AN2945
Application note

STM8S and STM32™ MCUs: a consistent 8/32-bit product line for painless migration

Introduction

Following the market launch of the award winning STM32™ microcontroller, STMicroelectronics completes the renewal of its microcontroller product line with the announcement of the STM8S family. Significant effort has been made to rationalize the MCU portfolio, in particular by capitalizing on common peripherals and software tools with the aim to easing product migration.

The cost, in terms of both time and money, of maintaining a development team to design in a new MCU family is a major criterion when selecting a microcontroller supplier. It is therefore an advantage to make this kind of non-recurring investment if it applies to a broad range of MCUs. With an MCU product line ranging from 20 to 144 pins, and memory sizes from 2 to 512 Kbytes, the 8-bit STM8S and 32-bit STM32 families bring a lot of flexibility when building a product portfolio. Should an 8-bit application run out of MIPS, there is an upgrade path to the STM32 family. Conversely, if you wish to cut costs on a 32-bit platform, it is relatively simple to switch to the STM8 family.

This document presents the similarities and common features of the STM8S and STM32 product lines, with a view of helping migration from one family to the other.

Contents

List of tables

List of figures

1 Core

The STM8™ CPU is a proprietary architecture that maintains the legacy of the previous ST7 core while being a breakthrough in terms of 8-bit CPU efficiency and code density. The STM32 is built around the industry standard ARM® Cortex™-M3 32-bit core and benefits from the complete ecosystem of development tools and software solutions associated with ARM processors. Although they may be perceived as radically different, these two processors indeed share many architectural similarities summarized in *Table 1*.

Table 1. STM8 and STM32: core comparison

	STM8	Cortex-M3
Data path	8-bit	32-bit
Drhystone MIPS (0WS)	0.29 DMIPS	1.22 DMIPS
Architecture	Harvard	Harvard
Pipeline	Yes, three-stage	Yes, three-stage
Instruction set	CISC	RISC
Program bus data width	32-bit	32-bit
Prefetch buffer	Yes, 2 × 32-bit, internal	Yes, 2 × 64-bit, in memory interface
Average instruction size	2 bytes	2 bytes
Interrupt type Latency	Vectorized 9 cycles, tail chaining supported	Vectorized 12 cycles, tail chaining supported
Low power modes	Slow, Wait for Event or interrupts, Halt, Halt on exit	Slow, Sleep (Wait for event or interrupt), Sleep on exit, Deep sleep
Debug interface	1-wire (SWIM)	2-wires or legacy JTAG

Both are based on the Harvard architecture. They have 3-stage pipelined execution that minimizes the execution time, a clock speed up to 24 MHz for the STM8S and up to 72 MHz for the STM32 family.

They are devised to be highly energy efficient, with several low power modes, and they benefit from memory interfaces wider than the average instruction length (32- and 64-bit wide busses, respectively). This minimizes the number of accesses to the memory bus and thus the consumption related to address bus toggling and non-volatile memory read accesses. Interrupt tail chaining and the Halt/Sleep on exit modes also help avoiding unnecessary stack accesses.

Finally, in terms of code density, both have excellent results, owing to the 8-bit CISC instruction set for the STM8S family and, to the 16-bit Thumb-2 mode introduced by the Cortex core for the STM32 family.

