
**Parallel synchronous transmission
using GPIO and DMA**

Introduction

The STM32 MCUs are able to emulate a parallel synchronous communication through the GPIO interface, using the embedded DMA IP. The frame transmitted consists of up to 16 bits of data and a synchronous data clock. The transmission requires usage of timers to generate the data clock and to control the transmission duration (timeout, number of data sent). STM32 MCUs may have different embedded IPs, for example DMA with or without double-buffering mode and FIFO. Also available Discovery boards may include or not some external memories (SDRAM). Such hardware differences will impact the way to implement the parallel synchronous communication and the possible performance that can be reached (max data frame length, max transmission frequency). Two different types of STM32 and Discovery boards have been used in this application note, to cover the above mentioned hardware differences and to explain the two different possible architecture implementations:

- the STM32L476G Discovery board (no DMA double-buffering mode, no DMA FIFO, no external SDRAM)
- the STM32F429I Discovery board (DMA with double-buffering mode and FIFO, external SDRAM)

The implementation architecture explanations will help to migrate the provided examples to other STM32-based boards. This application note is organized into two parts. It first gives an architecture implementation overview for both types of boards. Then it provides a description of the software examples, provided by the X-CUBE-PARAL-COM software package. This application note assumes that the reader is familiar with the DMA and timers of the STM32 as described in the STM32F42xxx reference manual, RM0329, and in the STM32L4x6 reference manual, RM0351 available from the www.st.com webpage.

The X-CUBE-PARAL-COM software package is delivered with this document and it contains the source code of the examples for the STM32 microcontrollers. It is available from the www.st.com webpage.

Reference documents and software

The following documents are available from the www.st.com webpage:

- *STM32L4x6 advanced ARM[®]-based 32-bit MCUs reference manual (RM0351)*
- *STM32F42xxx advanced ARM[®]-based 32-bit MCUs reference manual (RM0329)*
- *32L476GDISCOVERY Discovery kit with STM32L476VG MCU User manual (UM1879)*
- *32F429IDISCOVERY Discovery kit with STM32F429ZI MCU User manual (UM1662)*
- *Embedded software for STM32 L4 series (STM32CubeL4) Software package*
- *Embedded software for STM32 F4 series (STM32CubeF4) Software package*

Contents

- 1 Architecture overview 5**
- 1.1 Transmission 5
- 1.1.1 Data clock generation (using TIM PWM mode) 6
- 1.1.2 Data transmission (using DMA) 7
- 1.1.3 End of transmission (using DMA or TIM in master-slave mode) 9
- 1.2 Reception 10
- 1.2.1 Data reception (using DMA) 12
- 1.2.2 Data and clock alignment consideration 13
- 1.2.3 Detecting end of transmission (with TIM timeout) 15

- 2 Implementation code explanation 17**
- 2.1 STM32L476G example 18
- 2.1.1 Code usage 20
- 2.2 STM32F429I example 23
- 2.2.1 Code usage 25

- 3 Conclusion 28**

- 4 Revision history 29**

List of tables

Table 1.	STM32L476G example, achievable data frequency and associated tuning parameters	19
Table 2.	STM32L476G example, dma buffer size versus frequency for large data frame.	19
Table 3.	Transmission side parameters	20
Table 4.	Reception side parameters.	20
Table 5.	STM32F429I example, achievable data frequency and associated tuning parameters	24
Table 6.	Transmission side parameters	25
Table 7.	Reception side parameters.	25
Table 8.	Revision history	29

List of figures

Figure 1.	STM32L476 transmission architecture example.	5
Figure 2.	STM32F429 transmission architecture example.	6
Figure 3.	Transmission without double buffering using circular mode DMA	8
Figure 4.	Transmission with double buffering	9
Figure 5.	STM32L476 reception architecture example	11
Figure 6.	STM32F429 reception architecture example	11
Figure 7.	Reception without double buffering using circular mode DMA	12
Figure 8.	Reception with double buffering	13
Figure 9.	Data to clock synchronization	13
Figure 10.	Example with low-data frequency	14
Figure 11.	Example with high-data frequency	14
Figure 12.	STM32L476 timeout example.	15
Figure 13.	STM32F429 timeout example.	16
Figure 14.	The SW package structure	17
Figure 15.	Resistance R58 location on STM32L476 Discovery board.	21
Figure 16.	STM32L476G Discovery board connection and initialization configuration	21
Figure 17.	STM32L476G Discovery board after frame reception	22
Figure 18.	STM32L476G Discovery board CRC comparison	22
Figure 19.	Data frame storage in memory	24
Figure 20.	STM32F429I Discovery board connection and initialization configuration	26
Figure 21.	STM32F429I Discovery board CRC comparison	27

1 Architecture overview

1.1 Transmission

The data transmission requires to generate some data, create a data clock and present the generated data on GPIOs synchronously with the data clock.

The data is generated by the user and already stored in the memory (it could be internal SRAM or external memory). The number of data to transmit is not known by the receiving side, however the frequency (or frequency range) should be known to allow correct reception of the data frame.

The data clock transmitted is used to control the start and the end of the transmission.

Two different possible architectures have been considered, depending on which STM32 MCU is used (STM32L476 or STM32F429). These architectures can be re-used or adapted to other STM32 MCUs.

The details are discussed in next chapters, while the below block diagrams can be used as a reference for these chapters.

Note 1 In [Figure 1](#) and [Figure 2](#), the text inside the parenthesis refers to the implementation proposed in the code examples.

Note 2 In [Figure 1](#), the CPU and DMA buffer box are in dotted line and grey color because they have not been used in the implementation example. As it will be explained in [Section 1.1.2: Data transmission \(using DMA\)](#) a frame size < DMA 16-bit counter has been used and so there is no need to use circular DMA buffer mode.

Figure 1. STM32L476 transmission architecture example

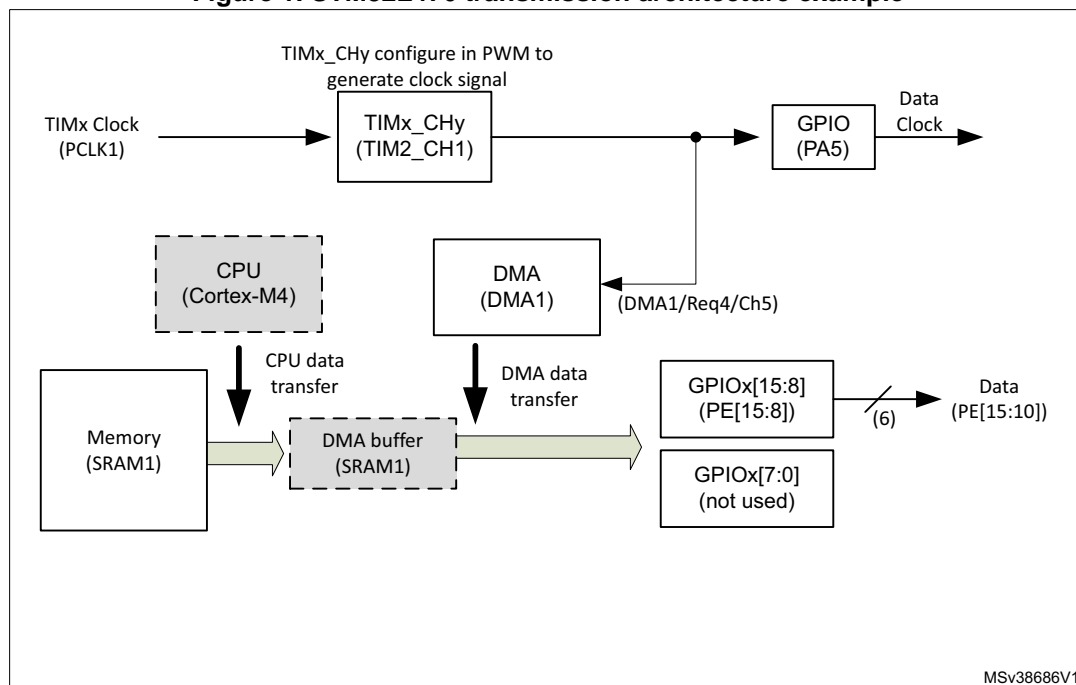
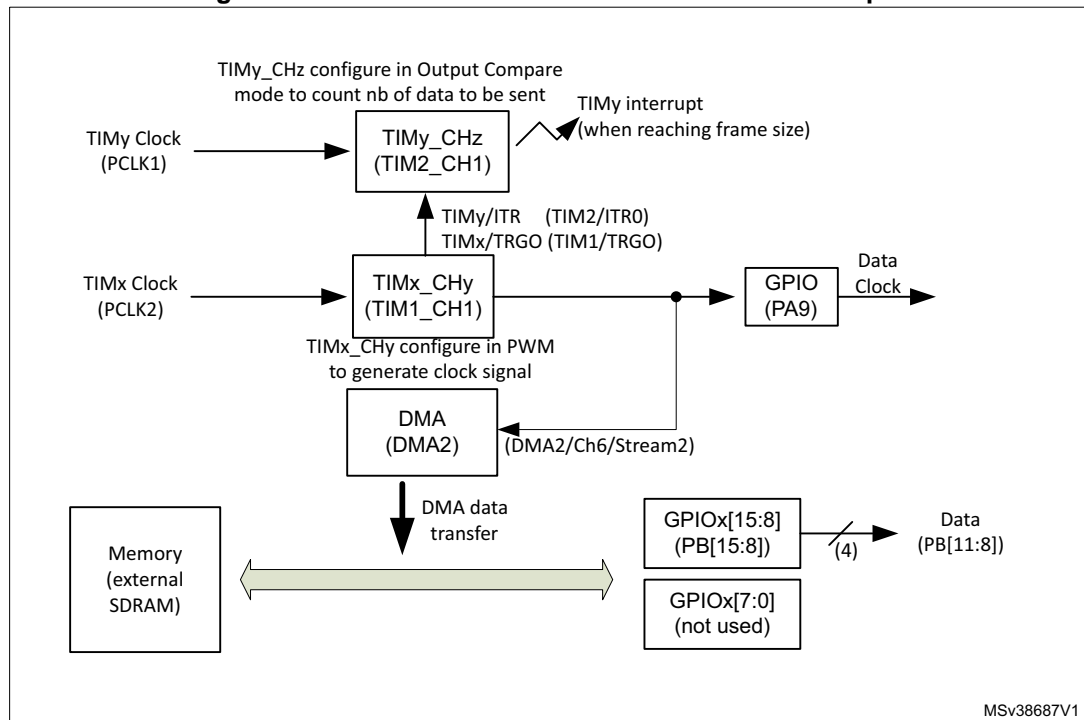


