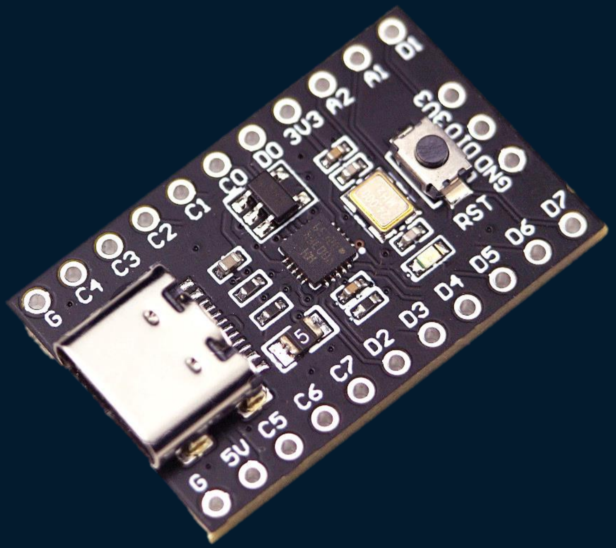
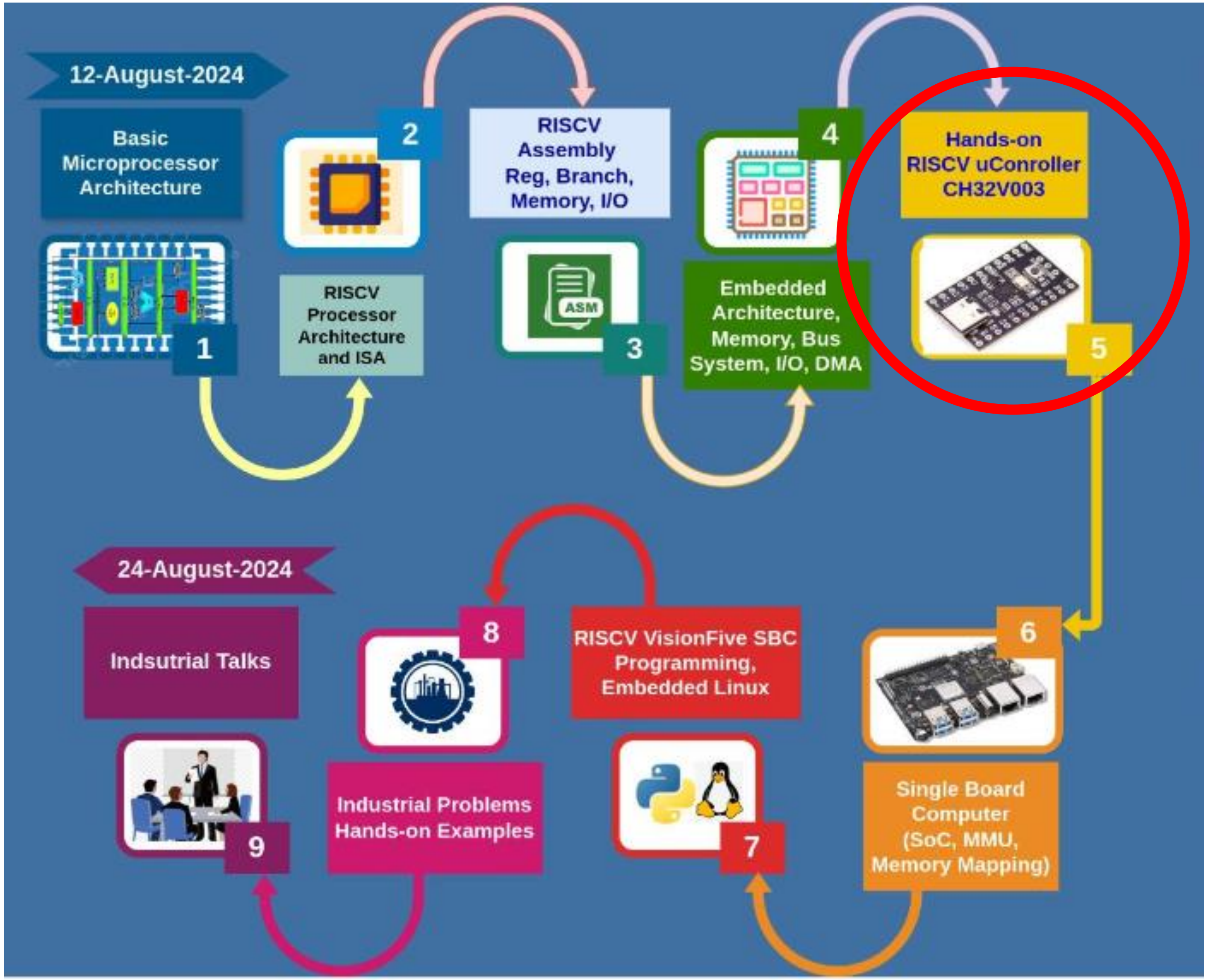
Embedded Systems

Day 4

Microprocessor Architecture
CH32V003



Program Flow



Contents

Contents

Characteristics of Low-Cost Microcontrollers

Cost

- **Definition of "Low-Cost"**
  - Typically refers to microcontrollers priced under $5.
- **Typical Price Range**
  - **Entry-Level**: $1 to $3
  - **Mid-Range**: $3 to $5

Features

- **Basic Processing Power**
  - Simple core with limited computational capabilities.
  - Low clock speeds, typically from a few MHz to 100 MHz.
- **Integrated Peripherals**
  - Basic set of peripherals such as GPIO, timers, and communication interfaces.
- **Limited Memory and Storage**
  - Flash memory: 1 KB to 64 KB.
  - RAM: 128 B to 8 KB.

Characteristics of Low-Cost Microcontrollers

Trade-offs

- **Performance vs. Cost**
  - Lower cost often means reduced processing power and fewer features.
- **Power Consumption**
  - Generally designed for low-power applications, often with power-saving modes.
- **Functionality**
  - Limited compared to higher-end microcontrollers; often lacks advanced features such as high-speed communication interfaces or complex timers.

Popular Low-Cost Microcontroller Families

Microcontroller Family	Typical Price Range	Key Features	Common Applications
Microchip PIC	$1 - $3	8-bit/16-bit, basic peripherals	Simple control systems, educational projects
Atmel AVR (e.g., ATtiny)	$1 - $3	8-bit, rich set of peripherals	Hobbyist projects, educational kits
STMicroelectronics STM32	$3 - $5	32-bit, various performance levels	Embedded applications, consumer electronics
NXP/Freescale Kinetis	$3 - $5	32-bit, balance of performance and cost	Embedded systems, automotive applications

Applications of Low-Cost Embedded Systems

01

Manufacturing Equipment

02

Domestic Appliances

03

Audio/Video Equipment

04

Gamming

05

Telecommunication

06

Medical Devices

07

Cars And Vehicles

08

Sensor Integration

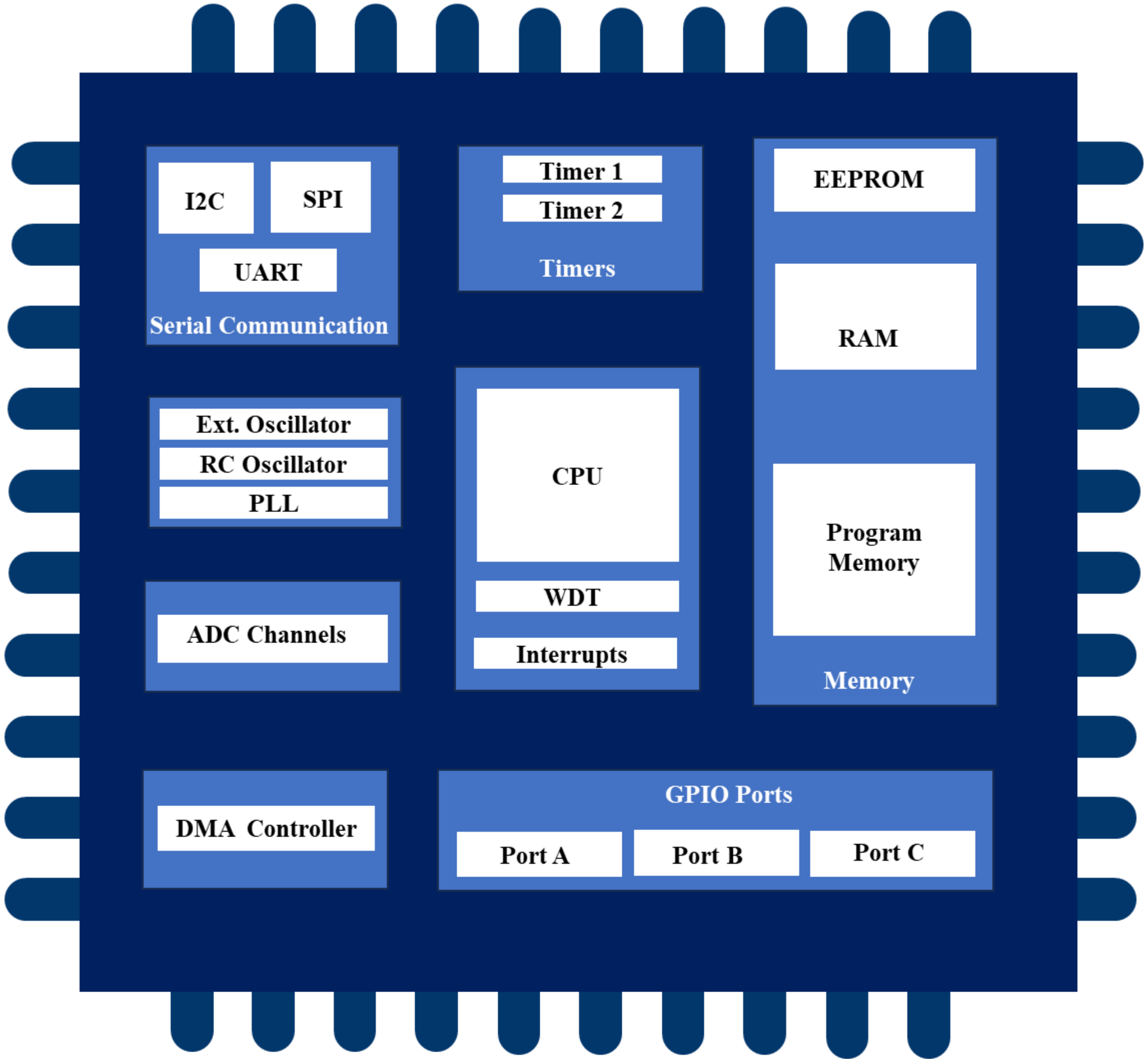
Basic Features in Low-Cost Microcontroller

1. **Basic Processing Power**
   - **Microcontrollers**: Often use simple, cost-effective microcontrollers with lower clock speeds and fewer cores.
   - **Limited Performance**: May have modest processing power suitable for basic tasks and control functions.
2. **Minimal Memory**
   - **Limited RAM**: Typically equipped with a small amount of RAM to reduce costs.
   - **Flash Storage**: Often use integrated flash memory for program storage, with limited capacity.
3. **Basic I/O Interfaces**
   - **Digital I/O**: Basic input and output capabilities for interacting with peripherals.
   - **Analog I/O**: Basic analog-to-digital converters (ADCs) for reading sensor data, if required.
   - **Serial Communication**: Commonly include UART, SPI, or I^2C interfaces for communication with other devices.
4. **Low-Power Operation**
   - **Energy Efficiency**: Designed to operate with minimal power consumption, often including sleep modes to extend battery life.

Basic Features in Low-Cost Microcontroller

5. **Simple Peripherals**
    - **Basic Sensors and Actuators**: Support for a limited number of standard sensors (e.g., temperature, humidity) and actuators (e.g., LEDs, motors).
    - **Limited Connectivity**: May not include advanced connectivity options like Wi-Fi or Bluetooth, focusing instead on basic wired or wireless communication.
6. **Cost-Effective Development Tools**
    - **Open-Source IDEs**: Use of free or low-cost integrated development environments (IDEs) and toolchains.
    - **Community Support**: Often benefit from strong community support and open-source libraries for development.
7. **Basic Operating Systems**
    - **RTOS**: May use a simple real-time operating system (RTOS) if an OS is required, but many operate without a full-fledged OS.
    - **Bare Metal**: Some systems run directly on hardware without any operating system, reducing overhead and cost.

