I²C to SPI Converter using SLG47011V
SLG47011

This application note shows the implementation of an I²C to SPI converter using the SLG47011V. The SLG47011V was chosen for this purpose because it contains all the necessary macrocells, such as Shift Registers, I²C Host Interface, Width Converter, Memory Table, and CNT/DLY. These features make the SLG47011V a perfect option for implementing an I²C to SPI converter.

Additionally, the SLG47011V offers high flexibility, allowing easy adaptation to various system requirements. This makes it not only a convenient but also a reliable solution for a wide range of applications.

Contents

References

For related documents and software, please visit: AnalogPAK | Renesas

Download our free Go Configure Software Hub [1] to open the .aap file [2] and view the proposed circuit design. Use the AnalogPAK development tools [3] to freeze the design into your own customized IC in a matter of minutes. Renesas Electronics provides a complete library of application notes [4] featuring design examples, as well as explanations of features and blocks within the Renesas IC.

[1] GreenPAK Go Configure Software Hub, Software Download and User Guide, Renesas Electronics
[2] AN-CM-409 I²C to SPI Converter using SLG47011V.aap, AnalogPAK Design File, Renesas Electronics
[3] GreenPAK Development Tools, AnalogPAK Development Tools Webpage, Renesas Electronics
[4] GreenPAK Application Notes, GreenPAK Application Notes Webpage, Renesas Electronics

Author: Andrii Velikhovskyi, Application Engineer, Renesas Electronics

Terms and Definitions

SHR	Shift register
CNT/DLY	Counter/Delay block

1. Introduction

In modern electronic systems, it's important for different devices to communicate effectively to achieve the best performance. A common issue is connecting devices that use different communication protocols, such as I²C and SPI. To solve this, an I²C to SPI converter can be used, which allows these two types of devices to communicate with each other. This application note shows how to implement an I²C to SPI converter using the SLG47011V IC.

The application note focuses on the practical steps for designing and using this converter. By following these instructions, you can connect I²C and SPI devices, improving compatibility and making electronic systems more flexible.

2. GreenPAK Design

The SLG47011V includes a Memory Table, Width Converter, I²C host interface and shift registers, which allows to create an I²C to SPI converter. Figure 1 shows the GreenPAK design for this converter it can both read and write SPI data through I²C. To write data, the Memory Table is used, which can store up to 4095 12-bit words. If you fill all 4095 12-bit words, you will no longer be able to transmit more data over SPI. The Width Converter then transform this data to MOSI SPI signal.

CNT3/DLY3 and LUT7 formed the internal oscillator, ensures writing, reading, and outputting SPI data. By default, the oscillator is set to 1 MHz, providing a stable and reliable clock source for the system. However, this frequency can be adjusted by rewriting the CNT3/DLY3 counter data via the I²C interface, providing flexibility in various applications.

For reading data, the SLG47011V uses six shift registers to store incoming SPI data. This design allows up to six bytes to be read at a time. CNT4/DLY4 and LUT5 limit the read data to six bytes, ensuring that the data is correctly read and processed. This internal limitation is important for keeping the data accurate and preventing overflow. The recording of the input data starts at the beginning. However, there is flexibility in adjusting the start of the recording by changing the counter data of CNT4/DLY4. For instance, it is possible to configure the design to start recording six bytes after the fifth byte is read. This feature allows for precise control over data capture, accommodating various timing and sequencing requirements in complex communication scenarios.