Figure 2. STM32F429 transmission architecture example



The user should note that in the above [Figure 1](#) and [Figure 2](#), the STM32L476 architecture is simplified, compared to the STM32F429 one. Indeed, in the STM32L476 example, the DMA counter is used to count the number of data to be sent, while in the STM32F429 example, an additional timer is used. This is due to the fact that in the STM32L476 example, only frames having less than 65535 data are sent, hence the 16-bit DMA counter is able to handle it. In the STM32F429 example a larger data frame, which requires a 32-bit counter implemented in the TIM2, is sent. It is possible to send a larger frame with STM32L476, using the same architecture as for the STM32F429. No additional timer is needed in case only small frame are transmitted.

1.1.1 Data clock generation (using TIM PWM mode)

The data clock is generated by a timer (TIMx) configured in Pulse Width Modulation (PWM) Edge Aligned mode on one of its channel (CHy).

So the TIMx_CHy output will toggle at a predefined frequency, and duty cycle.

The data clock frequency is computed from TIMx Auto-reload register (TIMx_ARR) and timer frequency:

$$\text{DataClockFreq} = (\text{TIMxfreq}) / (\text{TIMxARR} + 1)$$

The data clock duty cycle is a percentage computed from TIMx capture/compare register (TIMx_CCRy):

$$\text{DataClockDutycycle} = (\text{TIMxCCRy}) / (\text{TIMx1ARR} + 1) \times 100$$

The data clock duty cycle may be tuned to allow more time on reception side to handle receiving data detected on data clock edge. Refer to [Section 1.2.2: Data and clock alignment consideration](#).

The clock signal is an output for TIMx_CHy and should be connected to an available GPIO.

1.1.2 Data transmission (using DMA)

The data present in the memory must be transferred by DMA from the memory to GPIOs. Since GPIO bus is 16-bit wide, the max data size shall be 16 bits as well.

It is however possible to transmit (and receive) only on 8 MSB or 8 LSB of a GPIO bus if the complete GPIO bus is not available.

So the frame to be transferred will be split in 16 or 8-bit data.

The DMA will be configured in memory-to-peripheral mode.

DMA triggering:

The DMA requests will be generated by the TIMx_CHy, on each PWM data clock rising edge.

The TIMx OCyREF signal is used as DMA request.

The corresponding DMA request should be available for the TIMx_CHy channel in the DMA request table.

On STM32L4x6 code example (see [Section 2.1: STM32L476G example](#)) TIM2_CH1 has been used and is connected to DMA1/Channel5/Request4.

On STM32F429 code example (see [Section 2.2: STM32F429I example](#)) TIM1_CH2 has been used and is connected to DMA2/Stream2/Channel6.

DMA source – destination transfer width:

The source (memory) transfer width being word, half word or byte depending on memory used, and destination (GPIO) transfer width being half word or byte depending on available GPIOs, it is possible to have different source and destination transfer width depending on which STM32 microcontroller is used.

On STM32F429 code example (see [Section 2.2: STM32F429I example](#)) a word access on memory and byte access on GPIO has been used:

- STM32F429 DMA will write 4x8-bit data into the GPIO for each 32bit data read in external memory

On STM32L4x6 code example (see [Section 2.1: STM32L476G example](#)) byte access on both memory and GPIO has been used:

- STM32L476 DMA will write 8-bit data into the GPIO for each 8-bit data read in SRAM memory

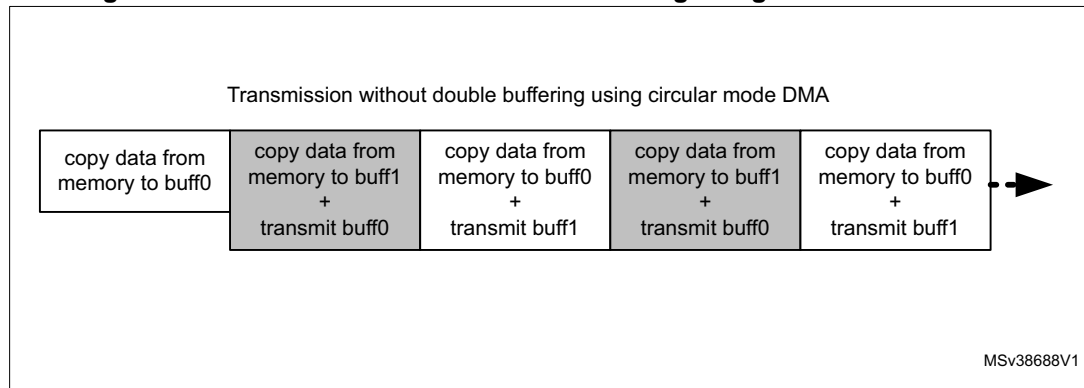
DMA normal or double-buffering mode:

On STM32L4, only the DMA circular mode is available but no double-buffering mode is possible.

So, if a continuous data transmission is needed, due to the 16-bit limitation of the DMA counter and no double-buffering mode is available, a single buffer circular mode must be used.

- The single buffer is split in two parts (buff0, buff1). While data is sent from buffer buff0, data is stored from memory by the CPU to buff1 and vice versa. This implies an overhead in terms of bandwidth on the buffer memory bus used by both DMA and CPU impacting the max possible transmission frequency.

Figure 3. Transmission without double buffering using circular mode DMA



In the STM32L476 example, a data frame length smaller than the 65535 DMA counter limit ($< 65535 * 8\text{-bit data}$) has been used, so that the whole data range can be accessed by the dma counter and there is no need to use circular buffering (this mode has been used only in the reception side to validate the maximum data frequency on reception).

On STM32F429, the DMA allows usage of the double-buffering mode.

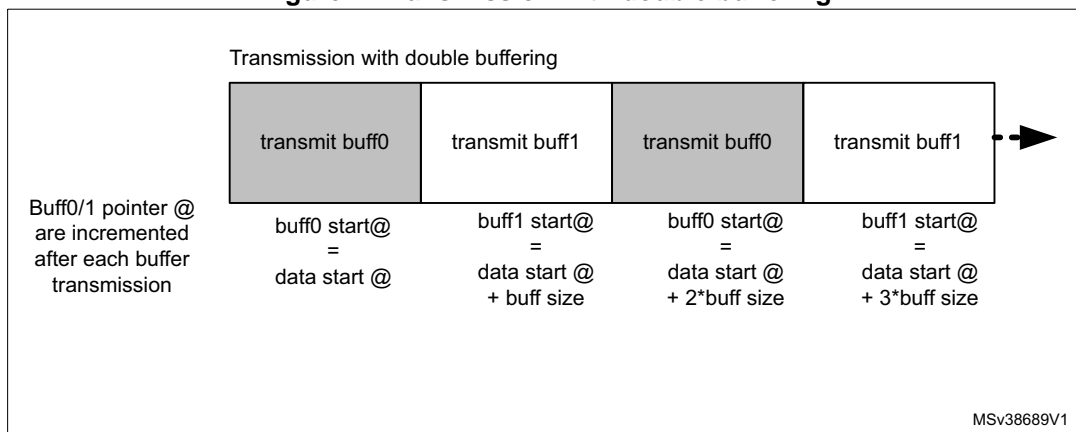
A double-buffer stream works as a regular (single buffer) stream with the difference that it has two memory pointers. When the double-buffering mode is enabled, the circular mode is automatically enabled (do not care about CIRC bit in DMA_SxCR) and at each end of transaction, the memory pointers are swapped.

In this mode, the DMA controller swaps from one memory target to another at each end of transaction. This allows the software to process one memory area while the second memory area is being filled or used by the DMA transfer. The double-buffer stream can work in both directions.

This feature is used in this case to transmit very long frame (continuous transmission), exceeding the 65535 size limit of the dma 16-bit counter, without firstly copying the data by the CPU in an intermediate DMA buffer.

- Using the double-buffering mode implies the usage of the circular mode. It means that the DMA buffer is split in two buffers (buff0, buff1) and that the buff0 and buff1 pointers addresses are increased at each circular loop to progressively address the full data address range. So this buffer points to the source memory location with no need for an additional intermediate memory location.

Figure 4. Transmission with double buffering



DMA FIFO consideration:

The STM32F429 DMA includes a FIFO (3x32 bits).

When the data source (read from memory) has a different size than the data destination (written on the GPIO) the DMA must be configured in FIFO mode (no data alignment is done)

In the STM32F429 example, the data are read from RAM with 32-bit size and written into GPIO with 8-bit size, so the FIFO mode has been used.

In the STM32L4, there is no FIFO mode available. It is possible to have different data size for source and destination, but the DMA applies some data alignment when necessary. This may imply the loss of some data. For example trying to read on 32 bits from memory and write 8 bits on GPIO would only write the 8 LSB bits of each memory access into the GPIO (see RM0351 DMA section for more details). In the example, memory is read and written into the GPIO using same 8-bit data size, hence maximizing the use of the SRAM memory storing the data.