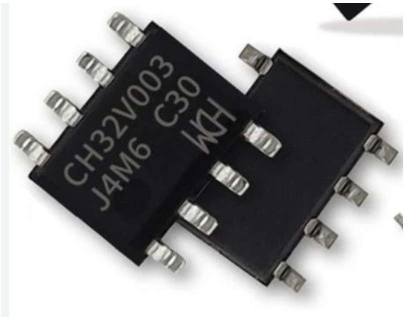
Basic Hardware in Low-Cost Microcontroller

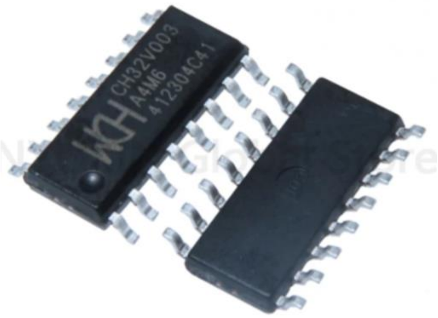


Contents

Introduction to CH32 Family of Microcontrollers

- CH32V003 series are industrial-grade general-purpose microcontrollers designed based on 32-bit RISC-V instruction set and architecture.
- It adopts QingKe V2A core, RV32EC instruction set, and supports 2 levels of interrupt nesting.
- The series are mounted with rich peripheral interfaces and function modules.
- Its internal organizational structure meets the low-cost and low-power embedded application scenarios.





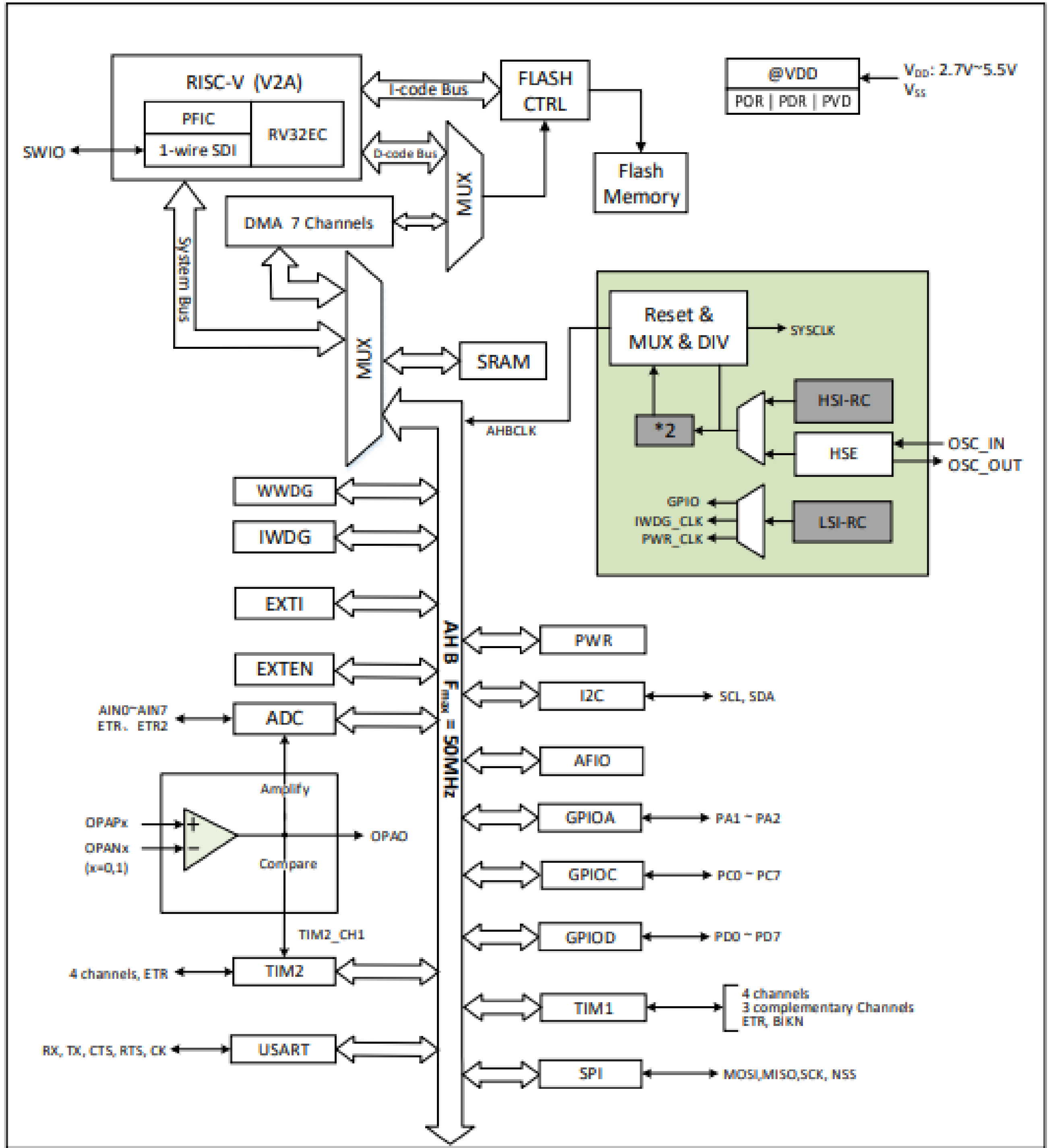
Introduction to CH32 Family of Microcontrollers

• RISC-V MCUs • Cortex-M MCUs • RISC Assembly MCUs • Featured Application MCUs • BLE MCUs • E8051 USB M

Part NO.	Freq	Flash	SRAM	GPIO	Adv/GP Timer	WDOG	RTC	ADC Unit/CH	Touchkey
CH32V003J4M6	48MHz	16K	2K	6	1/1	2	-	1/6	-
CH32V003A4M6	48MHz	16K	2K	14	1/1	2	-	1/6	-
CH32V003F4U6	48MHz	16K	2K	18	1/1	2	-	1/8	-
CH32V003F4P6	48MHz	16K	2K	18	1/1	2	-	1/8	-
CH32X035R8T6	48MHz	62K	20K	60	2/1	2	-	1/14	14
CH32X035C8T6	48MHz	62K	20K	46	2/1	2	-	1/10	10
CH32X035G8U6	48MHz	62K	20K	27	2/1	2		1/10	10
CH32X035G8R6	48MHz	62K	20K	26	2/1	2	-	1/11	11

System Architecture: CH32V003

- The CH32V003 series is designed based on the **RISC-V instruction set**, and its architecture interacts the core, arbitration unit, **DMA module**, **SRAM** storage and other parts through multiple buses.
- The design integrates a general-purpose **DMA controller** to reduce the CPU load and improve access efficiency, as well as data protection mechanisms, automatic clock switching protection mechanisms and other measures to increase system stability
- **Clock tree** hierarchy management to optimize power consumption of peripherals.
- **Data protection** mechanisms and clock security systems for enhanced stability.



RISC-V (V2A)
PFIC
1-wire SDI
RV32EC
SWIO
I-code Bus
D-code Bus
FLASH CTRL
Flash Memory
@VDD
POR | PDR | PVD
V_{DD}: 2.7V~5.5V
V_{SS}
DMA 7 Channels
MUX
System Bus
MUX
SRAM
Reset & MUX & DIV
SYSCLK
AHBCLK
*2
HSI-RC
HSE
OSC_IN
OSC_OUT
LSI-RC
GPIO
IWDG_CLK
PWR_CLK
WWDG
IWDG
EXTI
EXTEN
ADC
AIN0~AIN7
ETR、ETR2
Amplify
Compare
OPAPx
OPANx
(x=0,1)
OPAO
TIM2_CH1
TIM2
4 channels, ETR
USART
RX, TX, CTS, RTS, CK
AHB F_{max} = 50MHz
PWR
I2C
SCL, SDA
AFIO
GPIOA
PA1 ~ PA2
GPIOC
PC0 ~ PC7
GPIOD
PD0 ~ PD7
TIM1
4 channels
3 complementary Channels
ETR, BKIN
SPI
MOSI,MISO,SCK, NSS

Bus Architecture

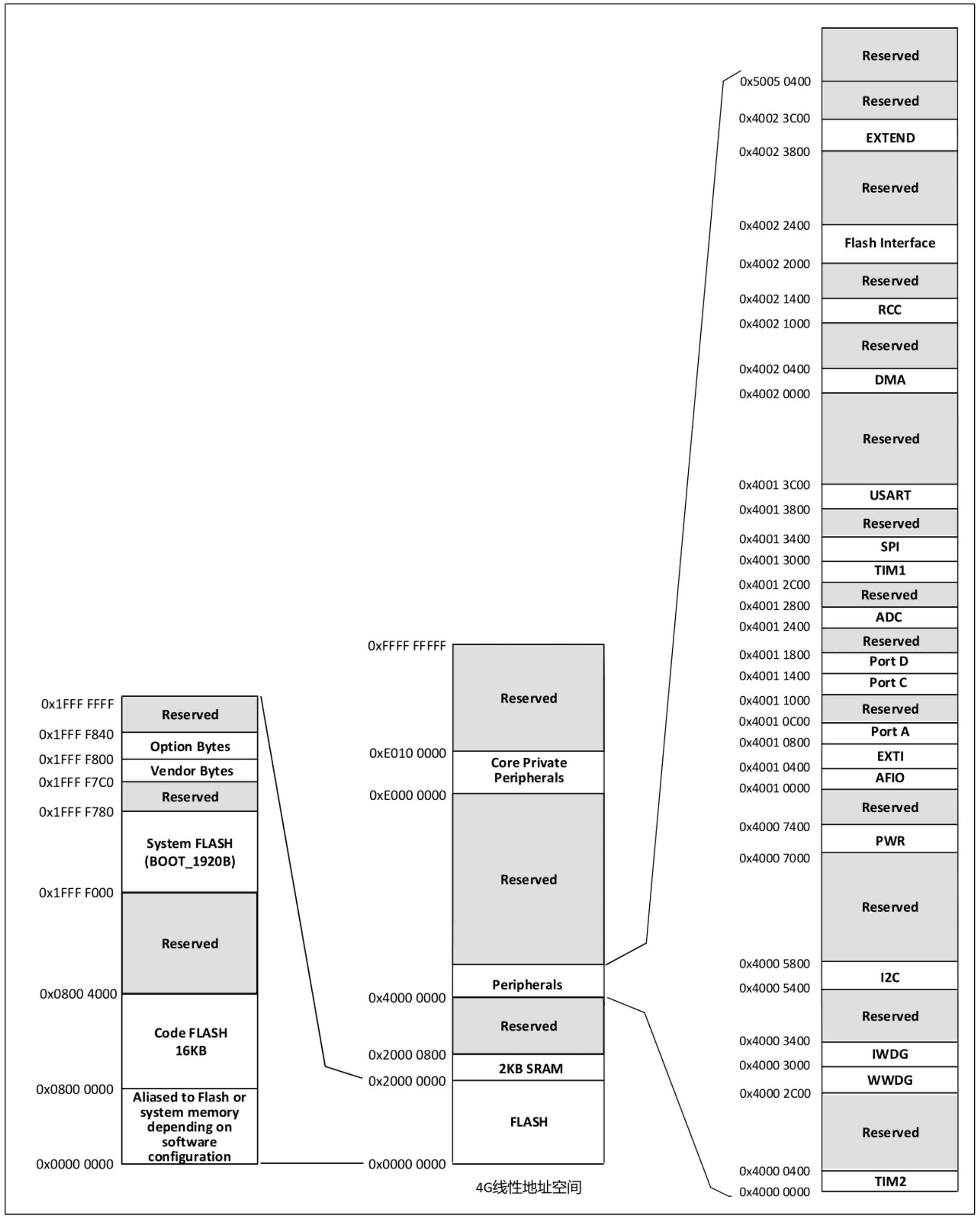
Bus Type	Connection	Function
Instruction Bus (I-Code)	Core to FLASH instruction interface.	Handles instruction prefetching and loading.
Data Bus (D-Code)	Core to FLASH data interface.	Manages constant data loading and debugging.
System Bus	Connects the core to the bus matrix.	Coordinates access to core, DMA, SRAM, and peripherals
DMA Bus	Links to the bus matrix.	Handles DMA operations through AHB master interface.
Bus Matrix	Coordinates access between system bus, data bus,	Handles DMA operations through AHB master interface.

Memory Architecture

- The CH32V003 family contains program memory, data memory, core registers, peripheral registers, and more, all addressed in a 4GB(2^32) linear space.
- System storage stores data in small-end format, i.e., low bytes are stored at the low address and high bytes are stored at the high address.

Memory Allocation

- Built-in 2KB SRAM, starting address 0x20000000, supports byte, half-word (2 bytes), and full-word (4 bytes) access.
- Built-in 16KB program Flash memory (Code Flash) for storing user applications.
- Built-in 1920B System memory (bootloader) for storing the system bootloader (factory-cured bootloader).
- Built-in 64B space for vendor configuration word storage, factory-cured and unmodifiable by users. Built-in 64B space for user-option bytes storage.

