AN3117
Application note

Using the STM8L15x/STM8L162 DMA controller

Introduction

This application note describes how to use the direct memory access (DMA) controller of the STM8L15x and STM8L162.

The DMA controller, the STM8™ core, and the memory system contribute to provide a high data bandwidth and to develop very-low latency response time software.

This application note also describes how to take full advantage of these features and ensure correct response times for different peripherals and subsystems.

Note: All examples and explanations in this document are based on the STM8L15x firmware library and STM8L15x/STM8L162 microcontroller family reference manual (RM0031). DMA refers to the STM8L15x and STM8L162 DMA1.

Contents

List of tables

List of figures

1 DMA controller description

The STM8L15x DMA controller is a system peripheral used for transferring data between the local memory and the main memory without the intervention of the central processor unit (CPU).

Once the DMA registers have been configured, large blocks of data can be transferred at high speed between peripherals and a memory, or from one memory location to another. Moving data with the DMA keeps CPU resources free for other operations and allows computation and data transfer concurrency.

This is especially useful in real-time computing applications where not stalling the CPU during concurrent operations is critical, and when data processing and transfer must be performed in parallel to achieve sufficient throughput.

1.1 DMA main features

- Four DMA channels shared among several peripherals
- Three transfer directions
  - Peripheral to memory
  - Memory to peripheral
  - Memory to memory
- Hardware and software channel priorities which allow arbitration
- Programmable number of “data to be transferred” (up to 255 data blocks)
- Capability to increment and decrement memory addressing mode
- Optional interrupt on half transactions and end of transactions
- Two transfer block sizes (8-bit and 16-bit data) which are programmable by software.
- Circular buffer management (auto-reload mode)
- Capability to suspend and to resume DMA transfers
- Capability to operate in Low power modes (Wait for interrupt or Wait for event)

Each channel is assigned to a unique peripheral (data channel) at a given time. Peripherals connected to the same DMA channel (see *Table 2*) cannot be used simultaneously when the DMA controller is active.

The DMA controller performs direct memory transfers by sharing the address and data bus with the STM8 core. The DMA request may stop CPU access to the bus for some bus cycles, for example, when the CPU and DMA are targeting the same destination (memory or peripheral).

The arbitration between DMA and CPU is performed inside the STM8 core. Refer to the STM8 core description for further information.

1.2 Peripherals supported by DMA

The different peripherals supporting DMA transfers are listed in *Table 2*. The peripherals served by the DMA and the bus system structure are represented in *Figure 1*.

Table 1. Peripherals served by DMA and channels allocation (medium+ and high density devices)

Peripheral	DMA Ch.0 request source	DMA Ch.1 request source	DMA Ch.2 request source	DMA Ch.3 request source
ADC1 (1)	EOC	EOC	EOC	EOC
SPI1		SPI1_RX	SPI1_TX	
I2C	I2C_RX			I2C_TX
USART1		USART1_TX	USART1_RX	
DAC		DAC_CH2TRIG		DAC_CH1TRIG
TIM2	TIM2_CC1	TIM2_U		TIM2_CC2
TIM3	TIM3_U	TIM2_CC1	TIM3_CC2	
TIM1	TIM1_CC3	TIM1_CC4	TIM1_U TIM1_CC1 TIM1_COM	TIM1_CC2
USART2	USART2_TX			USART2_RX
USART3		USART3_TX	USART3_RX	
SPI2	SPI2_RX			SPI2_TX
TIM5	TIM5_U		TIM5_CC1	TIM5_CC2
AES	AES_IN			AES_OUT
TIM4(2)	TIM4_U	TIM4_U	TIM4_U	TIM4_U

Note: 1 ADC1 can be mapped on each of the four channels: depending on the SYSCFG_RMPCR1[1:0] bits (please refer to the ADC chapter and the SYSCFG registers chapter in the RM0031 reference manual). The default mapping is Channel 0.

2 TIM4 can be mapped on each of the four channels: depending on the SYSCFG_RMPCR1[3:2] bits (please refer to Timer chapter athe SYSCFG registers chapter in the RM0031 reference manual). The default mapping is Channel 3.

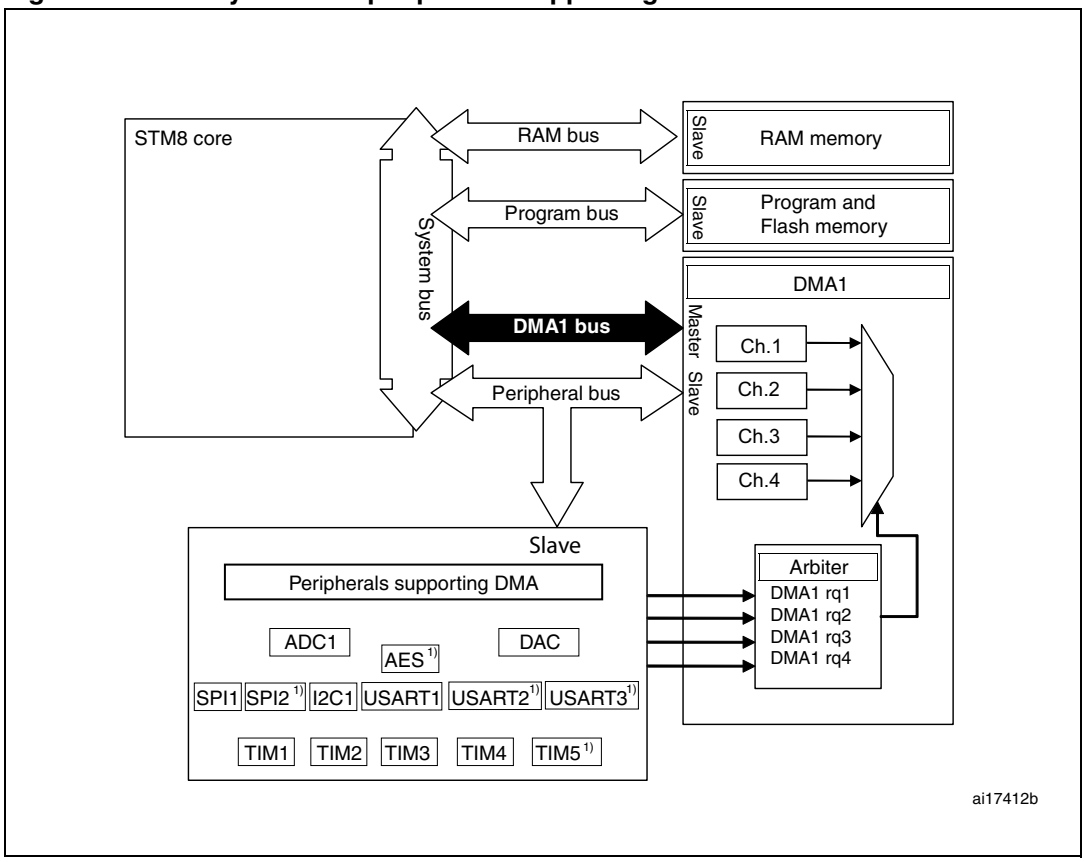
Table 2. Peripherals served by DMA and channels allocation (medium density devices)

Peripheral	DMA Ch.0 request source	DMA Ch.1 request source	DMA Ch.2 request source	DMA Ch.3 request source
ADC1	EOC	EOC	EOC	EOC
SPI1	–	RX	TX	–
I2C1	RX	–	–	TX
USART1	–	TX	RX	–
DAC	–	–	–	TRIG
TIM1	CC3	CC4	TIM1_U/CC1/COM	CC2
TIM2	CC1	TIM2_U	-	CC2
TIM3	TIM3_U	CC1	CC2	-
TIM4	TIM4_U	TIM4_U	TIM4_U	TIM4_U

Note: 1 ADC1 can be mapped on each of the four channels: depending on the SYSCFG_RMPCR1[1:0] bits (please refer to the ADC chapter and the SYSCFG registers chapter in the RM0031 reference manual). The default mapping is Channel 0.

2 TIM4 can be mapped on each of the four channels: depending on the SYSCFG_RMPCR1[3:2] bits (please refer to Timer chapter athe SYSCFG registers chapter in the RM0031 reference manual). The default mapping is Channel 3.

Figure 1. Bus system and peripherals supporting DMA



1. Peripherals available on medium+ and high density devices only. AES available on STM8L162 only.

The DMA is able to transfer data from the memory to a peripheral.

- For regular DMA channels (0, 1 and 2), data transfer can be from (or to) a part of the RAM memory (see note1).
- For memory DMA channel (channel 3), 2 cases are available :
  a) When a peripheral-to-memory or memory-to-peripheral transfer is performed, the memory areas can be all addressable areas (see note 2).
  b) When a memory-to-memory transfer is performed, the source memory areas are all addressable areas (see note 2), while the destination memory areas are the first 8 Kbytes (see note 3).

Note: 1 For medium and medium+ density devices, 2-Kbyte RAM memory is accessible (between 0x0000 and 0x07FF).
For high density devices, 4-Kbyte RAM memory is accessible (between 0x0000 and 0x0FFF).

2 Source memory addressable areas are comprised between 0x0000 and 0xFFFF for medium density devices and between 0x0000 and 0x17FFF for high density devices.

3 Destination memory addressable areas are comprised between 0x0000 and 0x1FFF, which corresponds to the RAM and the Data EEPROM.

2 Performance considerations

The DMA controller performs direct memory transfers by sharing the address and data bus with the STM8™ core. DMA requests may stop CPU accesses to the bus during some bus cycles, when the CPU and the DMA are both targeting memory and peripheral. The arbitration is performed inside the STM8 core.

In addition, the DMA controller can signal to the STM8 core that the current access must have priority over the CPU. There are two ways to do this:

- The application specifies the timeout duration (number of wait cycles starting from the latest request) by configuring TO[5:0] bits in the DMA_GCSR register. The DMA then waits until this timeout has elapsed before requesting from the core a high priority access to the bus.
- The application configures a channel so that it always takes priority over the CPU.

Refer to *Section 4.6: DMA channel priority transfer example* for more details.

2.1 DMA latency

Three operations are required to perform a DMA data transfer from peripheral to RAM (for example SPI reception):

1. DMA request arbitration and address computation
2. Reading data from the peripheral (DMA source)
3. Writing loaded data in RAM (DMA destination)

When transferring data from RAM to peripheral (for example SPI transmission), the operations are performed in the opposite order:

1. DMA request arbitration and address computation
2. Reading data from RAM memory (DMA source)
3. Writing data to the peripheral

The service time per channel for 8-bit data transfer, t_S, is given by the equation below:

$$t_S = t_A + t_{ACC} + t_{RAM}$$

where:

- t_A is the arbitration and address computation time
  t_A = 1 system clock cycle
- t_{ACC} is the peripheral access time
  t_{ACC} = 1 system clock cycle
- t_{RAM} is the RAM read or write access time
  t_{RAM} = 1 system clock cycle

As a result, the total latency for 8-bit data transfer is 3 system clock cycles.

The service time per channel for 16-bit data transfer, t_S, is given by the equation below:

$$t_S = t_A + t_{ACC16} + t_{RAM16}$$

where:

- t_A is the arbitration and address computation time
  t_A = 1 system clock cycle
- t_{ACC16} is the peripheral access time
  t_{ACC16} = 2 system clock cycles
- t_{RAM16} is the RAM read or write access time
  t_{RAM16} = 2 system clock cycles

As a result, the total latency for 16-bit data transfer is 5 system clock cycles.

When the DMA is idle, it compares the priorities of all pending DMA requests (software and hardware priorities, in this order). The highest priority channel is served and the DMA jumps to execute the second operation. While a channel is being served (operation 2 or 3 ongoing), no other channel can be served whatever its priority.

As a result, when a DMA transfer is on going, the DMA latency for the highest priority channel is the sum of the ongoing transfer time (without the arbitration phase) and the transfer time for the next DMA channel to be served (highest pending priority).

2.2 Data bus bandwidth

When the DMA has priority over the CPU, the maximum data transfer rate is obtained with 16-bit data transfers at 16 MHz. Two bytes are transferred in 5 clock cycles, which makes a transfer rate of 6.4 Mbytes/s.

For 8-bit data transfers, the maximum rate is 1 byte in 3 clock cycles, that is transfer rate of a 5.3 Mbytes/s.

The data rate is reduced when the CPU takes priority over the DMA for RAM and peripheral accesses.

3 Programming the DMA using the STM8L15x standard firmware library

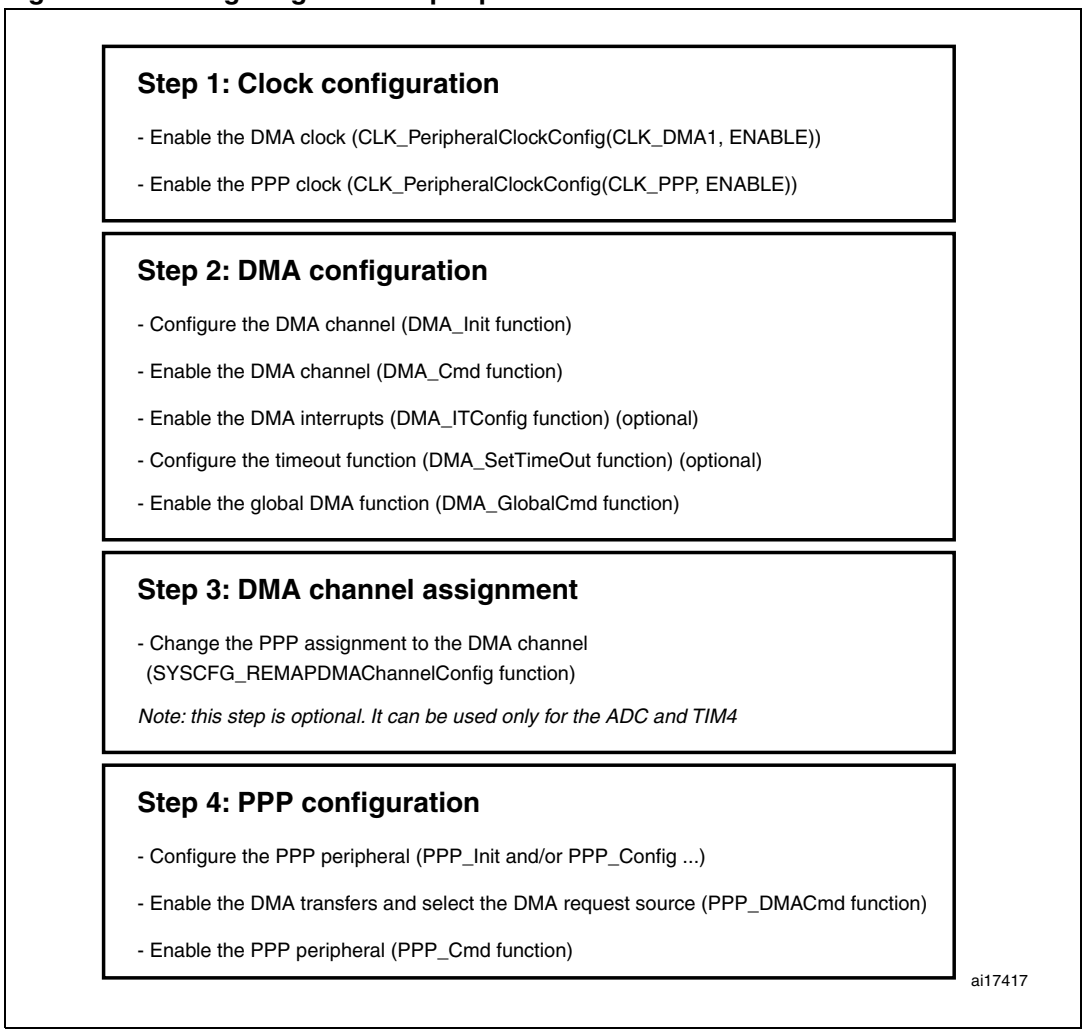
The STM8L15x standard peripheral library is an easy-to-use complete firmware package consisting of device drivers for all the standard device peripherals. It is available for download, together with this application note, from the STMicroelectronics website: http://www.st.com/mcu/familiesdocs-120.html.

Each device driver includes a set of functions covering full peripheral functionality. The STM8L15x DMA driver provides the user with an easy way to configure and supervise the DMA peripheral.

3.1 Configuring the DMA peripheral to transfer data

To configure the DMA peripheral to transfer data, three mandatory steps (steps 1, 2, and 4) and one optional step (step 3) are required. These steps are described in *Figure 2*. The user can adapt these steps to suit his application.

Figure 2. Configuring the DMA peripheral to transfer data



Note: PPP refers to a peripheral that supports DMA transfers, such as ADC1, DAC, USARTx (x=1...3), I2C, SPIx (x=1;2), TIMx(x=1...5) and AES.

3.1.1 Configuring the DMA channels

The DMA driver provides the **DMA_Init()** function which can be used to configure all DMA channels.

```
DMA_Init(DMA_Channelx. /* select Channel to be configured       */
         DMA_Memory0BaseAddr,/* RAM memory Base Address */
         DMA_PeripheralMemory1BaseAddr,
                          /* (case1) Peripheral base address for
                             memory to peripheral transfer
                             (case2) Memory base address for
                             memory to  memory transfer (this
                             feature exist only in Channel 3 */
         DMA_BufferSize,     /* number of transactions */
         DMA_DIR,            /* Direction of the transfer */
         DMA_Mode,             /*  simple or circular transfer. */
```

```
DMA_MemoryIncMode,      /*  memory increment mode */
DMA_Priority,           /*  software priority */
DMA_MemoryDataSize );   /* number of transfers */
```

Table 3 describes the DMA_Init function parameters and their corresponding parameter types.

Table 3. DMA_Init parameters

Parameter name	Parameter type
DMA_Channelx	DMA_Channel_TypeDef
DMA_Memory0BaseAddr	uint16_t
DMA_PeripheralMemory1BaseAddr	uint16_t
DMA_BufferSize	uint8_t
DMA_DIR	DMA_DIR_TypeDef
DMA_Mode	DMA_Mode_TypeDef
DMA_MemoryIncMode	DMA_MemoryIncMode_TypeDef
DMA_Priority	DMA_Priority_TypeDef
DMA_MemoryDataSize	DMA_MemoryDataSize_TypeDef

Note: Refer to the stm8l15x_dma.h file for more details about the DMA functions and their parameters.

3.1.2 Selecting the channel request source

USARTx, SPIx, I2C1

Each communication peripheral (USARTx (x=1..3), SPIx (x=1,2) and I2C1) has two event request sources: one for transmission (TX) and one for reception (RX).

Selection of the desired DMA request is performed by the driver of the communication peripheral. *Table 4* describes the DMA request configuration functions. *Table 5* lists the DMA channel allocation for the appropriate communication peripheral.

Table 4. DMA request configuration functions from the communication peripheral side

Peripheral	Function	Description
USARTx	**USART_DMACmd()**	Enables or disables the USART DMA requests. The USARTx (x= 1...3) TX or RX requests are selected independently.
SPIx	**SPI_DMACmd()**	Enables or disables the SPI DMA requests. The SPIx (x= 1, 2)TX or RX requests are selected independently.
I2C1	**I2C_DMACmd()**	Enables or disables the I2C DMA requests. Both I2C1 TX and RX requests are selected.

Table 5. Communication peripheral DMA channel allocations

Peripheral	RX request	TX request
USART1	DMA Ch.2	DMA Ch.1
USART2	DMA Ch.3	DMA Ch.0
USART3	DMA Ch.2	DMA Ch.1
SPI1	DMA Ch.1	DMA Ch.2
SPI2	DMA Ch.0	DMA Ch.3
I2C1	DMA Ch.0	DMA Ch.3

Note: Refer to the stm8l15x_spi.h, stm8l15x_usart.h, and stm8l15x_i2c.h files for further details about the SPI_DMACmd(), USART_DMACmd(), and I2C_DMACmd() functions and their parameters.

Digital-to-analog convertor (DAC)

A DAC channel DMA request is generated when a software or hardware trigger occurs while the trigger feature and the DAC DMA feature are enabled (using **DAC_Init()** and the **DAC_DMACmd()** functions). The DAC channel DMA request remains set until a DAC channel DMA acknowledge comes from the DMA controller. The DAC DMA request indicates that DAC data holding registers have been transferred to the DAC data output registers.

Analog-to-digital convertor (ADC)

ADC requests can be activated or deactivated by programming the DMA control feature (using the **ADC_DMACmd()** function).

As soon as an ADC end of conversion is detected, the ADC_EOC request is sent to the DMA, informing it that data are ready to be transferred.

By default, ADC1 is connected to DMA channel 3. However, this assignment can be remapped using the system configuration driver **SYSCFG_REMAPDMAChannelConfig()** function.

Note: Refer to the stm8l15x_syscfg.h file for further details about the SYSCFG_REMAPDMAChannelConfig() function and its parameters.

Timers

General procedure for all timers: DMA single mode

Timer DMA requests can be independently activated/deactivated by programming the DMA control feature (using the **TIMx_DMACmd()** functions where x =1, 2, 3 or 4).

Table 6 describes the timer DMA request sources and channel allocations.

Table 6. Timer DMA request sources and channel allocations

Timer	Update request	Capture Compare requests				COMmutation request
		CC1	CC2	CC3	CC4	
TIM1	DMA Ch.2	DMA Ch.2	DMA Ch.3	DMA Ch.0	DMA Ch.1	DMA Ch.2
TIM2	DMA Ch.1	DMA Ch.0	DMA Ch.3			
TIM3	DMA Ch.0	DMA Ch.1	DMA Ch.2			
TIM4	All DMA channels					
TIM5	DMA Ch.0	DMA Ch.2	DMA Ch.3			

Specific TIM1 feature: DMA Burst mode

The DMA can work in Burst mode with TIM1. In this mode, the DMA can transfer a block of data from/to a block of TIM1 registers. To configure Burst mode, use the STM8L15x TIM1 driver **TIM1_DMAConfig()** function.

Specific TIM4 feature: DMA channel affectation

For flexibility reasons, a DMA transfer from/to TIM4 can be remapped to any DMA channel (0, 1, 2 or 3).

By default, TIM4 is connected to DMA channel 0, however, this assignment can be changed using the system configuration driver **SYSCFG_REMAPDMAChannelConfig()** function.

Note: Refer to stm8l15x_syscfg.h file for further details about the SYSCFG_REMAPDMAChannelConfig() function and its parameters.

3.1.3 Enabling the DMA transfers

After configuring the DMA channels, the user should do the following:

1. Enable the corresponding DMA channel transfer (using the **DMA_Cmd()** function).
   ```
   DMA_Cmd(DMA_Channelx, ENABLE);
   ```
2. Enable the global DMA channel transfers (using the **DMA_GmobalCmd()** function).
   ```
   DMA_GlobalCmd(ENABLE);
   ```

Once the DMA request is enabled, the DMA transfer starts.

3.1.4 Configuring the arbitration delay

Arbitration between the DMA and CPU is performed inside the STM8 core. The DMA controller can inform the STM8 core that the current access should have priority over the CPU.

The DMA waits until the configured timeout (using the **DMA_SetTimeOut()** function) has elapsed before requesting from the core a high priority access to the system bus. One timeout duration is equal to one CPU cycle.

When the timeout delay is programmed to 0, it means there is no timeout. Once a request is served, the DMA immediately asks the CPU for a high priority access to the system bus.

3.2 Supervising the DMA transfer

3.2.1 Method 1: Current data counter

A DMA transfer consists of a single read/write operation from/to a data block. It cannot be interrupted.

A DMA transaction consists of a complete DMA read and write operation of a given number of data blocks. They can be divided into single transfers.

A data block consists of either 8-bits or 16-bits of data depending on the size of the block that has been programmed.

The number of transfers is indicated by the DMA_BufferSize parameter that the user chooses during DMA channel initialization (using the **DMA_Init**() function).

The STM8L15x DMA driver provides a function which returns the current data counter.

By reading the current data counter, using the **DMA_GetCurrDataCounter()** function, the user learns the number of remaining transfers.

3.2.2 Method 2: Flags/interrupts

The DMA peripheral provides two flags for each channel which are shown in *Table 7*.

Table 7. DMA flags

Flag	Description
HTIF	Half transfer interrupt flag
TCIF	Transfer complete interrupt flag

If enabled as interrupt (using the **DMA_ITConfig()** function), each of the above flags can generate an interrupt which alerts the user of the transaction status (using the **DMA_GetFlagStatus()** / **DMA_GetITStatus()** functions). As both flags are cleared by software, they must be cleared after their corresponding channel transfer (using **DMA_ClearFlag()** / **DMA_ClearITPendingBit()** functions).

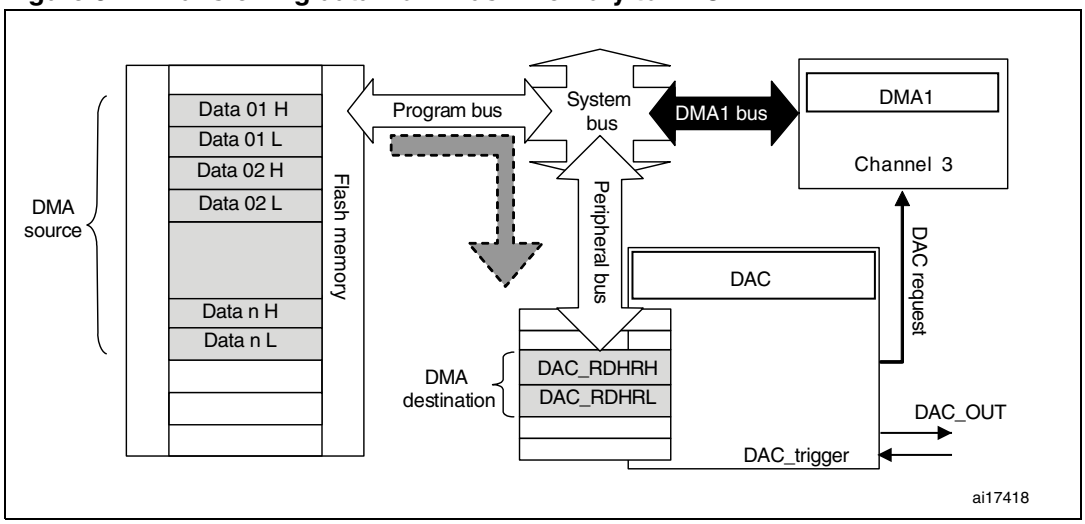
4 DMA programming examples

All the examples described below use the STM8L15x Standard peripheral library and are provided in the firmware package associated with this application note.

4.1 Memory to peripheral transfer example: transferring waveform data from Flash memory to DAC

This example uses the DAC peripheral to generate a signal (stored in a Flash memory buffer) using DMA memory to peripheral mode.

Figure 3. Transferring data from Flash memory to DAC



The DMA event is provided by the external hardware DAC trigger (TIM4_TRGO). Each time the external DAC trigger occurs, the DMA transfers data from the Flash memory buffer to the DAC data holding registers.

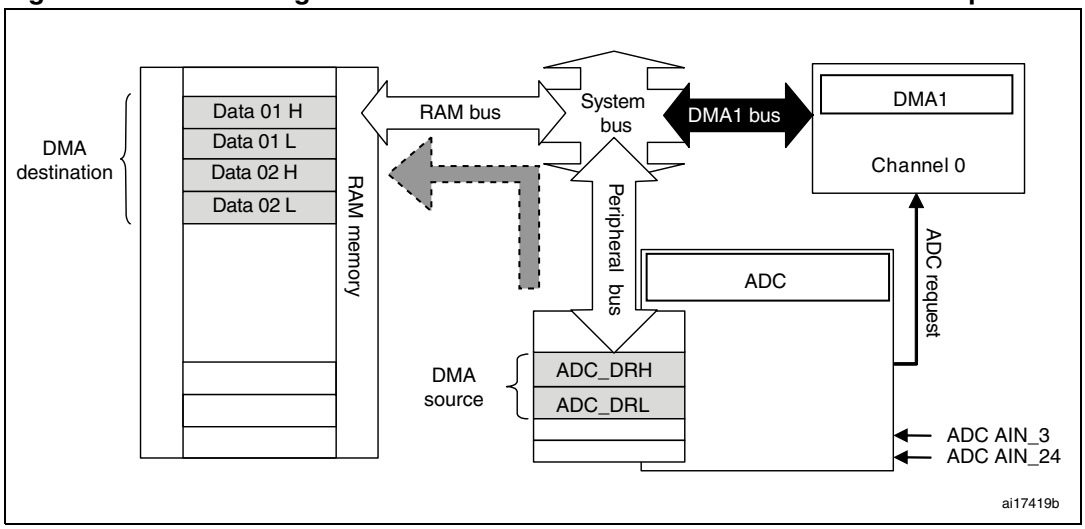
Since the DMA is configured in circular mode, the DAC is triggered periodically by TIM4. The DMA transfer from the Flash memory buffer to the DAC is repeated indefinitely.

The code for this example is provided in the STM8L15x standard peripheral library package available from http://www.st.com/mcu/familiesdocs-120.html. It is located under *\STM8L15x_StdPeriph_Lib\Project\STM8L15x_StdPeriph_Examples\DAC\DAC_SignalsGeneration*.

4.2 Peripheral to memory transfer example 1: transferring data from the ADC to the RAM via multi channel acquisitions

This example uses the ADC peripheral to convert two ADC input voltage channels in scan mode using DMA.

Figure 4. Transferring data from the ADC to the RAM via multi channel acquisition



The DMA event is provided by the ADC_EOC event. Each time the ADC completes the conversion of the first input channel, it sends the end of conversion request to the DMA. As soon as the DMA receives the request, it transfers the ADC data register value to the RAM buffer (position 01). During this time, the ADC completes the conversion of the second input channel and sends another end of conversion request to DMA. This request enables the transfer of the ADC data register values to the RAM buffer (position 02).

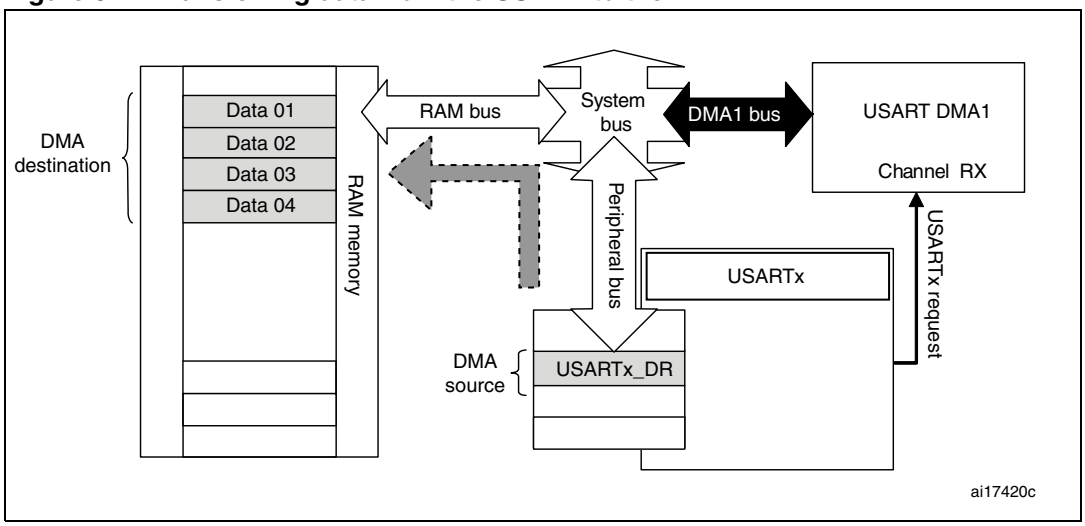
Since the ADC is configured in continuous mode and the DMA is configured in circular mode, ADC acquisition and transfer to the RAM are repeated indefinitely.

The code for this example is provided in the STM8L15x standard peripheral library package available from http://www.st.com/mcu/familiesdocs-120.html. It is located under *\STM8L15x_StdPeriph_Lib\Project\STM8L15x_StdPeriph_Examples\ADC\ADC_DMA*.

4.3 Peripheral to memory transfer example 2: transferring data from the USART to the RAM

This example uses the DMA peripheral in circular mode to transfer data from the USARTx peripheral to the RAM memory. When a character is received by USARTx, it is transferred to a specific RAM buffer using the DMA.

Figure 5. Transferring data from the USART to the RAM



The DMA event is provided by the USART_RX.

The code for this example is provided in the STM8L15x standard peripheral library package available from http://www.st.com/mcu/familiesdocs-120.html. It is located under *\STM8L15x_StdPeriph_Lib\Project\STM8L15x_StdPeriph_Examples\DMA\DMA_USARTtoRAM*.

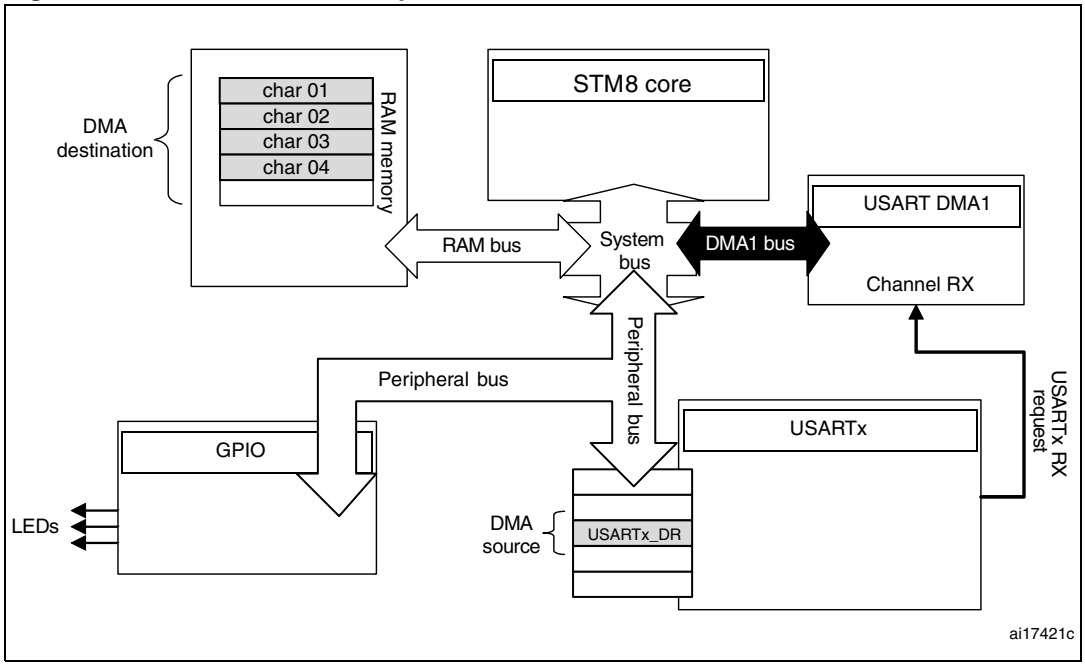
Note: USART DMA Channel RX can be DMA channel 2 or DMA channel 3 respectively for USART1 and USART2.

4.4 DMA transfer examples in low power mode

4.4.1 DMA transfer example in Wait for event (WFE) mode

This example uses the DMA peripheral in circular mode to transfer data from the USARTx peripheral to the RAM memory. When the user sends a character using USARTx, it is transferred to a specific RAM buffer using the DMA.

Figure 6. DMA transfer example in WFE mode



The DMA event is provided by the USARTx_RX.

After configuring USART DMA Channel RX to transfer 4 bytes of data from the USARTx data register to a RAM buffer, the MCU enters WFE mode.

As long as no “DMA transfer complete” event has occurred, the MCU is in WFE mode. As soon as the “DMA transfer complete” event occurs, the MCU returns to Run mode and executes a specific action (activate/deactivate LEDs) depending on the data in the RAM buffer. The MCU is then returned to WFE mode by software. Since USART DMA Channel RX is configured in circular mode, the same procedure is repeated indefinitely.

The code for this example is provided in the STM8L15x standard peripheral library package available from http://www.st.com/mcu/familiesdocs-120.html. It is located under *\STM8L15x_StdPeriph_Lib\Project\STM8L15x_StdPeriph_Examples\DMA\DMA_WFE*.

Note: USART DMA Channel RX can be DMA channel 2 or DMA channel 3 respectively for USART1 and USART2.

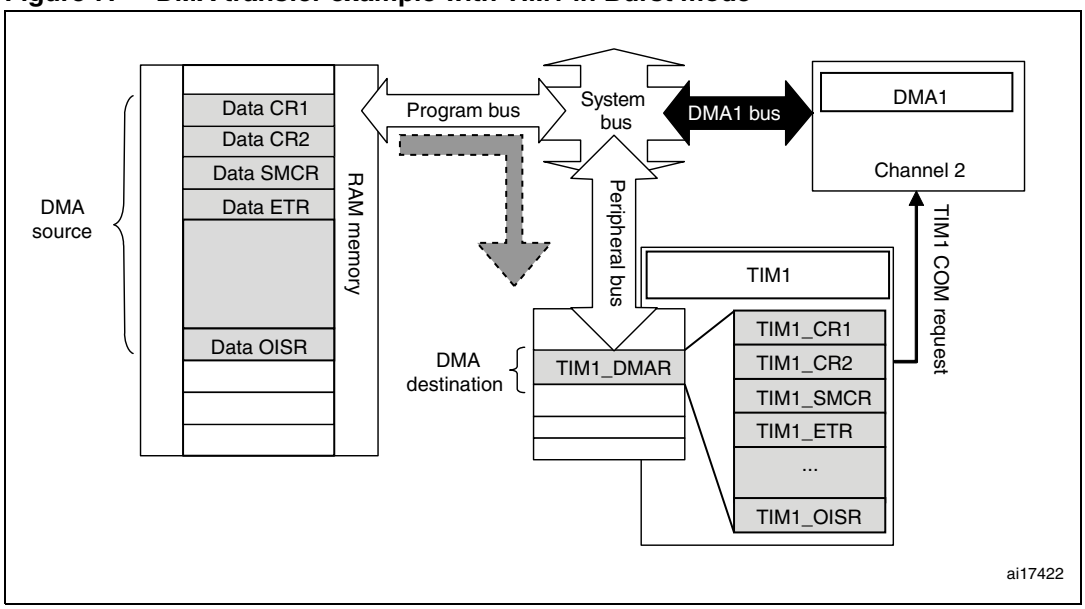
4.4.2 DMA transfer example in Wait for interrupt (WFI) mode

This example uses the DMA peripheral to transfer data in exactly the same way as the WFE example. The difference is that the treatment (LED activation/deactivation depending on the data in the RAM buffer) is performed by the DMA transfer complete interrupt routine, without returning to the main program.

The code for this example is provided in the STM8L15x standard peripheral library package available from http://www.st.com/stm8l. It is located under *\STM8L15x_StdPeriph_Lib\Project\STM8L15x_StdPeriph_Examples\DMA\DMA_WFI*.

4.5 DMA transfer example with TIM1 in Burst mode

Figure 7. DMA transfer example with TIM1 in Burst mode



This example uses the DMA peripheral in Burst mode to transfer data from the RAM memory to TIM1. TIM1 is configured in pulse width modulation (PWM) output mode and DMA channel 2 is configured to transfer data from the RAM memory to the TIM1 registers (TIM1_PSCRH, TIM1_PSCRL, TIM1_ARRH, TIM1_ARRL, TIM1_RCR, TIM1_CCR1H, TIM1_CCR1L). The goal of this data transfer is to update simultaneously TIM1 channel 1 frequency and duty cycle when an EXTI event triggered by the key button occurs.

The code for this example is provided in the STM8L15x standard peripheral library package available from http://www.st.com/stm8l. It is located under *\STM8L15x_StdPeriph_Lib\Project\STM8L15x_StdPeriph_Examples\DMA\DMA_TIM1BurstMode*.

4.6 DMA channel priority transfer example

In the example below, the priority of the DMA channels changes which has an impact on the transfers. All transfers are 8 bits in size and have a length of 1.

The DMA channels are configured to perform the following transfers:

- DMA channel 0: **Transfer A**: from RAM buffer 2 to TIM3 register
- DMA channel 1: **Transfer B**: from RAM buffer 1 to TIM2 register
- DMA channel 2: **Transfer C**: from TIM1 register to RAM buffer 1
- DMA channel 3: **Transfer D**: from TIM4 register to RAM buffer 2

Before starting any transfer, the RAM buffers and timer registers are initialized as shown in *Table 8*.

Table 8. RAM buffers and timer registers at t = t0

Time	RAM Buffer1	TIM1 ARRL	TIM2 ARRL	TIM3 ARRL	TIM4 ARR	RAM Buffer2
t0	0x00	0x11	0x22	0x33	0x44	0x55

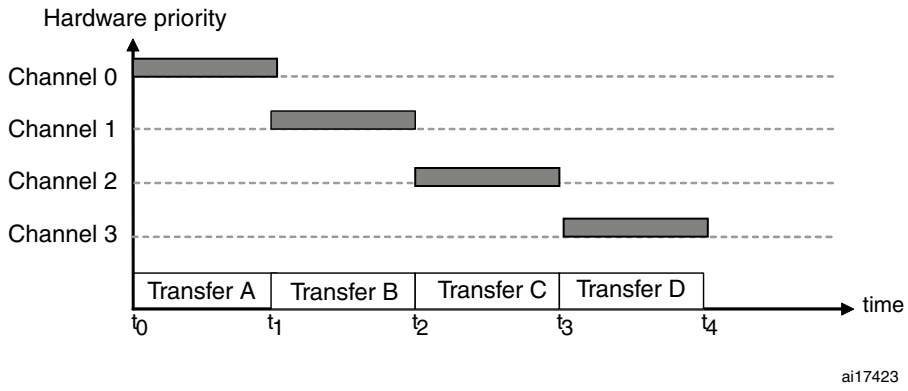
4.6.1 Transfer example case 1: hardware priority configuration of DMA channels

In this example, the priority of the channels is maintained at its default configuration: the hardware priority acts and the software priority stays the same for all channels.

Table 9. Example case1: priority configuration of the DMA channels

DMA channel	HW priority	SW priority
Channel 0	Very high	Low
Channel 1	High	Low
Channel 2	Medium	Low
Channel 3	Low	Low

Figure 8. Example case 1: DMA channel data transfers as a function of time



Transfer results are shown in *Table 10*.

Table 10. Results of example case 1

Time	RAM Buffer1	TIM1 ARRL	TIM2 ARRL	TIM3 ARRL	TIM4 ARR	RAM Buffer2
t0	0x00	0x11	0x22	0x33	0x44	0x55
t1	-	-	-	0x55	-	-
t2	-	-	0x00	-	-	-
t3	0x11	-	-	-	-	-
t4	-	-	-	-	-	0x44
Result	**0x11**	**0x11**	**0x00**	**0x55**	**0x44**	**0x44**

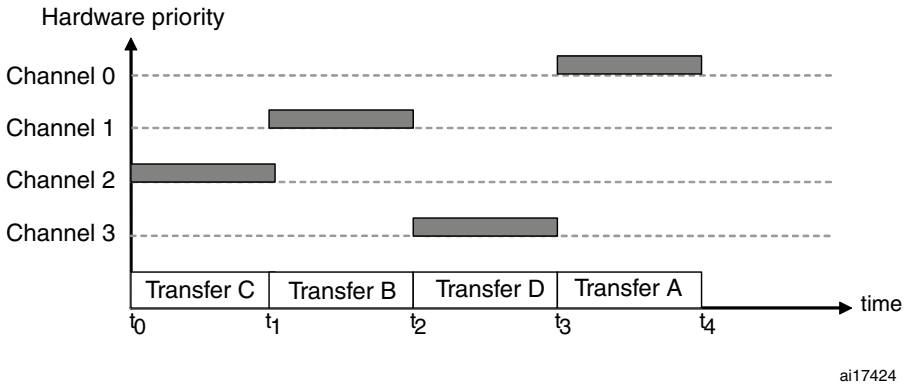
4.6.2 Transfer example case 2: software priority configuration 1 of the DMA channels

In this example, the software priority of the channels is updated as shown in *Table 11*.

Table 11. Example case 2: priority configuration of the DMA channels

DMA channel	HW priority	SW priority
Channel 0	Very high	Low
Channel 1	High	High
Channel 2	Medium	Very high
Channel 3	Low	Medium

Figure 9. Example case 2: DMA channel data transfers as a function of time



Transfer results are shown in *Table 12*.

Table 12. Results of example case 2

Time	RAM Buffer1	TIM1 ARRL	TIM2 ARRL	TIM3 ARRL	TIM4 ARR	RAM Buffer2
t0	0x00	0x11	0x22	0x33	0x44	0x55
t1	0x11	-	-	-	-	-
t2	-	-	0x11	-	-	-
t3	-	-	-	-	-	0x44
t4	-	-	-	0x44	-	-
Result	**0x11**	**0x11**	**0x11**	**0x44**	**0x44**	**0x44**

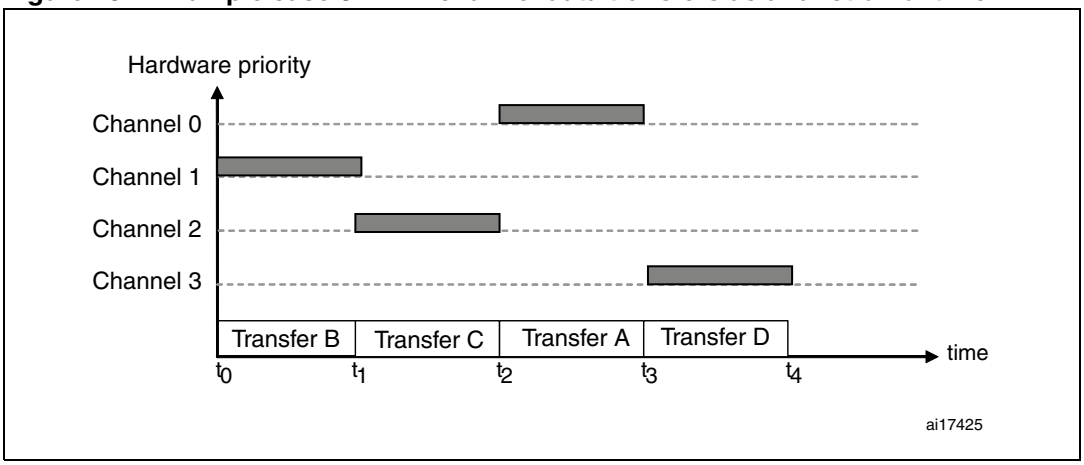
4.6.3 Transfer example case 3: software priority configuration 2 of the DMA channels

In this example, the software priority of the channels is updated as shown in *Table 13*

Table 13. Example case 3: priority configuration of the DMA channels

DMA channel	HW priority	SW priority
Channel 0	Very high	Medium
Channel 1	High	Very high
Channel 2	Medium	High
Channel 3	Low	Low

Figure 10. Example case 3: DMA channel data transfers as a function of time



Transfer results are shown in *Table 14*.

Table 14. Results of example case 3

Time	RAM Buffer1	TIM1 ARRL	TIM2 ARRL	TIM3 ARRL	TIM4 ARR	RAM Buffer2
t0	0x00	0x11	0x22	0x33	0x44	0x55
t1	-	-	0x00	-	-	-
t2	0x11	-	-	-	-	-
t3	-	-	-	0x55	-	-
t4	-	-	-	-	-	0x44
Result	**0x11**	**0x11**	**0x00**	**0x55**	**0x44**	**0x44**

The code for this example is provided in the STM8L15x standard peripheral library package available from http://www.st.com/mcu/familiesdocs-120.html. It is located under *\STM8L15x_StdPeriph_Lib\Project\STM8L15x_StdPeriph_Examples\DMA\DMA_Channels Priority.*.

5 Revision history

Table 15. Document revision history

Date	Revision	Changes
01-Feb-2010	1	Initial release
10-Sep-2010	2	Added “high density devices” information. Added *Table 1: Peripherals served by DMA and channels allocation (medium+ and high density devices) on page 7* Added one paragraph below *Figure 1: Bus system and peripherals supporting DMA on page 9* Modified *Section 3.1.1: Configuring the DMA channels on page 13* Added “high density devices” peripherals in *Table 4: DMA request configuration functions from the communication peripheral side on page 14*, *Table 5: Communication peripheral DMA channel allocations on page 15* Modified *Figure 4: Transferring data from the ADC to the RAM via multi channel acquisition on page 19*, *Figure 5: Transferring data from the USART to the RAM on page 20*, *Figure 6: DMA transfer example in WFE mode on page 21* Modified addresses where codes are available in *Section 4.4.1: DMA transfer example in Wait for event (WFE) mode on page 21*, *Section 4.5: DMA transfer example with TIM1 in Burst mode on page 22* and *Section 4.6.3: Transfer example case 3: software priority configuration 2 of the DMA channels on page 26* Modified *Section : Introduction* Modified title of *Table 1: Peripherals served by DMA and channels allocation (medium+ and high density devices)* Modified *Table 2: Peripherals served by DMA and channels allocation (medium density devices)* Modified footnote under *Figure 1: Bus system and peripherals supporting DMA* Modified note under *Figure 1: Bus system and peripherals supporting DMA* Modified title of *Section 3.1.2: Selecting the channel request source*

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZED ST REPRESENTATIVE, ST PRODUCTS ARE NOT RECOMMENDED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. ST PRODUCTS WHICH ARE NOT SPECIFIED AS "AUTOMOTIVE GRADE" MAY ONLY BE USED IN AUTOMOTIVE APPLICATIONS AT USER’S OWN RISK.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.

Information in this document supersedes and replaces all information previously supplied.

The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

© 2010 STMicroelectronics - All rights reserved

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com