1.1.3 End of transmission (using DMA or TIM in master-slave mode)

The number of data to be transmitted can be either counted by the DMA counter (if number of data is less than 65535), or, if the data frame size is too big, by a 32-bit counter in one of the 32-bit timer registers.

DMA counter:

In STM32L4 example, a DMA counter is used to detect the end of transmission. The DMA is programmed with the frame size value in the DMA channel number of data register (DMA_CNDTR). When the DMA has finished sending the last data, a DMA Transfer Complete interrupt is generated and the TIMx data clock generator is stopped as soon as possible to avoid additional clock toggling. This is done as the first instruction in the DMA interrupt handler.

However, when the data clock frequency is high the time needed between the generation of the interrupt and the TIMx stop is larger than a data clock period and additional clock cycles are generated. The number of additional clock cycles generated is provided, for the STM32L476 example, in the [Table 1: STM32L476G example, achievable data frequency and associated tuning parameters](#). The SW should take this into account and so that the receiving SW should know the expected frequency, to discard additional invalid clock cycles

at the end of the frame (in the example the number of data to be received is unknown and defined by the number of data clock received).

TIM 32-bit counter:

In the STM32F429 example, the frame data size can be larger than 65535, so an additional TIMy 32-bit counter is used to count the number of data to be sent.

The TIMy_CHz is used as slave of the TIMx clock signal generator. It is triggered by TIMx TRGO on its ITR input, which is activated at each TIMx_CHy output compare (at each cycle of the data clock output), and the TIMy Capture Compare Register (CCR) is pre-set with the frame length value.

When the TIMy counter reaches the CCR value, a TIMy_CHz Output Compare interrupt is generated and the TIMx is stopped. This is done as the first instruction of the interrupt handler to avoid additional clock cycle generation. Like for the “DMA counter” example above, when the data clock frequency is high, some additional clock cycles are generated and the number of additional clock cycles is provided, for STM32F429 example, in the [Table 5: STM32F429I example, achievable data frequency and associated tuning parameters](#).

1.2 Reception

The data reception stores receiving data on GPIO to internal or external memory based on the synchronous data clock received.

Like for transmission, the architecture is dependent on which STM32 is used (STM32L476 or STM32F429) and can be re-used or adapted for other STM32 microcontrollers.

The details are discussed in next chapters, while below block diagrams can be used as reference for those chapters.

Note: In below [Figure 5: STM32L476 reception architecture example](#) and [Figure 6: STM32F429 reception architecture example](#), the text under parenthesis refers to the implementation proposed in the code examples.

Figure 5. STM32L476 reception architecture example

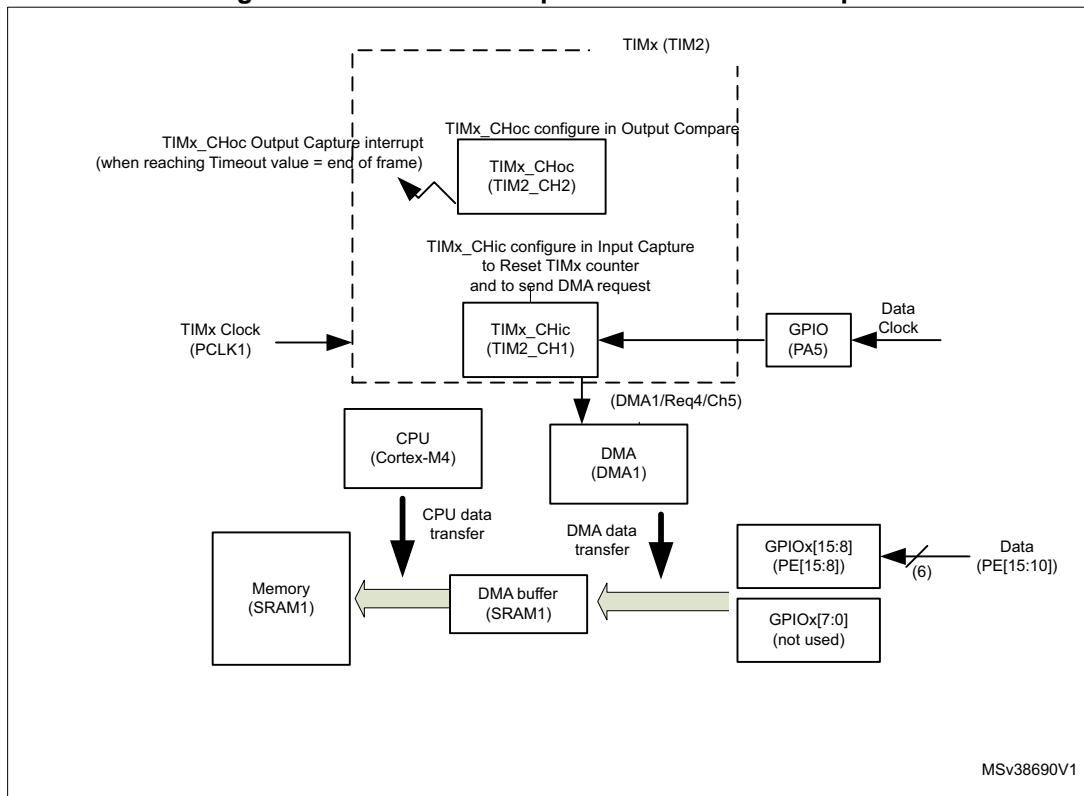
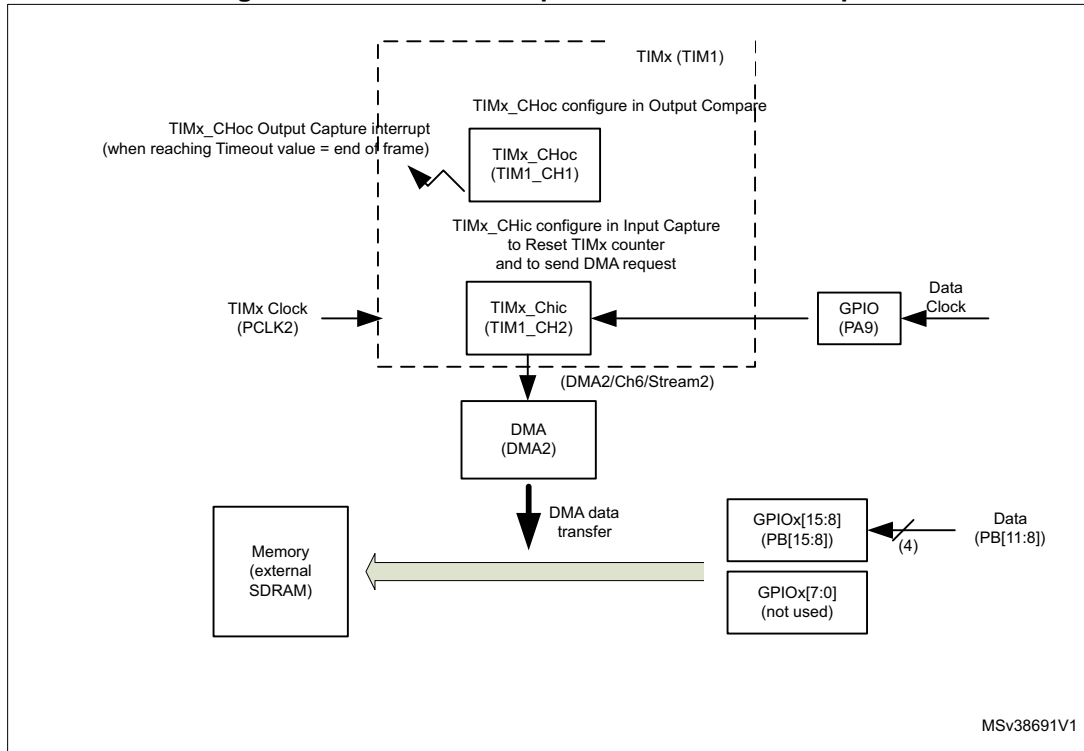


Figure 6. STM32F429 reception architecture example



1.2.1 Data reception (using DMA)

The data clock is received on a GPIO connected to a timer input channel (TIMx_CHic) configured in Input Capture mode. The reception of the signal data clock edge will trig a DMA request that will store GPIO data into memory (either SRAM for STM32L476 example or external SDRAM for the STM32F429 example).

Example:

On STM32F429 code example (see [Section 2.2: STM32F429I example](#)) TIM1_CH2 is used and is connected to DMA2/Stream2/Channel6.

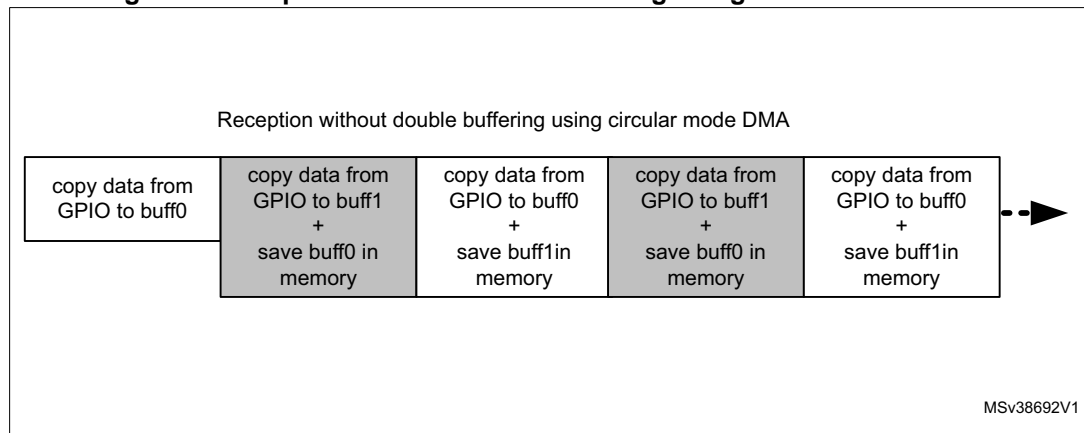
On STM32L4x6 code example (see [Section 2.1: STM32L476G example](#)) TIM2_CH1 is used and is connected to DMA1/Channel5/Request4.

