samsung. MIPI stands for Mobile Industry Processor Interface.
MIPI-DSI is their specification for a unidirectional, serial interface designed especially for connecting to a display. DSI stands for Display Serial Interface.

For the LCD-TFT I mentioned data is output as 24-bit parallel RGB data. This means that at least 24 pins on both the microcontroller and the display are required, and that 24 signal traces on the PCB are required for connecting them together. All of this adds more size to the product which is obviously a big deal for mobile products.

A standardized serial display interface, that only requires a fraction of the signals, acts to significantly reduce the size of the product.

MIPI-DSI has been widely adopted by the mobile industry. It is ubiquitous in smart phones and tablets. Having a microcontroller that supports MIPI-DSI opens up a huge selection of high resolution, full color, mobile displays. Most of these are high resolution AMOLED (Active Matrix Organic Light Emitting Diode) displays.

The MIPI-DSI interface on the STM32F469 can support up to 720p HD video at 30 frames per second. If 1080p HD is your goal then you will require a microprocessor or a specialized video processor.

Note that there is also a MIPI-CSI interface standard for connecting to cameras, but that is not supported by this microcontroller or any other one currently available. Hopefully that will change in the near future (are you listening ST?).

Analog-to-Digital Converter (ADC)

The STM32F030 included only one 12-bit ADC with a maximum sampling rate of 1 MSPS (Million Samples Per Second). The STM32F469 increases the number of converters to three, and increases the sampling speed up to 2.4 MSPS.

The STM32F469 even supports sampling rates up to 7.2 MSPS when using what is called triple interleave mode. In this high-speed mode, all three ADC units are working together to sample the same signal. This gives a top sampling rate equal to three times the standard rate, or 3 x 2.4 MSPS = 7.2 MSPS.

Debug mode

The STM32F030 only supported Serial Wire Debug (SWD). The STM32F469 still supports SWD but it also supports the more advanced protocol called JTAG.

The JTAG interface requires a lot more pins, and is used for more than just programming and debug. JTAG can be used during manufacturing to detect hardware defects.

Advanced connectivity

Inter-IC Sound (I2S)

The STM32F469 offers a special audio interface called I2S (Inter-IC Sound). This is not to be confused with I2C (Inter-Integrated Circuit).

Both are used for transmitting data synchronously (with a shared clock) between chips on a PCB. But, I2S is specifically used for transmitting stereo digital audio data, whereas I2C is for transmitting more generic, low-speed data.

If your product requires high-quality audio functionality then you will almost definitely want to select a microcontroller that supports I2S. To eliminate noise pickup it is usually best to leave audio information in the digital domain as long possible.

For example, if you need to route audio data across your PCB it is much better to do so as a digital data like I2S, not an analog signal.

Although the entry-level STM32 microcontrollers (like the STM32F030) don't support I2S, many intermediate versions do support it and you don't necessarily need to a controller as advanced as the STM32F469.

CAN Bus

The STM32F469 also supports two CAN bus interfaces. CAN bus is a serial communication protocol used in automotive applications. CAN is how the various subsystems in an automobile communicate with the master processor.

USB 2.0 OTG

The STM32F469 offers two USB 2.0 ports. One port supports high-speed USB communication up to 480 Mbits/sec which is the maximum transfer speed supported by USB 2.0.

The other port only supports full-speed USB which is limited to transfer speeds of 12 Mbits/sec.

Both ports support USB-OTG (USB On-The-Go). A device that supports USB OTG mode has the capability to serve either the host function or the peripheral function. Controllers that only support standard USB can't ever serve the host function.

For example, if your product offers a standard USB interface then you can only connect it to a device that includes a USB host controller such as a PC or tablet. Your product will always serve as the peripheral, so you can't connect it to a USB peripheral such as a printer.

But with OTG, if you connect your device to a USB peripheral like a printer then your device will assume the host controller role. The hardware required to serve as a USB host controller is significantly more complex than for a peripheral-only device.

Ethernet

Another feature that sets this microcontroller apart from a more entry level microcontroller is that it also includes an Ethernet controller.

To implement Ethernet functionality both a MAC (Media Access Control) layer and a PHY (physical) layer are required. The MAC layer is embedded in the microcontroller, but an external Ethernet transceiver is still required for the PHY layer.

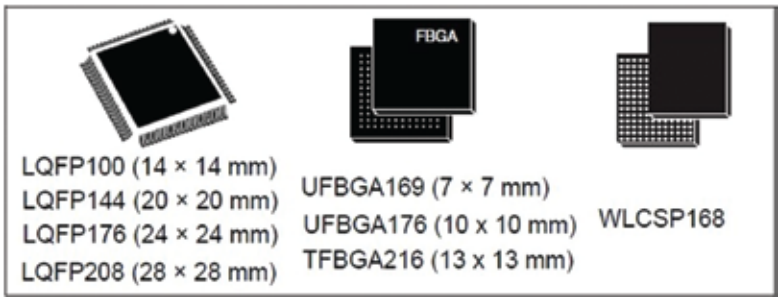
Parallel camera interface

A parallel camera interface allows a digital connection with a camera using between 8 and 14 bits of parallel data. This camera interface supports transfer speeds up to 54 MB/sec.

The speed of this interface is sufficient to support up to 720p HD video. As I mentioned already, to support 1080p HD video a more advanced microprocessor, or specialized video processor is required.

Packages

The STM32F469 is available in three types of packages: QFP, BGA, and CSP with varying pin counts depending on the number of GPIO pins.



Unless small size is absolutely critical for your product, I would in most cases recommend the QFP package. This is because it is a leaded package so all of the pins are easy to access. This can be critical for debugging purposes. Leaded packages are also cheaper to have soldered onto the PCB.

A BGA (Ball Grid Array) has all of the pins on the bottom of the package in a grid pattern. These make for smaller chip sizes, especially for those requiring high pin counts. But, they also complicate the PCB layout.

Connecting all of those closely placed pins requires a PCB with more layers. That increases the PCB cost. Also, soldering a BGA package on your PCB is more expensive than for a leaded QFP package.

So a BGA packages allows for a smaller PCB, but at the expense of extra design complexity and production cost.

A CSP (Chip-Scale Package) takes the concept of a BGA even one step further. It eliminates the plastic package altogether allowing for the absolute minimum size possible.

A CSP package has all of the same disadvantages as a BGA package, so only use this package if you must squeeze every fraction of a millimeter from your board size.

Summary

I think you can see that the STM32F469 is an extremely advanced microcontroller. It's in a whole other league than the STM32F030 that we looked at previously. There is a drastic difference in both speed and features.

PART 4 – Overview of the STM32H7

As impressive as the STM32F469 may be, it's raw processing performance is dwarfed by the STM32F7 series, and especially the STM32H7 series which we'll look at next.

The STM32H7 is the most powerful member of the popular STM32 family of 32-bit microcontrollers based on ARM Cortex-M cores, and offered by ST Microelectronics.

This microcontroller can be clocked at speeds up to 480 MHz with a benchmark performance greater than 1,000 DMIPS. This is one of the fastest, most powerful microcontrollers currently available on the market.

The STM32H7 is so fast and offers so many advanced features that it begins to blur the line between microcontroller and microprocessor.

The STM32H7 is at the top of the High Performance category. It is a single or dual-core microcontroller, consisting of a 480MHz Cortex M7 and an additional 240MHz Cortex M4 core for the dual core versions. The High Performance category offers the highest performance in code execution and data transfers.

The table below summarizes the main features of the various members of the STM32H7 offerings.