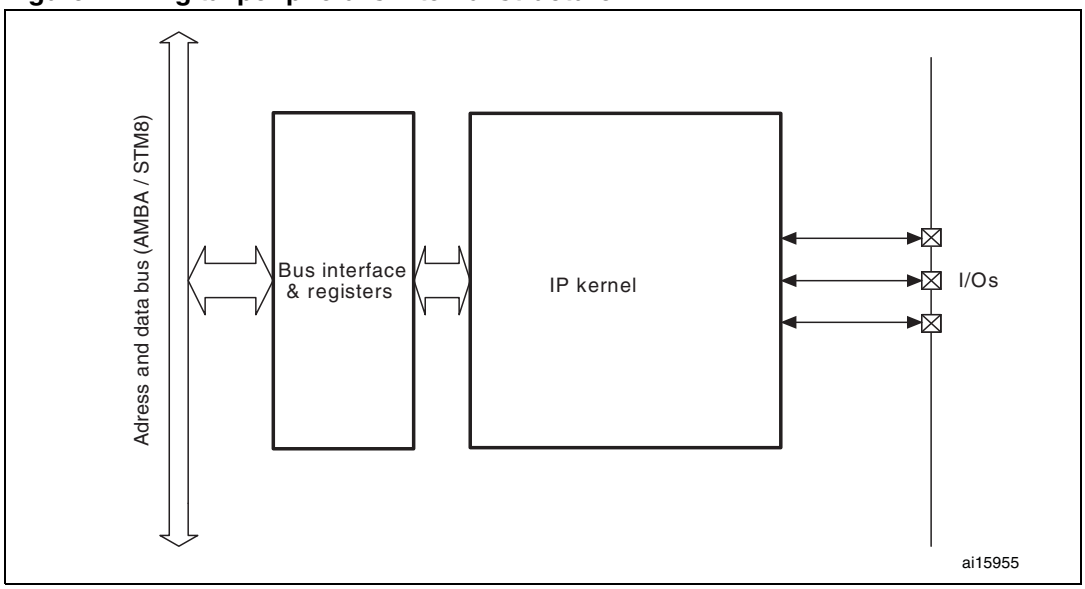
This short comparison demonstrates that both processors are state-of-the-art in terms of micro-architectural features. The STM8 is at the level of legacy of 16-bit processors, and the Cortex-M3 meets the requirements of applications currently using 32-bit down to mid/high-end 16-bit MCUs. The combination of the STM8 and STM32 therefore establishes a performance continuum, which is now also supported at tool levels by a third party offering a unified development platform for both product lines.

2 Peripherals

The MCU peripherals (also called IPs) are another example of the ST MCU consistency across the 8- and 32-bit product lines: most of the basic IPs have been defined and structured to be portable from one product family to the other. This was done by adapting simple, yet effective, 8-bit peripherals to the 32-bit world. It brings the benefits of cost- and power-effective, easy to understand resources, which are complemented at system level by wider busses and a DMA controller when higher performance is needed. Once the working principle of a peripheral is understood, it is applicable to both the STM8S and STM32 families, thus speeding up transition between devices.

Figure 1 shows a simplified representation of a digital peripheral.