The data clock triggering edge (rising/falling) is configurable in the SW depending on the frequency of the signal (see [Section 1.2.2: Data and clock alignment consideration](#)).

Like for transmission, the DMA is configured in different mode depending on the used product.

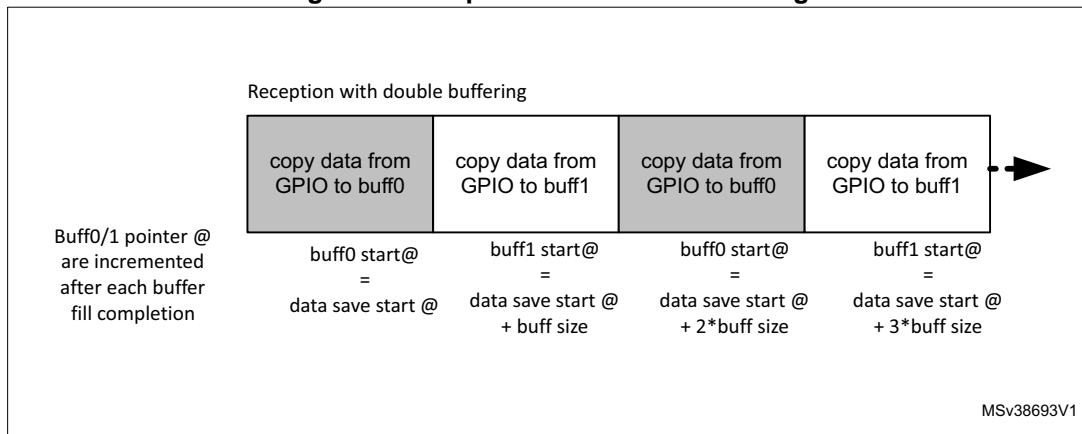
- On STM32L476 example, the DMA is configured in circular mode and no double-buffering mode is available. The single buffer is split in two parts (buff0, buff1). While data is written in buffer buff0 by the DMA, data is stored from buff1 to memory by CPU, then buff0 and buff1 are swapped. This mode allows to receive data in continuous mode, however, it adds bandwidth limitation to the DMA buffer bus, accessed by both CPU and DMA limiting overall data reception maximum frequency.

Figure 7. Reception without double buffering using circular mode DMA



- On STM32F429 example, the DMA is configured in double-buffering mode, the circular mode is used as well, but buff0 and buff1 pointer address are increased at each circular loop to progressively address the full data address range.

Figure 8. Reception with double buffering



1.2.2 Data and clock alignment consideration

The data and data clock received are synchronous, but not aligned. The data clock being generated by the transmission timer TIMx_CHy configured in PWM mode, while the data is sent only after the dma data transfer has occurred between memory and GPIO.

So on reception and depending on the data clock frequency, the data clock edge used to receive data “n” may occur before or after the data changing from data “n” to data “n+1”. Hence reception may want to use either clock “n” or clock “n+1” as trigger to start reception of data “n”. It could also use either rising or falling edge of clock signal.

Note that the data on reception side should be stable until the end of reception dma storage from GPIO to memory.

Figure 9. Data to clock synchronization

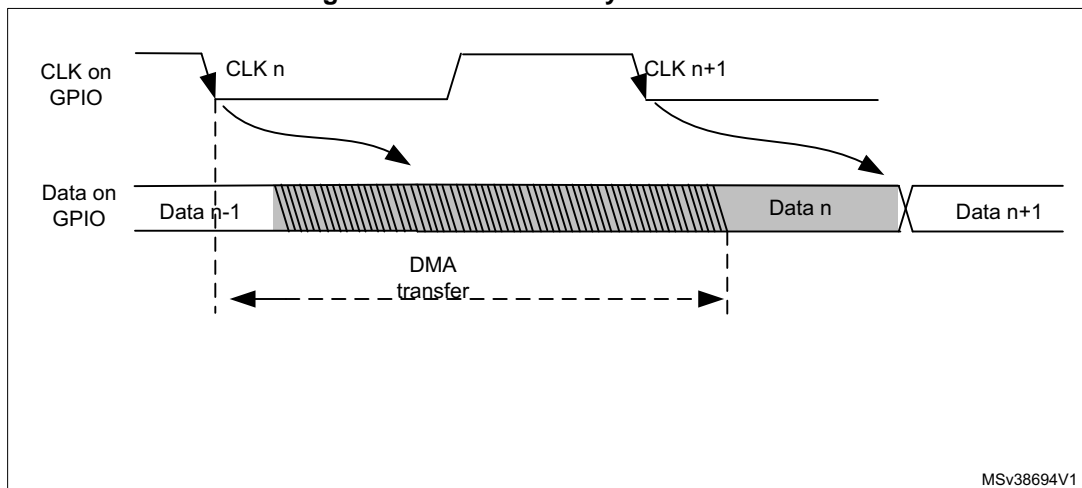


Figure 10. Example with low-data frequency

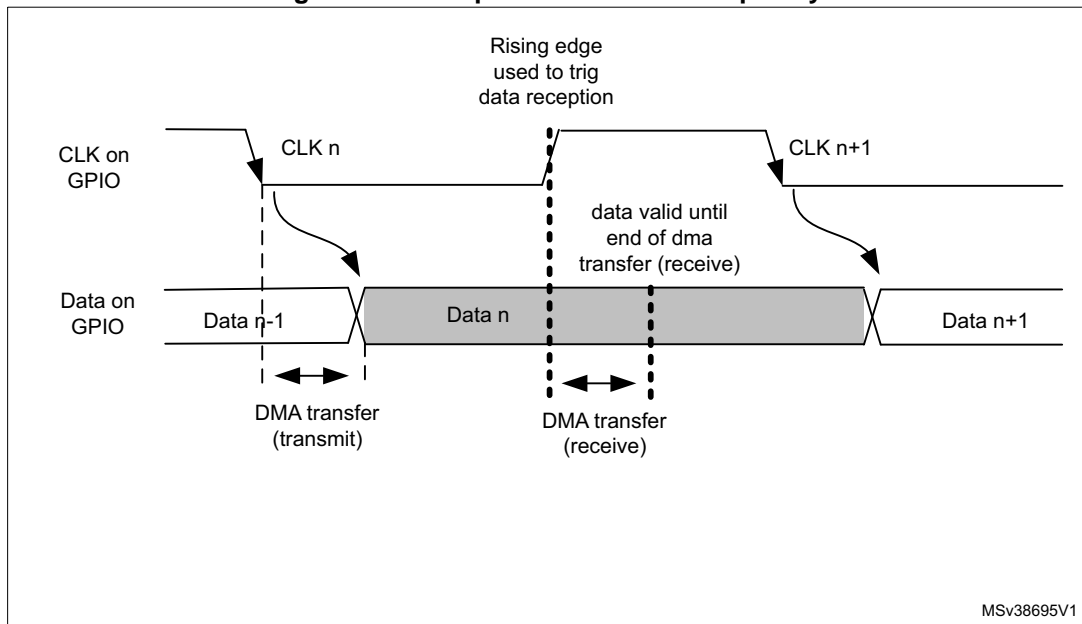
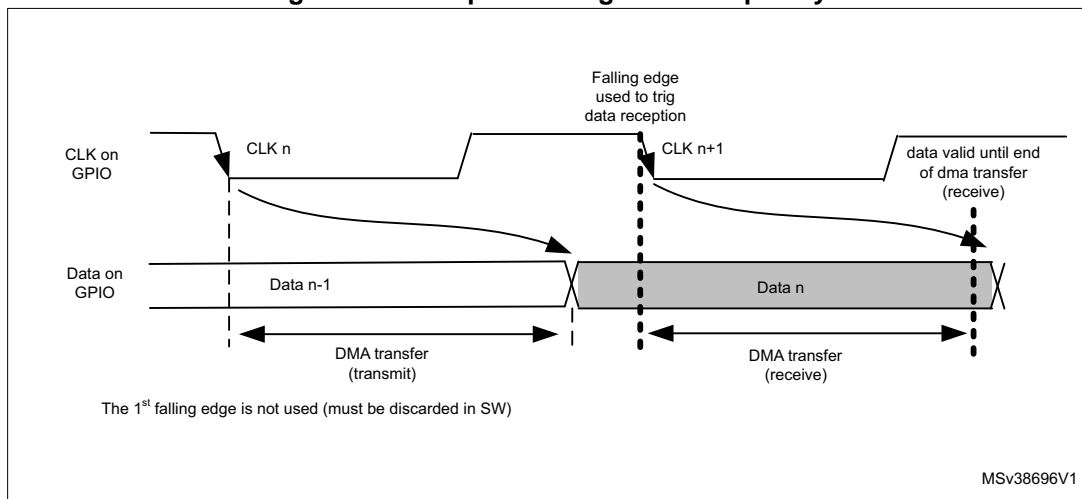


Figure 11. Example with high-data frequency



Depending on this data to clock relationship and so depending on the data frequency, the receiving SW should use the rising or falling edge of the data clock to trig the DMA reception data storage. It may also have to discard the first data received when first clock edge is not meaningful. Those values (rising/falling edge, discarded data versus data frequency range) for the two examples considered here (STM32L476 and STM32F429) are provided in [Table 1: STM32L476G example, achievable data frequency and associated tuning parameters](#) and in [Table 5: STM32F429I example, achievable data frequency and associated tuning parameters](#).