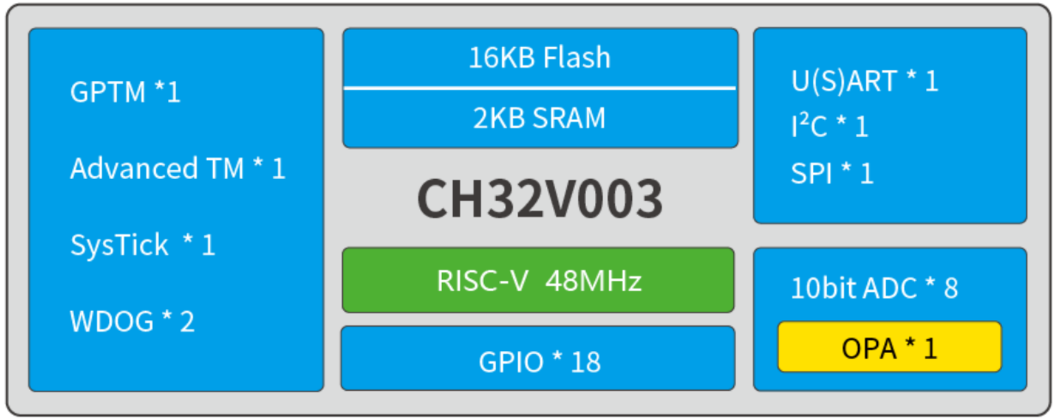


0x1FFF FFFF
Reserved
0x1FFF F840
Option Bytes
0x1FFF F800
Vendor Bytes
0x1FFF F7C0
Reserved
0x1FFF F780
System FLASH (BOOT_1920B)
0x1FFF F000
Reserved
0x0800 4000
Code FLASH 16KB
0x0800 0000
Aliased to Flash or system memory depending on software configuration
0x0000 0000
0xFFFF FFFFF
Reserved
0xE010 0000
Core Private Peripherals
0xE000 0000
Reserved
Peripherals
0x4000 0000
Reserved
0x2000 0800
2KB SRAM
0x2000 0000
FLASH
0x0000 0000
4G线性地址空间
Reserved
0x5005 0400
Reserved
0x4002 3C00
EXTEND
0x4002 3800
Reserved
0x4002 2400
Flash Interface
0x4002 2000
Reserved
0x4002 1400
RCC
0x4002 1000
Reserved
0x4002 0400
DMA
0x4002 0000
Reserved
0x4001 3C00
USART
0x4001 3800
Reserved
0x4001 3400
SPI
0x4001 3000
TIM1
0x4001 2C00
Reserved
0x4001 2800
ADC
0x4001 2400
Reserved
0x4001 1800
Port D
0x4001 1400
Port C
0x4001 1000
Reserved
0x4001 0C00
Port A
0x4001 0800
EXTI
0x4001 0400
AFIO
0x4001 0000
Reserved
0x4000 7400
PWR
0x4000 7000
Reserved
0x4000 5800
I2C
0x4000 5400
Reserved
0x4000 3400
IWDG
0x4000 3000
WWDG
0x4000 2C00
Reserved
0x4000 0400
TIM2
0x4000 0000

Contents

Detailed study of the CH32V003 Controller

- **Main Features**
- QingKe 32-bit RISC-V2A processor, supporting 2 levels of interrupt nesting
- Maximum 48MHz system main frequency 2KB SRAM,
- 16KB Flash Power supply voltage: 3.3/5V
- Multiple low-power modes: Sleep, Standby
- Power on/off reset, programmable voltage detector
- 1 group of 1-channel general-purpose DMA controller
- 1 group of op-amp comparator
- 1 group of 10-bit ADC
- 1×16-bit advanced-control timer, 1×16-bit general-purpose timer
- 2 WDOG, 1×32-bit SysTick
- 1 USART interface,
- 1 group of I2C interface,
- 1 group of SPI interface 18 I/O ports, mapping an external interrupt 64-bit chip
- unique ID 1-wire serial debug interface (SDI)



Detailed study of the CH32V003 Controller

The C-Type USB port cannot be used directly for programming purposes.

We need a programmer/debugger device to download program code on the microcontroller. There are different types of programming devices given by WCH but we have use one compatible with our microcontroller.

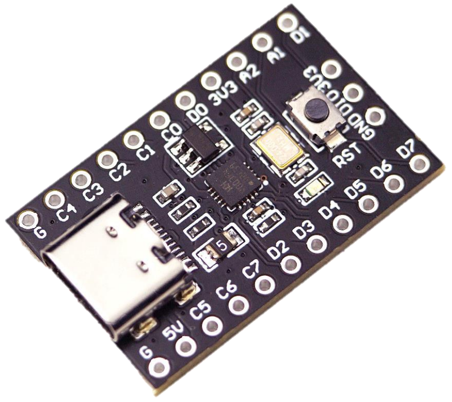
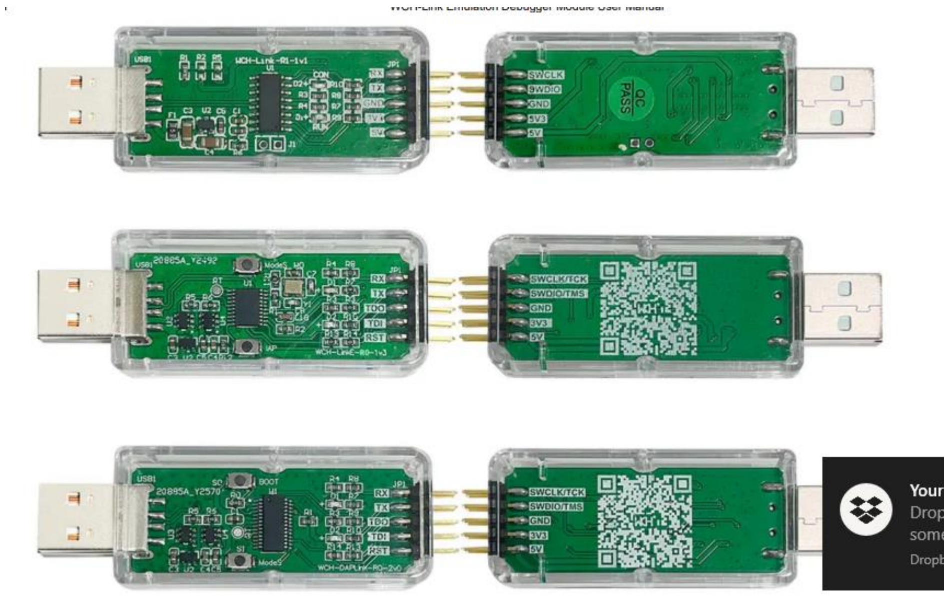
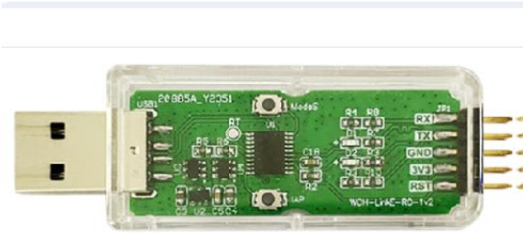


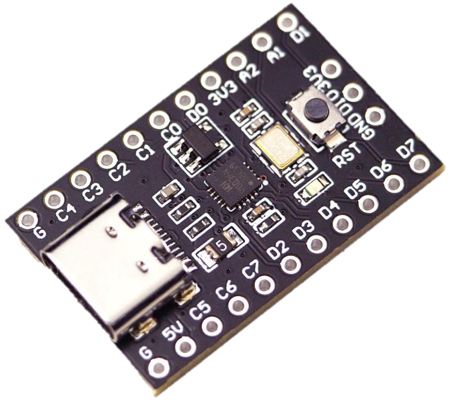
Table 6 Link supported chip model

Common chip models	WCH-Link	WCH-LinkE	WCH-DAPLink
CH32V003	×	√	×
CH32V10x/CH32V20x/cCH32V30x/CH569/CH573/CH583	√	√	×
CH32F10x/CH32F20x/CH579/friendly chips that support SWD protocol	√	√	√
friendly chips that support JTAG interface	×	√	√



Detailed study of the CH32V003 Controller





Host Machine
(Computer)

↔

WCH-LINKE
(Programmer)

1-Wire
SWDIO
↔

Debug Module
of CH32v003

Detailed study of the CH32V003 Controller

Mode	Status LED	IDE	Support chip
RISC-V	Blue LED is always off wh en idle	MounRiver Studio	WCH RISC-V core chips that support single/dual line debugging
ARM	Blue LED is always on wh en idle	Keil/MounRiver Studio	ARM core chips that support SWD/JTAG protocol

Detailed study of the CH32V003 Controller

RISC-V2A processor

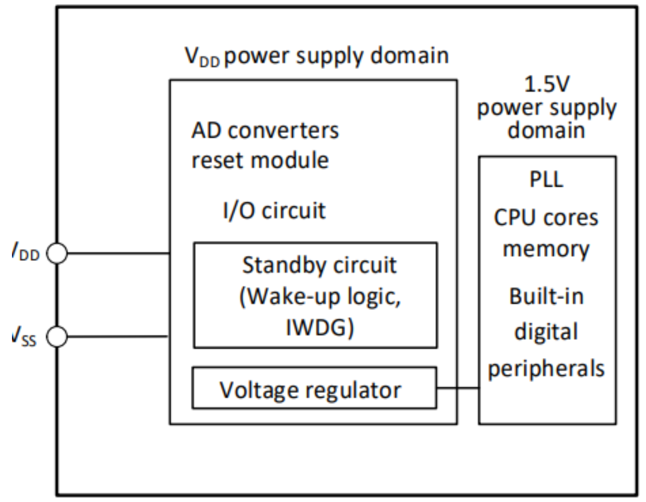
- The RISC-V2A supports the EC subset of the RISC-V instruction set.
- The processor is managed internally in a modular fashion and contains units such as a fast programmable interrupt controller (PFIC), extended instruction support, and more.
- The bus is connected to an external unit module to enable interaction between the external function module and the core.
- RV32EC instruction set, small-end data mode.
- The processor with its minimal instruction set, multiple operating modes, and modular custom expansion can be flexibly applied to different scenarios of microcontroller design, such as small area low-power embedded scenarios.
  - Support machine mode
  - Fast Programmable Interrupt Controller (PFIC)
  - 2-level hardware interrupt stack
  - 1-wire serial debug interface (SDI)
  - Custom extended commands

Detailed study of the CH32V003 Controller

Power supply scheme

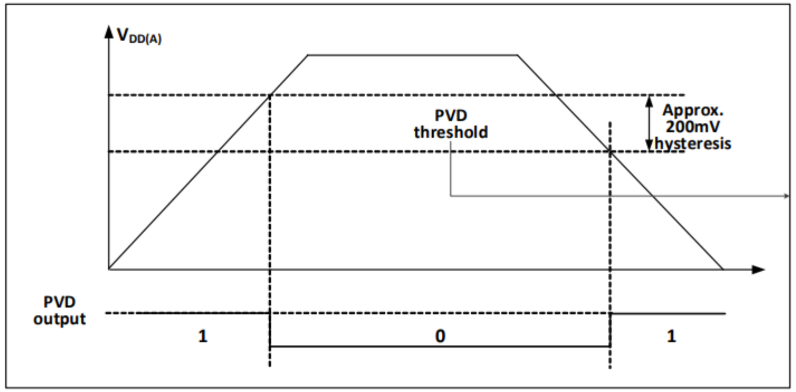
VDD = 2.7～5.5V:

Power supply for some I/O pins and internal voltage regulator (VDD performance gradually deteriorates if less than 2.9V when using ADC).



Power supply monitor

- This product integrates a power-on reset (POR)/power-down reset (PDR) circuit, which is always in working condition to ensure that the system is in supply.
- It works when the power exceeds 2.7V; when VDD is lower than the set threshold (VPOR/PDR), the device is placed in the reset state without using an external reset circuit.



Detailed study of the CH32V003 Controller

Programmable voltage monitor

- In addition, the system is equipped with a programmable **voltage monitor** (PVD), which needs to be turned on by software to compare the voltage of VDD power supply with the set threshold **VPVD**.
- Turn on the corresponding edge interrupt of PVD, and you can receive interrupt notification when VDD drops to the PVD threshold or rises to the PVD threshold. Refer to Chapter 4 for the values of VPOR/PDR and VPVD.