Figure 1. Digital peripheral's internal structure



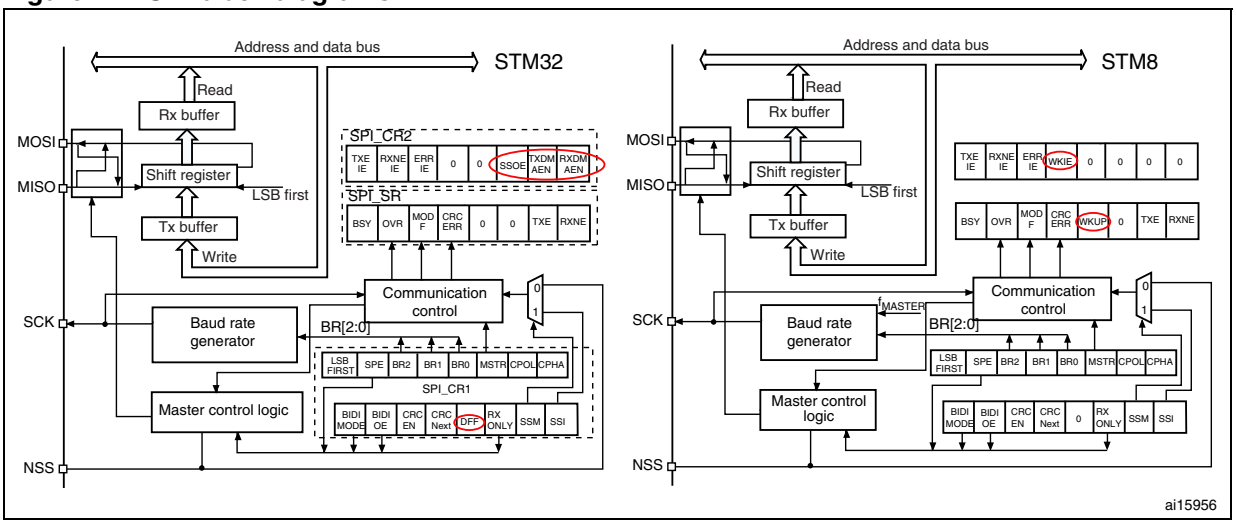
The peripheral can be partitioned into two main blocks. First, a kernel that contains the state machines, counters and any kind of combinatorial or sequential logic necessary to perform tasks that do not need the processor, such as low-communication layers, analog front-end management or timing-driven functions. If necessary, the kernel is connected to the external world via MCU ports. The external connection may consist of a few I/Os or complex busses. Second, the peripheral is initialized and controlled by the application through registers connected to an internal bus shared with the other MCU resources. In 8-bit microcontrollers, the processor directly writes to and reads from registers, whereas in 32-bit products, register read and write operations usually go through a bridge. The main difference between the two families, however, lies in the internal bus specification the peripheral has to comply with. This explains why STM8S and STM32 devices are able to share peripherals: these are based on the same kernel, and are only tailored to the two different bus interfaces. ARM processors and peripherals comply with the AMBA bus specification, with a 32-bit databus,

whereas STM8S devices use a simpler, yet efficient, 8-bit bus standard. From the functional point of view, they only differs by:

- the register size: 8 vs. 16 or 32-bit
- the maximum clock frequency that directly depends on the CPU operating speed
- the DMA that offloads the CPU from simple data management and increases the maximum data throughput
- few product-specific functions, such as I/O port management

Let us consider the STM8S and STM32 SPI block diagrams shown in *Figure 2*. At first glance, they look identical apart from a few differences in bits highlighted in red in *Figure 2*, for instance, at the level of the DMA.

Figure 2. SPI block diagrams



Now considering the register maps shown in *Table 2* and *Table 3*, they are clearly based on the same design: apart from a few differentiating bits and the register sizes, registers and bits have similar names and locations in registers.

Table 2. STM32 SPI register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**SPI_CR1**	Reserved																BIDIMODE	BIDIOE	CRCEN	CRCNEXT	DFF	RXONLY	SSM	SSI	LSBFIRST	SPE	BR [2:0]			MSTR	CPOL	CPHA
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x04	**SPI_CR2**	Reserved																								TXEIE	RXNEIE	ERRIE	Reserved		SSOE	TXDMAEN	RXDMAEN
	Reset Value																									0	0	0			0	0	0
0x08	**SPI_SR**	Reserved																								BSY	OVR	MODF	CRCERR	UDR	CHSIDE	TXE	RXNE
	Reset Value																									0	0	0	0	0	0	1	0
0x0C	**SPI_DR**	Reserved																DR[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**SPI_CRCPR**	Reserved																CRCPOLY[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
0x14	**SPI_RXCRCR**	Reserved																RxCRC[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x18	**SPI_TXCRCR**	Reserved																TxCRC[15:0]															
	Reset Value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1C	**SPI_I2SCFGR**	Reserved																				I2SMOD	I2SE	I2SCFG		PCMSYNC	Reserved	I2SSTD		CKPOL	DATLEN	CHLEN
	Reset Value																					0	0	0	0	0		0	0	0	0	0	0
0x20	**SPI_I2SPR**	Reserved																						MCKOE	ODD	I2SDIV							
	Reset Value																							0	0	0	0	0	0	0	0	1	0