Note: One possible option would be to write the first data on the GPIO using the CPU so that it is already present when the first clock edge is received and there would be no need to discard the first data received. In this case the DMA transfer would start directly with the second data.

1.2.3 Detecting end of transmission (with TIM timeout)

Since it is assumed that the number of transmitted data is not known on receiver side, a timeout is used to define the end of the frame.

The signal data clock edge will also reset the TIMx counter. This counter is used to detect a timeout on data reception through a channel (TIMx_CHoc) configured in Output Compare mode. The TIMx_CHoc is preset with a predefined timeout value of 1s.

The TIMx clock is set at 2KHz (HCLK=80MHz, TIMx prescaler= 40000)

The TIMx Capture Compare Register is set to 2000

So the timeout value is $2000/2\text{KHz} = 1\text{s}$

The TIMx counter is enabled by the input data clock trigger (so that the timeout counter only starts after a first data clock edge occurs).

In the STM32F429 example, this is done by setting the TIMx in Slave Mode Trigger (SMCR register, SMS bits set to 0x6) and defining the trigger as the TIMx_CHic (TIM1_CH2) input (SMCR register, TS bits set to 0x0060)

In the STM32L476 example, this is done by setting the TIMx in Slave Mode Combined Reset + trigger mode (SMCR register, SMS bits set to 0x1000) and defining the trigger as the TIMx_CHic (TIM2_CH1) input (SMC register, TS bits set to 0x0050). In this example, the timeout value could be much less than 1s, it has to be greater than 1 data clock cycle.

Note that on STM32L4, the Slave Mode Combined Reset + trigger mode allows to both reset and trig the counter at each data clock edge. On STM32F429, this mode does not exist, so the timeout is started on the first data clock edge and never reset, so the total frame length should be less than the predefined timeout value (1s in this example)

Figure 12. STM32L476 timeout example

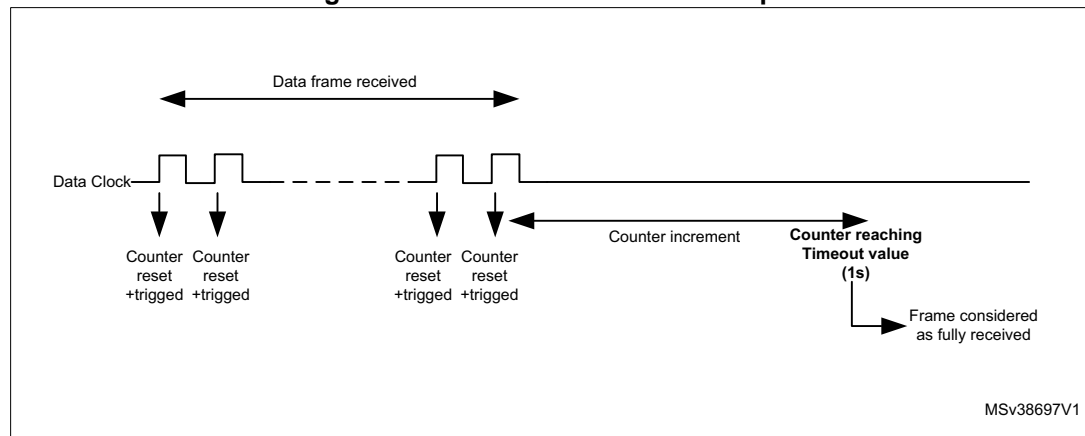
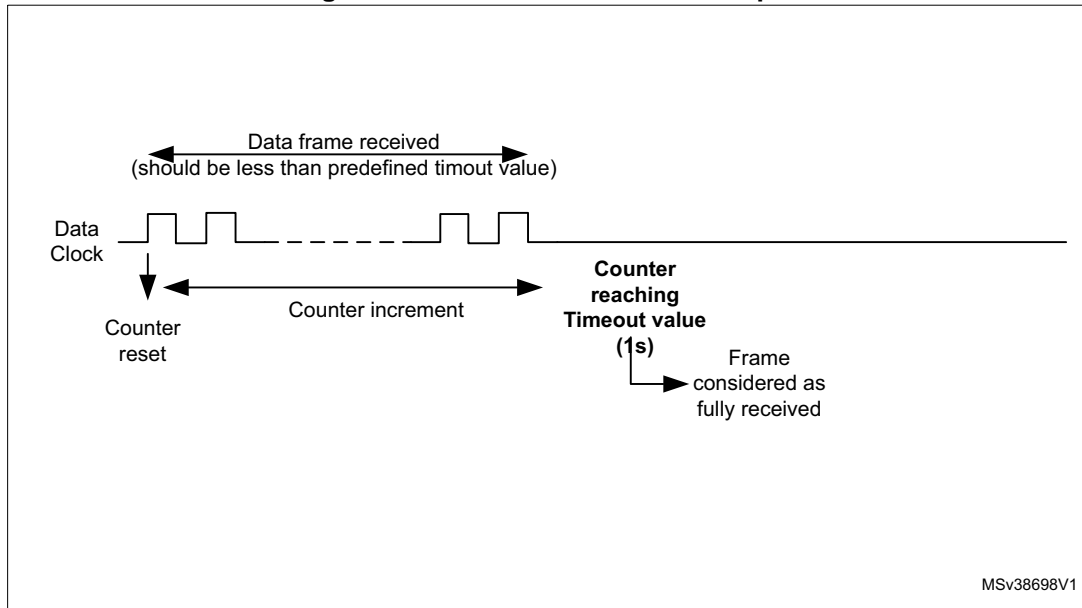


Figure 13. STM32F429 timeout example



2 Implementation code explanation

Two examples are available and can be adapted to specific use cases or other STM32 microcontrollers or application boards.

Both examples have been developed under:

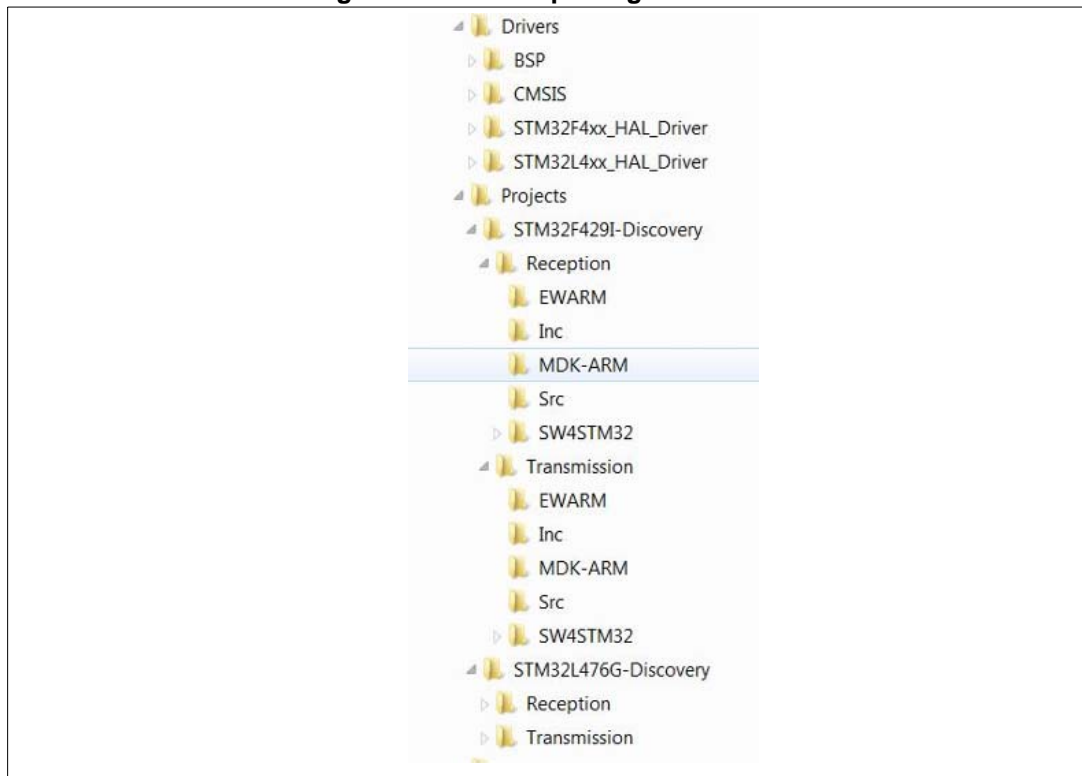
- RealView[®] Microcontroller Development Kit (MDK-ARM[™]) toolchain from ARM[®] Ltd using STMicroelectronics STM32Cube HAL libraries

and ported to:

- IAR Embedded Workbench[®] for ARM[®] (EWARM) toolchain from IAR Systems
- System Workbench for STM32 (SW4STM32) toolchain

The SW package structure is as follows:

Figure 14. The SW package structure



The STM32L476G-Discovery structure is the same as STM32F429I-Discovery.

Each example folder contains a readme.txt file.

All parameters are defined in the /Inc/main.h file.

On both examples, the transmitted data is created using the Random Number Generator (RNG) implemented in the STM32.

To verify the correct transmission, a signature is computed using the Cyclic Redundancy Check calculation unit (CRC) at the transmit side and again at the reception side. Both signatures are compared at the reception side. The transmission sends the signature at the end of the frame.

Some information are showed on the LCD (like number of data sent, CRC computed at transmission, number of data received or CRC computed at reception):

- On STM32L476G Discovery board only a simple 24 segment LCD is implemented
- On STM32F429I Discovery board a 2.4" QVGA TFT LCD is present allowing to show more information.