Arm® Cortex®-M7 – 480 MHz

CORE, MEMORIES AND ACCELERATION
- Single-core Cortex-M7 480 MHz
- Dual-core Cortex-M7 480 MHz and Cortex-M4 240 MHz (STM32H7x5 and STM32H7x7 only)
- Flash and RAM acceleration
- SP-FPU and DP-FPU
- 4 x DMA

CONNECTIVITY
- 2 x USB2.0 OTG FS/HS
- 2 x SDMMC
- USART, UART, SPI, I²C
- 2 x CAN (1 x FD and 1 x TT)
- HDMI-CEC
- FMC, Dual-mode Quad-SPI
- Ethernet MAC IEEE1588
- Camera I/F
- Analog (comp, AOP)

AUDIO
- 3 x I²S + audio PLL
- 4 x SAI
- 2 x 12-bit DAC
- SPDIF-RX

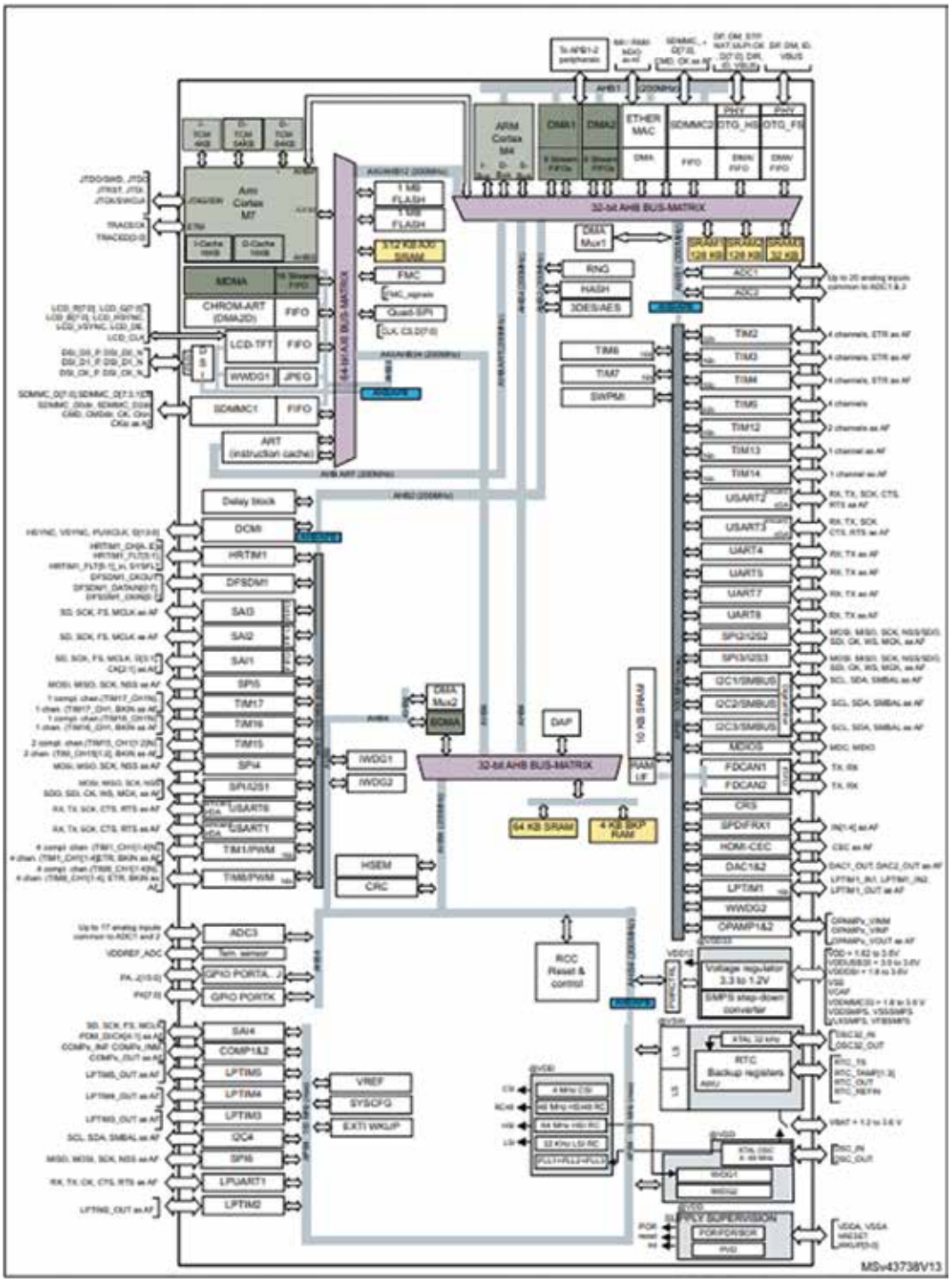
GRAPHIC
- Chrom-ART Accelerator™

OTHER
- Crypto/Hash (except H742)[1]
- Security services (except H742)
- TRNG
- DFSDM
- 16- and 32-bit timers
- 3 x 16-bit ADC (up to 3.6 MSPS)
- Voltage range 1.62 to 3.6 V (except 100-pin package : 1.71 to 3.6 V)
- Multi-power domains

STM32 H7 Product line	f_{CPU} (MHz)	Dual-Bank Flash memory (bytes)	RAM (bytes)	Graphic	Power supply	Ambiant temperature range
Dual-core lines						
STM32H7x7	480 + 240	Up to 2 Mbytes	1 Mbyte (incl.128 Kbytes DTCM + 64 Kbytes ITCM + 64 Kbytes backup1) + 4 Kbytes backup2	TFT-LCD JPEG codec MIPI-DSI	DCDC + LDO	Standard 85 °C
STM32H7x5	480 + 240	Up to 2 Mbytes	1 Mbyte (incl.128 Kbytes DTCM + 64 Kbytes ITCM + 64 Kbytes backup1) + 4 Kbytes backup2	TFT-LCD JPEG codec	DCDC + LDO	Standard 85 °C (Opt. Industrial CPN 125 °C)[2]
Single-core lines						
STM32H7x3	480	Up to 2 Mbytes	1 Mbyte (incl.128 Kbytes DTCM + 64 Kbytes ITCM + 64 Kbytes backup1) + 4 Kbytes backup2	TFT-LCD JPEG codec	LDO	Standard 85 °C
STM32H742	480	Up to 2 Mbytes	692 Kbytes (incl.128 Kbytes DTCM + 64 Kbytes ITCM + 16 Kbytes backup1) + 4 Kbytes backup2	no	LDO	Standard 85 °C
Value line						
STM32H750	480	128 Kbytes	1 Mbyte (incl.128 Kbytes DTCM + 64 Kbytes ITCM + 64 Kbytes backup1) + 4 Kbytes backup2	TFT-LCD JPEG codec	LDO	Standard 85 °C

Current members of the STM32H7 family and their salient features

The figure below shows the internal block diagram of the high-end dual core STM32H757.



Internal block diagram of the dual core SMT32H757 microcontroller

Because this line of STM32 microcontrollers has quite a few members, and each one has a varying number of features, here is a link to a product selection guide specifically related to the STM32H7 variants.

Hardware Development with the STM32H7

The STM32H7 sub-family consists of many members, some with single Cortex M7 cores and some with dual Cortex M7 and Cortex M4 cores. The different variants sport different numbers and combinations of the usual peripherals such as GPIO's, I2C's, I2S's, SPI, Timers, USARTS.

Some more advanced peripherals, that are not usually found in less performant microcontrollers, are also available. These include peripherals such as USB OTG, Ethernet and CAN controllers, Serial Audio Interfaces (SAI), Camera interface, TFT LCD interfaces and hardware graphics accelerators.

The family also supports cryptographic encryption engines in hardware, and also has features to ensure secure booting and code upgrade. There are no hard and fast rules for choosing the right microcontroller for a given application.

However, it should be quite obvious that the STM32H series is a candidate for applications that require high-end peripherals plus intense processing.

One of the best ways to learn about this family of processors is to spend some hands-on time with the hardware, running through some example development processes.

Toward this end, ST Microelectronics has many different evaluation boards, and the software development tools that go with them. Practicing with pre-defined examples will help you learn how to custom design applications.

I recommend theSTM32H7x3I-EVAL boards to evaluate the full set of features for the STM32H7 series.

The image below shows a picture of an actual STM32H753I-EVAL board. The TFT screen and Ethernet RJ-45 sockets can be clearly seen on the top right hand side of the picture. They can be purchased from the regular distributors such as Digikey.