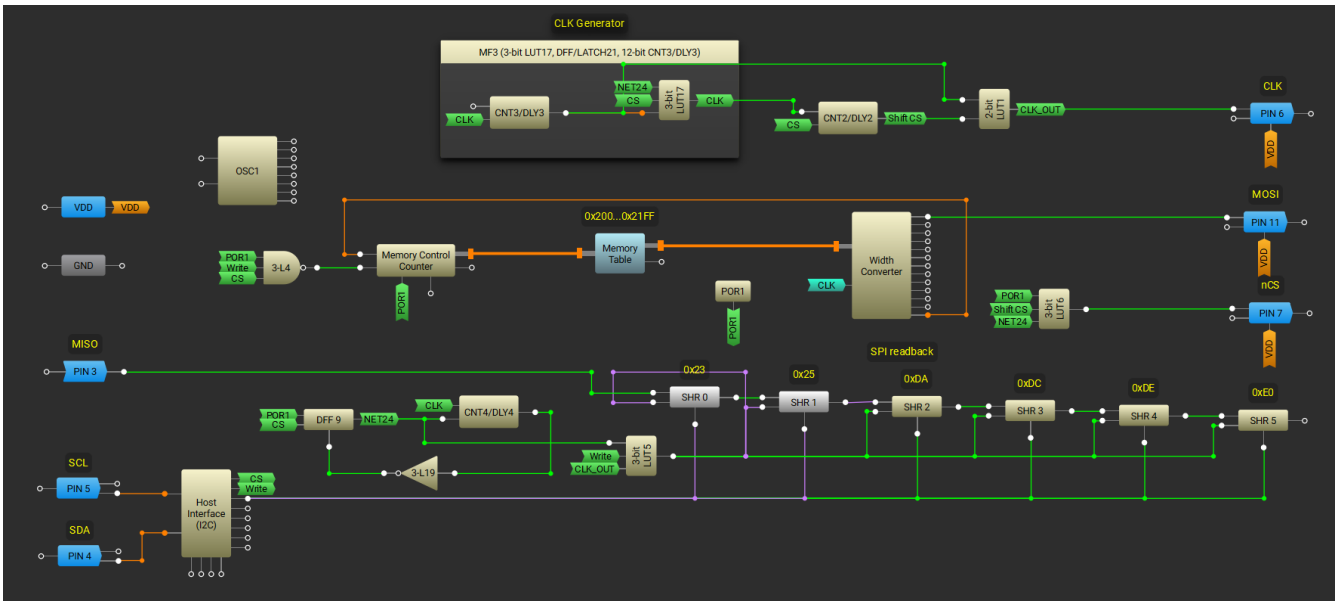


Figure 1. GreenPAK Design

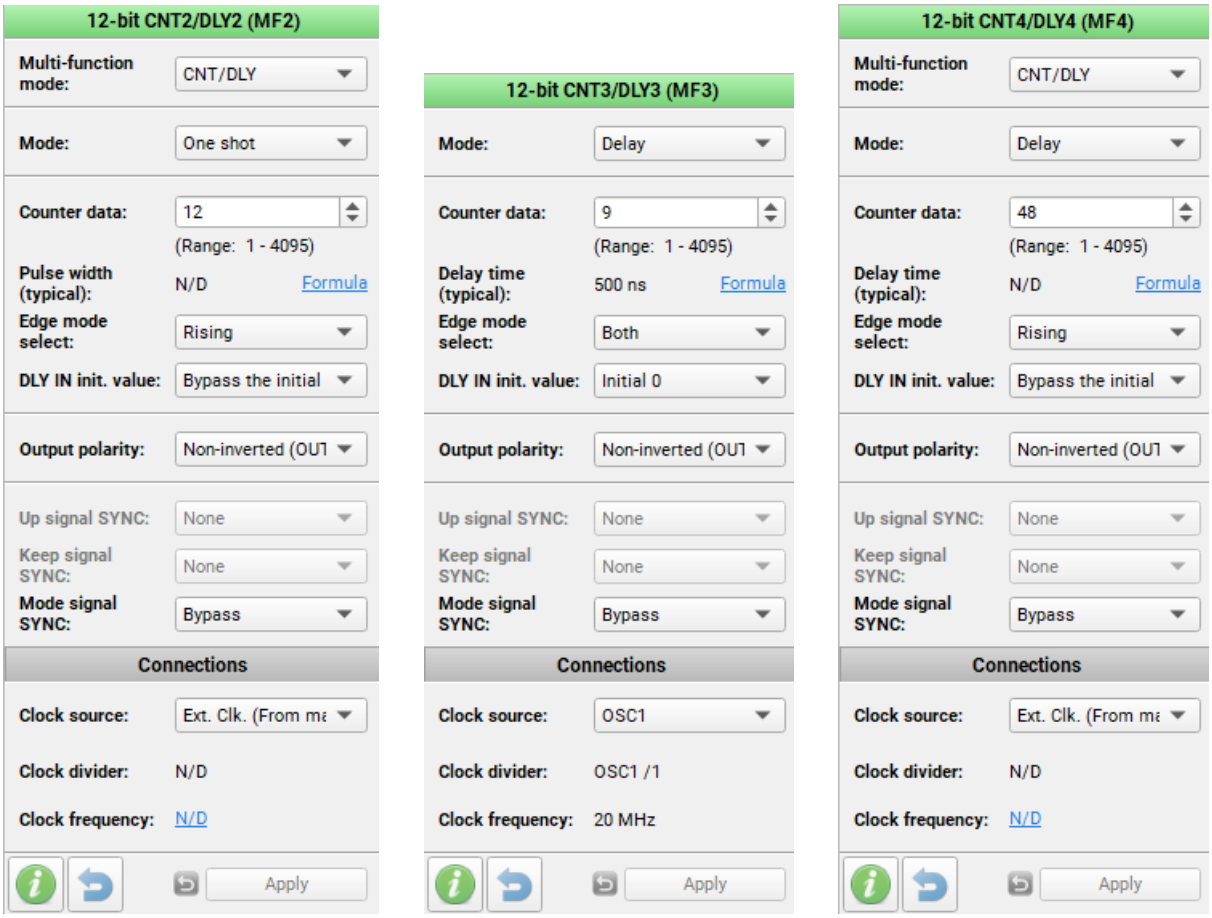


Figure 2. CNT/DLY marcocells settings

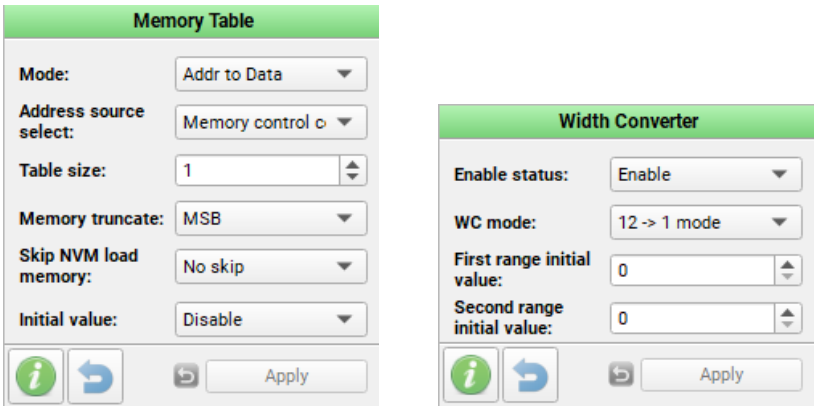


Figure 3. Memory Table and Width Converter marcocells settings

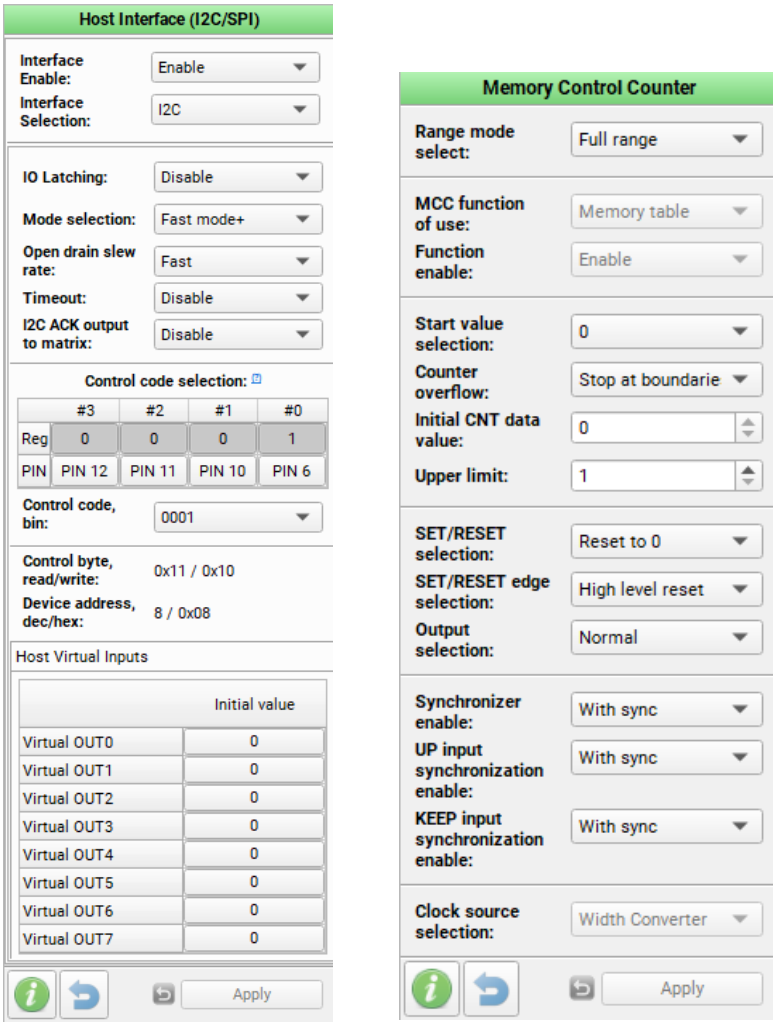


Figure 4. Design marcocells settings

2.1 Writing SPI data

Let's take a closer look at the SPI writing process. First, we need to write the value '0x84' to the 0x155 register via I²C, which switches the memory table to RAM mode and makes it possible to write the data that will be sent via SPI. Table 1 lists the I²C register control data which will help to choose correct register.

Table 1: Channel Selection vs different conditions

Address Byte	Register Bit	Block	Function
0x23	reg<280:287>	SHR0	Read current data in Shift Register0
0x25	reg<296:303>	SHR1	Read current data in Shift Register1
0xDA	reg<1744:1751>	SHR2	Read current data in Shift Register2
0xDC	reg<1760:1767>	SHR3	Read current data in Shift Register3