Voltage regulator

After reset, the regulator is automatically turned on, and there are 3 operation modes according to the application mode.

- ON mode: Normal operation, providing stable core power.
- Low-power mode: When the CPU enters Stop mode, system automatically enters Standby mode.

Detailed study of the CH32V003 Controller

Low-power Modes:

The system supports 2 low-power modes, which can be selected for low-power consumption, short start-up time and multiple wake-up events to achieve the best balance.

- **Sleep mode**
- **Standby mode**

External Interrupt/Event Controller

- External interrupt/event controller (EXTI) The external interrupt/event controller contains a total of 8 edge detectors for generating interrupt/event requests.
- Each interrupt line can be independently configured with its trigger event (rising or falling edge or double edge) and can be individually masked; the pending register maintains the status of all interrupt requests.
- EXTI can detect clock cycles with pulse widths less than the internal AHB. 18 general purpose I/O ports are optionally connected to the same external interrupt source.

Detailed study of the CH32V003 Controller

Fast Programmable Interrupt Controller (PFIC)

- The product's built-in Fast Programmable Interrupt Controller (PFIC) supports up to 255 interrupt vectors, providing flexible interrupt management capabilities with minimal interrupt latency.
- The current product manages 4 core private interrupts and 23 peripheral interrupt management, with other interrupt sources reserved. the registers of PFIC are all accessible in machine privileged mode.

- 2 individually maskable interrupts
- Provide a non-maskable interrupt NMI
- Hardware interrupt stack (HPE) support without instruction overhead
- Provide 2-way meter-free interrupt (VTF)
- Vector table supports address or command mode
- Support 2-level interrupt nesting Support break tail link function

Detailed study of the CH32V003 Controller

General-purpose DMA controller

- The system has built-in **1 group** of general-purpose **DMA controllers**, manages **8 channels** in total, and flexibly handles high-speed data transmission from
    1. **Memory to memory,**
    2. **Peripherals to memory,**
    3. **Memory to peripherals,**
- and supports ring buffer mode.
- Each channel has a dedicated hardware **DMA request logic** to support one or more peripherals' access requests to the memory.
- The **access priority**, **transfer length**, **source address** and destination address of the transfer can be configured.
- The main peripherals used by DMA include: general-purpose/advanced-control/basic timers TIMx, DAC, USART, I2C and SPI.

Detailed study of the CH32V003 Controller

Clock And Boot

The system clock source HSI is turned on by default. After the clock is not configured or reset, the internal 24MHz RC oscillator is used as the default CPU clock, and then an external 4~25MHz clock or PLL clock can be additionally selected.

Analog-to-digital Converter (ADC)

- CH32003 is embedded with a **10-bit analog/digital converter** (ADC) that shares up to **eight external channels** and two internal channel samples, with programmable channel sampling times for **single**, **continuous**, **sweep** or **intermittent** conversion.
- Provides analog watchdog function allows very accurate monitoring of one or more selected channels for monitoring channel signal voltages.
- Support for using DMA operation. Supports external trigger delay function. When this function is enabled, the controller delays the trigger signal according to the configured delay time when an external trigger edge is generated, and the ADC conversion is triggered as soon as the delay time is reached.

Detailed study of the CH32V003 Controller

Timers

The timers in the system include an advanced-control timer, a general-purpose timer, two watchdog timers and system time base timer.

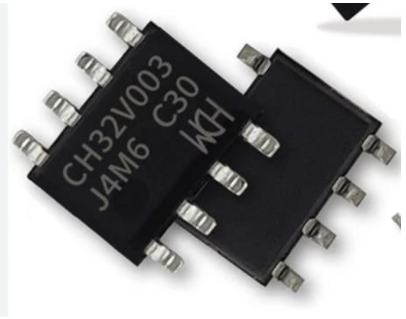
- **Advanced-control timer** The advanced-control timer is a **16-bit** auto-loading up/down counter with a 16-bit programmable prescaler
- **General-purpose timer** The general-purpose timer is a **16-bit** or **32-bit** auto-loading up/down counter with a programmable **16-bit prescaler** and **4 independent channels.**
- **Independent watchdog** The independent watchdog is a configurable 12-bit down counter that supports 7 frequency division factors.
- **Window Watchdog** The window watchdog is a **7-bit down counter** and can be set to free-running. It can be used to **reset the entire system** when a problem occurs. It is driven by the main clock and has an early warning interrupt function; in Debug mode, the counter can be **frozen**.
- **SysTick Timer** QingKe microprocessor core comes with a **32-bit incremental counter** for generating SYSTICK exceptions (exception number: 15), which can be used exclusively in real-time operating systems to provide a "heartbeat" rhythm for the system, or as a standard 32-bit counter. It has an automatic reload function and a programmable clock source.

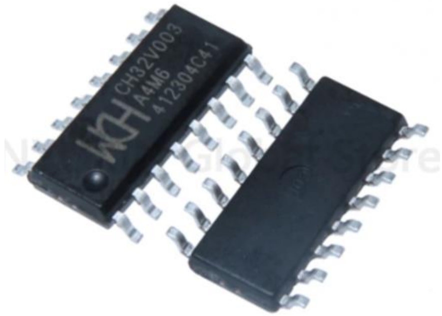
Detailed study of the CH32V003 Controller

Product Packages:

TSSOP20, QFN20, SOP16, SOP8

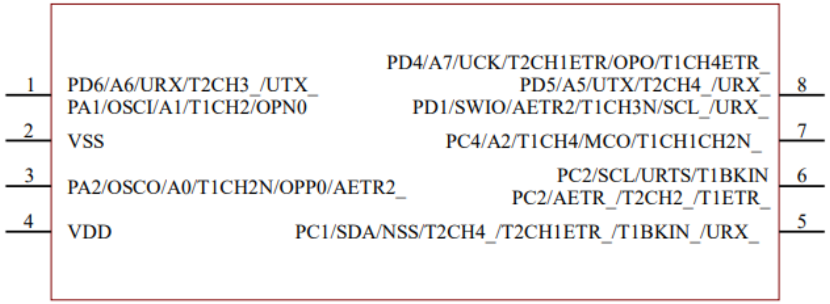
Part NO.	Freq	Flash	SRAM	GPIO	Timer				ADC (10bit) Unit/CH	OPA	U(S)ART	I²C	SPI	VDD	Package
					Adv (16bit)	GP (16bit)	WDOG	SysTick (32bit)							
CH32V003J4M6	48MHz	16K	2K	6	1	1	2	√	1/6	1	1	1	-	3.3/5.0	SOP8
CH32V003A4M6	48MHz	16K	2K	14	1	1	2	√	1/6	1	1	1	-	3.3/5.0	SOP16
CH32V003F4P6	48MHz	16K	2K	18	1	1	2	√	1/8	1	1	1	1	3.3/5.0	TSSOP20
CH32V003F4U6	48MHz	16K	2K	18	1	1	2	√	1/8	1	1	1	1	3.3/5.0	QFN20



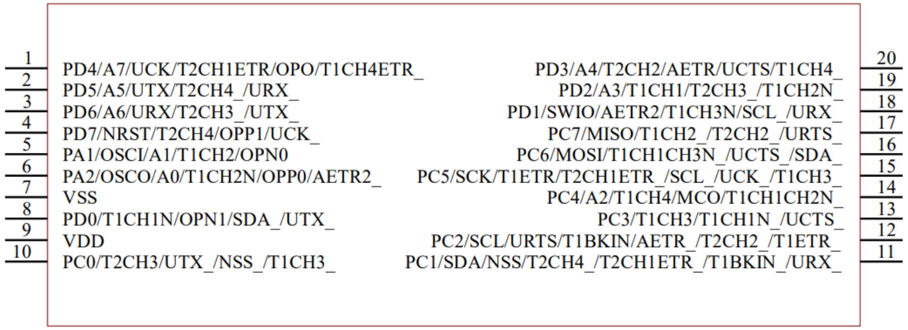


Detailed study of the CH32V003 Controller

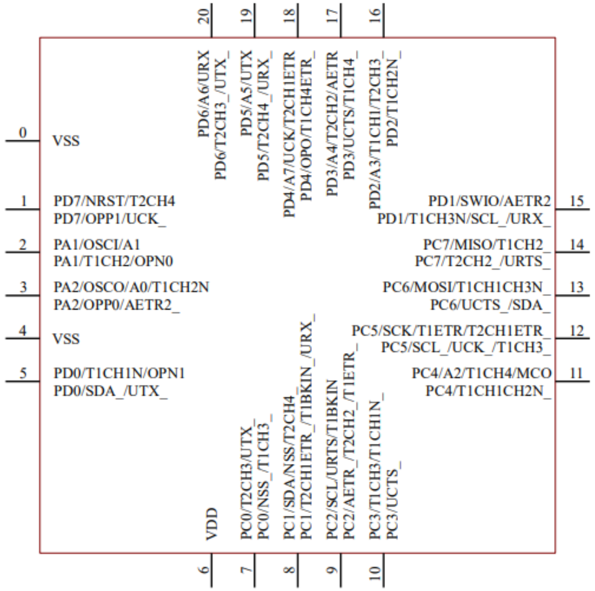
CH32V003J4M6



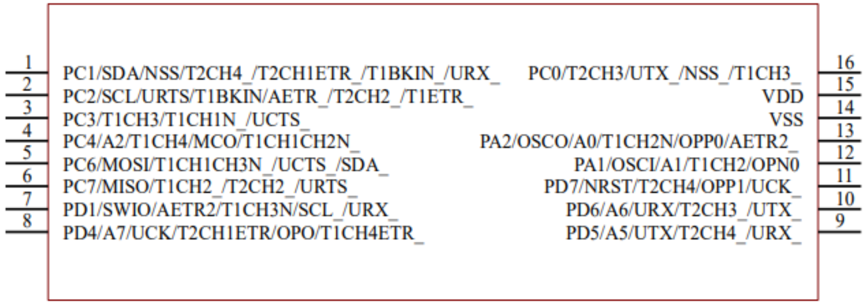
CH32V003F4P6



CH32V003F4U6



CH32V003A4M6



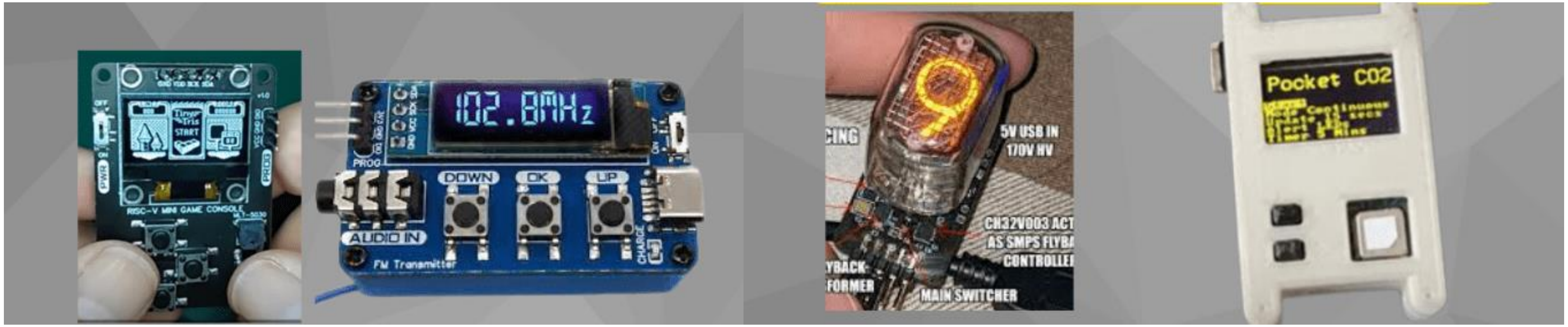
Contents

- Low Cost Microcontroller Architecture
- Introduction to CH32 Family of Microcontrollers
- Detailed study of the CH32V003 Controller
- Applications and Use Cases for CH32V003
- Debugging Tools and Techniques
- Introduction to System Clock
- Header File Explained

Applications and Use Cases for CH32V003

The CH32V003 Microcontroller is proving to be a powerhouse, as developers are creating impressively innovative projects with it. The complexity and sophistication of these projects are remarkable, considering the microcontroller's small size.