STM32H7531 evaluation board

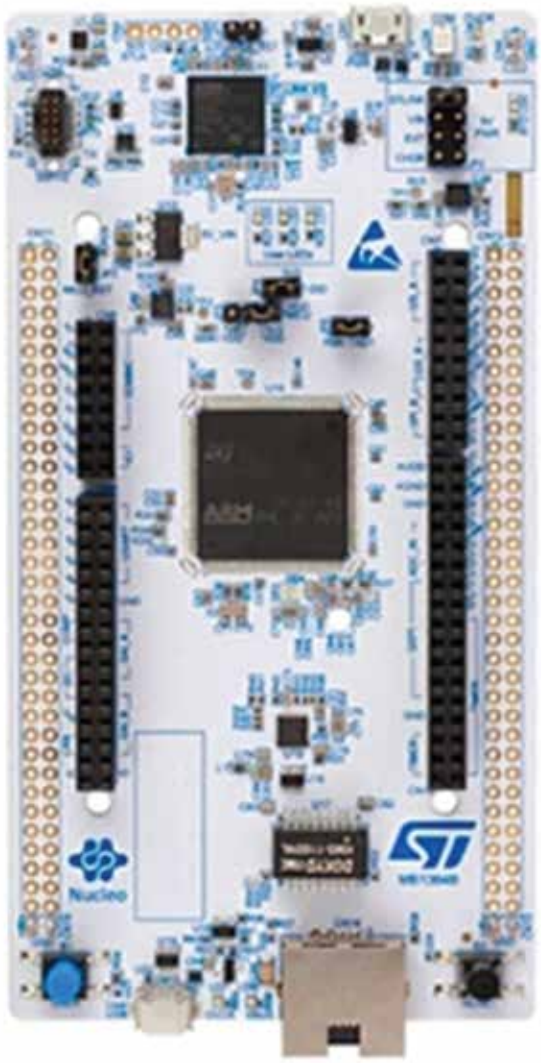
This board provides an evaluation platform for some of the more advanced peripherals of the STM32H7 family of devices.

A somewhat intermediate level board is the Discovery board for a given STM32 microcontroller. It doesn't have all the options of the evaluation board, but it does incorporate some on-board sensors.

The least expensive alternative to the STM32H7x3I-EVAL board is the Nucleo board (see image below). It is basically a bare-bones module with minimal additional hardware besides the microcontroller itself. It's a good introduction to getting familiar with a particular microcontroller.

Also, since the pins of the microcontroller are available on the headers on the left and right sides of the board, it is possible to interface this board to external hardware for additional prototyping.

Also, be aware that at the time of this writing, Discovery and Nucleo boards are not available for all STM32H7 devices. But STMicro is usually quite good at supporting their microcontrollers, so this should not be considered a blocking issue.



Nucleo board for one of the STM32H7 microcontrollers

All of the boards mentioned allow downloading example code, or user-written code, from a cross-development platform like Windows, Mac or Linux machines where the code is actually written, complied, and debugged first.

The actual hardware connection from the development platform to the board is USB. On the other hand, the actual microcontroller on the target board is programmed in-circuit through a Serial Wire Debug, or SWD, interface.

The SWD Interface is an alternative to the JTAG (Joint Test Action Group) industry standard for debugging and flashing microcontrollers. ST Microelectronics uses it to program its STM32 line of microcontrollers. The HW interface that allows the cross-development platform to download code to the microcontroller is the STLink-V3. It is built into all the boards mentioned previously.

Also, please note that the STLink-V3 is only found in boards sporting the latest STM32 microcontrollers such as the STM32H7 family. Older STM32 evaluation boards had STLINK-V2 interfaces, which are much

slower, among other things, at about 12Mbits/s download speed compared to the STLINK-V3 at 480 Mbits/s.

The STLINK-V3 has other improvements over the V2 as well. It is also available as a standalone product for use in user-developed hardware from the usual distributors.

Application Development for the STM32H7

As previously mentioned, the STM32H7 family is a complex, single-core, or dual-core, high performance microcontroller with advanced peripherals. Developing applications from scratch, as is usually done with smaller microcontrollers, is not going to be a viable approach in most cases.

Fortunately, this need not be the case here. This section looks at the various tools available to help with application development.

All that it takes for most people to develop applications from the SYM32H7 family is to install a suitable Integrated Development Environment, or IDE, on a cross-development platform.

Then, develop the code. Compile, link in the appropriate libraries, and download to the target through a suitable programmer such as the STLink-V2 or STLink-V3. Although the STLINK-V2 will work, it is recommended to use the V3 if for nothing other than the higher download speed.

However, the STLink-V3 offers much more than that. This short clip shows some of its capabilities. As for the IDE, one such example is TrueStudio from Atollic. It, of course, features an Atollic compiler.

Other major compilers are available for STM32H7 family from Keil and IAR Systems. To go totally freeware, choose the Eclipse IDE.

This IDE can accommodate many add-on compilers such as the GCC, GNU Compiler Collection, tool chain for the STM32. Just be aware that while they are mostly compatible with each other in terms of supporting regular C language, they still have differences in the way some things are accomplished.

Assuming that everything is fine so far, it's time to focus on the application software. It isn't possible to cover all user applications.

However, embedded applications typically require setting up the core hardware and the various built-in peripherals, such as timers or IO pins. This is in order to control external hardware, whether these are LED's, relays, I2C devices, or others.

As I mentioned earlier, manufacturers such as ST Micro license Cortex cores from ARM Holdings, and then integrate their own set of peripherals such as UARTs, CAN, I2C, Ethernet or SPI on single chips that

they then offer as their microcontrollers. So, some of the setup will be for the ARM core itself, and some will be directed to the peripherals.

Configuring the core, or any peripherals, involves writing configuration values to internal registers. Determining what values to actually write to these multiple registers involves two things:

- Understanding what a particular hardware block does, and how it should behave in a particular application.
- Determining which registers to setup for that particular hardware block, and what values to set the registers to.

The first one requires reading, and understanding, the device specifications. There is simply no way around it. For example, the ARM core can have many system clock sources, with Phase-Locked Loops (PLL's) to multiply these up and divisors to divide them down.

The reason for that is because various blocks such as the memories and the peripheral bus can run at different clock speeds.

As another example concerning peripherals, each of the multiple timers require a suitably scaled clock input source. They can operate in many modes such as Input Capture, Output Compare, Pulse Width Modulation (PWM), with or without interrupt, or automatic reload.

The same applies to the USARTs, SPI, I2C, USB, and other peripherals. Each one of these is a separate section in the datasheet.

While there are no shortcuts for the first part, there is help on the second part. Setting up the hardware involves writing the appropriate values to certain registers.

Most of these registers are further segmented into bit fields, where only some bits of a given register affect some part of a given hardware block, while other groups of bits affect other parts.

Worse, sometimes the settings of one register can affect how the bits of other registers are interpreted. So, it is usually a very time consuming exercise that involves a fully detailed understanding of each of these registers.

Fortunately, both ARM and STMicro have made life simpler by providing their own Hardware Abstraction Layers, or HAL's. HAL, as the name implies, is a firmware layer that abstracts the hardware below it, and presents a more uniform way to access this hardware to all FW above it.

For instance, registers that are used to configure a particular block of HW are given appropriate names. There are pre-defined functions that are then used to configure a particular aspect of the HW such as IO port, clock sources, or timers.

The ARM Cortex HAL is called CMSIS for Cortex Microcontroller Software Interface Standard. The STM peripherals are simply called STM HAL.

As good as the HAL is, the user still needs to call the appropriate functions from all the available HAL functions, with the appropriate values to configure the underlying hardware. So, ST has provided a tool to dispense with even this part.

It is called STMCubeMx. This tool provides a GUI that allows the user to essentially configure the HW without writing much code. The configuration C code is automatically generated by this tool, with some sections where the user can insert the actual application code.

STMCubeMx is available here, and this video describes the STMCube in general.

Again, STMCubeMx is a nice tool, but it still requires that the user knows how the device needs to be configured to match the application. Here are some examples specifically targeted to the STM32H7.

Conclusion

The STM32H7 blurs the line between the world of microcontrollers and the world of high-performance microprocessors. It's one of the fastest, most advanced, microcontrollers currently on the market.

For many products the STM32H7 would be overkill, but for products that require fast processing speeds it can be a fantastic solution that is much easier to implement than a custom microprocessor design.

PART 5 –Custom STM32 Board Design

In this final section you'll learn how to design your own custom microcontroller board based on the STM32F042. You can also watch this video showing me designing this board.

I'll break down the design process into three fundamental steps:

Step 1 - System Design
Step 2 - Schematic Circuit Design
Step 3 - PCB Layout Design

System / Preliminary Design

When developing a new circuit design the first step is the high-level system design (which I also call a preliminary design). Before getting into the details of the full schematic circuit design it's always best to first focus on the big picture of the full system.

Designing the system consists mainly of two steps: creating a block diagram and selecting all of the critical components (microchips, sensors, displays, etc.). A system design treats each function as a black box

In engineering, a black box is an object which can be viewed in terms of its inputs and outputs but without any knowledge of its internal workings. With a system-level design the focus is on the higher level interconnectivity and functionality.