As explained in [Section 1: Architecture overview](#), the SW on reception side has to be tuned (in main.h) depending on the data frequency, for two main reasons:

- When data frequency is high, the time needed to stop the TIMx PWM clock data generation at the end of the frame transmission (interrupt handler) is too long and some additional data clock cycles can be generated. In such case, the reception SW must discard some invalid clock received at the end of the frame (see [Section 1.1.3: End of transmission \(using DMA or TIM in master-slave mode\)](#)).
- Depending on the clock frequency, the data to clock arrival time is different and the receiving SW must use either rising or falling edge of the received clock and eventually discard the first clock edge (see [Section 1.2.2: Data and clock alignment consideration](#)). A table showing data frequency achievable and associated tuning on reception side (clock edge, number of data to discard) is provided in next sections for each example.

Additionally, four additional dummy data are sent at frame start (4 times 0x0), to avoid timing issues with the second data, which has a slightly different timing than other data and may cause timing issue for high frequency data clock (in fact only seen for clock ratio=9 @ 8.89MHz).

These four dummy data are discarded at the reception side.

It is important to note that those code examples are sensitive to the code execution timings (the time needed to stop the timer generating PWM clock signal at the end of frame on transmission side). Depending on the compiler used and on the optimization options used, the user may have to fine tune the main.h on reception side (adding or removing one "data to discard" from the baseline showed in the [Table 1: STM32L476G example, achievable data frequency and associated tuning parameters](#) and [Table 5: STM32F429I example, achievable data frequency and associated tuning parameters](#)).

2.1 STM32L476G example

On the STM32L476G-Discovery board, only GPIO PE[15:10] can be used for the data transfer, since other GPIO PE bits or other GPIO buses are already used or not accessible. PA[5] is used for the data clock transmission.

It is only possible to read/write on GPIO either on 16 bits or on 8 bits. In this case write (transmit) and read (receive) shall be done on 8 bits. It means that write/read will be done on GPIO PE[15:8] but GPIO PE[9:8] are not accessible on the Discovery board.

So only 6 bits in parallel can be transmitted. The non-transmitted bits are set to 0 when computing the CRC at transmission side and the same is done on the reception side.

The CRC is also transmitted on the last frame data of 6 bits and so non-transmitted CRC bits (CRC computed on 32 bits) are set to 0 before the CRC comparison at reception side.

At the end of transmission, the CRC computed is showed on the LCD.

At the end of reception, the number of data received is showed on the LCD.

The user should then press the user button and the CRC computed is showed.

The red LED is toggling in case of CRC mismatch.

The data is stored in SRAM in 8-bit table format (same size as the GPIO).

Table 1. STM32L476G example, achievable data frequency and associated tuning parameters

CLK range # (clock ratio range)	Data frequency range ⁽¹⁾	Number of start data to discard	Number of end data to discard	CLK latching edge
Range1 (80000 – 28)	1KHz – 2.86MHz	0	0	falling
Range2 (27 – 24)	2.96MHz – 3.33MHz	1	0	rising
Range3 (23-12)	3.48MHz – 6.67MHz	1	1	rising
Range4 (11-10)	7.27MHz – 8MHz	1	2	rising
Range5 (9-8)	8.89MHz - 10Mhz	0	3	falling

1. The data clock is the system clock at 80Mhz divided by the clock ratio.

Note: [Table 1](#) is valid for MDK-ARM compiler V5.14 used during the example development and may have to be fine-tuned for other compilers, especially for frequency values at the range limit.

Since the STM32L4 is using the DMA in circular mode, the buffer received into the SRAM final location must be copied, while the second buffer is being filled. This is adding some workload on the SRAM bus and for a high frequency, the buffer size must be increased a lot to get the transmission working especially for a large data frame.

Table 2. STM32L476G example, dma buffer size versus frequency for large data frame

Clock ratio range	Data frame length	Data frequency range	Buffer size	Comment
>= 16	65530	<= 2.86MHz	100	
15	65530	5.33MHz	200	
14	65530	5.71MHz	1000	
13	65530	6.15MHz	1500	
12	65530	6.67MHz	12000	
11	65530	7.27MHz	21000	
10	65530	8MHz	27000	

Table 2. STM32L476G example, dma buffer size versus frequency for large data frame (continued)

Clock ratio range	Data frame length	Data frequency range	Buffer size	Comment
9	60500	8.89MHz	32000	Reaching limit of SRAM1 size for this frame length ⁽¹⁾
8	55000	10Mhz	40000	Reaching limit of SRAM1 size for this frame length ⁽²⁾

1. For a very large buffer size (for clock ratio = 9, F = 8.89Mhz) the MAX_FRAME_SIZE in main.h should be reduced from 65535 to 60510 and data frame length set to 60500 max to fit in SRAM1 size of 96KB.
2. For a very large buffer size (for clock ratio = 8, F = 10Mhz) the MAX_FRAME_SIZE in main.h should be reduced from 65535 to 55005 and data frame length set to 55000 max to fit in SRAM1 size of 96KB.

2.1.1 Code usage

User should tune the below parameters in the main.h file, as showed in [Table 3](#) and [Table 4](#):

Table 3. Transmission side parameters

Trasmission side	Tuning
#define CLK_RATIO	from 8 @10Mhz to 80000 @1KHz
#define FULL_FRAME_SIZE	max 65530 for CLK_RATIO < 9, max 60500 for CLK_RATIO=9 max 55000 for CLK_RATIO = 8

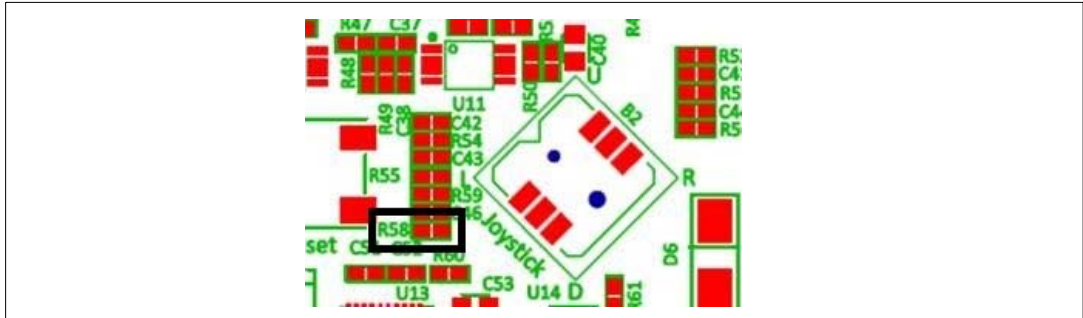
Table 4. Reception side parameters

Reception side	Tuning
#define CLK_RANGE _x	x=1 to 5 depending on expected frequency according Table 1: STM32L476G example, achievable data frequency and associated tuning parameters
#define BUFFER_SIZE	depending on expected frequency according to Table 2: STM32L476G example, dma buffer size versus frequency for large data frame , given for max data frame size according to Table 1: STM32L476G example, achievable data frequency and associated tuning parameters
#define MAX_FRAME_SIZE	max 65535 for CLK_RATIO < 9, max 60510 for CLK_RATIO=9 max 55010 for CLK_RATIO = 8, according Table 2: STM32L476G example, dma buffer size versus frequency for large data frame notes.

Other parameters in main.h do not need to be modified. It is important to remove resistance R58 on the STM32L476G-Discovery (see [Figure 15: Resistance R58 location on STM32L476 Discovery board](#)). R58 is a pull down connected on PA5, used for JOYSTICK

DOWN button. After removing R58, JOYSTICK DOWN feature will not be available anymore.

Figure 15. Resistance R58 location on STM32L476 Discovery board

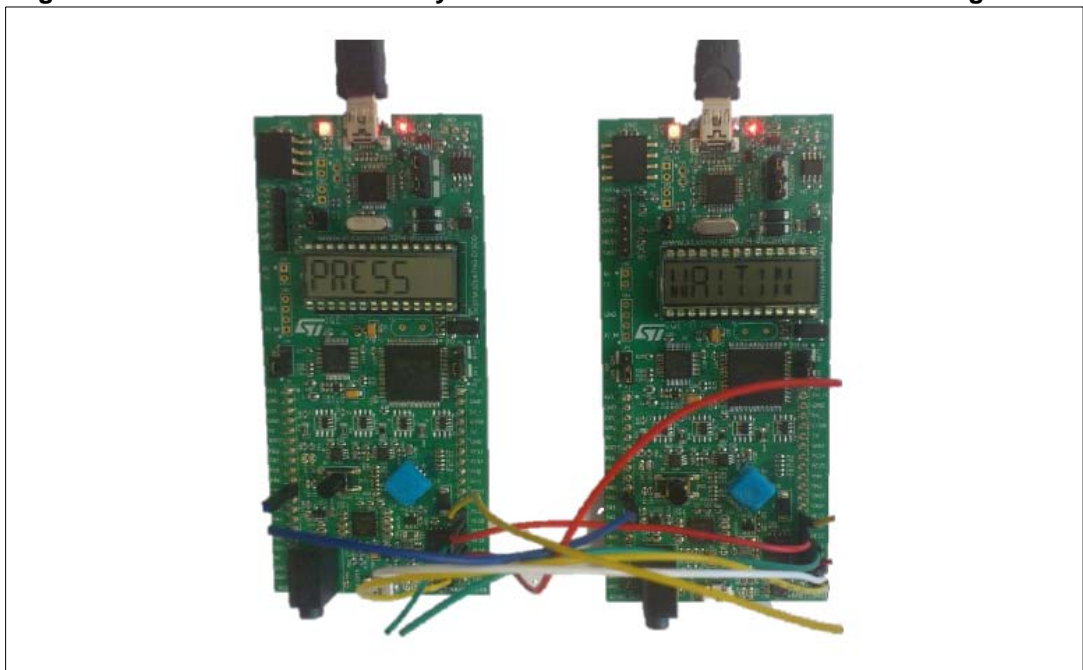


The pins PA[5] (data clock) and PE[15:10] should be connected together between the transmission and reception boards.

Both boards loaded with their respective code and powered through the ST-LINK USB cable connected to the PC.