0xDE	reg<1776:1783>	SHR4	Read current data in Shift Register4
0xE0	reg<1792:1799>	SHR5	Read current data in Shift Register5
0x61	reg<776>	I²C Virtual input 0	Set CS signal
	reg<777>	I²C Virtual input 1	Set write signal

	reg<778>	I²C Virtual input 2	Reset the Shift registers
0x155	reg<2728>	Memory table mode switch	Switch between RAM and ROM mode
0x200-0x21FF	reg<4096:69631>	Memory table	Read data from memory table
0xF4	reg<1952:1959>	CNT3/DLY3 counter data	Set frequency (default 1MHz)

The data in the memory table is stored in registers 0x200 – 0x21FE. It is important to note that registers in the memory table are organized in 12-bit segments. For example, you can store 12 bits in the registers at address 0x200 and 0x201, and this applies similarly to each word.

So, if we need to write several 8-bit words into the memory table, the first word will be fully written into the first 12-bit segment, leaving the last 4 bits empty. The first 4 bits of the next 8-bit word will then be written into these last 4 bits of the first 12-bit segment. The remaining 4 bits of this second 8-bit word will be written into the second 12-bit segment then the third 8-bit word will be fully written to the second 12-bit segment, and so on

Let’s look at an example. Imagine we need to send the following sequence of 8-bit words: 0xAB, 0xCD, and 0xEF. You would start by writing 0xAB into the first 8 bits of register 0x200. Then, the first 4 bits of the next register, 0x201, would be used to store the first 4 bits of 0xCD. Consequently, the remaining 4 bits of 0xCD would be stored in the first 4 bits of register 0x202. The last 4 bits of register 0x202 and the first 4 bits of register 0x203 would be used to store the entirety of 0xEF. Figure 5 shows how this example appears in the Memory Table Data Editor."

This process of overlapping the storage of 8-bit words into 12-bit registers ensures efficient use of memory space and alignment with the chip’s data transmission protocol.