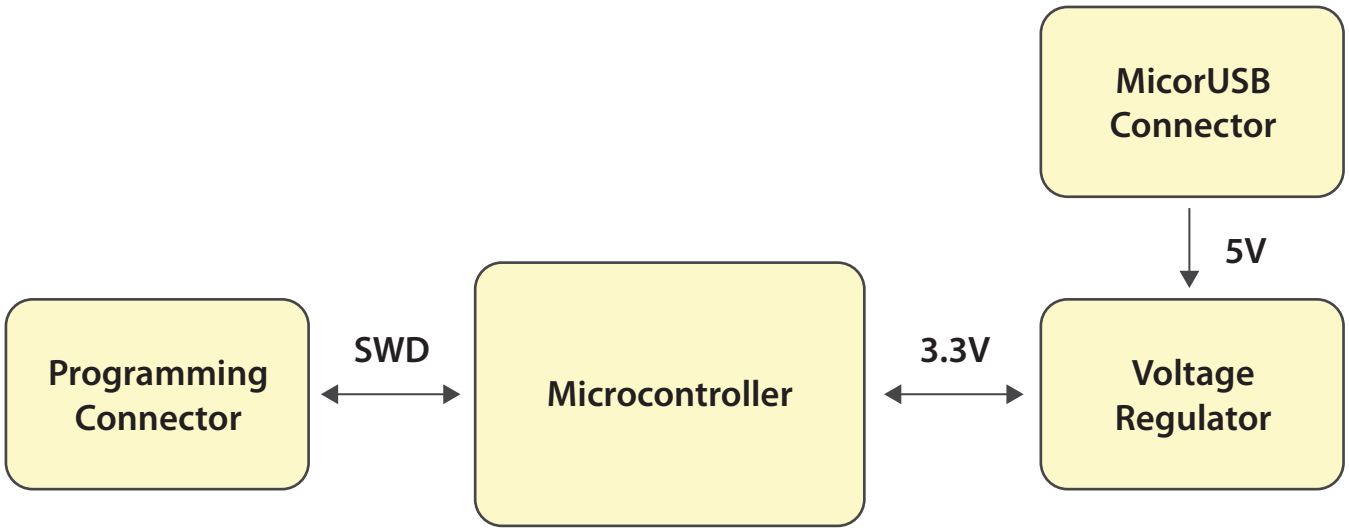
Block Diagram

Below is the block diagram that we'll be working from in this tutorial series. As I mentioned, for this first tutorial we'll focus just on the microcontroller itself. In future tutorials we'll expand the design to include all of the functionality shown in this block diagram.

A block diagram should include a block for each core function, the interconnections between the various blocks, specified communication protocols, and any known voltage levels (input supply voltage, battery voltage, etc.).

Later, once all of the components have been selected and the required supply voltages are known I like to add the supply voltages to the block diagram. Including the supply voltage for each functional block it allows you to easily identify all of the supply voltages you'll need as well as any level shifters.

In most cases when two electronic components communicate they need to use the same supply voltage. If they are supplied from different voltages then you'll usually need to add in a level shifter.



System-level block diagram

Now that we have a block diagram we can better understand the necessary requirements for the microcontroller. Until you've mapped out everything that will connect to the microcontroller it's impossible to select the appropriate microcontroller.

Select Microcontroller

When selecting a microcontroller (or just about any electronic component) I like to use an electronics distributor's website like Newark.com. Doing so allows you to easily compare various options based on a variety of specifications, pricing, and availability. It's also an easy way to quickly access the component's datasheet.

I'm a big fan of ARM Cortex-M microcontrollers. Arm Cortex-M microcontrollers are easily the most popular line of microcontrollers used in commercial electronic products. They have been used in tens of billions of devices.

Microcontrollers from Microchip (including Atmel) may dominate the maker market but Arm dominates the commercial product market.

Arm doesn't actually manufacture the chips directly themselves. They instead design processor architectures that are then licensed and manufactured by other chip makers including ST, NXP, Microchip, Texas Instruments, Silicon Labs, Cypress, and Nordic.

The ARM Cortex-M is a 32-bit architecture that is fantastic choice for more computationally intensive tasks compared to what is available from older 8 bit microcontrollers such as the 8051, PIC, and AVR cores.

Arm microcontrollers come in various performance levels including the Cortex-M0, M0+, M1, M3, M4, and M7. Some versions are available with a Floating Point Unit (FPU) and are designated with an F in the model number such as the Cortex-M4F.

One of the biggest advantages of Arm Cortex-M processors is their low price for the level of performance you get. In fact, even if an 8-bit microcontroller is sufficient for your application you should still consider a 32-bit Cortex-M microcontroller.

There are Cortex-M microcontrollers available with very comparable pricing to some of the older 8-bit chips. Basing your design on a 32-bit microcontroller gives you more room to grow should you want to add additional features in the future.

Although numerous chip makers offer Cortex-M microcontrollers, my favorite by far is the STM32 series from ST Microelectronics. The STM32line of microcontrollers is quite expansive with just about any feature and level of performance you would ever need.

The STM32 line can be broken down into several subseries as shown in the table below.

STM32 Series	Cortex-Mix	Max clock (MHz)	Performance (DMIPS)
F0	M0	48	38
F1	M3	72	61
F3	M4	72	90
F2	M3	120	150
F4	M4	180	225
F7	M7	216	462
H7	M7	400	856
L0	M0	32	26
L1	M3	32	33
L4	M4	80	100
L4+	M4	120	150

Comparison of various STM32 microcontroller variants

The STM32F subseries is their standard line of microcontrollers (versus the STM32L subseries which is specifically focused on lower power consumption). The STM32F0 has the lowest price but also the lowest performance. One step up in performance is the F1 subseries, followed by the F3, F2, F4, F7, and finally the H7.

For this tutorial I have selected the STM32F042K6T7 which comes in a 32-pin LQFP leaded package. I selected a leaded package primarily because it simplifies the debugging process because you have easy access to the microcontroller pins.

Whereas with a leadless package, like a QFN, the pins are hidden away underneath the package making access impossible without test points.

A leaded package also allows you to more easily swap out the microcontroller if it were to become damaged. Finally, leadless packages cost more to solder on to the PCB so they increase both the prototyping and manufacturing costs.

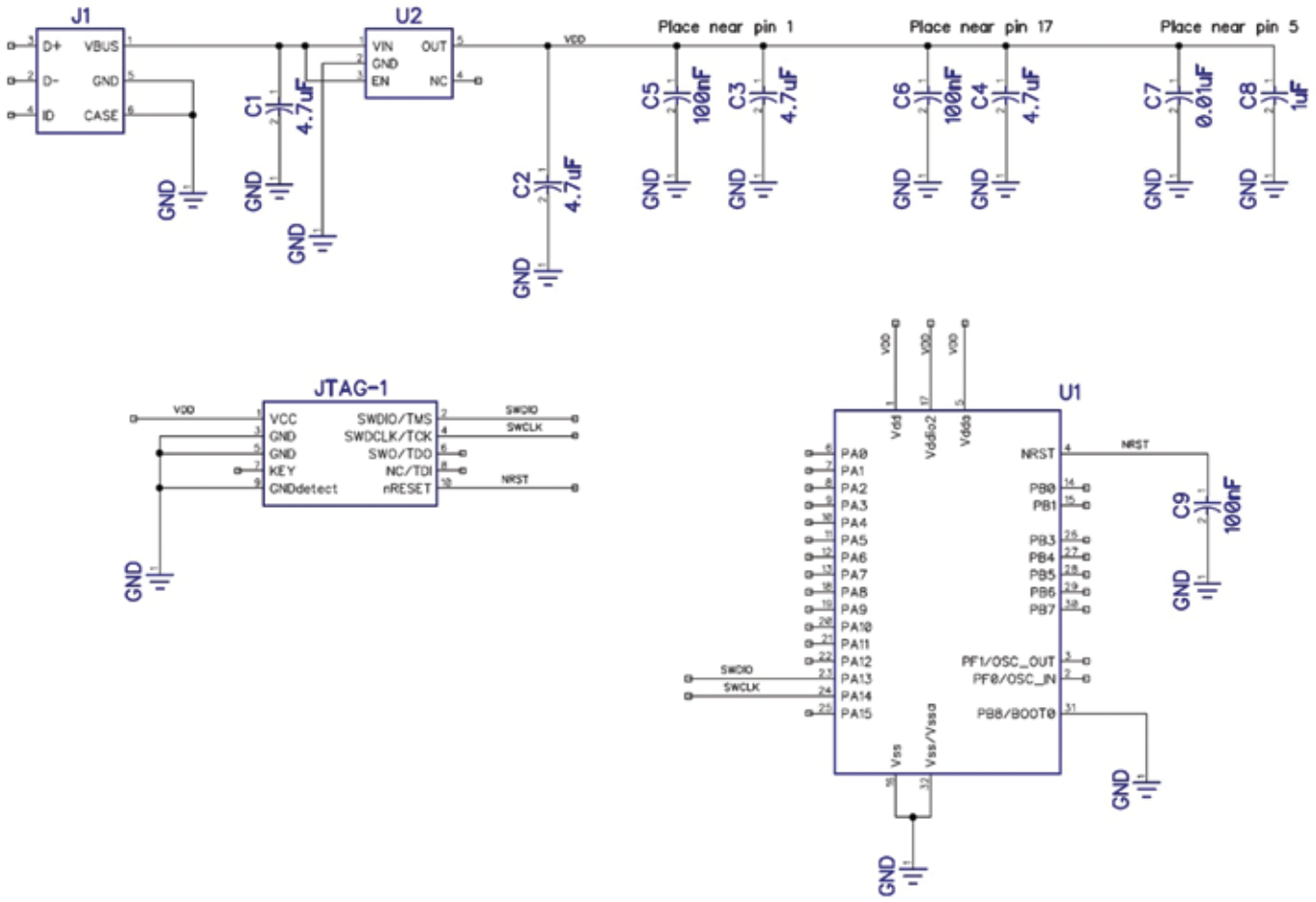
I selected the STM32F042 because it offers moderate performance, a good number of GPIO pins, and various serial protocols including UART, I2C, SPI and USB.