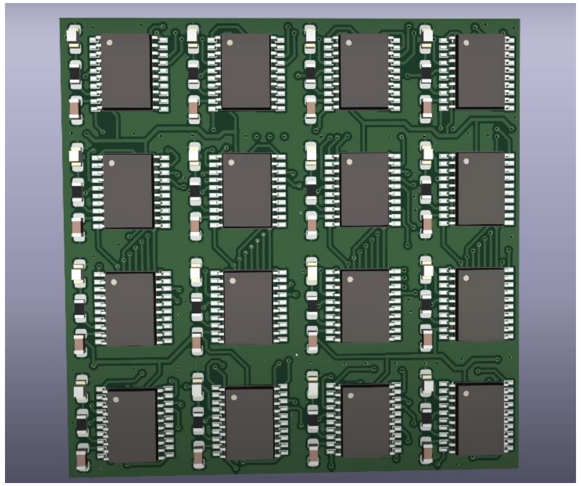
Lets see some interesting projects by ENGINEERS.

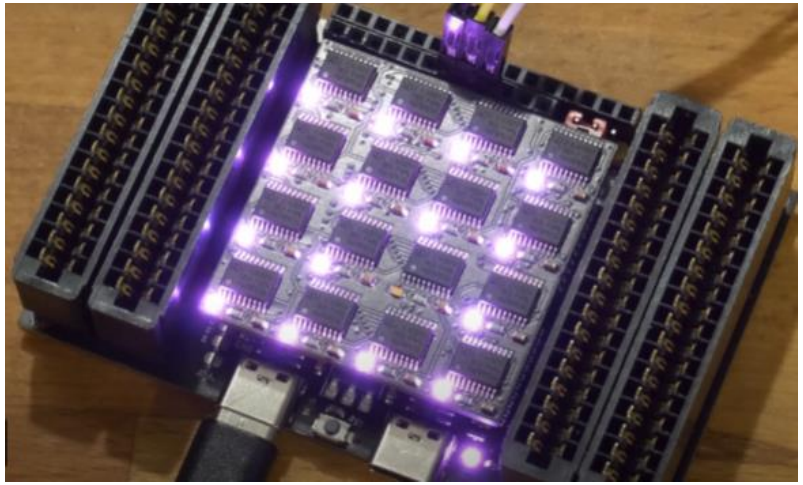


Applications and Use Cases for CH32V003

1) CH32V003 based Cheap RISCV Supercluster

A small scale low-cost computing cluster built using 16 CH32V003 microcontrollers, each priced at just 10 cents, on a single PCB. This project explores the potential of low cost Clusters and pushing the boundaries of what's possible with small, affordable hardware.





Applications and Use Cases for CH32V003

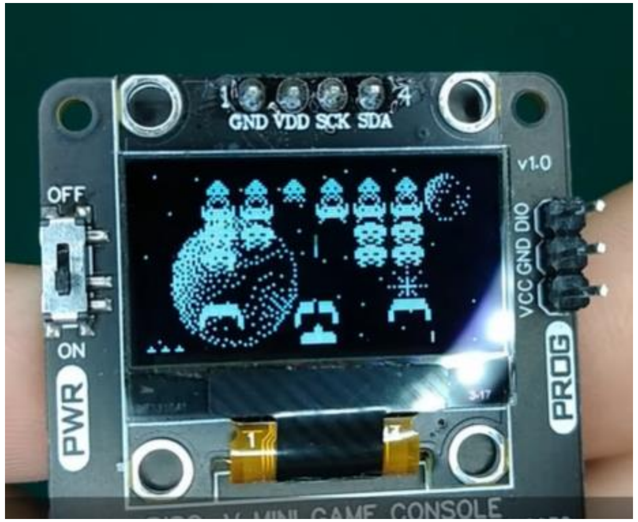
2) CH32V003 based FM Transmitter

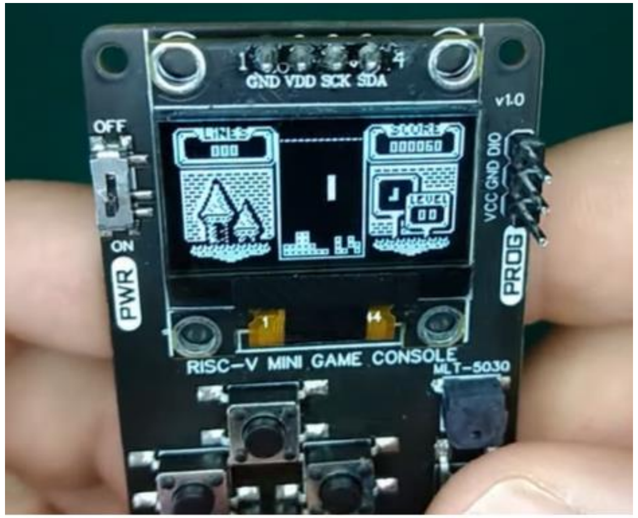
It is compact battery powered FM transmitter KT0803K or KT0803L Radio-Station-on-a-Chip, this is the core of the project along with CH32V003 which is the main MCU. The KT0803K/L is a low cost Digital Stereo FM Transmitter ASIC, It takes audio signal input and transmits the modulated FM signal over a short range.

Applications and Use Cases for CH32V003

3) RISCV Mini Game Console

Mini Game Console utilizing the CH32V003J4M6 ultra-cheap (10 cents by the time of writing) 32-bit RISC-V microcontroller, an SSD1306 128x64 pixels OLED display and CR/LIR2032 coin cell battery holder.





Applications and Use Cases for CH32V003

4) Battery Powered Pocket CO2 Sensor

The project is a pocket size battery powered CO2 Sensor or monitor. It is built using the incredibly cheap CH32V003 microcontroller. The Project uses 128×64 SSD1306 OLED Display, TP4057 3.7V LiPo Battery charge controller, and a Sensirion SCD40 CO2 sensor, Micro USB port for charging the battery.

Applications and Use Cases for CH32V003

The CH32V003's low cost, small size, and RISC-V architecture make it an attractive option for a wide range of applications, from simple DIY projects to complex industrial systems.

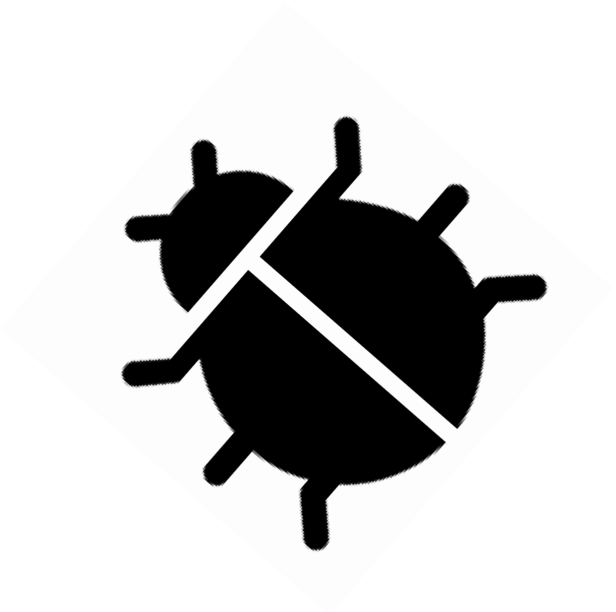
Here's an expanded list of potential uses for the CH32V003 microcontroller:

1. IoT Devices: Home automation, smart sensors, and wearables
2. Robotics: Control and navigation systems for small robots
3. Industrial Automation: Monitoring and control of industrial processes
4. Medical Devices: Portable medical devices, health monitors, and fitness trackers
5. Consumer Electronics: Smart home devices, gaming consoles, and multimedia players
6. Automotive Systems: In-vehicle infotainment, navigation, and sensor systems
7. Educational Projects: Robotics, electronics, and programming learning platforms
8. Wearable Technology: Smartwatches, fitness trackers, and health monitors
9. Security Systems: Access control, surveillance, and alarm systems
10. Environmental Monitoring: Air and water quality monitoring, weather stations

Applications and Use Cases for CH32V003

11. Portable Instruments: Multimeters, oscilloscopes, and signal generators
12. Smart Energy Management: Energy monitoring and control systems
13. Communication Devices: Modems, routers, and wireless communication modules
14. Gaming Platforms: Handheld game consoles and gaming peripherals
15. Scientific Instruments: Data loggers, spectrometers, and laboratory equipment
16. Standalone Systems: Self-contained devices for specific tasks (e.g., GPS trackers, digital signage)
17. Edge Computing: Distributed computing nodes for real-time data processing
18. Machine Learning: TinyML applications, neural network processing, and AI-enabled devices
19. Networked Devices: Ethernet-enabled devices for industrial control, monitoring, and automation
20. Wireless Sensor Networks: Distributed sensor nodes for industrial,

Contents



Debugging and Analysis Techniques

Embedded systems are specialized computer systems designed for specific purposes. They

- **Control**
- **Monitor**
- **Assist**

in the operation of equipment, machinery, or a larger system. These systems are present in various industries, such as automotive, consumer electronics, aerospace, and medical devices.

- **Debugging** is a crucial aspect of embedded systems development. As these systems are responsible for critical operations, any error or malfunction can have severe consequences.
- Debugging helps identify and fix errors, ensuring the system functions as expected.
- Moreover, it contributes to the overall quality, reliability, and performance of the embedded system

Debugging and Analysis Techniques

Understanding Debugging

Debugging is the process of

- **Identifying,**
- **Analyzing,**
- **Resolving issues within a software or hardware system.**

It involves

- **finding the root cause of problems,**
- **understanding their impact, and**
- **implementing solutions to ensure proper functioning.**

Debugging and Analysis Techniques

Goals and Objectives of Debugging

- The primary goal of debugging is to ensure that a system functions as intended. This involves identifying and fixing errors, optimizing performance, and enhancing stability. Debugging aims to:
- Locate and resolve software bugs and hardware issues.
- Improve system performance and efficiency.
- Enhance the user experience by fixing usability issues.
- Ensure compliance with industry standards and best practices.
- Maintain system stability and reliability.

Debugging and Analysis Techniques

Importance of Debugging in Embedded Systems

Debugging plays a vital role in embedded systems development. Due to the specialized nature of these systems, errors can lead to severe consequences, such as equipment malfunction or even safety hazards.

- Debugging helps ensure the proper functioning of embedded systems by:
- Eliminating errors that can compromise system performance and safety.
- Optimizing resource usage, which is crucial in systems with limited resources.
- Enhancing system stability and reliability.
- Improving overall system quality and user satisfaction.

By thoroughly understanding and mastering debugging techniques, embedded systems developers can create high-quality, reliable, and efficient systems that meet the demands of various industries.

Debugging and Analysis Techniques

Common Debugging Challenges in Embedded Systems

1. Limited Resources and Processing Power

Embedded systems often operate under strict resource constraints, such as limited memory, processing power, and power consumption.

Debugging in such environments can be challenging, as developers must balance the need for debugging tools and techniques with the available resources. This may require creative approaches and careful planning to ensure effective debugging without impacting system performance.

2. Real-Time Constraints

Many embedded systems operate in real-time, meaning they must respond to events and inputs within strict time constraints. Debugging real-time systems can be challenging, as developers must not only identify and resolve issues but also ensure that the system continues to meet its real-time requirements.

This often involves analyzing and optimizing the timing and synchronization aspects of the system.

Debugging and Analysis Techniques

Complex Hardware and Software Interactions

Embedded systems typically involve complex interactions between hardware and software components. Debugging these systems requires a deep understanding of both domains, as well as the ability to analyze and trace issues across the hardware-software boundary. This can be challenging, particularly when dealing with proprietary or custom hardware.

Concurrency Issues

Many embedded systems rely on concurrent processing to achieve their goals, whether through multi-threading, multi-processing, or other parallel processing techniques.

Debugging concurrent systems introduces additional complexity, as developers must identify and resolve issues related to synchronization, race conditions, and other concurrency-related challenges.

Debugging and Analysis Techniques

Debugging Techniques for Embedded Systems

1. Static Code Analysis

Static code analysis involves examining the source code of a system without executing it. It helps identify potential issues such as syntax errors, memory leaks, and coding standard violations. The benefits of static code analysis include early detection of errors, improved code quality, and reduced development time.

Some popular static code analysis tools for embedded systems include:

PC-Lint: A widely used tool for analyzing C and C++ code

Cppcheck: An open-source tool for detecting bugs in C and C++ code

CodeSonar: A commercial tool for analyzing C, C++, Java, and Ada code

MISRA-C: A set of coding standards for embedded systems development in C

Debugging and Analysis Techniques

2. Dynamic Analysis

Dynamic analysis involves monitoring the behavior of a system during runtime. It helps identify issues such as memory corruption, race conditions, and performance bottlenecks. The benefits of dynamic analysis include real-time error detection, improved system performance, and increased reliability.