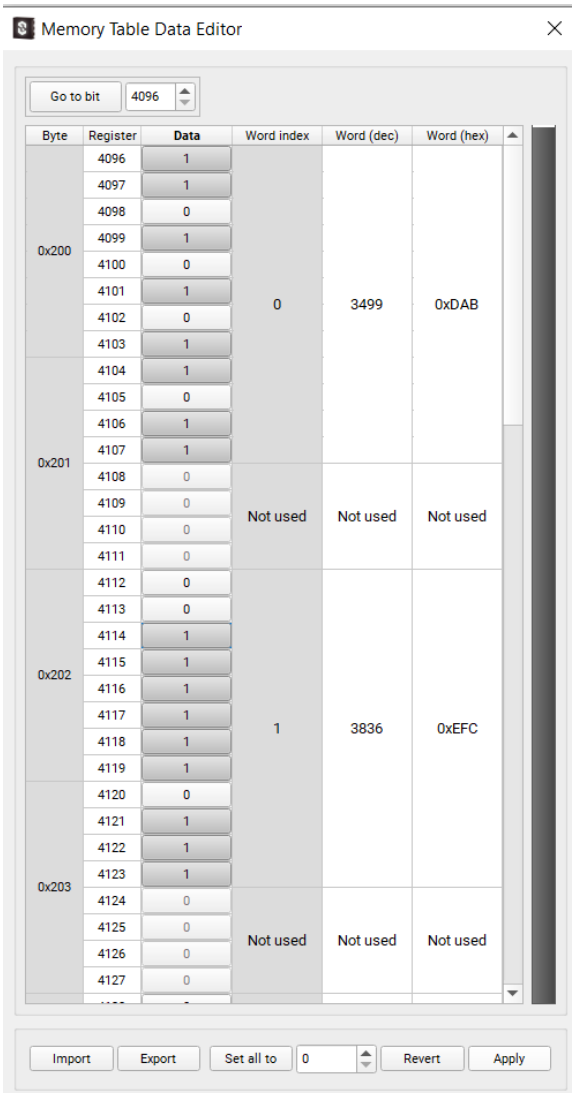


Figure 5. Example sequence of storing 8-bit words: 0xAB, 0xCD, and 0xEF in the memory table

This method of data storage requires careful attention to detail, particularly in managing the overlap between registers. Any miscalculation or oversight can lead to data corruption or misalignment, which would subsequently result in errors during data transmission over SPI. It is crucial to implement a precise algorithm for handling the writing process, ensuring that each 8-bit word is correctly partitioned and allocated to the appropriate registers without overlapping which can cause errors.

Additionally, it is beneficial to implement verification steps to check the integrity of the written data before it is transmitted via SPI. This can be achieved by reading back the data from the memory table and comparing it with the intended values. Any discrepancies can be corrected before the final transmission, which will ensure the reliability and accuracy of the data communication process.

After writing the data to the memory table, we need to write the value ‘0x8F’ to the 0x155 register. This switches the memory table to ROM mode and enables the ability to output the written data via SPI.

Finally, we write the value ‘0x03’ to the 0x61 register to set the I²C virtual inputs 0 and 1 (CS and write signal, respectively) to a HIGH level. This initiates the internal 1 MHz oscillator and begins sending SPI data via the MOSI signal.

2.2 Reading SPI data

To read data via SPI, we must first set I²C virtual input 2 to a HIGH level and subsequently set input 0 to a HIGH level. This command set can be sent without an additional time gap between them. Once I²C virtual input 0 is set to HIGH, the internal clock starts. After 12 cycles of the internal clock, this signal reaches Pin 6, and at this point, the data read via SPI is written to the shift registers. Table 1 lists the addresses for all shift registers from which you can read SPI data. After 48 cycles of the internal clock, the data will no longer be written. This means any data transfer operations must be completed within this 48-cycle window to ensure data integrity. However, even after this period, the oscillator will continue to operate until I²C virtual input 0 is set to LOW.

To optimize performance, ensure that all necessary preparations are made before initiating the data transfer. This includes configuring the SPI and I²C interfaces correctly, verifying all timing requirements, and ensuring that any other system dependencies are met. Additionally, consider implementing error-checking mechanisms to handle any potential issues that may arise during the data transfer process. This could include verifying the integrity of the data before and after transfer, monitoring the internal clock cycles to ensure proper timing, and implementing retries or error correction protocols as necessary. By taking these precautions, we can enhance the robustness of our system and reduce the likelihood of data corruption or communication failures.

Here is an I²C command example of writing and reading the SPI data

Writing SPI data

[start] [0x08] [w] [0x00] [0x61] [0x00] [stop] //Set CS and write signal to 0, reset the shift registers.

[start] [0x08] [w] [0x01] [0x55] [0x84] [stop] //Set Memory Table to “Storage” mode

[start] [0x08] [w] [0x02] [0x00] [0xB0] [0x0B] [stop] // write the data that will be converted to SPI

[start] [0x08] [w] [0x01] [0x55] [0x8F] [stop] // Set Memory Table to “Addr to Data” mode

[start] [0x08] [w] [0x00] [0x61] [0x07] [stop] // Set CS and write signal to 1, remove reset signal from shift registers.

Reading SPI data

[start] [0x08] [w] [0x00] [0x23] [start] [0x08] [r] [xxxxxxxx] [stop] // read SHR 0 data.

[start] [0x08] [w] [0x00] [0x25] [start] [0x08] [r] [xxxxxxxx] [stop] // read SHR 1 data.

[start] [0x08] [w] [0x00] [0xDA] [start] [0x08] [r] [xxxxxxxx] [stop] // read SHR 2 data.