This is a fairly entry-level STM32 microcontroller with only 32 pins, but with a wide variety of features. More advanced versions come with as many as 216 pins which would be quite overwhelming for an introductory tutorial.

Schematic Circuit Design

Now that we have selected the microcontroller it's time to design the schematic circuit diagram. For these tutorials I'll be using a PCB design tool called DipTrace.

There are dozens of PCB tools available but when it comes to ease of use, price, and performance I find that DipTrace is hard to beat, especially for beginners.



Schematic circuit diagram for this first tutorial showing the STM32 microcontroller, linear regulator, USB connector, and programming connector.

If you don't have a PCB design package then you may want to consider downloading the free version of DipTrace so you can follow along closely with this tutorial. The best way to learn something is always to actually do it.

Nonetheless, this tutorial will be focusing on the process for designing a custom microcontroller board, and not on how to use any specific PCB design tool. So regardless of which PCB software you end up using you'll still find these tutorials are just as useful.

The first step in designing a schematic is to place all of the key components. For this initial design this includes the microcontroller chip, a voltage regulator, a microUSB connector, and a programming connector.

For more complex designs it usually makes more sense to completely design each sub-circuit first, then merge them all together. Depending on the design complexity (and personal preference) you may also want to place each sub-circuit on its own separate sheet. This keeps the schematic from becoming a huge, overwhelming monster on a single sheet.

Capacitors

Next, we'll place all of the various capacitors. For the most part you can think of capacitors as tiny little rechargeable batteries that hold electrical charge and help to stabilize the voltage on a supply line.

We'll start by placing a 4.7uF capacitor on the input pin of the linear regulator. This is the 5VDC input voltage supplied by an external USB charger. This voltage is fed into a TLV70233 linear regulator which steps the voltage down to 3.3V since the microcontroller can only be supplied by a maximum of 3.6V.

Another 4.7uF capacitor is placed on the output of the regulator as close to the pin as possible. This capacitor serves to store charge to supply transient loads and it acts to stabilize the internal feedback loop of the regulator. Without an output capacitor most regulators will begin to oscillate.

Decoupling capacitors must be placed as close as possible to the microcontroller supply pins (VDD). It's always best to refer to the microcontroller datasheet in regards to their recommendations for decoupling capacitors.

The datasheet for the STM32F042 recommends a 4.7uF and a 100nF capacitor be placed next to each of the two VDD pins (input supply pins). It also recommends 1uF and 10nF decoupling capacitors be placed near the VDDA pin.

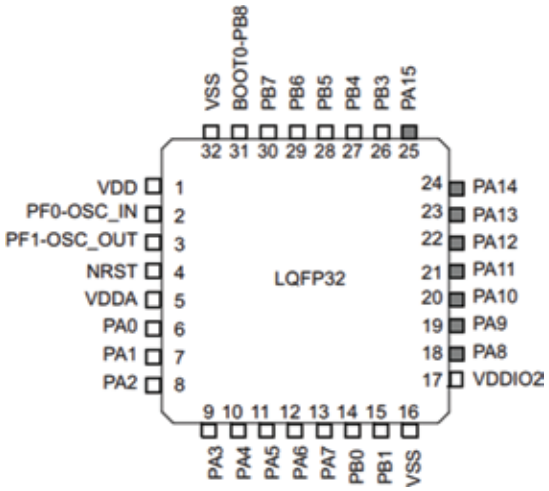
The VDDA pin is the supply for the internal analog-to-digital (ADC) converter and must be especially clean and stable. We're not using the ADC in this first tutorial but we will in a future one.

Note that you'll commonly see two capacitor sizes specified together for decoupling purposes. For example, 4.7uF and 100nF capacitors.

The larger 4.7uF can store more charge which helps stabilize the voltage when large spikes in load current are required. The smaller capacitor serves mainly to filter out any high-frequency noise.

Microcontroller pinout

Although the STM32F042 offers a wide variety of functions such as UART, I2C, SPI, and USB communication interfaces, you won't find any of these functions labeled on the microcontroller pinout. This is because most microcontrollers assign a variety of functions to each pin so as to reduce the number of pins required.



Pinout for the STM32F042 microcontroller in a 32-pin LQFP leaded package.

For example, on the STM32F042 pin 9 is labeled as PA3 which means it is a GPIO pin. Upon startup this function is automatically assigned to this pin. But it also has alternative functions that can be specified in the firmware program.

Pin 9 can be programmed to serve the following functions: receive input pin for UART serial communication, an input to the Analog-to-Digital Converter (ADC), a timer output, or an I/O pin for the capacitive touch sensor controller.

Refer to the pin definition table in the microcontroller datasheet (page 33 for the STM32F042) which shows all of the various functions available for each pin. Always be sure to confirm that two functions required for your product don't overlap on the same pins.

Clock

All microcontrollers require a clock for timing purposes. This clock is just an accurate oscillator. Microcontrollers execute programmed commands sequentially with each tick of the clock.

The simplest option, if available on the selected microcontroller, is to use the internal clock. This internal clock is known as an RC oscillator clock because it uses the timing characteristics of a resistor and capacitor.

The major downside to an RC oscillator is accuracy. Resistors and capacitors (especially those embedded inside a microchip) vary significantly from unit to unit causing the oscillator frequency to vary. Temperature also significantly impacts the accuracy.

An RC oscillator is fine for simple applications, but if your application requires accurate timing then it won't be sufficient. For this initial tutorial we're going to use the internal RC clock to keep things simple. In future tutorials we'll improve the design by adding a much more precise, external crystal-based oscillator.

Programming Connector

Programming an STM32 is done via one of two protocols: JTAG or Serial Wire Debug (SWD). More advanced versions of the STM32 (STM32F1 and higher) offer both JTAG and SWD programming interfaces. The STM32F0 subseries offers only the simpler SWD programming interface so that is what we will focus on for this tutorial.

The SWD interface requires only 5 pins. They are SWDIO (data input/output), SWCLK (clock signal), NRST (reset signal), VDD (supply voltage) and ground.

Unfortunately the ST-LINK programmer device that you'll use to program the STM32 uses a 20-pin JTAG connector (with SWD functionality). This connector is quite large and is not practical for smaller board designs.

Instead, you can use a 20-pin to 10-pin adapter board such as this one from Adafruit so you can use a smaller 10-pin connector on your board.

For this tutorial we will use the 10-pin connector. If that is still too large for your project then you can always use a 5-pin header and jumper wires from the 20-pin programmer output to connect only the 5 lines required for SWD programming.

Power

The last part of the schematic we'll cover is the power section. The STM32 microcontroller can be powered with a supply voltage from 2.0 to 3.6V. Unless you have a variable power supply, you'll need an on-board regulator to provide the appropriate supply voltage.

For this design we'll power the board using an external USB charger which outputs 5 VDC. This voltage will then feed into a linear voltage regulator (TLV70233 from Texas Instruments) which steps it down to a stable 3.3V.