Table 3. STM8 SPI register map and reset values

Address offset	Register name	7	6	5	4	3	2	1	0
0x00	SPI_CR1 Reset value	LSBFirst 0	SPE 0	BR2 0	BR1 0	BR1 0	MSTR 0	CPOL 0	CPHA 0
0x01	SPI_CR2 Reset value	BDM 0	BDOE 0	CRCEN 0	CRCNEXT 0	Reserved 0	RXONLY 0	SSM 0	SSI 0
0x02	SPI_ICR Reset value	TXIE 0	RXIE 0	ERRIE 0	WKIE 0	Reserved 0	Reserved 0	Reserved 0	Reserved 0
0x03	SPI_SR Reset value	BSY 0	OVR 0	MODF 0	CRCERR 0	WKUP 0	Reserved 0	TXE 1	RXNE 0
0x04	SPI_DR Reset value	MSB 0	- 0	- 0	- 0	- 0	- 0	- 0	LSB 0
0x05	SPI_CRCPR Reset value	MSB 0	- 0	- 0	- 0	- 0	- 1	- 1	LSB 1
0x06	SPI_RXCRCR Reset value	MSB 0	- 0	- 0	- 0	- 0	- 0	- 0	LSB 0
0x07	SPI_TXCRCR Reset value	MSB 0	- 0	- 0	- 0	- 0	- 0	- 0	LSB 0

Table 4 lists the common peripherals, highlighting the coherency between products at register, bit and feature level.

Table 4. Peripherals shared between STM8 and STM32 devices

Peripheral names	
STM32	**STM8**
Independent watchdog (IWDG)	
Window watchdog (WWDG)	
Serial peripheral interface (SPI)	
Inter-integrated circuit (I^2C) interface	
Universal synchronous/asynchronous receiver/transmitter (USART)	
Advanced-control timers	16-bit advanced-control timer
General-purpose timer	16-bit general-purpose timers
Basic timer	8-bit basic timer

Although the timers seem different with many distinct configurations, their architecture across and within the product families is the same. There are only variations of a single timer architecture. From the superset, sub-blocks can optionally be stripped to decrease the number of capture/compare channels or remove options necessary only for a few specific applications such as motor control.

3 System features

Today's MCUs are complex SoCs (systems on chip) that not only include a lot of peripherals, but also advanced-system features aiming at reducing the bill of material or enhancing the products' safety and robustness. This is true for both 8- and 32-bit platforms.