Some popular dynamic analysis tools for embedded systems include:

Valgrind: An open-source tool for detecting memory management issues

GDB: The GNU Debugger, a widely used debugger for various programming languages

JTAG: A hardware debugging interface used for on-chip debugging and programming

Tracealyzer: A commercial tool for visualizing and analyzing real-time system behavior

Debugging and Analysis Techniques

4. In-Circuit Debugging

In-circuit debugging involves connecting a debugger directly to a running embedded system, allowing developers to monitor and control its execution. Benefits include real-time debugging capabilities, improved system visibility, and the ability to debug hardware-related issues.

Some popular in-circuit **debugging tools** for embedded systems include:

JTAG: A widely used hardware debugging interface

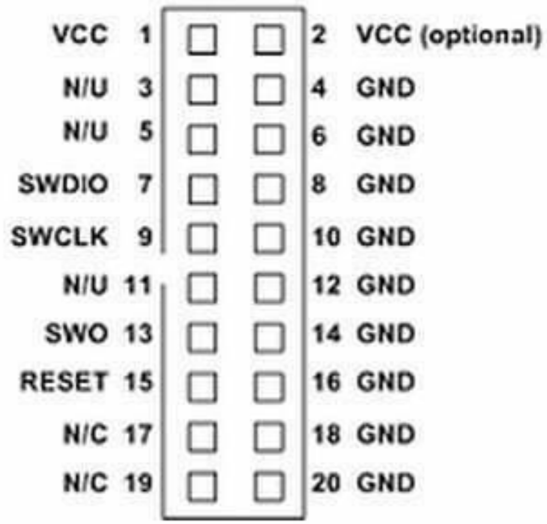
Segger J-Link: A popular JTAG/SWD debugger for ARM-based systems

P&E Micro: A provider of in-circuit debugging solutions for various microcontroller platforms

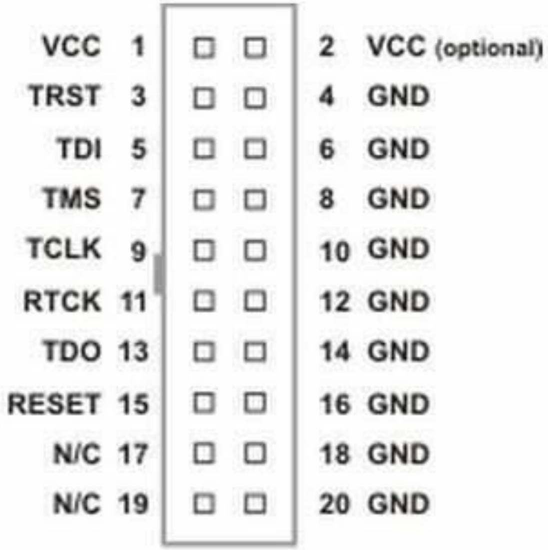
Atmel-ICE: An in-circuit debugger and programmer for Atmel microcontrollers.

SWD: Single wire Debug (used my WCH MCus)

Debugging and Analysis Techniques

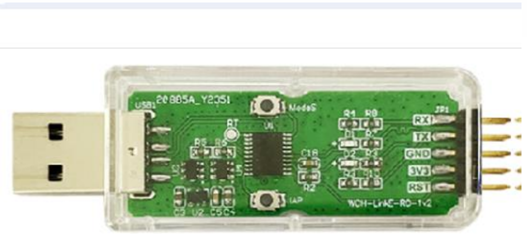


SWD



JTAG

J-Link



Debugging and Analysis Techniques

5. Hardware Debugging

Hardware debugging involves diagnosing and fixing issues related to the physical components of an embedded system, such as circuitry, sensors, and actuators. Benefits include improved system reliability, reduced development time, and the ability to identify and resolve hardware-specific issues.

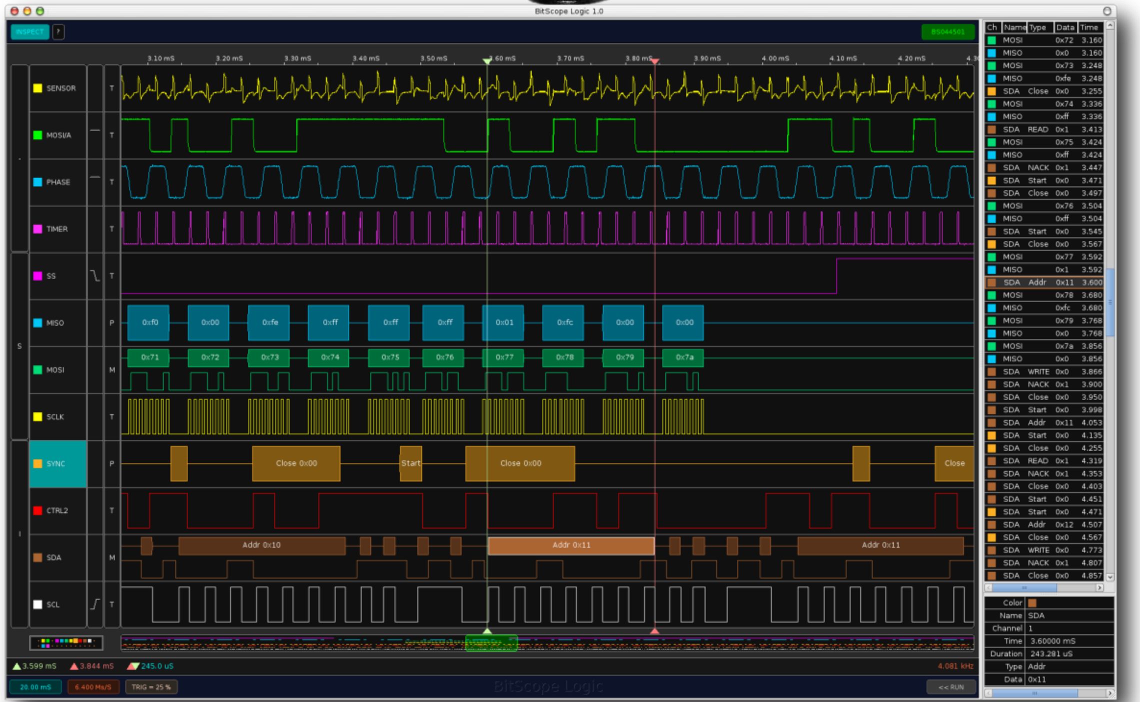
Some popular hardware debugging tools for embedded systems include:

- **Oscilloscopes:** Essential tools for analyzing and troubleshooting electrical signals
- **Logic Analyzers:** Devices used for monitoring and analyzing digital signals
- **Protocol Analyzers:** Tools for capturing and analyzing communication data between system components
- **Power Analyzers:** Instruments for measuring and analyzing power consumption in embedded systems

Debugging and Analysis Techniques

5. Hardware Debugging Tools

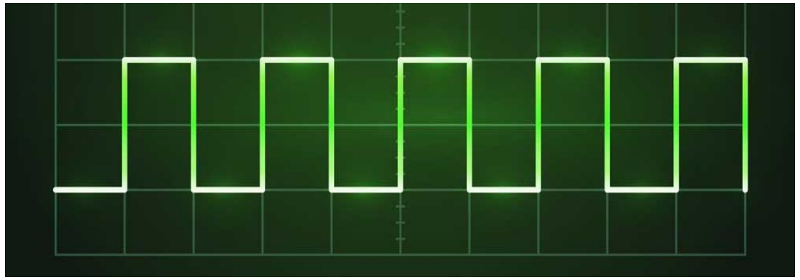
Logic Analyzers



Oscilloscope

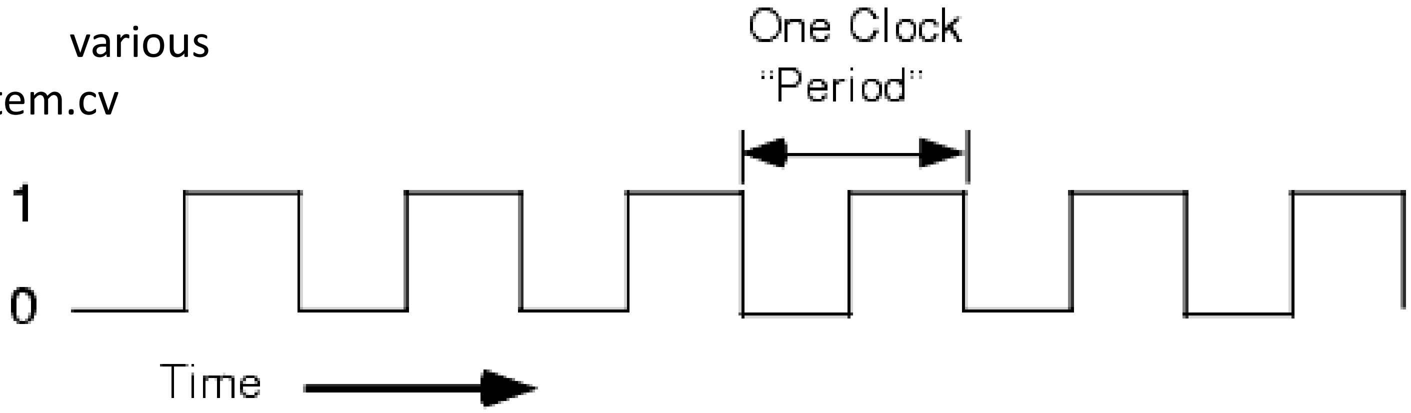
Contents

Systems Clock



What is a **Clock** ?

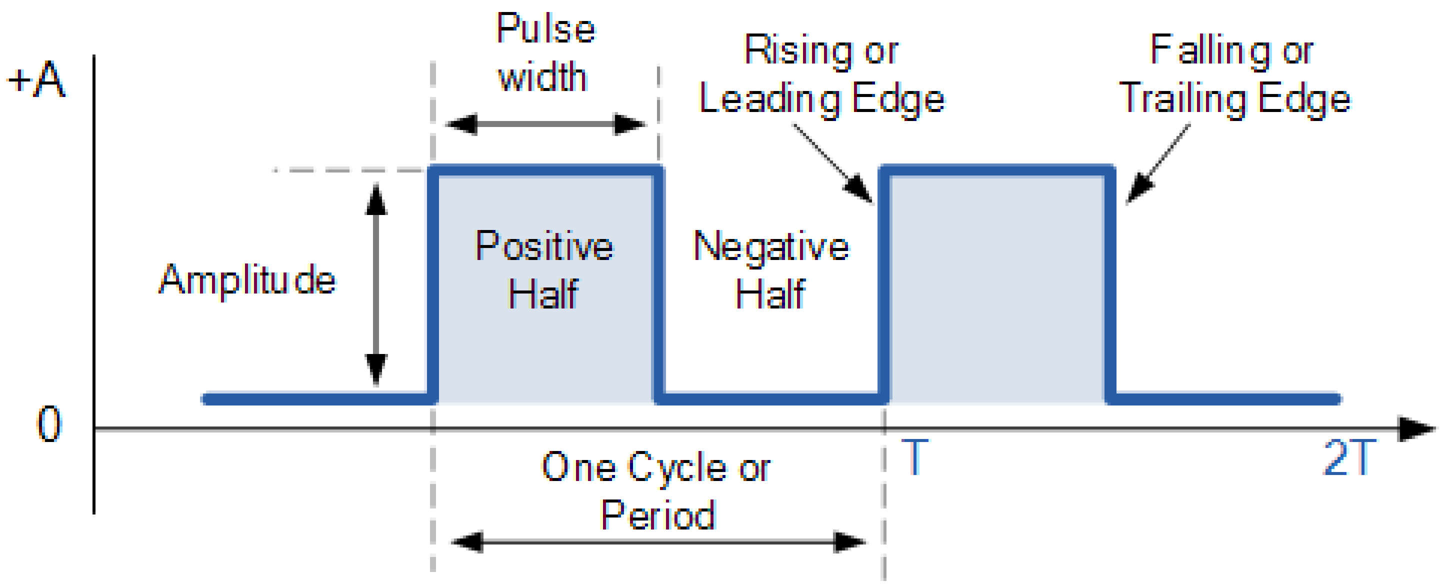
The clock generates a continuous sequence of pulses or oscillations, which are used to synchronize the operations of various components within the embedded system.cv



Clock Speed/Frequency: The frequency of the clock (measured in Hertz, Hz) determines the speed at which the processor and other components operate. A higher clock speed typically allows the system to perform tasks more quickly, but also requires more power and can generate more heat.