The STM32 requires a maximum of only 24mA assuming none of the GPIO pins are sourcing any current (each GPIO pin can source up to 25mA). The absolute maximum current the STM32 will ever require is 120mA assuming various GPIO pins are sourcing current.

The TLV70233 is rated for up to 300mA which should be more than sufficient for this initial design. In future tutorials, as we add additional functions, we may need to revisit this to ensure the regulator can handle the required system current.

In future tutorials we'll significantly improve the power circuit design by adding in a rechargeable lithium battery that can be recharged via the USB port.

Electrical Rules Check

The final step of designing the schematic circuit diagram is to perform a verification step called an Electrical Rules Check (ERC). This verification step checks for errors such as shorts between nets, nets with only one pin, superimposed pins, and unconnected pins.

You can also setup various pin type errors. For example, if an output pin is connected to another output you will get an error. Or if an output pin is connected to a power supply line you will get an error. DipTrace uses a colored grid matrix that allows you to define which pin type connections will give you errors or warnings.

Printed Circuit Board (PCB) Design

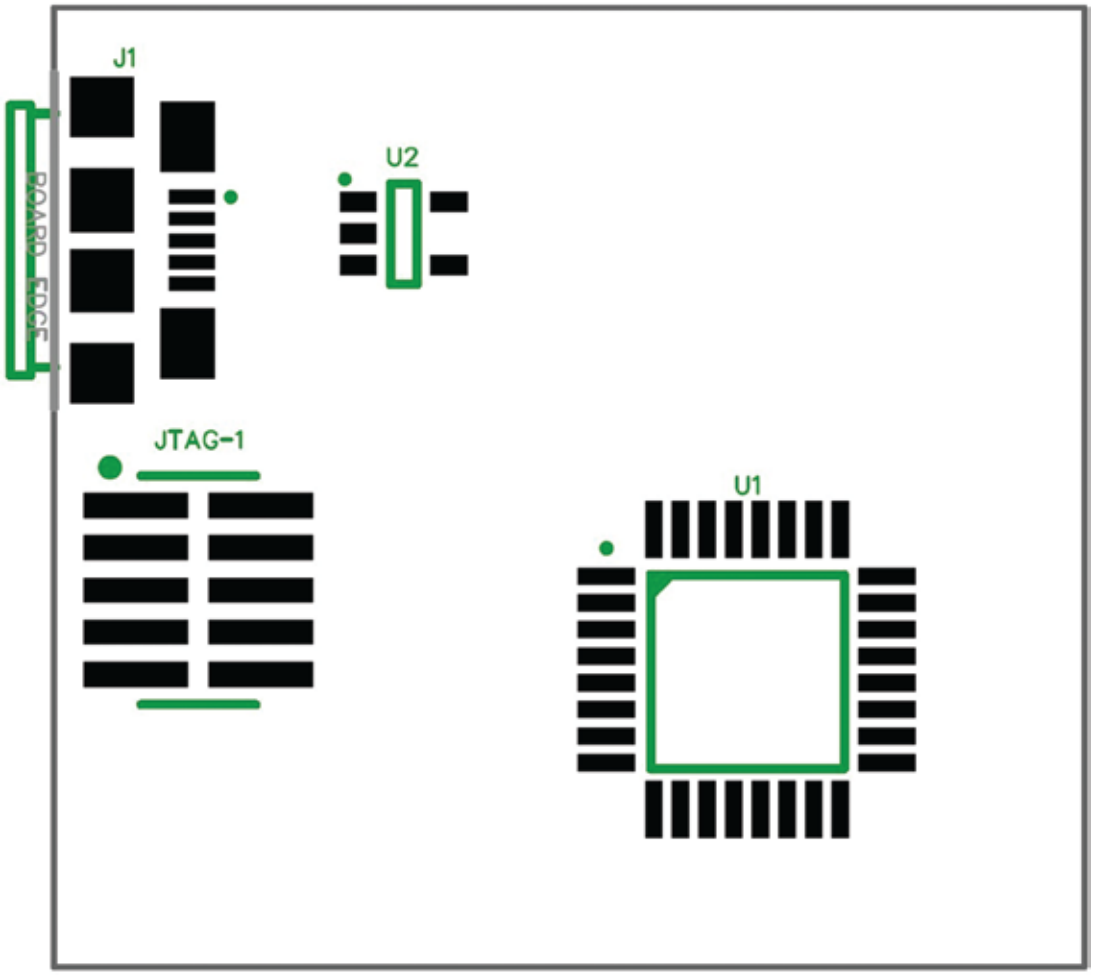
Once the schematic design is completed, it's time to design the Printed Circuit Board. Begin by inserting all of the components into the PCB layout. In DipTrace, you can use the "Convert to PCB" function in the schematic to automatically create the PCB with all of the components inserted.

Component Placement

Although all of the components have been inserted, it's your job to determine exactly where each component is placed on the PCB.

Most PCB design software packages include an auto-placement feature that places components with the goal of minimizing routing lengths. But I never use it, and it's almost necessary to manually place the components in the best layout.

For our initial tutorial circuit, the component placement is pretty simple. Place the microUSB connector next to the linear regulator with its output as close as possible to the input supply pins (VDD) on the microcontroller. Finally, place the programming connector anywhere that is convenient.

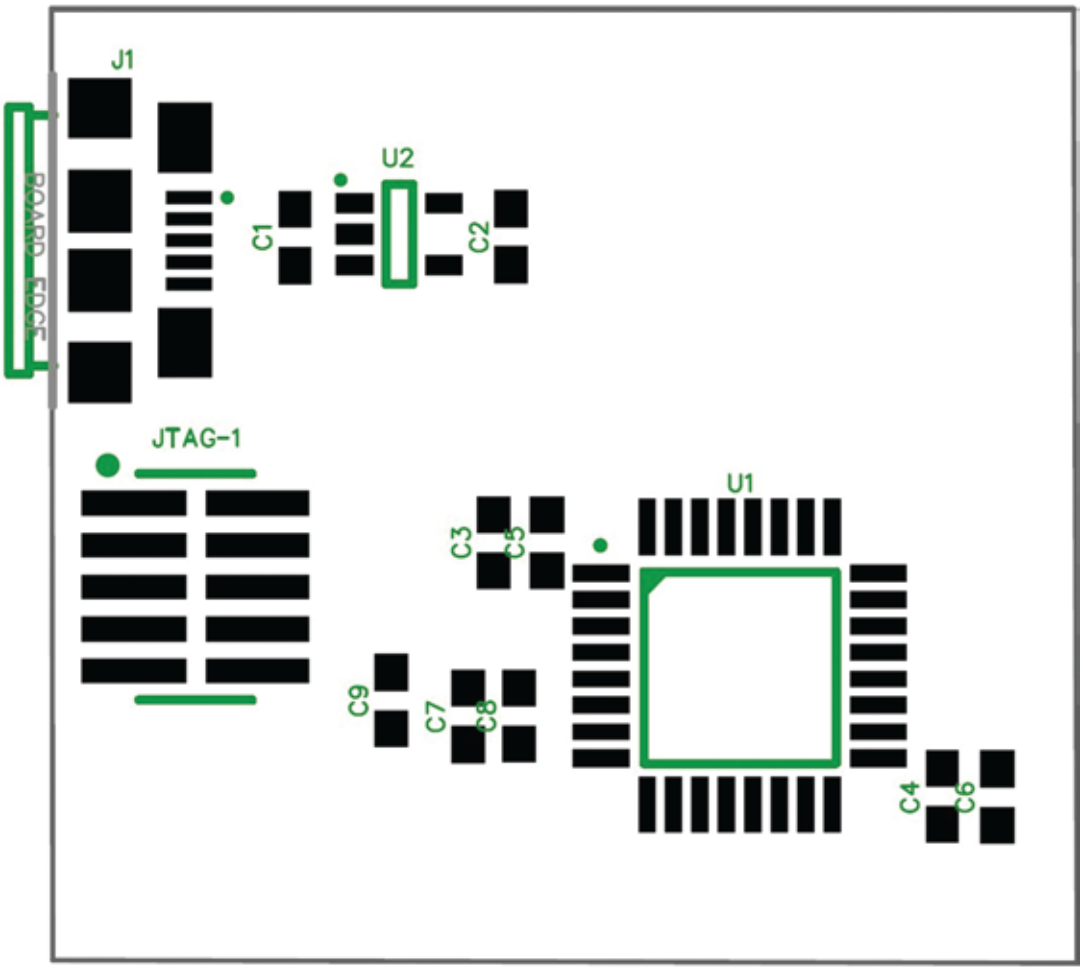


Placement of the critical components in this initial design: the microcontroller (U1), the regulator (U2), the micro USB connector (J1), and the programming connector (JTAG-1).