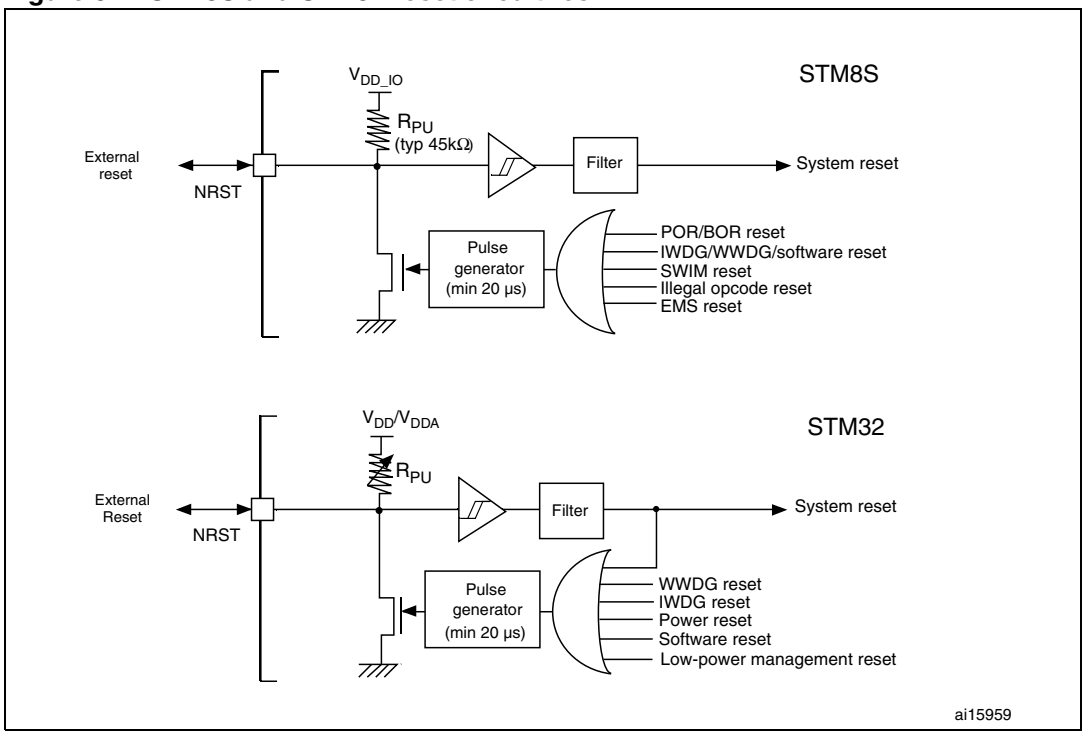
3.1 Reset

As shown in *Figure 3*, the STM8S and STM32 devices have the same reset circuitry, with only slight differences.

The NRST pin is both an input and an open-drain output with a built-in pull-up resistor. For EMS (electromagnetic sensitivity) robustness purposes, a filter is inserted to avoid glitch propagation into the digital circuitry. There are three advantages with having a bidirectional reset:

- for multi-MCU systems, bidirectional reset ensures than all subprocessors are correctly synchronized at startup or in case of a warm reset
- the voltage supervisors (power-on reset and brownout reset) embedded in the MCU can also be used at system level for other ICs
- it is of a great help during debugging when spurious internal resets are generated

Figure 3. STM8S and STM32 reset circuitries



3.2 Clock

From the clock system standpoint, the two products have three main clock sources in common, that share similar electrical characteristics. See *Table 5* for details.

The oscillator handles both the crystal and resonators, and is called the HSE (for high-speed external). It can also be bypassed to feed the MCU with an external clock. This is used for applications that have stringent requirements in terms of accuracy and stability, for communication purposes for instance.

An application can run at a high frequency without an external crystal by using the HSI clock (for high-speed internal). This source has a consumption 10 times lower than the HSE and a very low percentage of accuracy error. It can also be used as a PLL input on the STM32 to increase the internal frequency to up to 64 MHz.

Finally, the low-speed internal clock (LSI) is an ultralow internal power source (a few µA), that can be permanently enabled to clock an auto-wakeup peripheral during the Halt or Stop mode. It can also clock a secondary on-board watchdog (refer to *Section 3.4: Safety* for further details), and be used as the CPU clock on the STM8 products. It is not accurate (error of a few tens of percents), but it can be measured periodically using the precise HSI clock to compensate for chip manufacturing variations or the drift due to temperature for instance.

Table 5. STM8S/STM32 clock source characteristics (indicative data[1])

System clock source	Frequency		Accuracy error	Consumption
	STM8S	STM32		
High-speed external (HSE)	1-24 MHz	4-16 MHz	Crystal dependent, down to a few tens of ppm	1 to 2 mA
High-speed internal (HSI)	16 MHz	8 MHz	1% typical	100 to 250 µA
Low-speed internal	110-146 kHz	30-60 kHz	20 to 50%	1 to 5 µA

1. Refer to product datasheet for detailed electrical characteristics.

3.3 Memory

Both product lines are based on non-volatile memories and have an option byte loader. This mechanism replaces the legacy fuses for MCU power-up configuration: the user can select several options at programming time, which are written alongside the program binary image.

Several features are available on all news microcontrollers:

- Reset in Halt, Stop or Standby mode: this is to avoid a deadlock situation in case the MCU enters a low power mode by accident, for applications not designed to handle such a configuration
- Hardware/Software watchdog, to have the possibility of starting the watchdog by hardware, right after the reset sequence
- Memory readout protection, to prevent any piracy on the program content
- Memory write protection, to protect part of the memory, if it contains a critical code. Usually, this applies to the boot code or an IAP (in application programming) driver

These options allow automatically enabled safety and robustness features, so that the application can recover even if a disturbance or an attack occurs before the very first instruction is fetched by the CPU.