Terms related to Clock

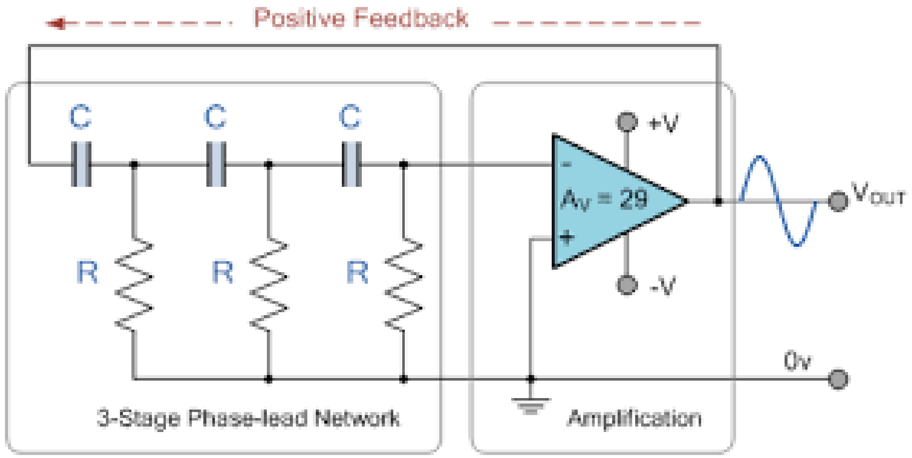
- **Amplitude:** The maximum voltage level of a clock signal from its baseline.
- **Positive Half:** The time interval during which the clock signal is high.
- **Negative Half:** The time interval during which the clock signal is low.
- **Rising Edge:** The transition of the clock signal from low to high.
- **Falling Edge:** The transition of the clock signal from high to low
- **Pulse Width:** The duration of time the clock signal remains at its high level during one cycle.

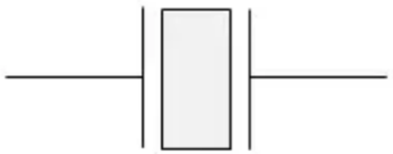


Sources of Clock in Embedded Systems

Clock Sources: Embedded systems can use different types of clock sources, including:

1. **RC Oscillators**: Less precise but can be cheaper and more compact.
2. **Crystal Oscillators**: Commonly used for their stability and accuracy.





RC Oscillators vs Crystal Oscillators

Feature	RC Oscillators	Crystal Oscillators
Principle	Uses resistors and capacitors to generate oscillations.	Uses the mechanical resonance of a quartz crystal to produce oscillations.
Accuracy	Less accurate; frequency can drift with temperature and supply voltage changes.	Highly accurate; stable frequency due to the crystal's precise resonance.
Stability	Less stable; affected by temperature, aging, and supply voltage variations.	Highly stable; minimal drift with temperature and voltage changes.
Frequency Range	Can be designed for a wide range of frequencies but typically less precise.	Provides precise frequencies, usually in specific ranges (e.g., MHz to GHz).
Size	Generally smaller and simpler in design.	Larger and more complex due to the crystal's packaging.
Cost	Generally cheaper to produce and implement.	Typically more expensive due to the precision required.

Reset and Clock Control (RCC)

The controller provides different forms of resets and configurable clock tree structures based on the division of power areas and peripheral power management considerations in the application.

Main Features:

- Multiple reset forms
- Multiple clock sources, bus clock management
- Built-in external crystal oscillation monitoring and clock security system
- Independent management of each peripheral clock: reset, on, off
- Supports internal clock output

Reset:

There are two types of Resets provided by the system:

- Power-on Rest
- System Reset

Reset and Clock Control (RCC)

➢ **Power Reset**

When a power Reset occurs, it will reset all registers. A power Reset is generated when the following event occurs:

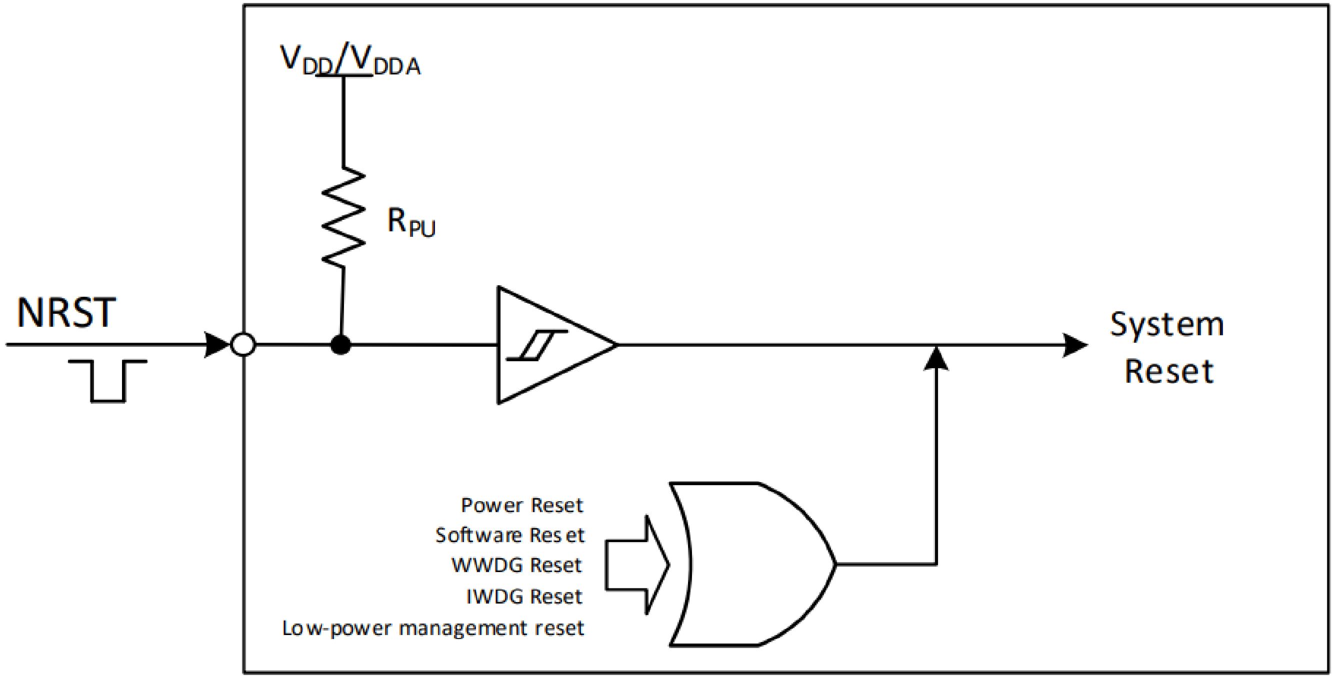
- **Power-up/power-down reset (POR/PDR)**

➢ **System Reset**

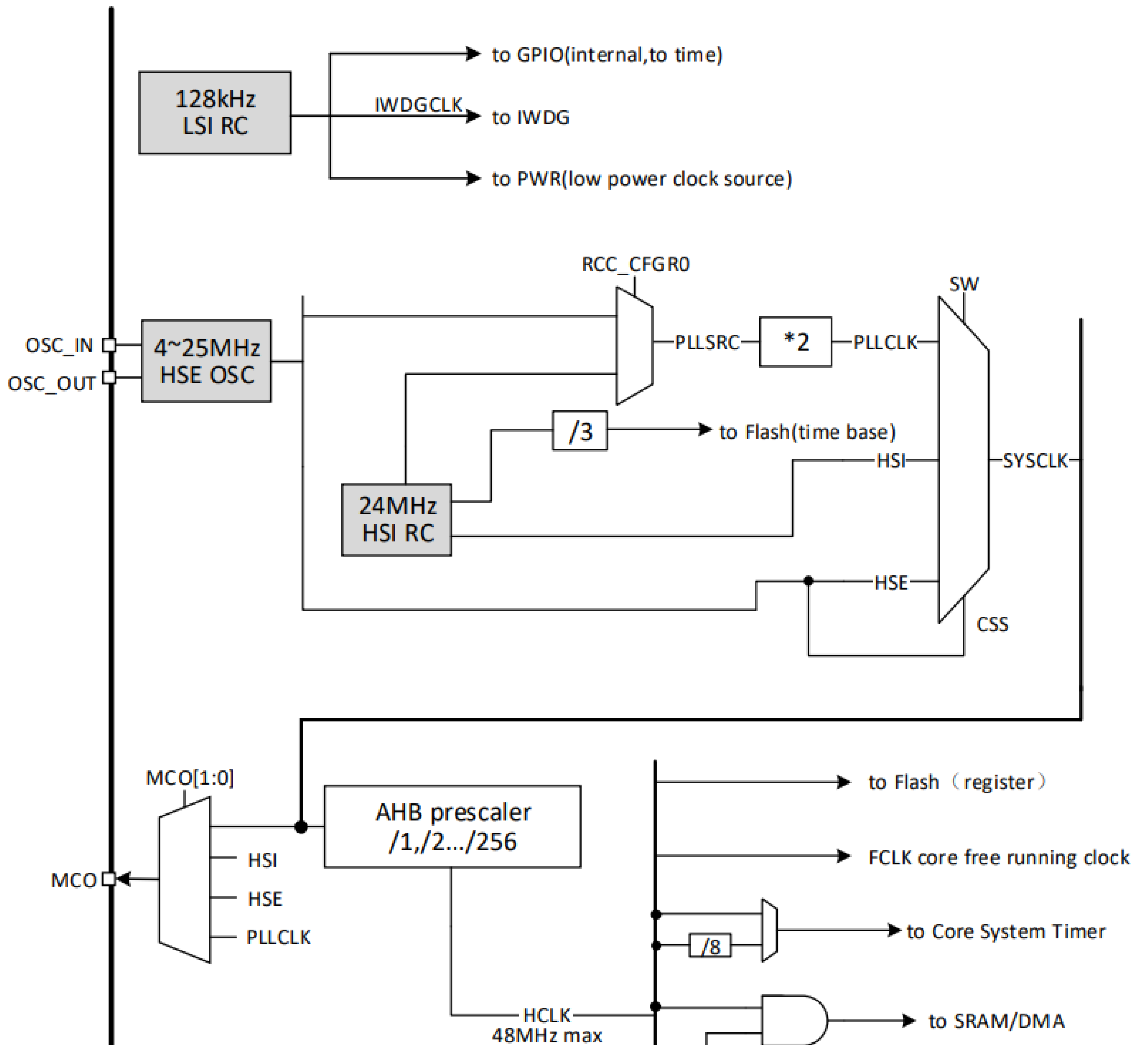
When a system Reset occurs, it will reset the reset flag in addition to the control/status register **RCC_RSTSCKR** and all the registers.

The source of the reset event is identified by looking at the reset status flag bit in the **RCC_RSTSCKR** register.

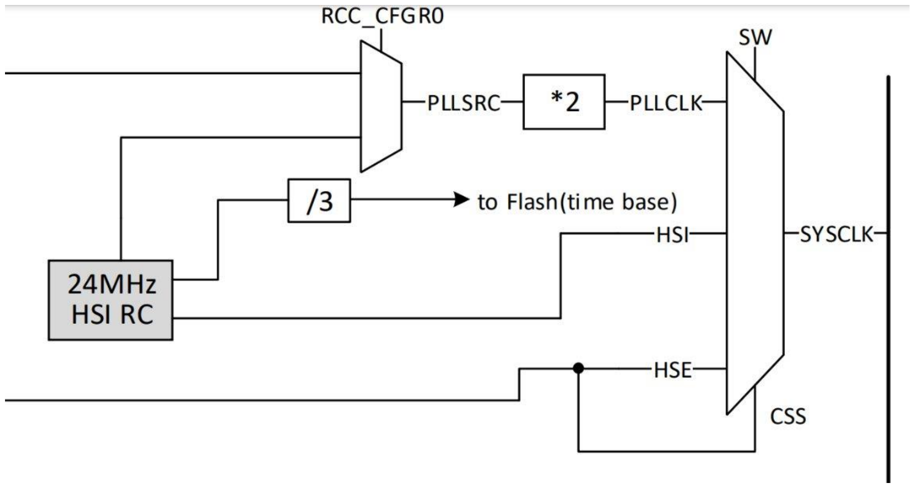
System reset structure



Reset and Clock Control (RCC)



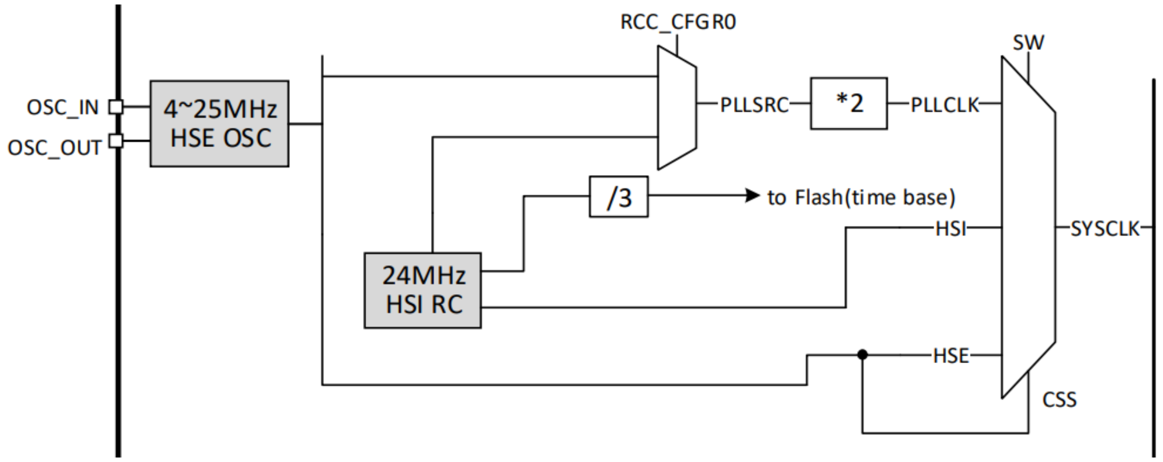
High-Speed Clock (HSI/HSE)



HSI (High-Speed Internal Clock)

- **Description**:
    - Internal 24 MHz RC oscillator.
    - Provides system clock without external devices.
    - Short start-up time.
- **Control and Status Register Bits**:
    - **HSION**: Enables or disables HSI (bit in RCC_CTLR register).
    - **HSIRDY**: Indicates stability of HSI (bit in RCC_CTLR register).
    - **HSIRDYIE**: Generates interrupt on HSI ready status (bit in RCC_INTR register).
- **Default Settings**:
    - **HSION**: Set to 1 (enabled by default).
    - **HSIRDY**: Set to 1 (HSI stable by default).
- **Backup Clock**:
    - HSI used as a backup clock source if HSE crystal oscillator fails.