Once all of the core components are properly placed, your next step is placing all of the passive components (resistors, capacitors, and inductors). For this initial design the only passive components are capacitors.

One key aspect of designing electronics that you need to learn is the concept of parasitics. Parasitics are passive components (resistors, capacitors, and inductors) that you don't intentionally design into your circuit. But, nonetheless, they are there and impact performance.

For example, although a signal trace is intended to be a perfect short, it in fact has some finite resistance, capacitance, and inductance all of which become more significant as the trace length increases, and as the number of bends and vias increase.



Placement of all critical components (U1, U2, J1, and JTAG-1) and passive components (capacitors).

So this means that if a voltage source is located far away from the load, which is the STM32 microcontroller in this case, there is essentially a resistor between the load and the source (neglecting any capacitance and inductance).

If the microcontroller all of a sudden requires a fast spike of current then it will cause a voltage drop across this trace resistor.

So even though the voltage regulator's output may be a perfect 3.30V, the voltage at the microcontroller pin will be lower during this current surge. Decoupling capacitors are used to solve this problem.

Remember, capacitors are like little batteries that store electrical charge. Placing them right at the microcontroller's supply pins allows them to supply any fast, transient current needs of the microcontroller.

Once the transient load disappears the capacitors are recharged by the power supply so they are ready for the next transient increase in load current.

PCB Layer Stack

A printed circuit board is made up of stacked layers. Conducting layers are separated by insulating layers. The minimum number of conducting layers is two. This means the top layer and the bottom layer can be used for routing signals, and these two layers are separated by an internal insulating layer. For this tutorial we'll start with a 2-layer board to keep things simple. But as the circuit complexity increases you'll find it necessary to add additional layers.

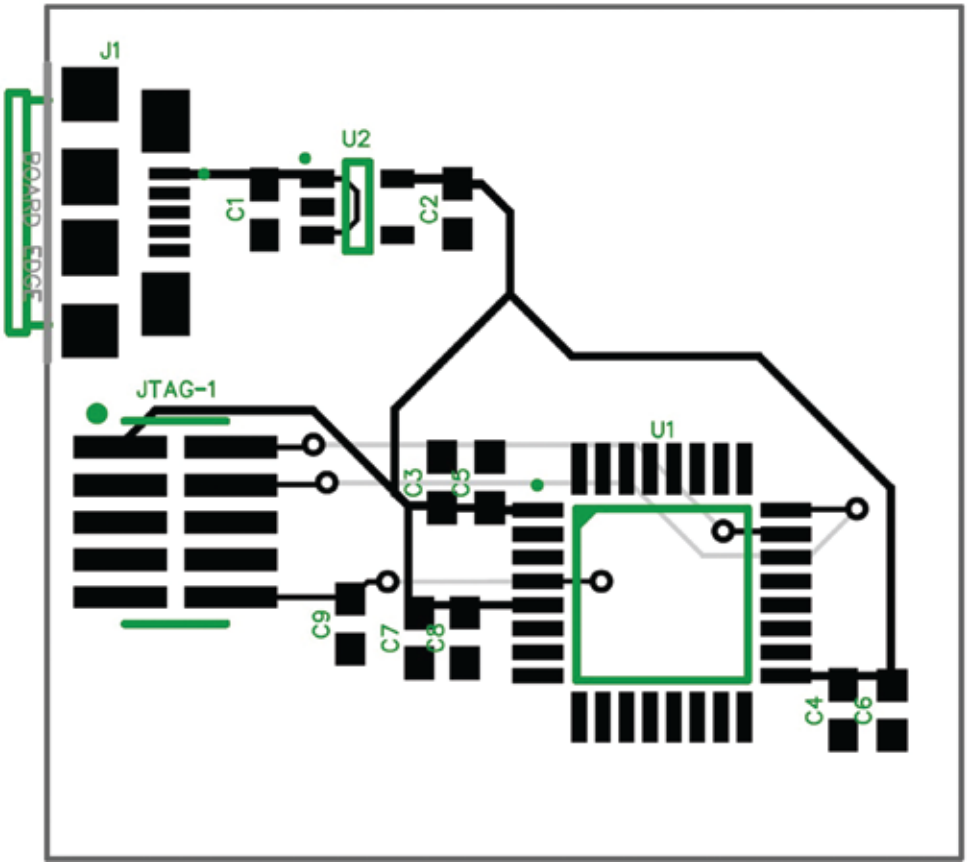
The number of conducting layers is always an even number, so you can have a board with 2,4,6,8,10,12 conducting layers. Most designs will require 4-6 layers, and more advanced designs may require 8 or more layers.

Routing

Once all of the components have been properly placed it's now time to perform the necessary routing. There are two options for routing: manual and automatic.

For auto-routing in DipTrace you simply select Route -> Run Autorouter and the software will automatically do all of the routing.

Unfortunately, auto-routers in general do a horrible job, and in almost all cases you will need to manually do all of the routing. For this tutorial we will be doing all of the routing manually.

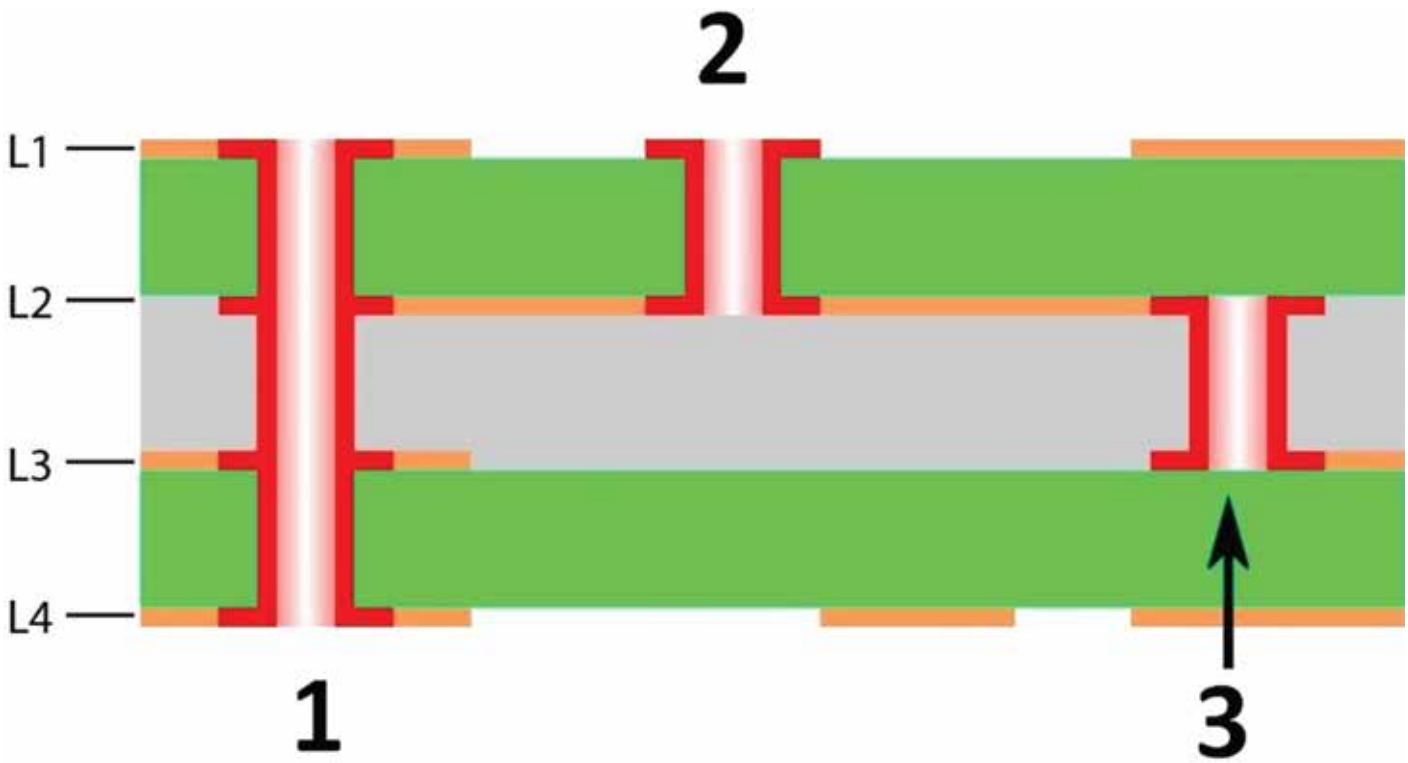


Routing the PCB (black traces on the top layer, gray traces on the bottom layer)

When routing on a PCB you want to minimize the length of each trace as much as possible. You also want to minimize the number of vias and avoid any 90 degree bends in the traces. These recommendations are especially critical for high power traces and high speed signals.