The STM8S and STM32 devices have an embedded boot loader, making it possible reprogram the internal Flash memory with an on-board serial interface (the UART for instance). Any PC with a serial COM interface can then be used as a programming tool to program or update the Flash and data EEPROM memory content. ST provides a software utility to perform all operations supported by the boot loader.

3.4 Safety

The automotive industry first pushed for increasing the reliability of MCU-based electronic controls. This has been followed by similar requests from the industrial segments, and household appliances now have to comply with a specific standard, IEC60335-1. Both the STM32 and STM8S devices are Class B compliant according to this standard. Compliance is obtained by using dedicated self-test libraries certified by an independent test institute, and also with the help of some specific hardware circuitry. Both the software and hardware contribute to significantly reduce the development and qualification time of applications with stringent functional safety requirements.

- Watchdogs

  The MCUs embed two watchdogs:

  – The Window Watchdog is intended to monitor the main loop and check that loop time is within a given time frame. It runs on the system clock.

  – In parallel, an independent watchdog can be activated to increase the system's robustness This watchdog will indeed continue to operate even in the case of a main clock failure (for instance due to a broken crystal).

- Clock monitoring

  The standard also requires the detection of crystal failure or oscillations at harmonics/ subharmonics. This is achieved using the clock system described in *Section 3.2: Clock*, to periodically measure the external crystal or resonator frequency with the internal clock source. Finally, a clock security system (CSS) also monitors the HSE source and automatically switches back the system clock to the internal HSI clock in case of a failure.

3.5 Low power

On top of the core's intrinsic low power modes, both the STM8S and STM32 devices are able to reduce the overall consumption at system-on-chip level.

The power consumption in the Run and Wait modes can be reduced by one of the following means:

- Slowing down the system clocks: the consumption can thus be adjusted according to the performance required by the application. This is done using the prescalers included in the clock controller.
- Gating the clocks of the peripherals when they are not used to minimize the dynamic consumption related to the clock tree switching activity.

The two products embed regulators to supply the internal logic at 1.8 V. These regulators have a significant operating current in Run mode (a few tens of µA) where they are able to deliver currents in the mA or tens of mA range. In order to further reduce consumption, it is possible to configure the regulator in low power mode, and minimize its quiescent consumption, when the current necessary to supply the logic is in the µA range, typically during the Halt or Stop mode. This mode offers the lowest consumption, with a wakeup time slightly longer than the configuration using the regulator in Run mode.

4 Software library

Peripheral compatibility throughout ST's STM8 and STM32 MCU families promotes platform design and helps significantly switch from one product line to the other. When it comes to development time, however, software support is essential. Extensive software libraries are available for both the STM8S and STM32 devices, providing the user with a hardware abstraction layer (HAL) for all MCU resources. Moreover, there is not a single control/status bit that is not covered by a C function or an API.

The software library covers three abstraction levels, and it includes:

1. a complete register address map with all bits, bit fields and registers declared in C. By providing this map, the software library makes the designers' task much lighter and, even more importantly so, it gives all the benefits of a bug-free reference mapping file, thus speeding up the early project phase.
2. a collection of routines and data structures in API form, that covers all peripheral functions. This collection can directly be used as a reference framework, since it also includes macros for supporting core-related intrinsic features and common constant and data type definition. Moreover, it is compiler agnostic and can therefore be used with any existing or future toolchain. It was developed using the MISRA C automotive standard.
3. a set of examples covering all available IPs (85 examples so far for the STM32 family, 57 for the STM8S family), with template projects for the most common development toolchains. With the appropriate hardware evaluation board, only a few hours are needed to get started with a brand new microcontroller.

It is then up to you to choose how to use the library. You can either pick up the files useful for the design, use examples to get trained or quickly evaluate the product. You can also use the API to save development time.