Firstly the transmission board must be reset and secondly the reception board. The two boards should be in the state showed in [Figure 16](#).

Figure 16. STM32L476G Discovery board connection and initialization configuration

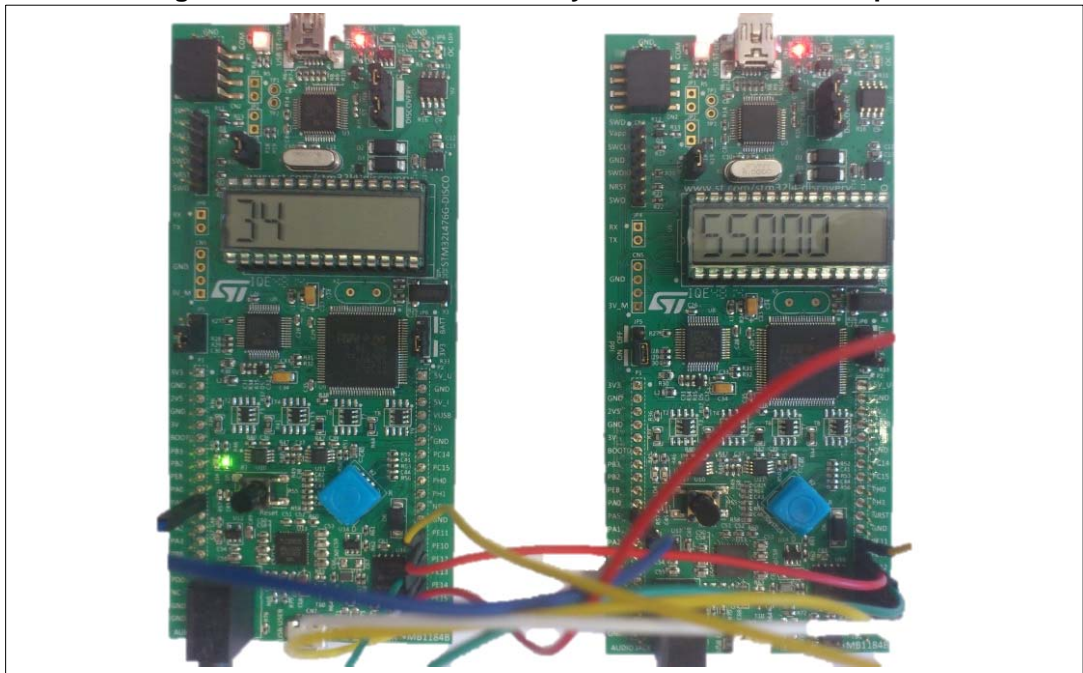


Then the transmission board user button (blue button) shall be pressed to send the frame.

The computed CRC is showed on the transmission board LCD, the number of data received is showed on the reception board LCD and should match the number indicated in the transmission SW main.h (FULL_FRAME_SIZE parameter).

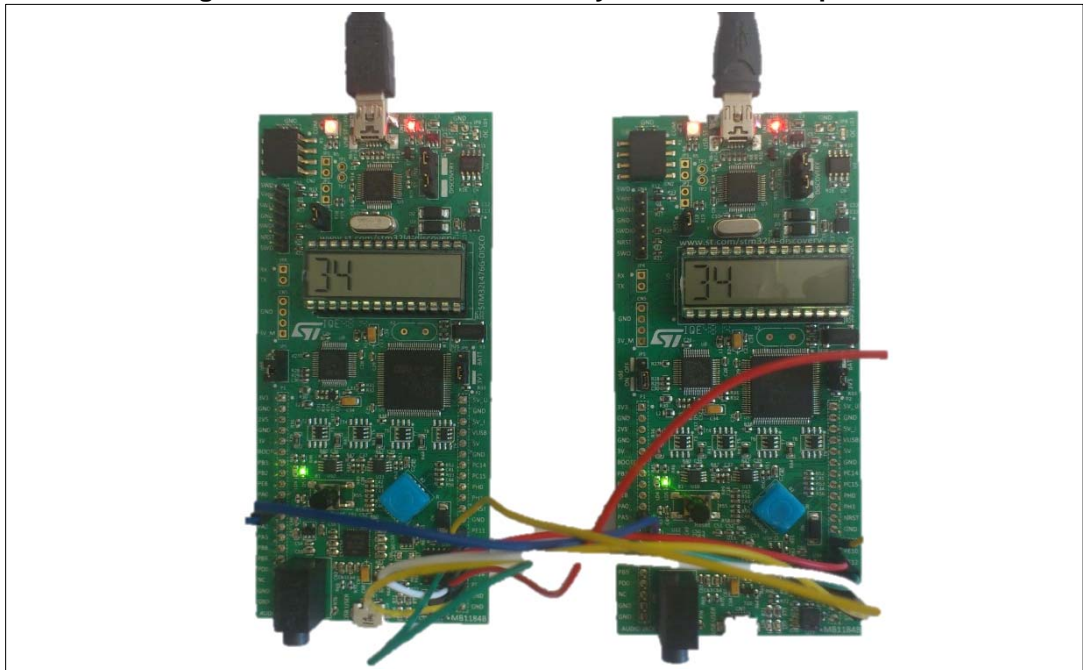
On [Figure 17](#) the CRC computed is 34 and the number of data sent is 55000.

Figure 17. STM32L476G Discovery board after frame reception



Then the reception board user button shall be pressed to show the received computed CRC that should match the CRC showed on the transmission board. In case of mismatch the red led will toggle, otherwise the green led will toggle (see [Figure 18: STM32L476G Discovery board CRC comparison](#)).

Figure 18. STM32L476G Discovery board CRC comparison



2.2 STM32F429I example

On the STM32F429I-Discovery board, only GPIO PB[11:8] can be used since other PB bits or other GPIO bus are already used or not accessible. PA[9] is used for the data clock transmission.

It is only possible to read/write on GPIO either on 16 bits or on 8 bits. In this case write (transmit) and read (receive) shall be done on 8 bits. It means that write/read will be done on GPIO PB[15:8] but GPIO PB[15:12] bits are not accessible on the Discovery board.

Note that the PB[11:8] bits are used for LCD, and so it is not possible to fully use the LCD while the data is transmitted or received.

At the transmission side, the PB[11:8] bits GPIO are configured as output mode as soon as first "PRESS BUTTON" message is displayed (and not as LCD alternate function). The impact is that the LCD message is slightly dimmed but readable.

On the reception side the LCD GPIO are not configured and LCD not initialized before the frame is received. So there is no initialization message on the reception board.

This ensures that no noise is generated by the LCD conflict between the two boards on PA[9] GPIO that could trig the clock signal detection.

Also, at the end of the transmission, while both LCD are active, the GPIO PB[15:8] may conflict which means that the transmission board LCD will show some artifacts on its display (number of data sent, CRC computed), but it is readable.

It is important to reset first the transmission board (until first message is displayed) then reset the reception board.

Since only 4 bits in parallel can be transmitted, the non-transmitted bits are set to 0 when computing the CRC at the transmission side and the same is done on the reception side.

The CRC is also transmitted on the last 4 frame data of 4bits each and so non-transmitted CRC bits are set to 0 before the CRC comparison at reception side (the 32-bit CRC result is masked with 0x0F0F0F0F).

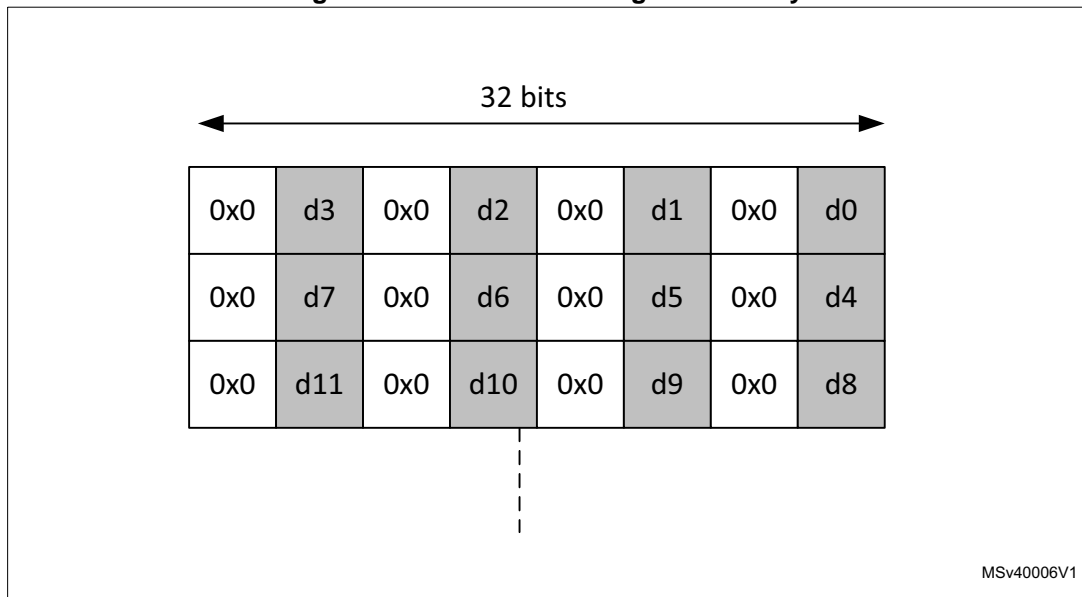
At the end of the transmission, the number of data sent and CRC are showed on the LCD.

At the end of the reception, the number of data received, the CRC received and the CRC computed are showed on the LCD.

The data is stored in the external SDRAM on word (32 bits) format and read/write on/to the GPIO on 8 bits. So each 32-bit word stored in memory contains 4 data of 8 bits and each of those 8-bit data includes only 4 relevant bits and 4 bits set to 0 (because not transmitted)

It implies that the number of data sent should be a multiple of 4 to allow the SW to work properly.