A via is a hole between layers with conducting material that allows you to connect together two traces on different layers. Most vias are what are known as through vias which means the via tunnels through all layers of the board.

Through vias are the simplest type to manufacture because they can be drilled after the entire PCB layer stack-up is assembled.



*Via #1 is a classic **through via**, via #2 is a **blind via**, and via #3 is a **buried via**.*

Vias that only tunnel through a subset of layers are known as buried and blind vias. Blind vias connect an external layer to an internal layer (so one end is hidden inside the PCB stack-up). Buried vias connect two internal layers and are completely hidden on the assembled PCB.

Blind and buried vias allow you to pack a design more tightly. This is because they don't take up space on the layers not using them. Through vias on the other hand consume space on all layers.

However, be aware that blind and buried vias drastically increase the prototype cost for your board. In most situations you should restrict yourself to only using through vias. Only exceptionally complex designs, that must fit in an exceptionally small space, should likely ever require these more advanced via types.

When routing any high current power lines you need to ensure the trace width is capable of carrying the necessary current. If you run too much current through a PCB trace it will overheat and melt causing the board to become defective.

To determine the necessary trace width I like to use a PCB trace width calculator. To determine the required trace width you need to first know the trace thickness for your specific PCB process.

PCB manufacturers allow you to select various conducting layer thicknesses, usually measured in ounces per square foot (oz/ft2) but also measured in mils (a mil is one thousandth of an inch) or millimeters.

A common conducting layer thickness is 1 oz/ft2. In this tutorial I've made the power supply lines 10 mils wide. Using the calculator linked to above shows that a 1 oz/ft2 trace measuring 10 mils wide can actually carry almost 900mA of current.

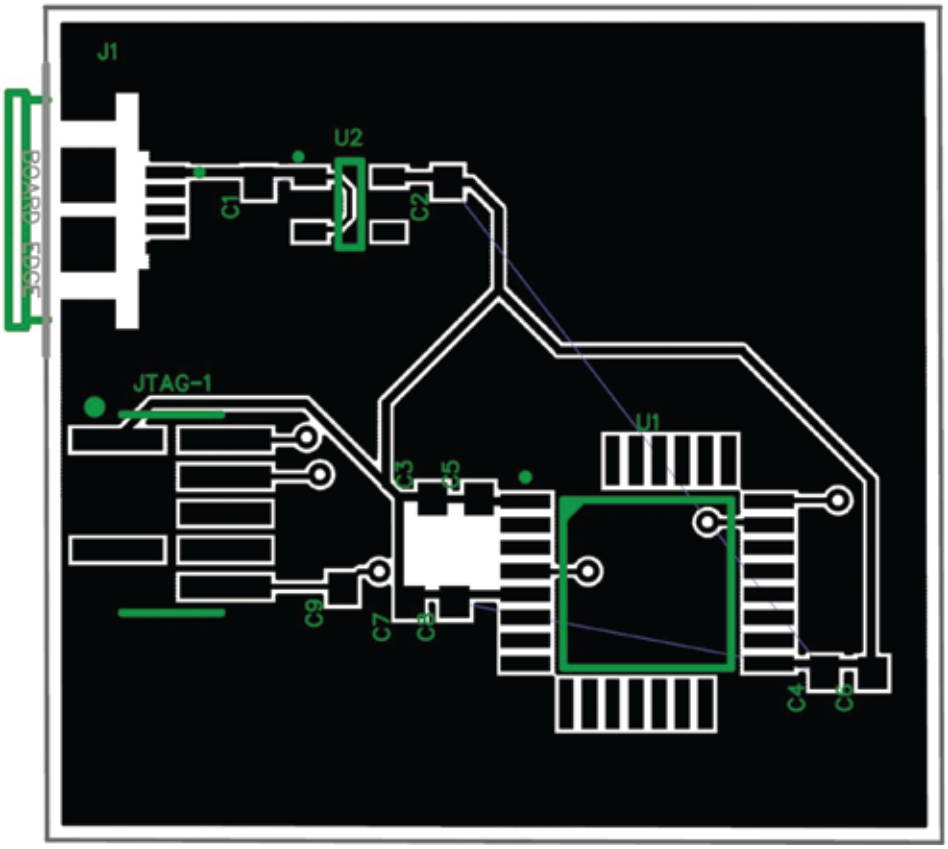
This is obviously much more than we'll need, and I could have easily made the supply lines much more narrow. In the first tutorial I showed that the absolute maximum current required by the STM32F042 is 120mA. Perhaps surprisingly, to handle 120mA we only need a trace width of 0.635 mils!

The minimum trace width allowed by most processes is 4-6 mils. The minimum width traces can be easily used for the supply lines in this design. That being said, the wider the trace the less the resistance and the more stable the supply voltage at each component.

Unless space is extremely tight you should always over design the power supply traces. In fact, in many cases you'll want the power supply routing on its own layer so you can maximize the routing width.

Finally, in the calculator you'll notice the requirements are different for internal layers versus external layers. For this simple, 2-layer design both layers are external so the "Results for External Layers in Air" is what we need to use.

Internal layers can carry much less current because they don't get the cooling effect of being exposed to air so the traces will overheat with much less current.



The completed Printed Circuit Board (PCB) layout for this initial tutorial.

Verification

Once all of the routing is done it's time to perform verifications to ensure everything is correct. This is where automation really works well and any PCB design tool will offer automatic verification features.

There are broadly two types of verification: design rules check (DRC) and schematic comparison.

The DRC verifies that all PCB process design rules have been followed. This includes rules such as the minimum allowed trace width, the minimum spacing allowed between traces, the minimum spacing between a trace and the board edge, and so on.

In order to run a DRC verification its necessary that you first obtain all of the design rules for the specific PCB process you will be using.

Each PCB prototyping process has slightly different rules so you must have the correct rules before proceeding. You can get the design rules for your specific process from your PCB prototype supplier. In DipTrace, you define the design rules by selecting Verification->Design Rules. Once all rules have been correctly defined, you can run the DRC by selecting Verification->Check Design Rules.

After you have verified that your PCB layout adheres to all of the process design rules, it's now time to verify that your PCB design matches your schematic diagram. To do this in DipTrace you simply select Verification->Compare to Schematic.

In future tutorials, I'll show you various types of DRC and schematic comparison errors, and how to fix them.

Generating Gerbers

Once you have verified the design adheres to the process design rules and matches the schematic diagram, it's time to order your PCB prototypes.

In order to do this you'll need to convert your PCB layout design (which is currently stored in a proprietary file format) into the industry standard file format known as Gerber.

The Gerber format outputs each layer of your PCB design as a separate file. The generated layers are much more than just the conducting layers of your board. Some of those layers include:

1) ***Silk layers*** – Includes the text and component designators.
2) ***Assembly layers*** - Similar to silk layers but with specific assembly instructions.
3) ***Solder mask layers*** - Indicates the green stuff on a PCB that covers up any con-ductors that you don't want to solder to. This prevents accidental shorts during soldering.
4) ***Solder paste layers*** - Used to precisely place solder paste where soldering will occur.

You will also need to generate what is known as a Pick-and-Place file which includes the coordinates and orientation for all of the components. This file is used by the manufacturer's automatic component placement machines.

Finally, you need to output a drill file that provides the exact location and size of any holes such as vias and mounting holes.

Once you have the Gerbers, the Pick-and-Place file, and the drill file you can send those files to any prototype shop or manufacturer for production of your board.

Finally, you need to output a drill file that provides the exact location and size of any holes such as vias and mounting holes.

Once you have the Gerbers, the Pick-and-Place file, and the drill file you can send those files to any prototype shop or manufacturer for production of your board.

Summary

In this section you've learned how to design a system-level block diagram, select all of the critical components, design the full schematic circuit diagram, design the Printed Circuit Board (PCB) layout, and order prototypes of your completed microcontroller PCB design.

PREDICTABLE
DESIGNS