Let us now have a look at the few key files and concepts. Two separate libraries support the STM8S and STM32F devices. In the file names below, you simply need to replace the "*stmxxx_*" prefix by "*stm32f10x*" or "*stm8s*" depending on the chosen product.

- *stmxxx_.h*

  This file is the only header file that must be included in the C source code, usually in *main.c*. This file contains:

  – data structures and address mapping for all peripherals
  – macros to access peripheral register hardware (for bit manipulation for instance), plus STM8S core intrinsics
  – a configuration section used to select the device implemented in the target application. You also have the choice to use or not the peripheral drivers in the application code (that is code based on direct access to registers rather than through API drivers)

- *stmxxx_conf.h*

  This is the peripheral driver configuration file, where you specify the peripherals you wants to use in your application, plus a few application-specific parameters such as the crystal frequency.

- *stmxxx_it.c*

  This file contains the template IRQ handler to be filled, but this is already the first development step!

Once you have understood the above operating principle and file organization, for simple applications, you could virtually switch from one product to the other without referring to the reference manual.
Let us take a practical example: an SPI peripheral configured in master mode, used to read from/write to an external EEPROM.

Figure 4 and *Figure 5* below show the initialization code (using the software library) for an STM8S and an STM32 product, respectively.

Figure 4. STM8S code example

```
/* --------------- Initialize SPI in Master mode -------------- */
SPI_Init(SPI_FIRSTBIT_MSB,
SPI_BAUDRATEPRESCALER_4,
SPI_MODE_MASTER,
SPI_CLOCKPOLARITY_LOW,
SPI_CLOCKPHASE_2EDGE,
SPI_DATADIRECTION_1LINE_TX,
SPI_NSS_SOFT,
0x07); /* CRC Polynomial */
/* ------------------------- Enable SPI ----------------------- */
SPI_Cmd(ENABLE);
```

Figure 5. STM32 code example

```
  /* Private variables ------------------------------------- */
  SPI_InitTypeDef SPI_InitStructure;

/* ---------------------- SPI1 Master ------------------------- */
SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_4;
SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
SPI_InitStructure.SPI_Direction = SPI_Direction_1Line_Tx;
SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
SPI_InitStructure.SPI_CRCPolynomial = 7;
SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
SPI_Init(SPI1, &SPI_InitStructure);
/* ---------------------- Enable SPI1 ----------------------- */
SPI_Cmd(SPI1, ENABLE);
```

All parameters are identical, and the procedure is similar with two function calls for both configuration and startup. The main difference lies in the way the parameters are passed into the function. STM32 devices use a structure passed by address whereas, for STM8S devices, parameters are passed directly to minimize the amount of RAM needed during the initialization phase (this is necessary with devices with down to 1 Kbyte of RAM).
Another difference is the possibility, when using STM32 devices, of specifying the data size (8- or 16-bit) for the SPI. In the above example (*Figure 5*), the data size is explicitly defined, however, the library is done so that it can be omitted: if this field is not initialized in the structure, the 8-bit data size is used by default to maintain compatibility with STM8S devices.

5 Conclusion

This application note discusses the points that ease the transition from the 8-bit STM8S to the 32-bit STM32 devices, and vice versa. Based on the 8- to 32-bit core performance continuum, the new STM32 and STM8S MCU families have a lot of common features. At the peripheral level, they share standard IPs like timers and communication interfaces. At the system level, they have identical features, reducing the external component count (clock and reset systems, safety features, etc).

These common features are complemented by a set of software libraries that come as a major help to get started for new development. The libraries can also serve as foundations for a unified development platform supporting both 8- and 32-bit MCUs owing to the abstraction level they both offer.

Finally, the common features of the STM8S and STM32 devices with the benefits of their software libraries maximize design re-use and decrease time to market, specially if the application has derivatives with various requirements in terms of processing power, connectivity or control function complexity.

6 Revision history

Table 6. Document revision history

Date	Revision	Changes
28-Jul-2009	1	Initial release.