High-Speed Clock (HSI/HSE)



High-Speed External Clock (HSE)

Description:

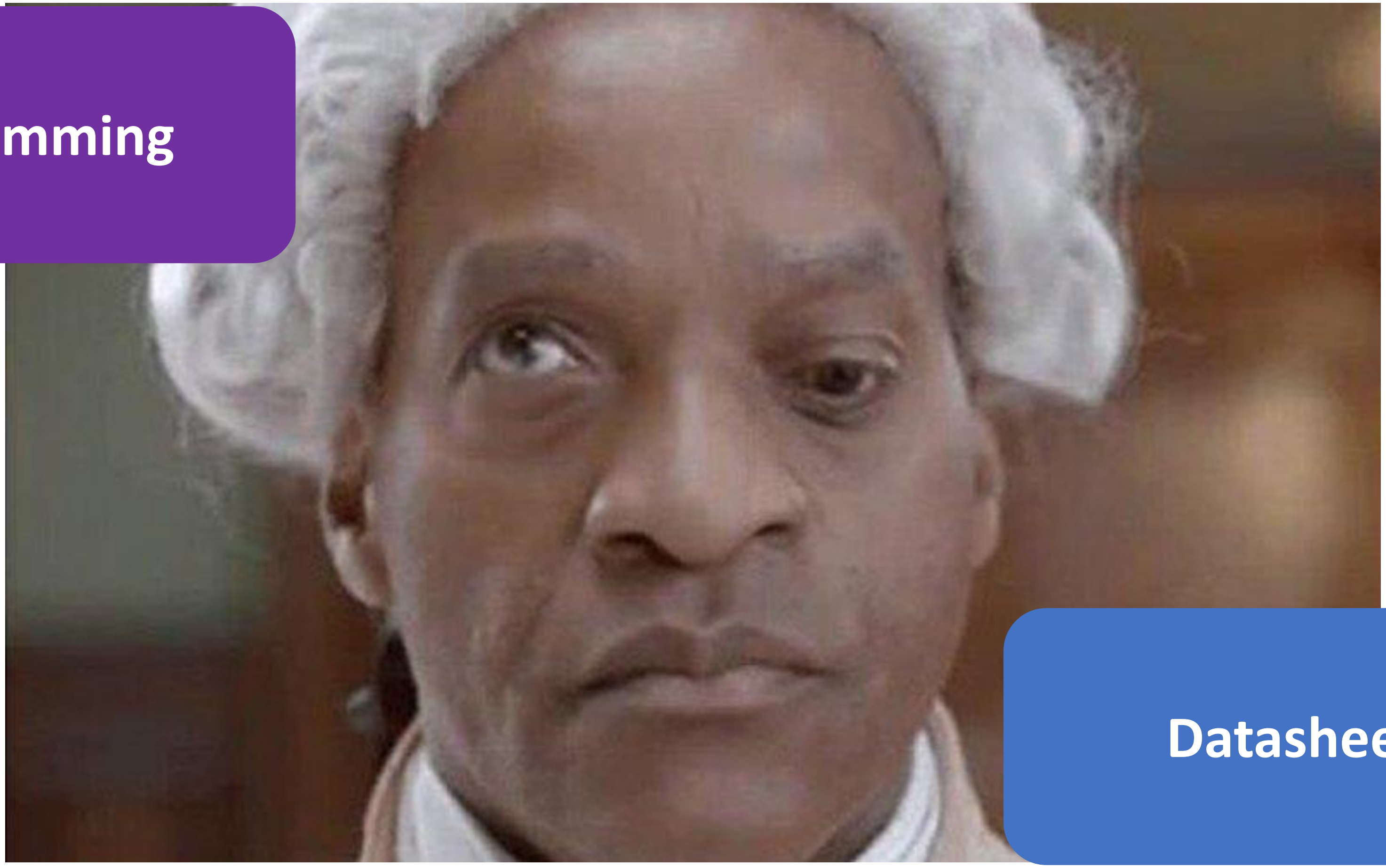
- **External High-Speed Clock**:
    - Can be sourced from an external crystal or ceramic resonator.
    - Alternatively, an external high-speed clock signal can be fed directly into the system.

    External Crystal/Ceramic Resonator (HSE Crystal):

- **Frequency Range**:
    - 4-25 MHz
    - Provides a more accurate clock source compared to internal oscillators.
- **Control and Status**:
    - **HSEON**: Enables or disables HSE (bit in RCC_CTLR register).
    - **HSERDY**: Indicates stability of HSE crystal oscillation (bit in RCC_CTLR register).
        - Clock fed into the system only after HSERDY is set to 1.
    - **HSERDYIE**: Generates interrupt on HSE ready status (bit in RCC_INTR register).

Condition of Hardware Engineers

Programming



Datasheet

Clock Registers

Name	Access address	Description	Reset value
R32_RCC_CTLR	0x40021000	Clock control register	0x0000xx83
R32_RCC_CFGR0	0x40021004	Clock configuration register 0	0x00000020
R32_RCC_INTR	0x40021008	Clock interrupt register	0x00000000
R32_RCC_APB2PRSTR	0x4002100C	APB2 peripheral reset register	0x00000000
R32_RCC_APB1PRSTR	0x40021010	APB1 peripheral reset register	0x00000000
R32_RCC_AHBPCENR	0x40021014	AHB peripheral clock enable register	0x00000004
R32_RCC_APB2PCENR	0x40021018	APB2 peripheral clock enable register	0x00000000
R32_RCC_APB1PCENR	0x4002101C	APB1 peripheral clock enable register	0x00000000
R32_RCC_RSTSCKR	0x40021024	Control/status register	0x0C000000

3.4.1 Clock Control Register (RCC_CTLR)

Offset address: 0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved						PLLRDY	PLLON	Reserved				CSSON	HSEBYP	HSERDY	HSEON

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Reserved	HSIRDY	HSION

Clock Registers

3.4.7 APB2 Peripheral Clock Enable Register (RCC_APB2PCENR)

Offset address: 0x18

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	USART1 EN	Reserved	SPI1 EN	TIM1 EN	Reserved	ADC1 EN	Reserved			IOPD EN	IOPC EN	Reserved	IOPA EN	Reserved	AFIO EN

Bit	Name	Access	Description	Reset value
[31:15]	Reserved	RO	Reserved	0
14	USART1EN	RW	USART1 interface clock enable bit. 1: Module clock is on; 0: Module clock is off.	0
13	Reserved	RO	Reserved	0
12	SPI1EN	RW	SPI1 interface clock enable bit. 1: Module clock is on; 0: Module clock is off.	0
11	TIM1EN	RW	TIM1 module clock enable bit. 1: Module clock is on; 0: Module clock is off.	0
10	Reserved	RO	Reserved	0
9	ADC1EN	RW	ADC1 module clock enable bit. 1: Module clock is on; 0: Module clock is off.	0
[8:6]	Reserved	RO	Reserved	0
5	IOPDEN	RW	PD port module clock enable bit for I/O. 1: Module clock is on; 0: Module clock is off.	0
4	IOPCEN	RW	PC port module clock enable bit for I/O. 1: Module clock is on; 0: Module clock is off.	0
3	Reserved	RO	Reserved	0
2	IOPAEN	RW	PA port module clock enable bit for I/O. 1: Module clock is on; 0: Module clock is off.	0
1	Reserved	RO	Reserved	0
0	AFIOEN	RW	I/O auxiliary function module clock enable bit. 1: Module clock is on; 0: Module clock is off.	0

Contents

Embedded C Programming for RISC-V Micro-controller

Understanding Header files of CH32v003

1) Header Guards

```
#ifndef __CORE_RISCV_H__
#define __CORE_RISCV_H__
```

These lines ensure that the contents of the header file are only included once in a compilation unit. If __CORE_RISCV_H__ is not defined, it defines it and includes the rest of the file. This is a common practice in C/C++ header files.

2) Conditional Compilation for C++

```
#ifdef __cplusplus
extern "C" {
#endif
```

This section checks if the code is being compiled with a C++ compiler and adds the extern "C" to ensure proper linking with C code.

Embedded C Programming for RISC-V Micro-controller

Understanding Header files of CH32v003

```
/* IO definitions */
#ifdef __cplusplus
       #define __I volatile /* defines 'read only' permissions */
#else
       #define __I volatile const /* defines 'read only' permissions */
#endif
#define __O volatile /* defines 'write only' permissions */
#define __IO volatile /* defines 'read / write' permissions */
```

These macros define the permissions for I/O (Input/Output) operations. __I is for read-only, __O is for write-only, and __IO is for read/write

Embedded C Programming for RISC-V Micro-controller

Standard Peripheral Library old types

```
/* Standard Peripheral Library old types (maintained for legacy purpose) */
typedef __I uint32_t vuc32; /* Read Only */
typedef __I uint16_t vuc16; /* Read Only */
typedef __I uint8_t vuc8; /* Read Only */
```

These typedefs define old types for backward compatibility. For example, vuc32 is a volatile read-only 32-bit unsigned integer.

Enumerating Error status/ Functional States/ Flag status

```
typedef enum {NoREADY = 0, READY = !NoREADY} ErrorStatus;
typedef enum {DISABLE = 0, ENABLE = !DISABLE} FunctionalState;
typedef enum {RESET = 0, SET = !RESET} FlagStatus, ITStatus;
```

Embedded C Programming for RISC-V Micro-controller

Memory-Mapped Register Structures

```
typedef struct{
 // ... (fields for Program Fast Interrupt Controller - PFIC)
 } PFIC_Type;
```

This defines a structure for the Program Fast Interrupt Controller (PFIC) with its various registers as fields. PFIC seems to be a component responsible for managing interrupts.

PFIC is a macro that is defined to represent a specific memory-mapped structure. Let's take a closer look at the relevant definition:

```
 #define PFIC ((PFIC_Type *) 0xE000E000)
```

Here, PFIC is defined as a pointer to a structure of type PFIC_Type, and it is initialized with the memory address 0xE000E000. The type PFIC_Type is a user-defined structure type that likely represents the memory layout of a specific hardware peripheral, possibly related to interrupt handling.

Embedded C Programming for RISC-V Micro-controller

6.5.2 PFIC Registers

Table 6-4 List of PFIC-related registers

Name	Access address	Description	Reset value
R32_PFIC_ISR1	0xE000E000	PFIC interrupt enable status register 1	0x0000000C
R32_PFIC_ISR2	0xE000E004	PFIC interrupt enable status register 2	0x00000000
R32_PFIC_IPR1	0xE000E020	PFIC interrupt pending status register 1	0x00000000
R32_PFIC_IPR2	0xE000E024	PFIC interrupt pending status register 2	0x00000000
R32_PFIC_ITHRESDR	0xE000E040	PFIC interrupt priority threshold configuration register	0x00000000
R32_PFIC_CFGR	0xE000E048	PFIC interrupt configuration register	0x00000000