After reading the SPI data, the data needs to be processed.

Let’s go through an example. Suppose the SPI device sends the following words: 0x02, 0x80, 0xFE, and 0x50, which are captured in the shift registers. When this data is read, we get the following results: from SHR0 (address 0x23), we read 0xE5; from SHR2 (address 0x25), 0x0F; and from SHR2 (address 0xDA), 0x28. As we can see, the SPI data is split between registers: the last 4 bits of the first SPI word (0x2) are written to the first 4 bits of SHR0, while the next 4 bits contain the first 4 bits of the next word (0x08). To retrieve the original SPI data, we need to merge the shift register data and then split it. Figure 6 illustrates the merged data, with the expected SPI byte highlighted in red.

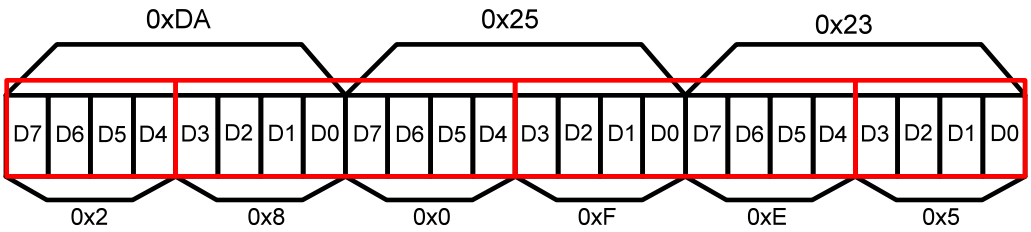
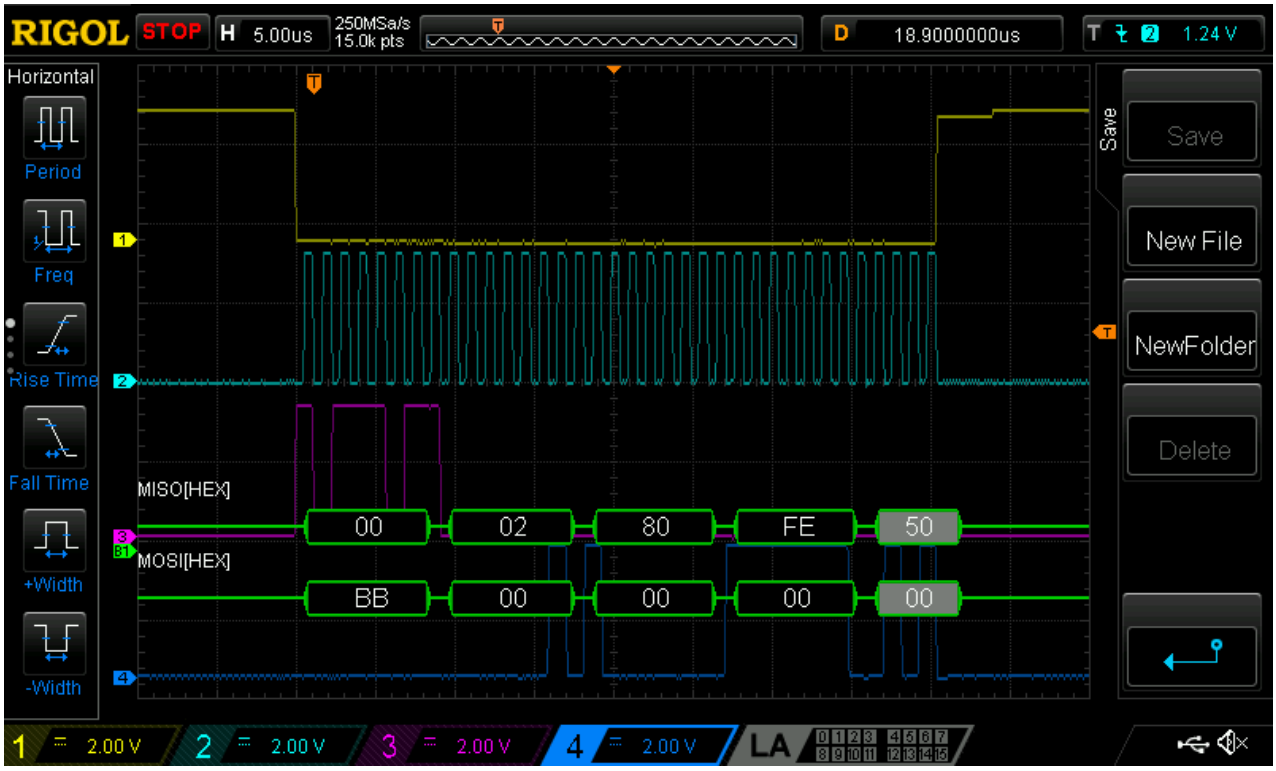


Figure 6. Merged data read via SPI data.

3. Results

Channel 1 (yellow/top line) – PIN# 7 (nCS).
Channel 2 (cyan line) – PIN# 6 (CLK).
Channel 3 (purple line) – PIN# 11 (MOSI).
Channel 4 (blue line) – PIN# 3 (MISO).

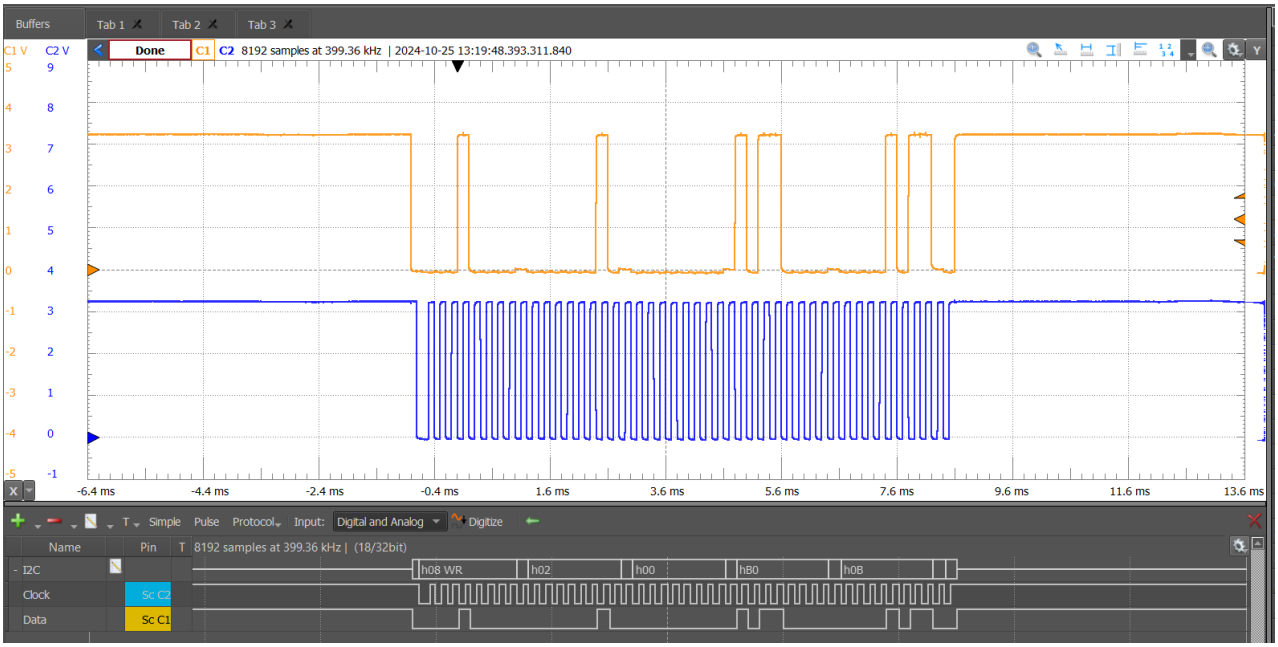
1. Design functionality. The process showing communication with the SPI device.



Channel 1 (orange/top line) – PIN# 4 (SDA).

Channel 2 (blue line) – PIN# 5 (SCL).

2. Writing data to the SPI device address via I²C.



3. Reading data from the SPI device through I²C. The data is stored in SHR0.

4. Reading data from the SPI device through I²C. The data is stored in SHR1.

5. Reading data from the SPI device through I²C. The data is stored in SHR2.

4. Conclusion

Overall, the SLG47011V offers a robust and flexible solution for I²C to SPI conversion, with its comprehensive set of features designed to ensure reliable and efficient data transfer. Its ability to store substantial amounts of data, coupled with flexible timing adjustments and accurate data conversion, makes it an ideal choice for applications demanding high-performance communication between I²C and SPI interfaces.

5. Revision History

Revision	Date	Description
1.00	May 20, 2025	Initial release.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.