TUNKU ABDUL RAHMAN
UNIVERSITY COLLEGE

BAME2123 Microcontroller with Peripherals

Serial Peripheral Interface

Course: Microelectronic with Embedded Technology (RMB2)

Group Member: 1) CHU JAAN HORNG (15WAR09633)

Tutor: Poh Tze Ven

Table of Contents

Introduction

Serial Peripheral Interface (SPI) is a serial communication bus developed by Motorola back in 20th century which was first used externally in microcontroller to communicate with the external peripherals. SPI is mostly used in communicating with the peripherals whenever speed is needed. When data streams, slow rate of communication like Inter-Integrated Circuit (I^2C) will be a bad idea to use. Thus, the existence of SPI will be a great help because of the significant high speed rate that SPI has achieved. SPI-compatible interfaces often range into ten megahertz or higher.

Although SPI is good in communication, SPI needs more effort and more hardware resources than I^2C when more than one slave is involved. In order to communicate with several slaves, SPI needs multiple connections. Thus, the setup for the connection between master and slave takes time and the area too. As this report goes along, the introduction to the SPI and the experiment of using SPI to communicate will be discussed and document. By the way, Figure 0(a) and Figure 0(b) are the setup in hardware for master and master with slave.

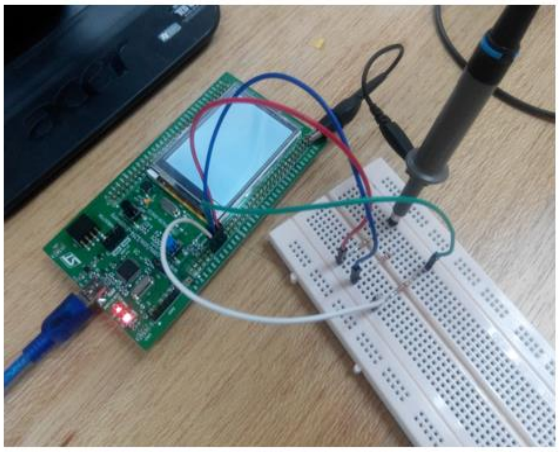

Figure 0(a)

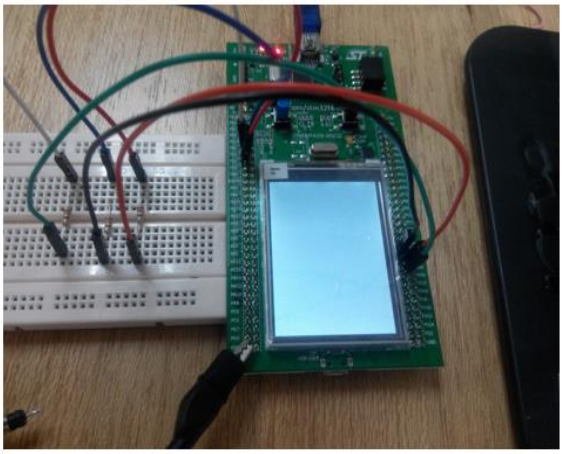

Figure 0(b)

Figure 0(a) is connected in a way that only master is doing the transmission and reception of data. For Figure 0(b), master and slave is connected through connecting the SCK, NSS, MOSI and MISO pins together.

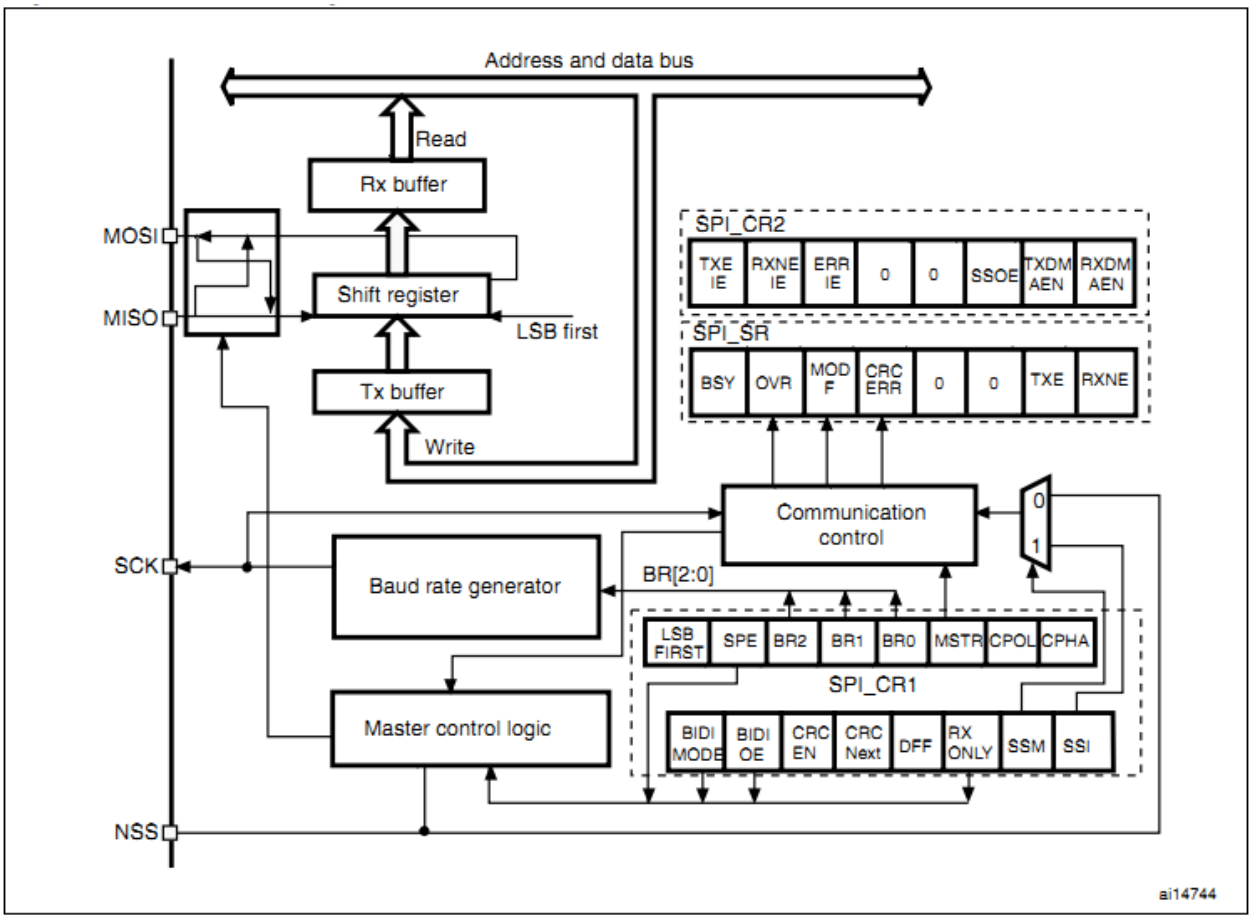

Figure 1Block Diagram of SPI [*Accessed from: http://www1.coocox.org/cox/manual/STM32F1xx/group___s_t_m32_f1xx___s_p_i.html]*

Figure 1 shown the internal circuitry of how SPI is built. From the diagram shown, four pins are shown which is Master out Slave in (MOSI), Master in Slave out (MISO), Serial Clock (SCK) and Slave Select (NSS) or Chip Select (CS). For MOSI and MISO, both pins are not necessary be used together which have to depend on the setup of transmission and reception.

Pins of SPI:

1. MOSI: This pin is used for master to send out the data to the specific slave.
2. MISO: This pin is used for slave to send out the data to the master.
3. SCK : This pin is used to output the clock for master and input for slave. The slave will not use any clock except the clock output from master, even the clock the slave generated.
4. NSS : This is used the select slave. In other word, without this pin, master and slave cannot be communicated. Initially, this pin is high as in logic 1. To connect with the chosen slave, NSS needs to be low (logic 0) to indicate that the chosen slave is in communication with the master and other slaves cannot communicate with master.

Transmission:

For transmitting any data to the destination through using SPI, data will be first stored in the Transmission buffer (Tx buffer) whenever data is written to Data Register (DR). Once Tx buffer is full, Transmit Buffer Empty (TXE) flag will be cleared to indicate the data is ready to send. Then, the data will send to MOSI or MISO serially through the shift register. Depending on the situation, either MOSI or MISO will be chose to send out data or both will be used for the communication.

Reception:

For receiving any data sent from the destination though SPI, the data will be stored at the reception buffer (Rx buffer) whenever DR is read. Once data is stored inside the Rx buffer, Receive Buffer Not Empty (RXNE) flag is set to indicate the data has not yet read. Then, the data will serially shifted towards the targeted bus get processed. Depending on the situation, either MOSI or MISO will be chose to receive the data or both will be used for the communication.

Serial Clock:

Clock is important in any sequential circuits because flip-flop needs clock pulse to output the data whenever clock triggered from time to time. Thus, SPI has this serial clock which only generated by master. Serial clock is made up by this clock phase and clock polarity. This clock can be configured at Control Register 1 (CR1) in SPI setting with four options. (Refer to Table 1.) The purpose of having this four options of clock is because some devices need this kind of pattern of clock in order to communicate between the source and the destination and able to sample the data at the correct timing. Figure 2 will show the pattern of clock from the four options of clock.

Table 1 Four options of clock pattern of the serial clock.

Options / Pattern	Clock phase (Bit 0 in CR1)	Clock polarity (Bit 1 in CR1)
00	First edge of SCK	Idle 0
01	First edge of SCK	Idle 1
10	Second edge of SCK	Idle 0
11	Second edge of SCK	Idle 1

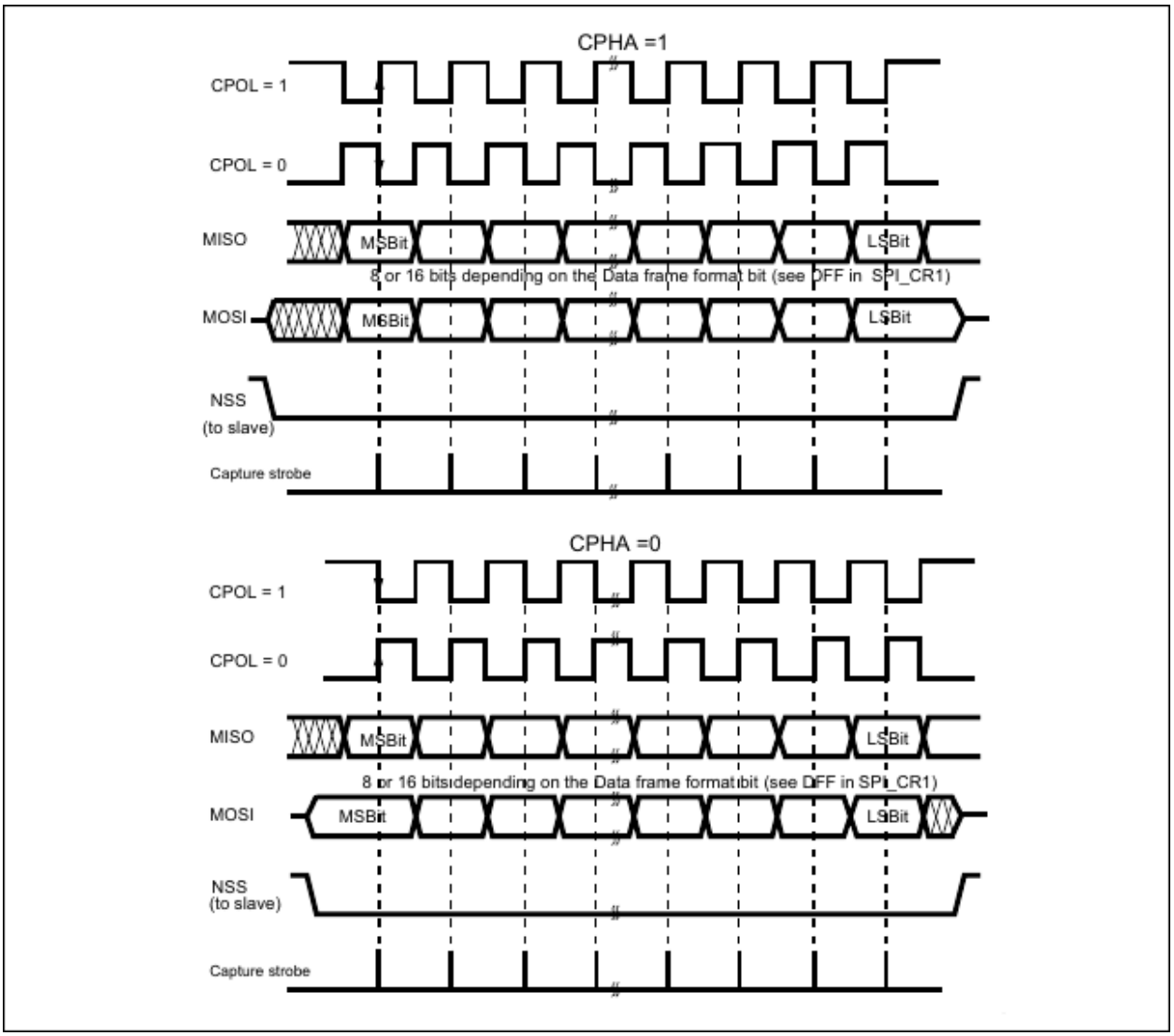

Figure 2 Data Clock Timing Diagram [*Accessed from: http://wiki.csie.ncku.edu.tw/embedded/SPI*]

Methodology

In order to use SPI in communication, SPI will need to configure in an appropriate setting. Thus, the control register 1 needs to be configured properly. However, there is a bit in control register 2 needs to be configured. That bit is called Frame Format (bit 4). This bit is to decide what kind of behavior that SPI have to behave — TI (Texas Instrument) mode and Motorola mode. By selecting TI mode, some of the configuration like LSBFIRST bit in CR1 will be set as according to TI mode. As for Motorola mode, there will not be any restriction will configuring CR1 which is a full custom configuration. However, there will not be any different between these two modes under the free conditions. Thus, this experiment is conducted in TI mode because there is not any condition like clock restriction in the communication. So, by choosing TI mode, some of the configuration will be configured by this TI mode and errors will be less. Of all the bits, bit MSTR is the most important because this bit is going to decide the SPI to become a Master or Slave. The following is the procedure to configure the SPI in either master mode or slave mode.

1. Slave mode:

   In the slave configuration, the serial clock is not important because the clock that the slave used is transmitted from master. This is due to the priority as a master. Thus, no matter what value that is set for the clock in the slave, data transfer rate will not be affected by the slave unless the clock from master has some alter. The following procedure is the way to setup SPI as a slave.

   1.1. The data frame format bit (DFF) in CR1 is set to define the data frame as 8-bit or 16-bit data frame format. By setting this bit, the output data will be sent in 8-bit data frame or 16-bit data frame.
   1.2. The CPOL and CPHA bits have to be set to define one of the four relationships between the data transfer and the serial clock. In order to transfer the data correctly, the configuration of CPOL and CPHA bits in CR1 for both master and slave need to be same. Otherwise, the data will be sampled at wrong timing and output the wrong data. However, if SPI is configured as TI mode, these two bits can be ignored and these two bits will be set by default.
   1.3. The frame format (LSBFIRST) needs to be set to define the data to output the Least Significant Bit (LSB) first or Most Significant Bit (MSB) first. However, this bit needs to be same for both master and slave. This is because the data will be different when received if both master and slave are not the same for this bit. Example, master send out data (0x69), data received at slave will become 0x96.
   1.4. In hardware mode, the NSS pin needs to be connected to a low level signal during whenever communication is needed for both master and slave. In software mode, Slave Select Management (SSM) bit has to be set and Internal Slave Select (SSI) needs to be cleared in CR1. However, these two bits can be ignored is TI mode is selected.
   1.5. The Frame Format (FRF) bit in CR2 needs to be set to define as TI mode for serial communications. By selecting TI mode, bit SSM, SSI, LSBFIRST, CPOL and CPHA will be ignored and these bits will be set as default. However, FRF can be cleared as Motorola mode if a full control of configuration of SPI is needed.
   1.6. Master (MSTR) bit is cleared to become a slave and set SPI Enable (SPE) bit at the end of configuration if all the setting is done. By enabling this SPE bit, the SPI is well to communicate with the master. Any changes made after the enable of SPI will not be accepted. This is to ensure to prevent any mismatch of errors occur after sent or received.

Lastly, in this slave configuration, MOSI will become as an input pin and MISO will become as an output pin.

2. Master mode:

In the master configuration, the serial clock is generated on the SCK pin. This clock will be used by master to transfer the data and also for the slave. This is to let the master and slave be synchronous when sending data and receiving data. As for this master configuration, there is not much different as compared to slave configuration. The following procedure is the configuration for master.

2.1. The Baud Rate Control (BR) bits in CR1 needs to be configure to define the serial clock baud rate.
2.2. Configuration for CPOL and CPHA have to be configured and same with the slave. These bits will be ignored if TI mode is selected.
2.3. DFF bit has to be configured to define as 8- or 16-bit data frame format. This is also needs to be same as the slave.
2.4. LSBFIRST bit needs to be configured to define the frame format as LSB first or MSB first. This bit has to be same as slave and will be ignored if TI mode is selected.
2.5. If the NSS pin is required in input mode, in hardware mode, the NSS pin has be connected to a high-level signal during the complete byte transmit sequence. In NSS software mode, SSM and SSI bit needs to be set in CR1. However, if NSS is needed in output mode, Slave Select Output Enable (SSOE) needs to be set. This step is not required if TI mode is selected.
2.6. FRF is set as TI mode for serial communication. Otherwise, Motorola mode is another option and also a full configuration for CR1.
2.7. Lastly, MSTR bit needs to be set to define the SPI as master and set SPE if all the configuration has done. Highly restriction and not recommended for any changes made after SPE is set.

In this configuration, MOSI will become as an output pin and MISO will become as an input pin.

Direction of communication for SPI

SPI can be configured in either half duplex (1-line bidirectional data wire) or full duplex (2-line unidirectional data wire) as shown in Figure 3. By configuring SPI in full-duplex mode, the data transfer rate is actually 2x times faster than half duplex. This is because data can be exchanged between master and slave at the same time in full-duplex due to two wire connection them but not for half-duplex. Transmit and receive in half-duplex can only be done either one is finished due to one wire connection.

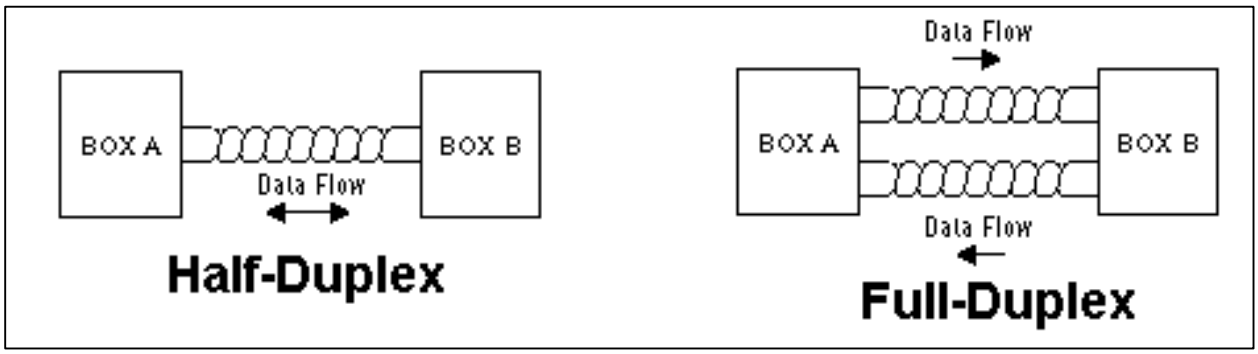

Figure 3 Half-Duplex versus Full-Duplex [*Accessed from: http://www.embarcados.com.br/comunicacao-spi-parte-2/*]

In order to configure SPI in half-duplex or full-duplex mode, the following procedure is the configuration for either half-duplex or full-duplex.

1. Half-duplex:
   1.1. Bidirectional Data Mode Enable (BIDIMODE) bit in CR1 needs to be set.
   1.2. Output Enable in Bidirectional mode (BIDIOE) bit in CR1 is cleared or set. For clearing BIDIOE will define SPI as receive-only mode due to output is disabled. In opposite, setting BIDIOE will define SPI as transmit-only mode.
2. Full-duplex:
   2.1. BIDIMODE will be cleared in order to communication in full-duplex mode.
   2.2. RXONLY bit in CR1 will need to be cleared to enable the transmission and reception. If RXONLY bit is set, master or slave can only receive data.

Both half-duplex and full-duplex must be the same in configuration for both master and slave. Otherwise, communication between the master and slave might not work.

Cyclic Redundancy Check (CRC):
CRC is a check that help to reduce any errors during transmission and reception. CRC is implemented for communication reliability because CRC is calculated using programmable polynomial serially on each bit. This CRC has two modes — CRC8 and CRC16. The selection of these two modes depend on the data frame format (DFF). If DFF is set to define for 8-bit communication, CRC will become CRC8. This is the same for CRC16.

CRC will be functioned whenever CRC is enabled by setting CRCEN bit in CR1. However, by setting CRCEN bit is not enough. Value has to be assigned in CRC polynomial register (CRCPR). This register will help to calculate CRC and attached to the data when transmit. But, CRCPR has an original value inside. Thus, assigned or not will not be a matter. Then, CRC Transfer Next (CRCNEXT) has to be set as soon as the last data is transmitting. This is to tell CRC to be ready and send out once the last data is finished in transmitting. If CRCNEXT bit is cleared, CRC will not be sent out whenever the last data is sent out.

In every transmission of a pack of data, CRC will be sent out once the last data is sent out. If a new set of data is going to send out with CRC enabled. CRC needs to be reset. This is because CRC will mess up with the next data if CRC has not reset yet but the new transmission of data is ongoing. Thus, the following procedure is the step to clear CRC before sending any new data.

Clear CRC:

1. Disable SPI by clearing SPE bit (SPE = 0).
2. Clear CRCEN bit.
3. Set CRCEN bit.
4. Enable SPI by setting SPE bit (SPE = 1).

Result and Discussion:
Throughout the whole experiment, TI mode is used due to the ease of work in configuration as the experiment is communicating in serial communication. However, in either TI mode or Motorola Mode is used, the result is same. All the results obtained as shown in Figures below were observed through oscilloscope.

In order to send out any data, GPIO pins for MOSI pin, MISO pin, NSS pin, and SCK pin need to be activated. Besides that, peripheral clock needs to be asserted too. Otherwise, data could not send out. The configuration for GPIO pins, peripheral clock and configuration for master and slave are attached at appendix.

Transmission:
1. 8-bits data transmission:

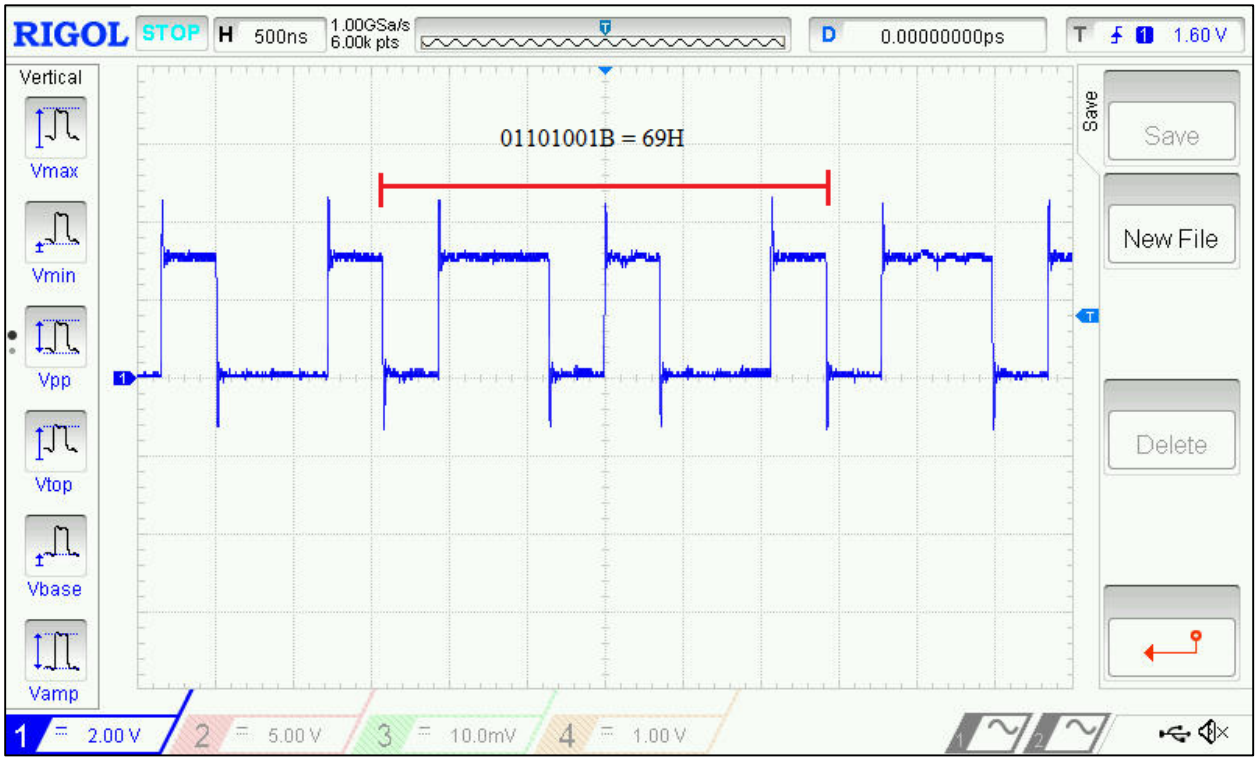

Figure 4 Transmission data of 69H

Figure 4 shows the result of transmitting 8 bits data, 69H. In order to send the data, transmit buffer (TXE) flag needs to be checked. When TXE flag is set, this indicates transmit buffer is emptied. Hence, data can be stored into transmit buffer by writing the data to data register. Once everything is ready, data will send through the output pin, either MOSI or MISO, to the destination.

In Figure 4, noise can be seen. Noise can be any unwanted signals like light. Whenever there is a hanger wire or similar that can be acted like an antenna, noise will be received that antenna-liked. To solve this problem, resistor can be a good helper. By placing a resistor at the output, noise will be trimmed.

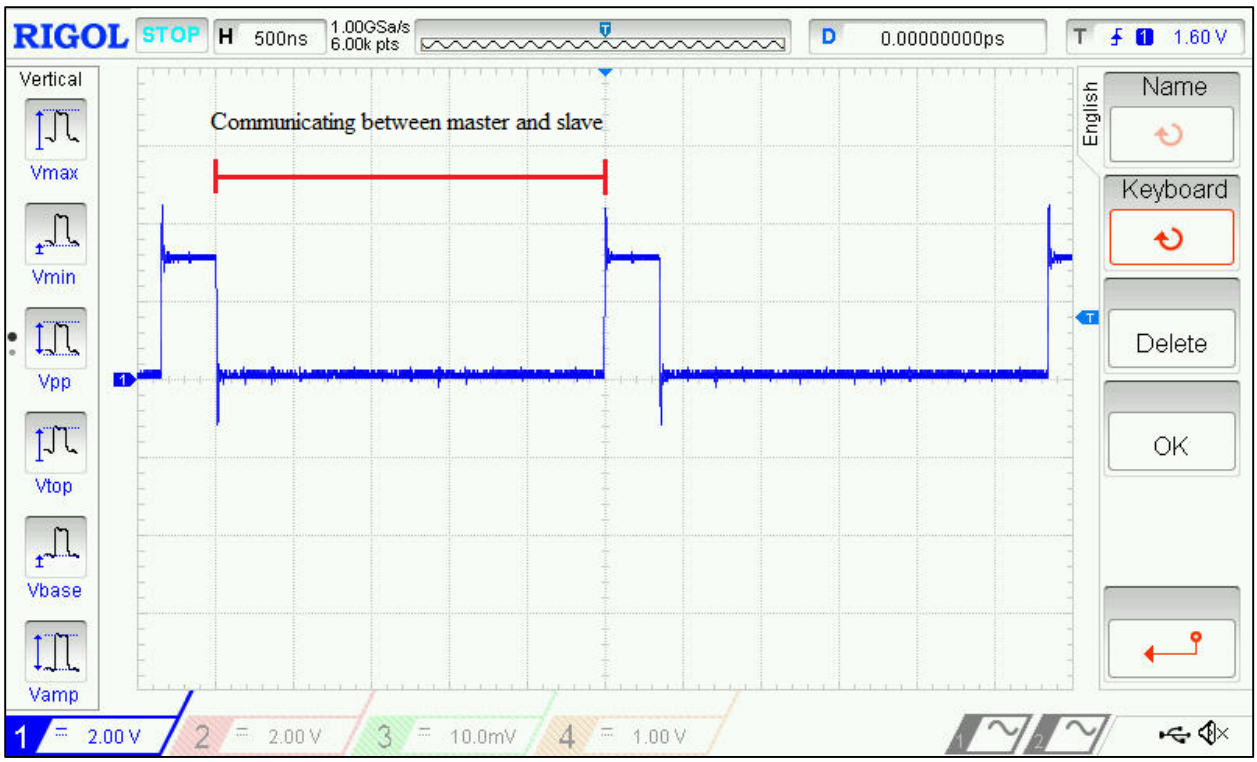

Figure 5 NSS signal between master and slave

NSS pin between master and slave is initially high level signal (logic 1). In order for master to communicate with the slave, NSS pin needs to put out a low level signal (logic 0) between master and slave. This is to indicate that the master is communicating with the specific slave if multiple slaves are connected to the master (multislave system). During the communication, any communication from other source are not allowed except the specific slave. This is to prevent any signal collision occurs that will cause the loss of main data. Thus, this is also an advantage of using SPI as communication tool. However, if multiple slaves are connecting the master, area will be a problem.

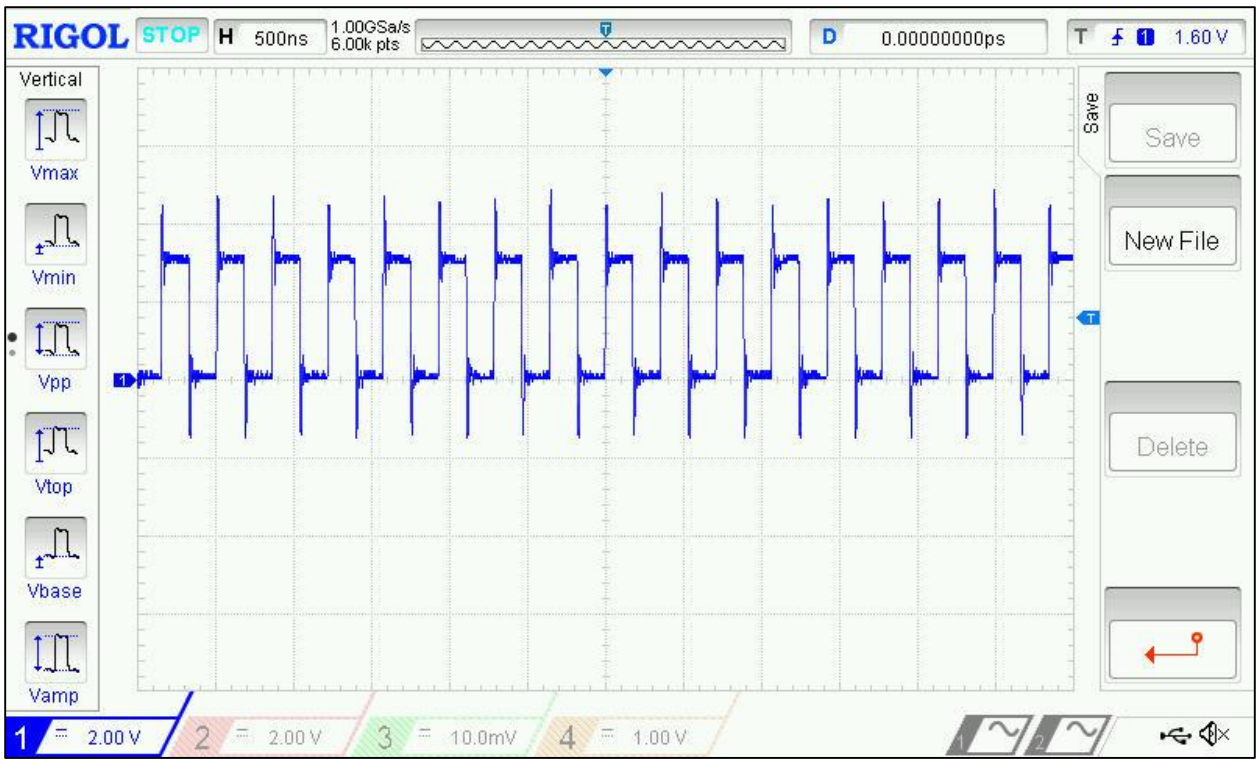

Figure 6 Serial Clock (SCK) generated by master and transmitted to slave

Figure 6 shows the serial clock generated by the master. From Figure 6, serial clock was generated continuously because the data was kept on transmitting to slave once TXE flag is set and RXNE is cleared which indicated new data can be sent and received. In order to generate this clock, peripheral clock for SPI needs to be asserted. If not, SPI will not be functioning and hence, no data can be sent and received.

2. 16-bits data transmission:

In order to send out a 16-bits data to the destination, DFF bit needs to be set. This is to set the data frame for the source to 16-bits, indicating that the source is about the send out or receive any data to or from the destination.

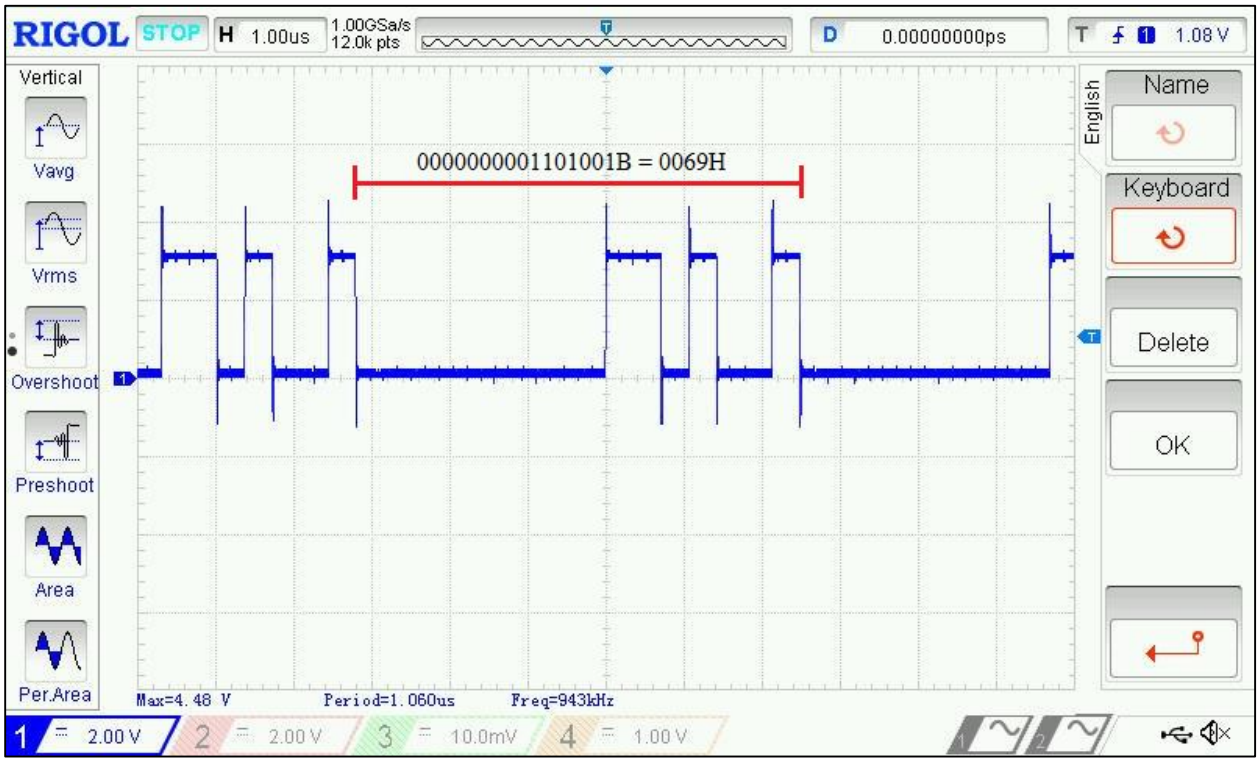

Figure 7 Transmission of data (69H) in 16-bits DFF

From Figure 7, data 69H is sent to the destination continuously in 16-bits data frame. Due to 69H data is an 8-bits data, the other 8-bits is not used. Thus, zeros are sent along with the data, causing the result as shown in Figure 7. Sending data in 8-bits frame or 16-bits frame has to depend on the how large the data is going to send at a time. In this case, either 8-bits data frame or 16-bits data frame will do the job because the data is only an 8-bits large data. As for NSS and SCK, these two behave the same as Figure 5 and Figure 6. The only difference for transmitting 8-bits data and 16-bits data is the data frame format that matters.

All the while, 2 unidirectional wire full-duplex communication is used. As for 1 bidirectional wire half-duplex communication, the results obtained through experiment are the same. The results are attached at appendix.

Reception for 8-bits and 16-bits data:

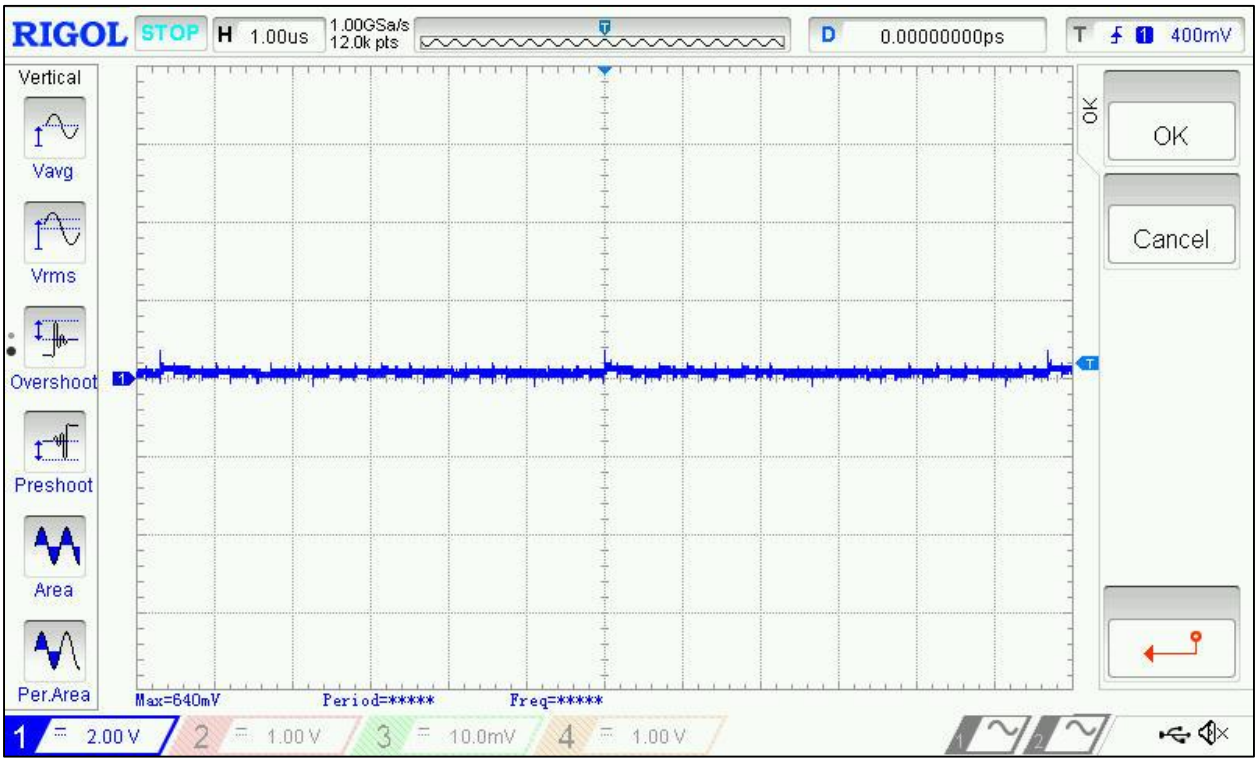

Figure 8 Reception of 8-bits and 16-bits data

Figure 8 shows the result obtained for receiving data from source in 8-bits and 16-bits data frame format. As shown in Figure 8, nothing is received because source did not send any data. This is to show the receiving mode in unidirectional mode and bidirectional mode in destination. When SPI is configured as destination to receive any data, SPI will be readied to receive data whenever receiver buffer is emptied which is represented by RXNE flag. However, whether the SPI is working or not, by observing the SCK and NSS will do the job. In this case, the receiving mode for SPI as destination to receive data is working because the SPI was able to receive the SCK signal generated by the source and SPI was connected to the source as shown in Figure 9 and Figure 10.

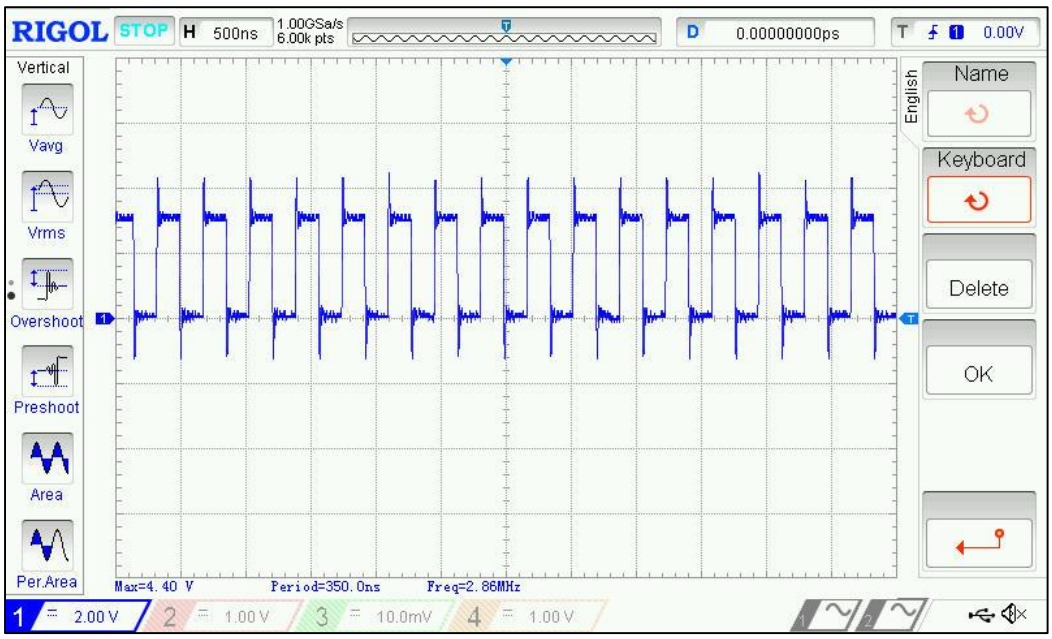

Figure 9 SCK received from source

As shown in Figure 9, destination SPI was able to receive the clock signal generated by the source. This is to show that the SPI is working.

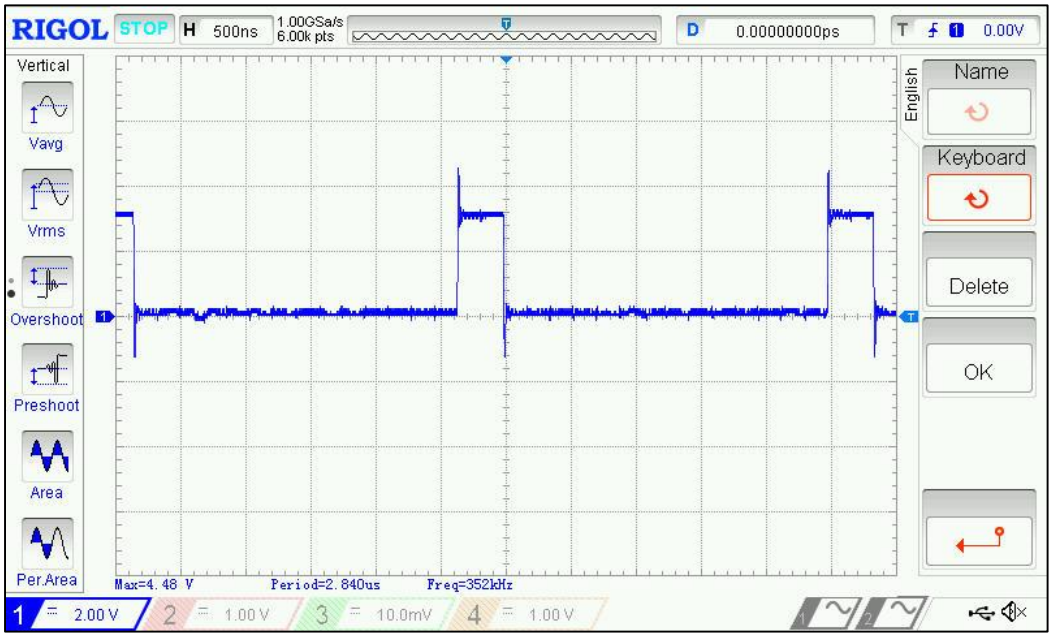

Figure 10 NSS pin signal between source and destination

Figure 10 shows NSS pin signal between source and destination. As shown in Figure 10, low signal level was presented. This shows that the SPI is connected and communicating.

Figure 9 and Figure 10 proved that the SPI is working. Just that no data is sent from the source of SPI because this is a test to test the functionality of SPI in receiving mode. Once the transmission and reception are tested working fine, data from source can send to destination for sure.

Cyclic Redundancy Check (CRC):

CRC is implemented in SPI for communication reliability. When CRC is enable, CRC will perform the CRC calculation using a programmable polynomial serially on each bit. Before that, a value can be assigned to CRC Polynomial Register (CRCPR) to do the CRC calculation. However, with or without a value assigned to CRCPR, CRC will perform the calculation because there is a default value (0007H) inside CRCPR. Then, the calculated CRC value will be stored in the transmit CRC register (TXCRCR) and receive CRC register (RXCRCR). In order to send the CRC value, CRCNEXT bit needs to be set as soon as the last data is written to the data register. Once the last data is sent out, CRC value will be sent right after the last data. In SPI, SPI offers two kinds of CRC calculation standard which depend directly on the data frame format selected for transmission and/or reception. Those CRC calculation standard is CRC8 and CRC16.

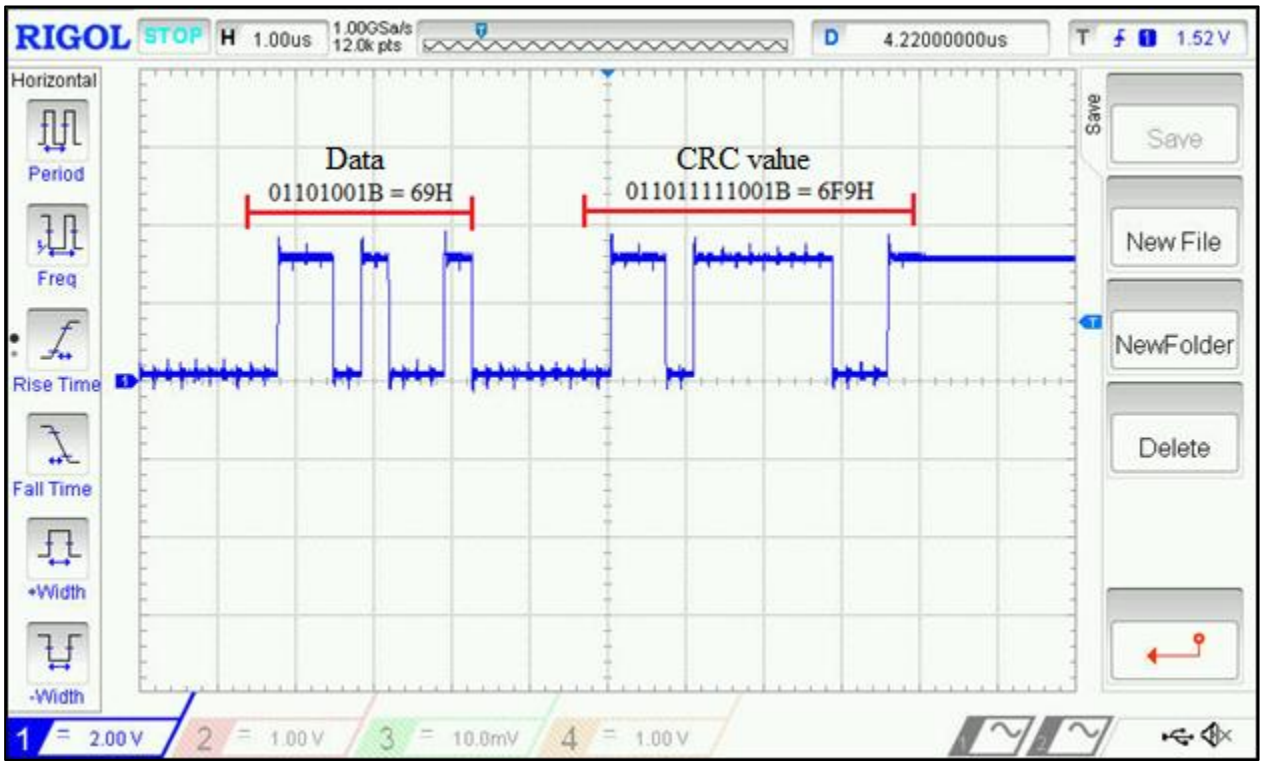

Figure 11 Data with CRCNEXT transfer

In Figure 11, CRC value was sent together with the data. In this case, 11H was written to CRCPR. With this value, CRC will perform the CRC calculation that combine the data and CRC polynomial value. Once the last data was sent, computed CRC value (6F9H) will be sent. CRC value can be affected by DFF. In Figure 11, DFF is 16-bits. Thus, CRC16 was used in the CRC calculation. When DFF is set as 8-bits data frame format, CRC will use CRC8 standard to compute and send out the computed CRC value.

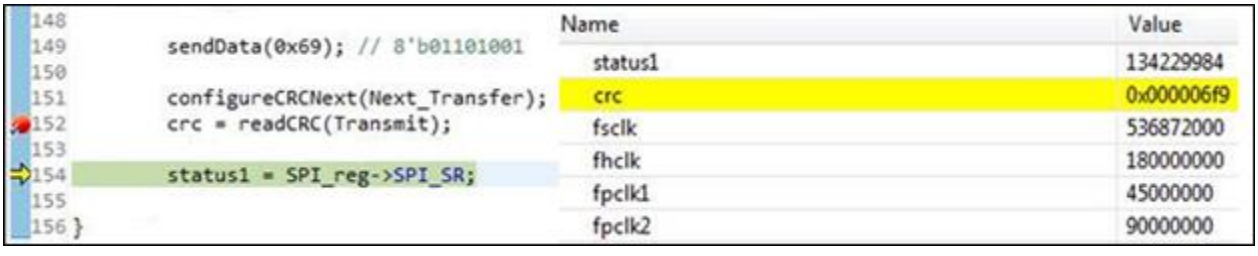

Figure 12 computed CRC value computed in CRC16 standard

Figure 12 shows the CRC value that was stored at TXCRCR by using the debugger of coIDE. As shown in Figure 12, the CRC value stored in TXCRCR is 6F9 in hex which is the same as shown in Figure 11. Once the polynomial in CRCPR is programmed by writing a value to CRCPR, a computed CRC value will be stored into TXCRCR and RXCRCR. Value in TXCRCR will be sent along with the data right after the last data is sent. Value in RXCRCR will be used to check the received CRC value from the source. If there was a mismatch between the CRC value received and the value in RXCRCR, error occurs and CRCERR flag will be set. This will generated an interrupt if error interrupt (ERRIE) is enabled.

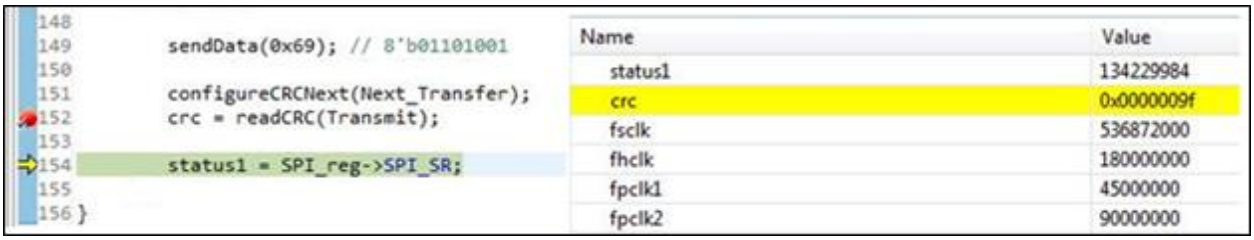

Figure 13 computed CRC value computed in CRC8 standard

Figure 13 shows the computed CRC value computed using CRC8 standard. This is because DFF was set to 8-bits data frame format. Thus, calculation uses CRC8 as standard to calculate the CRC value. In Figure 13, the CRC value in TXCRCR was observed through the use of debugger in coIDE which is shown as 9F in hex.

Direct Memory Access (DMA):
DMA is an intermediate device that helps the device in transferring the data. DMA is used in order to provide a high speed data transfer rate between peripherals and memory and between memory and memory. In normal situation where DMA was not used, transferring and receiving data will need CPU action which will slow down the processing power of the CPU. In order to prevent the processing power of CPU to reduce, DMA is capable to replace CPU to do all the transmission and reception of data because data can be quickly moved by DMA without any CPU action. This keeps the CPU resources free for other operations. DMA has two types, DMA1 and DMA2. Types of DMA used needs to depend on the peripheral used in communication. For this experiment, DMA2 was used because SPI4 is used in communication.

For DMA usage, SPI has implemented DMA accessibility that enable SPI to cooperate with DMA to send and/or receive data. In order to use DMA as an intermediate device, Transmit buffer DMA Enable (TXDMAEN) bit and/or Receive buffer DMA Enable (RXDMAEN) bit has to be enabled. Both of bits can be found in Control Register 2 (CR2). Besides that, DMA and the peripheral clock have to be configured as well in order to activate DMA as a helper to move the data along. Once everything is done, SPI and DMA will work fine.

Furthermore, some important conditions need to pay attention. Table 1 and Table 2 shows the important conditions that user needs to pay attention when configuring DMA.

Table 1 DMA2 Request Mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A(1)	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A(1)	ADC1	SAI1_B(1)	TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2	SAI1_B(1)	SPI6_TX(1)	SPI6_RX(1)	DCMI
Channel 2	ADC3	ADC3		SPI5_RX(1)	SPI5_TX(1)	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4	SPI4_RX(1)	SPI4_TX(1)	USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX	SPI4_RX(1)	SPI4_TX(1)		USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX(1)	SPI5_TX(1)	TIM8_CH4 TIM8_TRIG TIM8_COM

1. These requests are available on STM32F42xxx and STM32F43xxx.

[Accessed from: http://www.mcustep.ru/init/stm32f4/46-dma-v-stm32f4-opisanie.html]

Table 1 shows the DMA Request Mapping where user needs to take note because Table 1 shows the specific stream and channel for a peripheral. In this experiment, SPI4 was used. Thus, the colored square boxes show the stream and channel that SPI4 can be used. For the experiment, the stream and channel was used as according to

the red square box. These stream and channel for each peripheral are fixed in DMA. If the configuration for stream and channel in DMA was configured wrongly, sure no output can be obtained.

Table 2 Source and Destination Address

Bits DIR[1:0] of the DMA_SxCR register	Direction	Source address	Destination address
00	Peripheral-to-memory	DMA_SxPAR	DMA_SxM0AR
01	Memory-to-peripheral	DMA_SxM0AR	DMA_SxPAR
10	Memory-to-memory	DMA_SxPAR	DMA_SxM0AR
11	reserved	-	-

[Accessed from: http://www.mcustep.ru/init/stm32f4/46-dma-v-stm32f4-opisanie.html]

Table 2 shows the source and destination address where user needs to assigned to by knowing the transferring direction. For SPI with accessed to DMA, user needs to configure the transferring direction of DMA to Memory-to-Peripheral in order for SPI to transfer data. If this configuration configured incorrectly, SPI cannot transfer any data. Besides that, Number of Data to Transfer (NDT) in Number of Data Register (NDTR) needs to be assigned with a value. This is to indicate the number of data to be transferred that DMA will do until NDT is zero.

Transmission data of SPI with DMA:
When TXDMAEN is enabled in CR2, a DMA request will be issued each time TXE is set to logic 1. Then, DMA will write the data to data register. This will then clear the TXE flag and wait till the data is sent. After the data is sent, TXE flag will be set again. This shows that the transfer direction for DMA and peripheral is Memory-to-Peripheral.

1. 8-bits data frame format

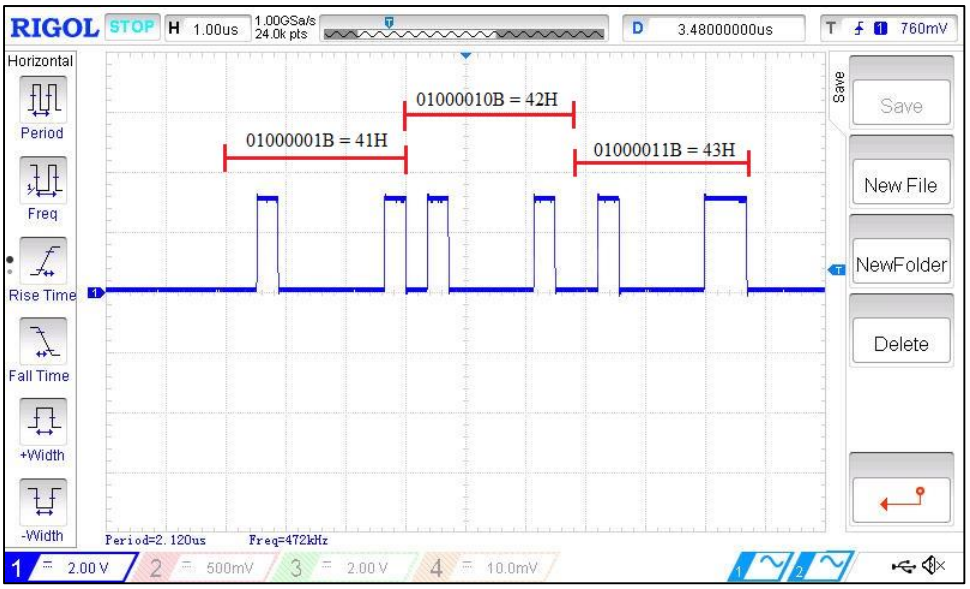

Figure 14 DMA transfer with 8-bits data frame format

Figure 14 shows the data transmission of SPI in 8-bits data frame format with the help of DMA. Initially, values "ABC" (41H, 42H, and 43H) are stored in the memory. When TXDMAEN is enabled and a DMA request is issued, DMA will write the data from memory to data register. The data then transmits to the destination (master or slave).

2. 16-bits data frame format

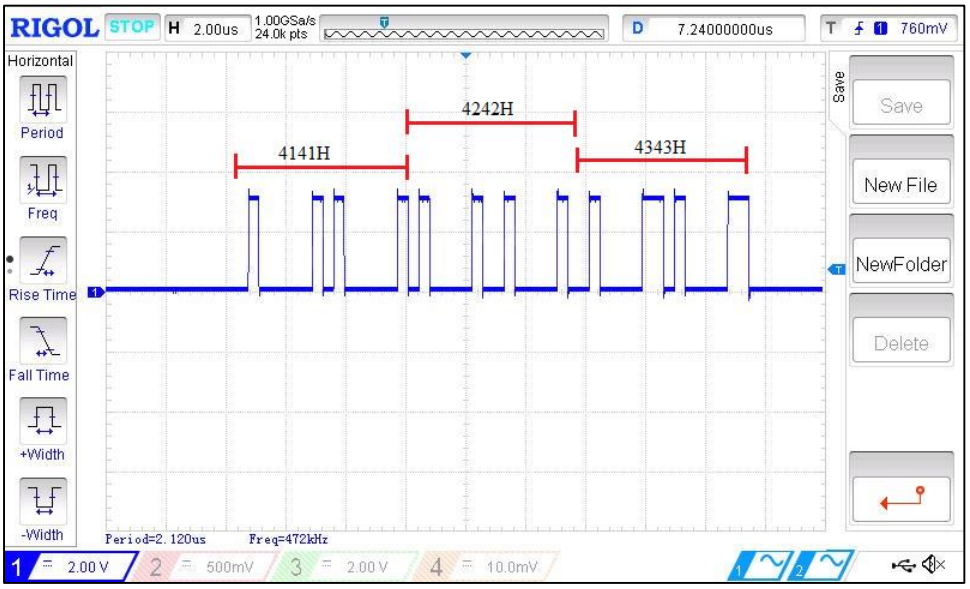

Figure 15 DMA transfer with 16-bits data frame format

Figure 15 shows the data transmission of SPI in 16-bits data frame format with the help of DMA. The data stored in memory are of 8-bits data size. When these data are sent in a 16-bits data frame format, DMA will fill up the unused space with the same data. Figure 15 shows the situation where a data is used in filling up the unused space. If not, the result obtained will be the same as Figure 14.

Reception data of SPI with DMA:
When RXDMAEN is enabled in CR2, a DMA request will be issued each time RXNE is set to logic 1. After that, DMA will read the data from data register. This clears the RXNE flag. This is to indicate that the Rx buffer has moved to the destination. The transfer direction for DMA and peripheral is Peripheral-to-Memory.

1. 8-bits data frame format

Figure 16 Data received in 8-bits data frame format

Figure 16 shows the data reception of SPI with DMA. Initially, memory only stored a character 'A' (41H). When the data is sent from the source and received by the destination (master or slave), DMA will read the data from data register when a DMA request is issued. Figure 16 shows the data received by destination through observing from oscilloscope. Besides that, the NDT was set to 1. Once the NDT become zero, no more data can be sent to the destination.

Name	Value
rxBuffer	0x20000460
rxBuffer[0]	'A'
rxBuffer[1]	0
rxBuffer[2]	0

Figure 17 DMA reception observed in debugger mode

Figure 17 shows the DMA moved the received data to memory (rxBuffer). This was observed through the debugger in coIDE.

2. 16-bits data frame format

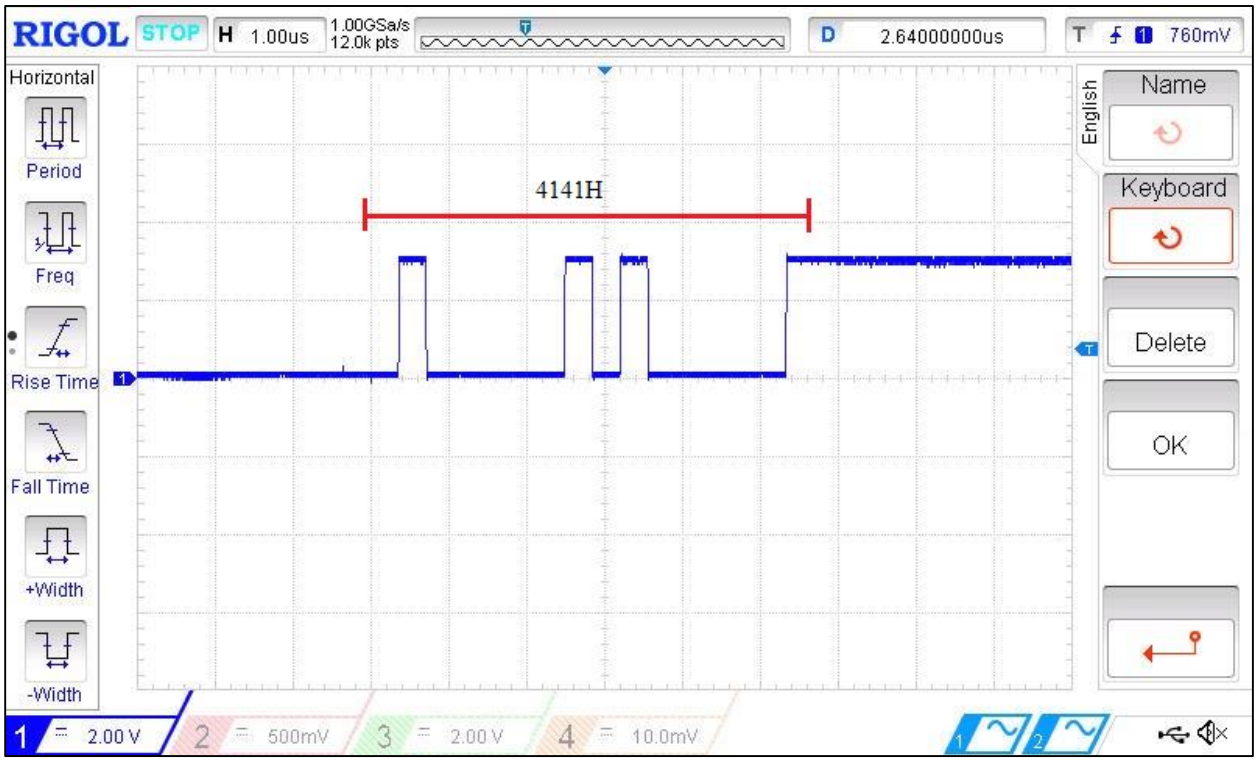

Figure 18 DMA reception in 16-bits data frame format

In Figure 18, 4141H was received and observed through oscilloscope. The data stored in memory is 41H. Due to DMA will fill up the unused space when writing the data into the data register in 16-bits data frame format and sent to destination, data received at destination will be the same as shown in Figure 18. Once NDT is zero, no more data can be sent to destination. In this case, NDT was assigned with data size 1. Thus, the result shown in Figure 18 is a data that only transferred once but not twice.

DMA with CRC:
When SPI communication is enabled with CRC communication and DMA mode, the transmission and reception of CRC at the end of communication will be automatically sent and received without the use of CRCNEXT. This is due to the help of DMA that actually transmits and receives the CRC value for each transmission and reception of data.

1. Transmission and Reception of Data in 8-bits data frame format

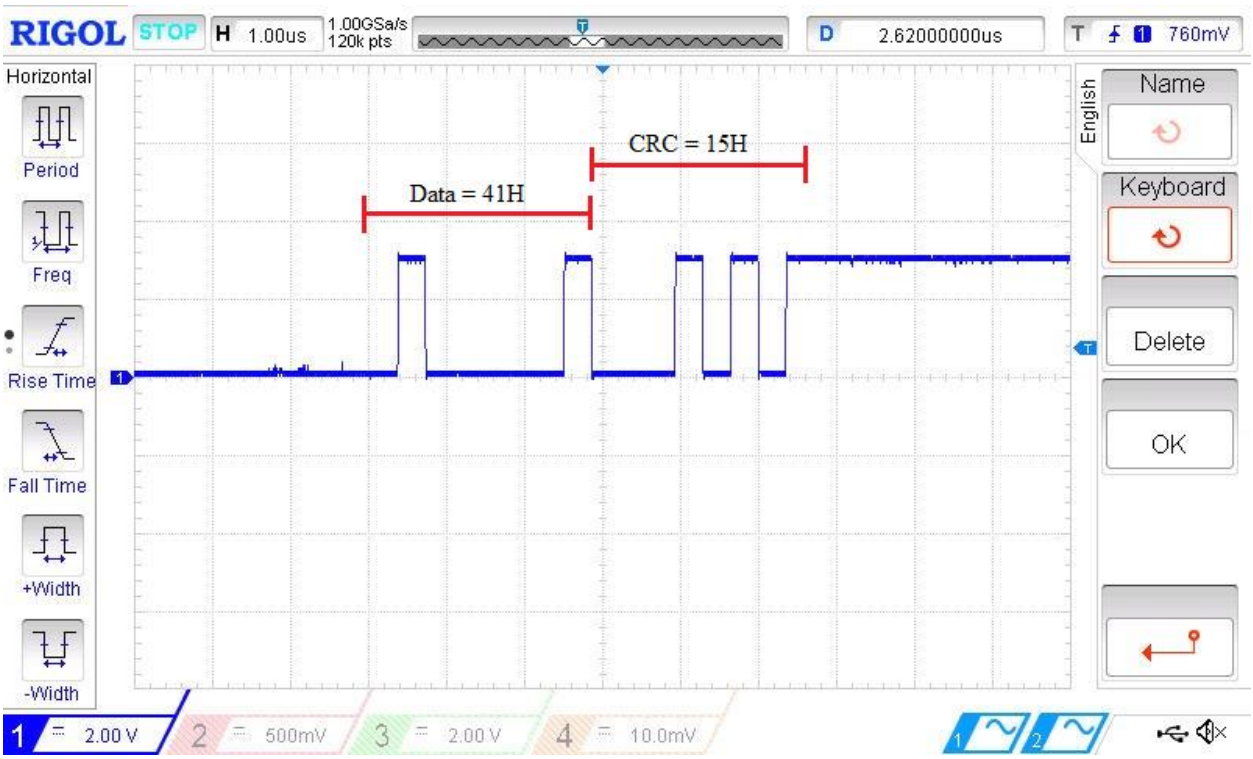

Figure 19 Data Transmission and Reception with CRC and DMA mode in 8-bits data frame format

Figure 19 shows the data transmission and reception with CRC enabled and DMA mode in 8-bits data frame format. CRC was transmitted and received without the assertion of CRCNEXT. DMA is able to detect the last data due to NDT that was assigned in NDTR. Once NDT is 0, CRC value will be sent. This is the same as receiving the data. If the received CRC value did not match the value in RXCRCR, CRCERR flag will be set.

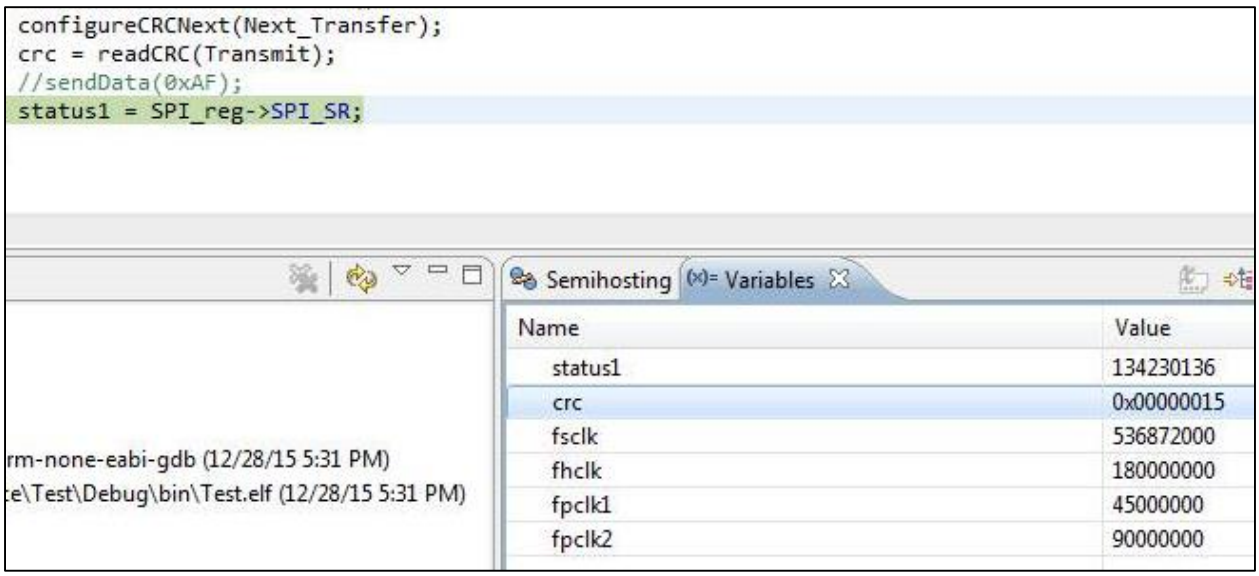

Figure 20 CRC value in TXCRCR with DMA mode

Figure 20 shows the CRC value stored in TXCRCR after CRC has computed the polynomial in CRCPR where the polynomial assigned to CRCPR is 11H. Figure 20 is used to prove the signal observed by oscilloscope for the CRC value that was transmitted and received in DMA mode in Figure 19 is the same as Figure 20. The "crc" in Figure 20 shows the CRC value in TXCRCR.

2. Transmission and Reception of Data in 8-bits data frame format

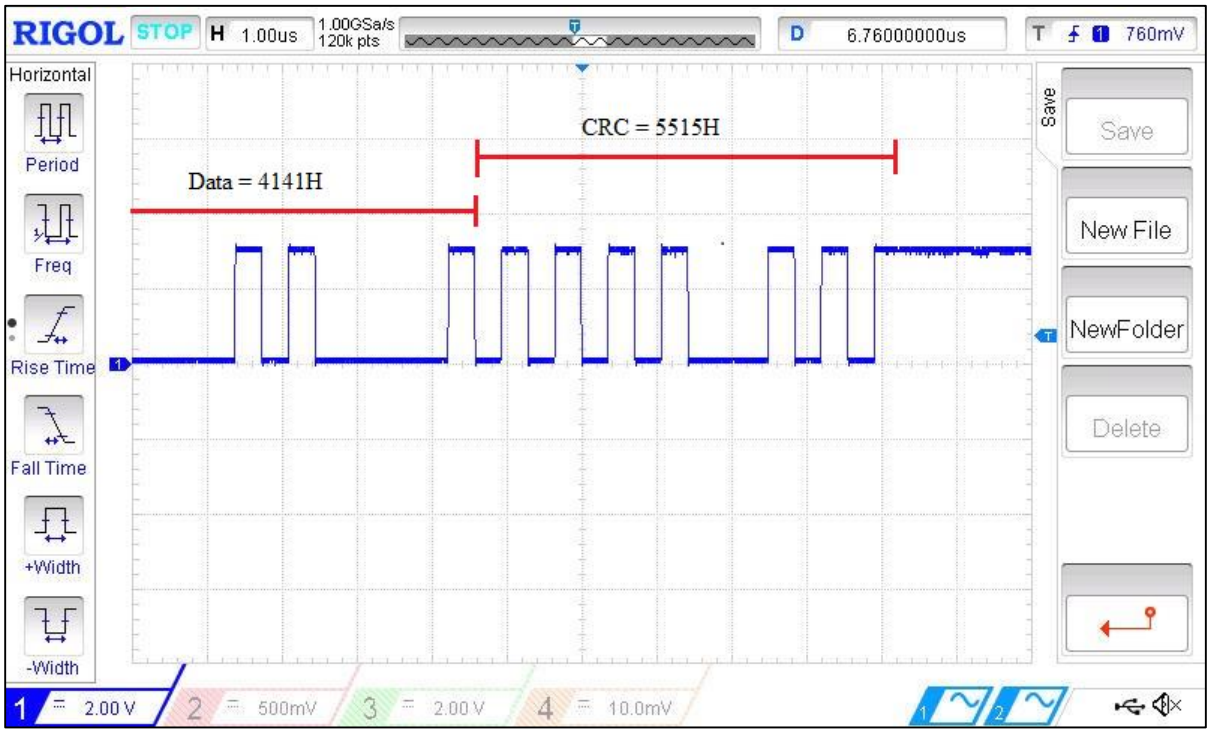

Figure 21 Data Transmission and Reception with CRC and DMA mode in 16-bits data frame format

Figure 21 shows the data transmission and reception with CRC and DMA mode in 16-bits data frame format. Due to some mistake, a part of the signal is missed in capturing the data through oscilloscope. The actual data that is going to be observed in 4141H which is expected because data is transmitting in 16-bits data frame format where

DMA will fill up the unused space. In this case, the CRC value that was transmitted and received is 5515H. Data transmission and reception with CRC and DMA mode in either 8-bits or 16-bits data frame format matters less. The only difference is the data sent might be duplicated where DMA filled up the unused space with the same data that cause duplication and the CRC value that is transmitted and received.

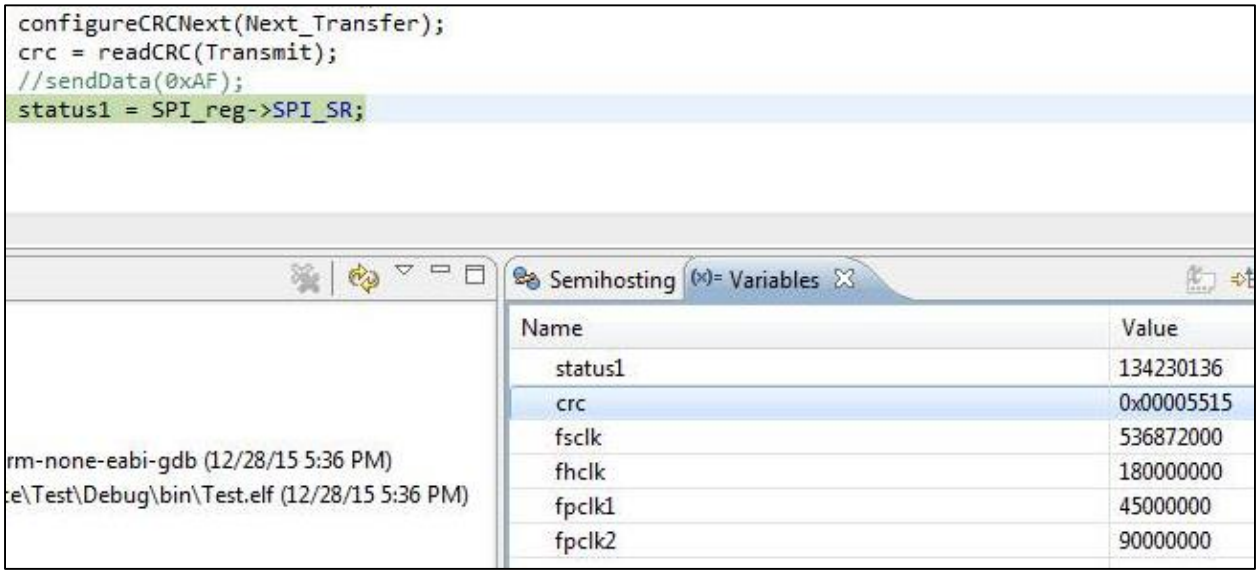

Figure 22 CRC value in TXCRCR with DMA mode

Figure 22 shows the CRC value stored in TXCRCR after CRC has computed the polynomial in CRCPR where the polynomial assigned to CRCPR is 11H. Figure 22 is used to prove the signal observed by oscilloscope for the CRC value that was transmitted and received in DMA mode in Figure 21 is the same as Figure 22. The "crc" in Figure 22 shows the CRC value in TXCRCR.

Interrupt:

The use of interrupt is to indicate an event needs an immediate attention. In SPI, two kind of interrupts are used, Transmit buffer empty interrupt (TXEIE) and Receive buffer not empty interrupt (RXNEIE), and error interrupt (ERRIE). However, ERRIE is not used in the experiment. Thus, only TXEIE AND RXNEIE are used. Besides that, DMA also does have an interrupt feature. Whenever the interrupt is generated, the specific status flag can be observed in status register. Interrupt between SPI and DMA are unable to be used together. Hence, selection in either using interrupt of SPI or interrupt of DMA needs to be selected. This selection can be done in selecting the flow control in DMA. By setting the flow control in DMA flow controller, the interrupt used will be DMA. Otherwise, interrupt of SPI will be used if peripheral flow controller is selected.

In order to use the interrupt features, Nested Vectored Interrupt Controller (NVIC) is needed. Thus, the specific interrupt function and specific interrupt handler are used. Both the interrupt function and interrupt handler can be found in "startup_stm32f429xx.s" file and "stm32f429xx.h" file respectively. By the way, the interrupt handler needs to be enabled by the function HAL_NVIC_EnableIRQ( ).

Whenever interrupt is enabled, an interrupt is generated when the condition is fulfilled. This situation occurred inside the system where the signal is hard to observe through oscilloscope. Thus, flowchart is used to indicate the flow of generating an interrupt in SPI and DMA.

1. Flow of interrupt in SPI

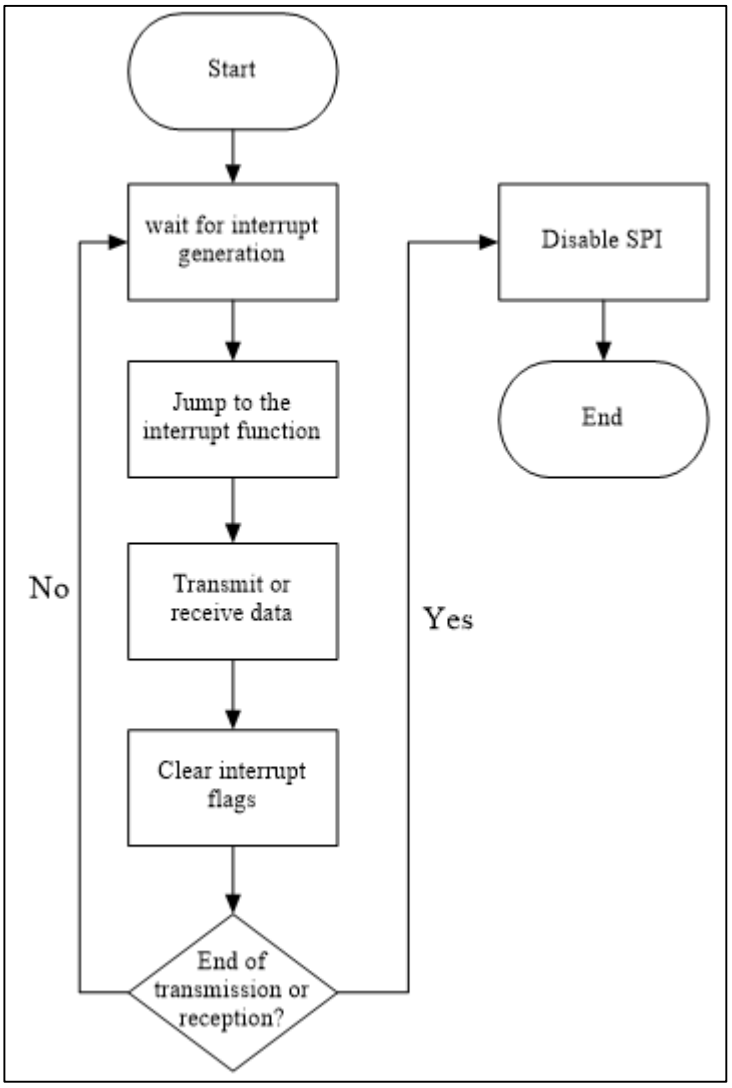

Figure 23 Flow chart for TXE AND RXNE interrupt

Whenever the interrupt is enabled, an interrupt will be generated when the condition is met. Once the interrupt is generated, the current event will be stopped and jump to the interrupt function. At the interrupt function, data transmission or reception will depend on the TXE and RXNE flag. After that, the interrupt/status flag will be cleared. Then, if the transmission or reception has not done yet, then the flow will be loop back to wait for an interrupt to be generated. Otherwise, SPI will be disabled by clearing the SPE bit in CR1 and end of communication.

2. Flow of interrupt in DMA

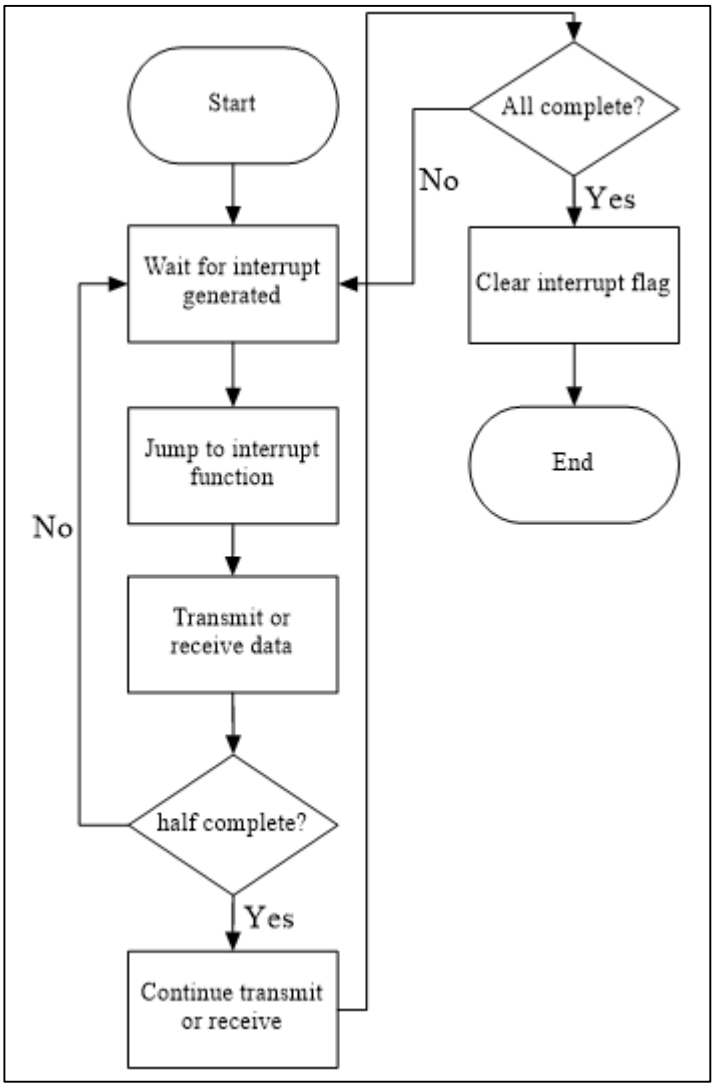

Figure 24 Flow chart for DMA interrupt

Figure 24 shows the flowchart of DMA interrupt. Whenever the interrupt is enabled and generated, the current event will be stopped and enter the interrupt function. Data transmission or reception will be done in this interrupt function. If the progression of either transmitting or receiving data is half complete, an interrupt will be generated and a Half Transfer Interrupt Flag (HTIF) will be set.

After the progression is complete, an interrupt will be generated and a Transfer Complete Interrupt Flag (TCIF) will be set. Once these two flags are set, this indicated that the data transmission or reception is done. Then, the corresponding bits in either Low Interrupt Status Register (LISR) or High Interrupt Status Register (HISR) need to be cleared. DMA will then be free again and able move the data around.

Conclusion:

Master is able to communicate with the slave with SPI communication. Several testing like DFF8, DFF16, CRC8 and CR16 are tested with SPI communication. Lastly, SPI is capable of using CRC for reliability communication and DMA for maximum transfer rate.