Figure 19. Data frame storage in memory



The data frame length is 1228800, which is the required size for a VGA picture data transmission (640x480 on 16 bits = 307200 * 16 bits = 1228800 * 4 bits, in this example data are sent on 4 bits).

Table 5. STM32F429I example, achievable data frequency and associated tuning parameters

CLK Range # (clock ratio range)	Data frequency range ⁽¹⁾	Number of start data to discard	Number of end data to discard	CLK latching edge
Range1 (400 – 55)	450KHz – 3.27MHz	1	0	rising
Range2 (54 – 44)	3.33MHz – 4.09MHz	1	1	rising
Range3 (43-28)	4.18MHz – 6.43MHz	0	2	falling
Range4 (27-23)	6.67MHz – 7.82MHz	0	3	falling

1. The data clock is the system clock (transmission side) at 180Mhz divided by the clock ratio.

Note:1 *Table 5 is valid for the MDK-ARM compiler V5.14 used during example development and may have to be fine-tuned for other compilers, especially for frequency values at the range limit.*

Note:2 *The maximum reachable data frequency is less in the STM32F4 example than in the STM32L4 example. This is due to the fact that the STM32L476G Discovery board does not include an external SDRAM and so the data is stored in the internal SRAM. Storing data in the internal SRAM is faster than storing into an external SDRAM, but it does not allow a very long data frame and it limits the usability of the internal SRAM for other processes.*

2.2.1 Code usage

User should tune the below parameters in the main.h file:

Table 6. Transmission side parameters

Trasmission side	Tuning
#define CLK_RATIO	from 23 @7.82Mhz to 400 @450KHz
#define FULL_FRAME_SIZE	should be multiple of 4 max tried is VGA on 16 bits = (640x480x4)x4 bits sent = 1228800

Table 7. Reception side parameters

Reception side	Tuning
#define CLK_RANGE _x	x=1 to 4 depending on expected frequency according Table 5: STM32F429I example, achievable data frequency and associated tuning parameters

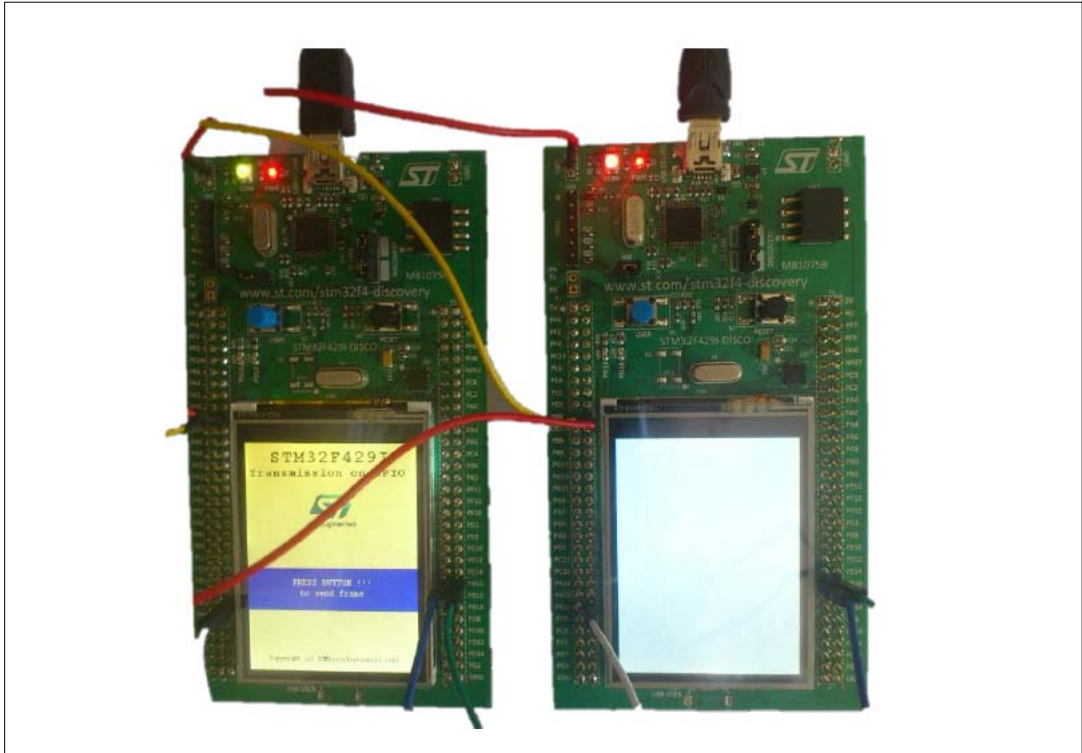
Other parameters in main.h do not need to be modified.

The pins PA[9] (data clock) and PB[11:8] should be connected together between the transmission and reception boards.

Both boards loaded with their respective code and powered through the ST-LINK USB cable connected to the PC.

User must reset first the transmission board and then the reception board. The two boards should be in the state showed in [Figure 20: STM32F429I Discovery board connection and initialization configuration](#)

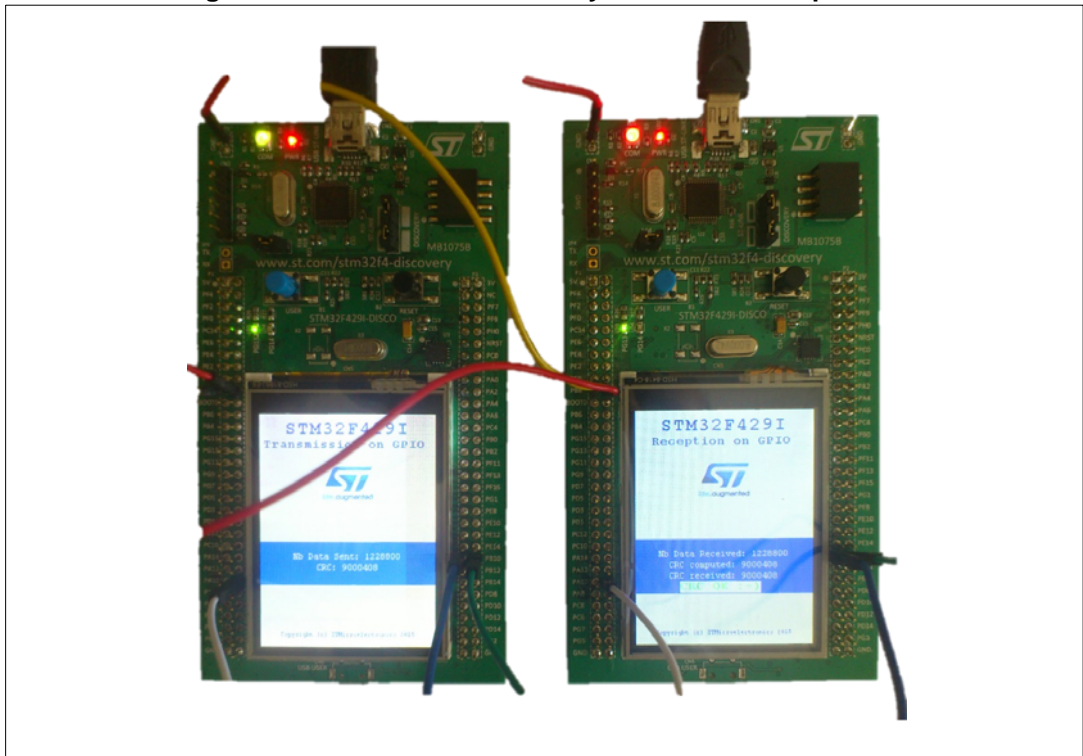
Figure 20. STM32F429I Discovery board connection and initialization configuration



Then the transmission board user button (see blue button in [Figure 20](#)) shall be pressed to send the frame.

The number of data sent and the computed CRC are showed on the transmission board LCD. After the timeout period, the number of data received, CRC computed, CRC received are showed on the reception board LCD with a “CRC OK” message if the CRCs match. The green LED also toggles if CRC match. This is as showed in [Figure 21: STM32F429I Discovery board CRC comparison](#).

Figure 21. STM32F429I Discovery board CRC comparison



3 Conclusion

In this application note, an implementation of a synchronous data-clock transmission through the GPIOs has been demonstrated on both STM32L476G-Discovery and STM32F429I-Discovery boards.

The two examples use an architecture based on timers and DMA that can be adapted to other STM32 microcontrollers.

The transmission frequency proved successful up to 10 MHz (HCLK/8 with HCLK=80 MHz) on STM32L476G-Discovery and up to 7.82 MHz (HCLK/23 with HCLK=180MHz) on STM32F429I-Discovery.

The lower frequency reached on the STM32F429I-Discovery is due to the fact that the data is not stored in the internal SRAM like on the STM32L476G-Discovery but in an external SDRAM to allow longer data frame.

The STM32L476G-Discovery example stores data into the SRAM1 and uses the DMA 16-bit counter on transmission side, to count the number of data sent and hence limiting data frame length to 65530. It would, however, be possible to have a continuous transmission (long data frame) by following the architecture proposed in the [Section 1.1.2: Data transmission \(using DMA\)](#) using a circular buffer mode DMA but adding CPU data copy from memory to DMA buffer. On reception side, this continuous reception architecture using circular mode DMA and CPU data copy from DMA buffer to memory has been used.

The STM32F429I-Discovery example stores data into the external SDRAM and uses a 32-bit timer counter on transmission side allowing very large data frame (in the proposed example the frame length was equivalent to a VGA frame size = $1228800 * 4 \text{ bits} = 640 * 480 * 16 \text{ bits}$), this is done using the DMA double-buffering mode available on the STM32F4 series.

4 Revision history

Table 8. Revision history

Date	Revision	Changes
13-Jan-2016	1	Initial release

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved