AN1285: RS9116W SPI Protocol Application Note

Version 1.2

10/12/2020

Table of Contents

1 About

This document provides information on the Hardware Design and Software configurations for SPI communications.

2 Introduction

Serial peripheral interface (SPI) is one of the most widely used interfaces between microcontrollers and peripheral ICs, such as sensors, ADCs, DACs, Shift Registers, SRAM, and others. Every SPI system consists of one master and one or more slaves, where a master initiates the communication by asserting the CSN line. When a slave device is selected, the master starts clocking out the data through the MOSI line and receives the data through the MISO line. The master sends and receives one bit for every clock edge. One byte can be exchanged in eight clock cycles. The master finishes communication by de-asserting the CSN line.

The SPI is a protocol without an acknowledgment mechanism for checking received or sent data. For safe communication, flow control can be implemented in the communications protocol at a higher level.

2.1 Features

- Full duplex communication in the default version of this protocol.
- Push-pull drivers (as opposed to open-drain) provide good signal integrity and high speeds.
- Higher throughput than I²C or SMBus. Not limited to any maximum clock speed, enabling potentially high speeds.
- Complete protocol flexibility for the bits transferred.
- Extremely simple hardware interfacing.
- Uses only four pins on IC packages, and wires in board layouts or connectors, fewer than in parallel interfaces.
- At most, one unique bus signal per device (chip select); all others are shared.
- Signals are unidirectional allowing for easy galvanic isolation.
- Simple software implementation.
- SPI Slave Interface supports 8-bit and 32-bit data granularity.
- It also supports the gated mode of SPI clock, and the Low, High and Ultra high-frequency modes.

3 Prerequisites

1. Windows PC
2. MCU/Host with SPI interface. (eg: STM32)
3. SPI Header (Recommended to use the cable length not more than 2 inches)
4. Any Logic analyzer for analyzing the data lines (eg: salae)
5. IDE to create Application for HOST (eg: Keil, cubeIDE)
6. Silicon Labs EVK with power cable

4 Terminology

SPI - Serial Peripheral Interface

MOSI - Master Out - Slave In

MISO - Master In - Slave Out

CSN - CHIP SELECT

PC - Personal Computer

IDE - Integrated Development Environment

5 Topic: SPI App Note for Hardware and Software Configurations

5.1 Description

The SPI slave interface is supported only in WiSeConnect™ mode. RS9116 detects the host interface automatically after connecting to respective host controllers like SDIO, SPI, UART, USB and USB-CDC. SPI interface is detected through the hardware packet exchanges. Below are the signal descriptions. For more details about the pin names and descriptions, please refer to the datasheet.

Signal Name	Supply Domain	Direction	Initial State (Power-up, Active Reset) & Sleep state	Description[1,2,3,4] (All signals are in Default states unless otherwise mentioned)
SPI_CLK	SDIO_IO_VDD	In	High-Z	Serial Clock Input
SPI_CSN		In	High-Z	Active-low Chip Select signal initiated by the master to select a Slave device.
SPI_MOSI		In	High-Z	Master-Out-Slave-In signal for data transfer from Master to Slave
SPI_MISO		Out	High-Z	Master-In-Slave-Out signal for data transfer from Slave to Master
SPI_INTR		Out	High-Z	Interrupt signal to Master for indicating data available with the Slave device. Upon interrupt, Master has to initiate SPI transaction and read the available data on the SPI_MISO line. For more details about this signal, read below.
SPI_ERR_INTR		Out	Initial state: Pull-up Sleep state: High-Z	This signal is not available in the Default state. Check its availability in the Software. If SPI core logic within the device has gone to a state where it is not able to recover and process SPI transactions from the external Master, then SPI_ERR_INTR is asserted to the external Master about this status. It is an active high output signal. Once this signal is asserted by the devices, then the external host must initialize SPI and start the transactions again.

Note

1. **"Default"** state refers to the state of the device after initial boot loading and firmware loading is complete.
2. **"Sleep"** state refers to the state of the device after entering Sleep state which is indicated by Active-High "SLEEP_IND_FROM_DEV" signal.
3. Please refer to the **"RS9116W SAPI Programming Reference Manual"** for software programming information in embedded mode.

Ensure that the input signals, SPI_CSN and SPI_CLK are not floating when the device is powered up and reset is de-asserted. This can be done by ensuring that the host processor configures its signals (outputs) before de-asserting the reset. SPI_INTR is the interrupt signal driven by the slave device. This signal may be configured as Active-high or Active-low. If it is active-high, an external pull-down resistor may be required. If it is active-low, an external pull-up resistor may be required. This resistor can be avoided if the following action needs to be carried out in the host processor.

1. To use the signal in the Active-high or Active-low mode, ensure that, during the power-up of the device, the Interrupt is disabled in the Host processor before de-asserting the reset. After de-asserting the reset, the Interrupt needs to be enabled only after the SPI initialization is done and the Interrupt mode is programmed to either Active-high or Active-low mode as required.
2. The Host processor needs to disable the interrupt before the ULP Sleep mode is entered and enable it after the SPI interface is reinitialized upon wakeup from ULP Sleep.

SPI modes

There are 4 SPI modes as shown in the below table.

SPI Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)	SPI Mode description (Driven by source, sampled by Destination)	
			Source	**Destination**
0 *	0	0	Data-driven on the rising edge	Data sampled on falling edge
1	0	1	Data-driven on falling edge	Data Sampled on the rising edge
2	1	0	Data-driven on falling edge	Data Sampled on the rising edge
3 *	1	1	Data-driven on the rising edge	Data sampled on falling edge

*** Note: RS9116 supports Mode-0 & Mode-3 only in the above.**

As of now by default, we are supporting only Mode 0, and there is no API available for changing modes.

The following are the example diagrams for Mode-0 and Mode-3. The data is shown on the MOSI and MISO lines. The start and end of transmission are indicated by the dotted green line. The dotted orange line indicates the data-driven by the source, and the dotted blue line indicates the data sampled by the destination.

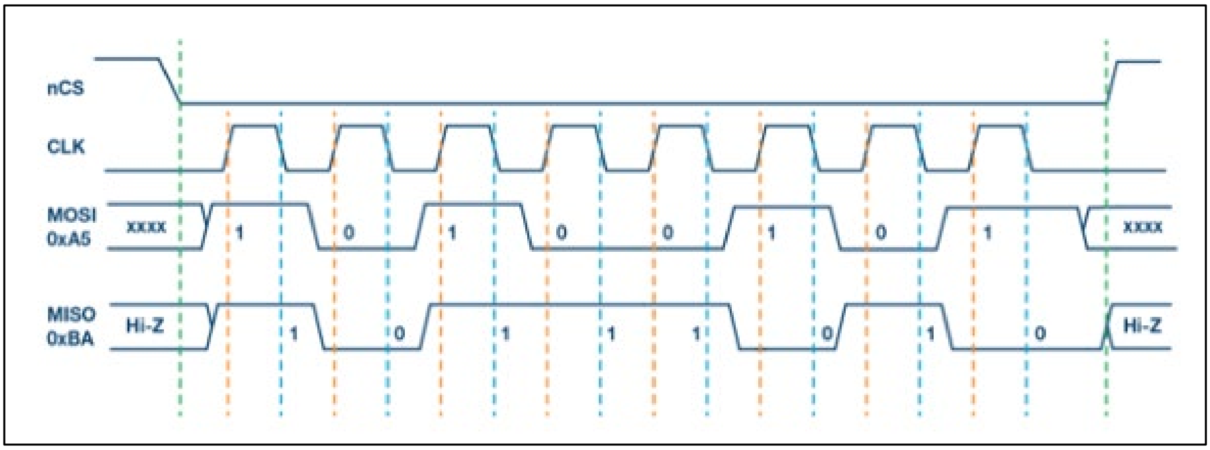


Figure 1 - SPI Mode-0

In Mode-0, clock polarity is ‘0’ which indicates that the idle state of the clock signal is low. The clock phase in this mode is ‘0’. Data transmission occurs during the rising edge of the clock.

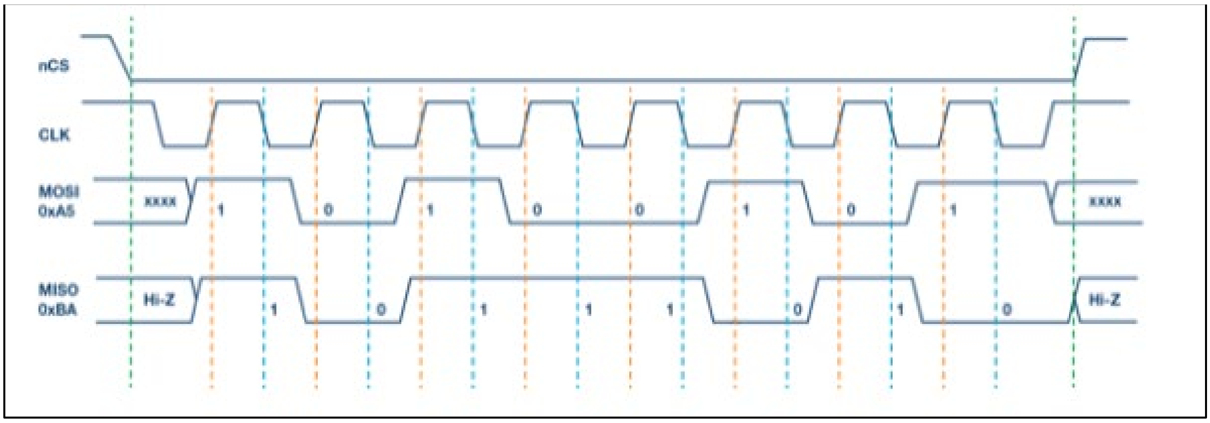


Figure 2 - SPI Mode-3

In this mode, the clock polarity is ‘1’ which indicates the idle state of the clock signal is high. The clock phase in this mode is 1. Data transmission occurs during the rising edge of the clock.

> **Note**
>
> **For more information on SPEC please refer to the respective Module Datasheet.**

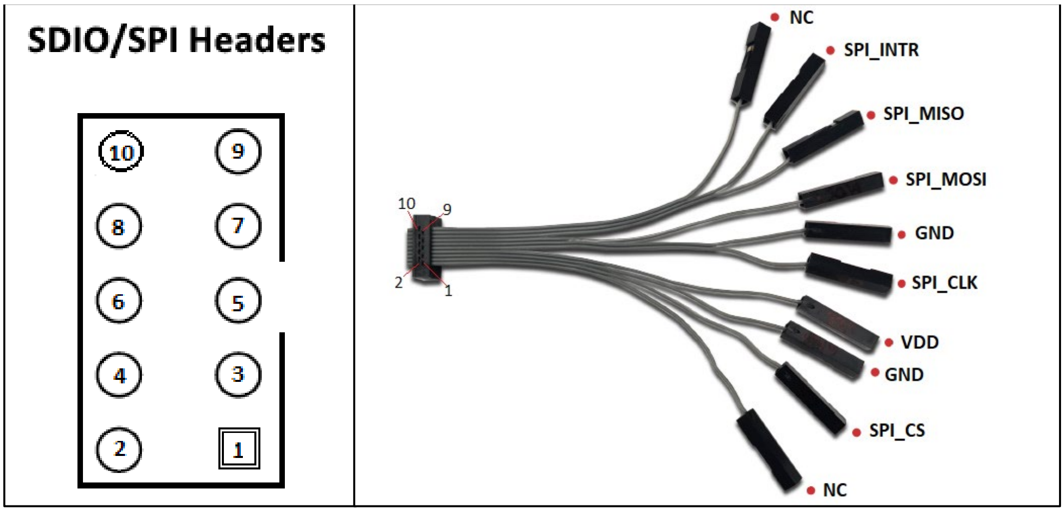
5.2 Use Case

This use case was executed with Silicon Labs EVK and third party host MCU STM32. Hence, before running the use case, we need to have basic information about hardware and SPI connections.

5.3 Hardware Setup

a. Connect the Micro A/B-type USB cable between a USB port of a PC/Laptop and the micro-USB port labeled POWER on the EVB.

b. Connect the 10-pin header of the SPI Adaptor Cable to the EVB. Connect the other wires of this connector to the SPI signals of a Host MCU platform. The details of the Header are given below:



Pin Number	Pin Name	Direction	Description
1	NC	-	This pin must be left open.
2	SPI_CS	Input	SPI slave select from host (active low)
3	GND	-	Ground
4	VDD	-	Supply voltage
5	SPI_CLK	Input	Serial clock in from the host. The clock can be up to80 MHz
6	GND	-	Ground
7	SPI_MOSI	Input	SPI data input
8	SPI_MISO	Output	SPI data output
9	SPI_INTR	Output	Active high-level triggered interrupt, used in SPI mode. The interrupt is raised by the EVB to indicate there is data to be read by the Host.
10	NC	-	No Connect

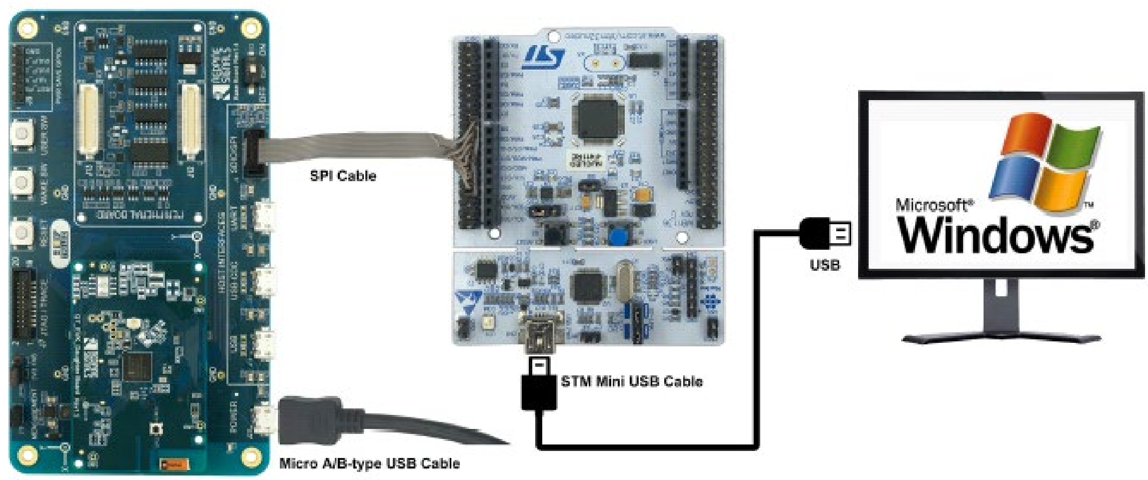


Figure 3 - EVK-STM32 Interface

SPI connection between Silicon Labs EVK and STM32 Board :

7	4	VCC
9	3	GND
11	5	CLK
13	8	MISO
15	7	MOSI
17	2	CSN
21	9	INT

14	10	RESET

Note

External reset control from MCU is always recommended to avoid Device Init failure.

Note

It is not mandatory to use Salae Logic analyzer, any logic analyzer can be used for capturing the data line info.

5.4 Description

Memory read command with SPI to read the HOST_INTF_REG_OUT register after power-up.
The following are captures of the SPI transfers after a module hardware reset.

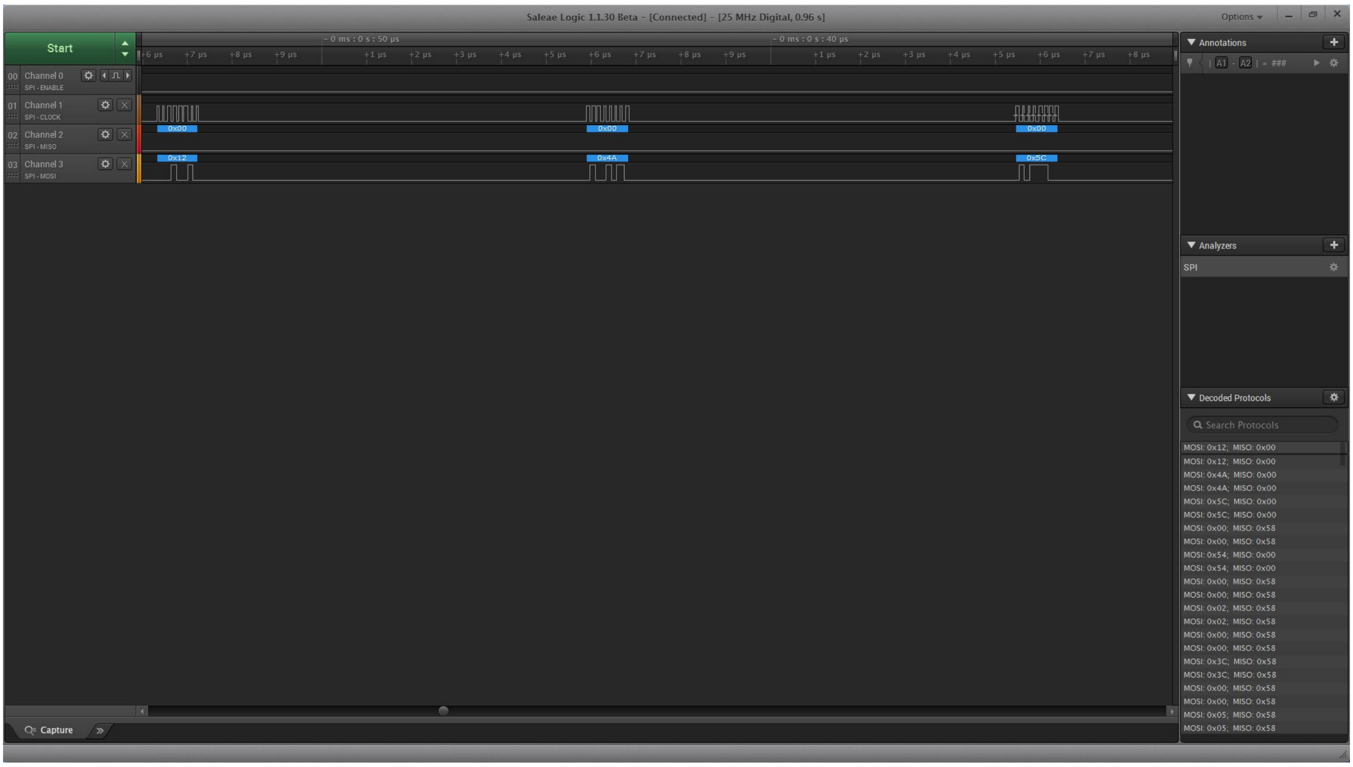
```
***************
TX 12 4A 5C 00
RX 00 00 00 58 // successful SPI initialization

TX 54 00 04 00 3C 00 05 41 00 00 00 00 00 00
RX 00 58 58 58 58 58 58 78 58 55 00 00 00 00 // memory read command
****************
```

There are three issues:
1) The module response includes 0x78. This is undocumented behavior.
2) The data following the start token (0x55) is all zero.
3) SPI Frequency in logic analyzer tool - 5MHz

6 Execution Steps

SPI Init was successful, but when we try to perform a memory read command with SPI to read the HOST_INTF_REG_OUT register, the module response includes 0x78.

- As part of troubleshooting try increasing the frequency to 25 MHz in the logic analyzer**.**
- Once the hardware setup is ready (connecting EVK with STM32, SALAE analyzer with STM32 SPI pins, start the Logic analyzer tools and make all the settings which are required (as per general settings of Logic analyzer tools). Then, compile the code in IDE and flash the code in the host. After that run the code and check the captures in 4 data lines on the logic analyzer.
- While setting the logic analyzer tool, set the clock frequency. If the SPI frequency is 25 MHz, then run the logic analyzer in 50 MHz frequency.
- Then, check whether the SPI initialization is successful or not,. if SPI initialization is successful, then the output will be as follows.
- Also, see the SPI initialization image below:
  TX 12 4A 5C 00
  RX 00 00 00 58 // successful SPI initialization



Then analyze the next memory read command with SPI to read the HOST_INTF_REG_OUT register.
TX 54 00 04 00 3C 00 05 41 00 00 00 00 00 00
RX 00 58 58 58 58 58 58 **|58|** 58 55 00 00 00 00 // memory read command (In bold we have got 58 instead of 78). (It is a kind of troubleshooting, by changing the logic analyzer to 25mhz)

7 Expected Result

In the above-mentioned use case, the logic analyzer frequency was configured as 5 MHz. If the frequency increases to 25 MHz based on clock changes, then we can achieve the expected result as **58.**

8 Recommendations Based on Software Configurations

1. The interrupt from the module is active high, and the host has to be configured to interrupt in the level-triggered mode.
2. It is recommended to port the external interrupt GPIO Pin for interrupt status in the **SPI HAL** layer.
    a. In order to configure soft reset, you need to map GPIO out pins of the host to the reset_ext in GPIO header.
    b. You need to send the reset sequence to the module in the function rsi_bl_module_power_cycle().
3. The following are the possible reasons for SPI busy
    a. A command is sent before reading the complete response to the last command.
    b. A received packet is not completely read but the next send command is being sent.
    c. The packet intended to be sent is not sent completely.
    d. A glitch in SPI lines.
4. When evaluating the EVK, ensure the power is provided using the POWER port on the EVK.
5. While porting MCU HAL, ensure the data which is sent to MCU HAL in SPI Transfer is placed in a buffer and its address is sent.

9 Summary

SPI Communication behavior is dependent on Hardware and Software. Based on the above Hardware Design, and use case for Software Configuration, and analysis of ISSUE, a basic check can be done. HOST and RS9116 WSC modules should be properly interfaced to avoid data loss.

10 References and Related Documentation

Refer to "**UG454: RS9116W with STM32 User's Guide**" from https://docs.silabs.com/rs9116.

11 Troubleshooting

11.1 Hardware Perspective

Follow the below guidelines for any issues faced on-board. Use the latest datasheet as a reference for all the below.

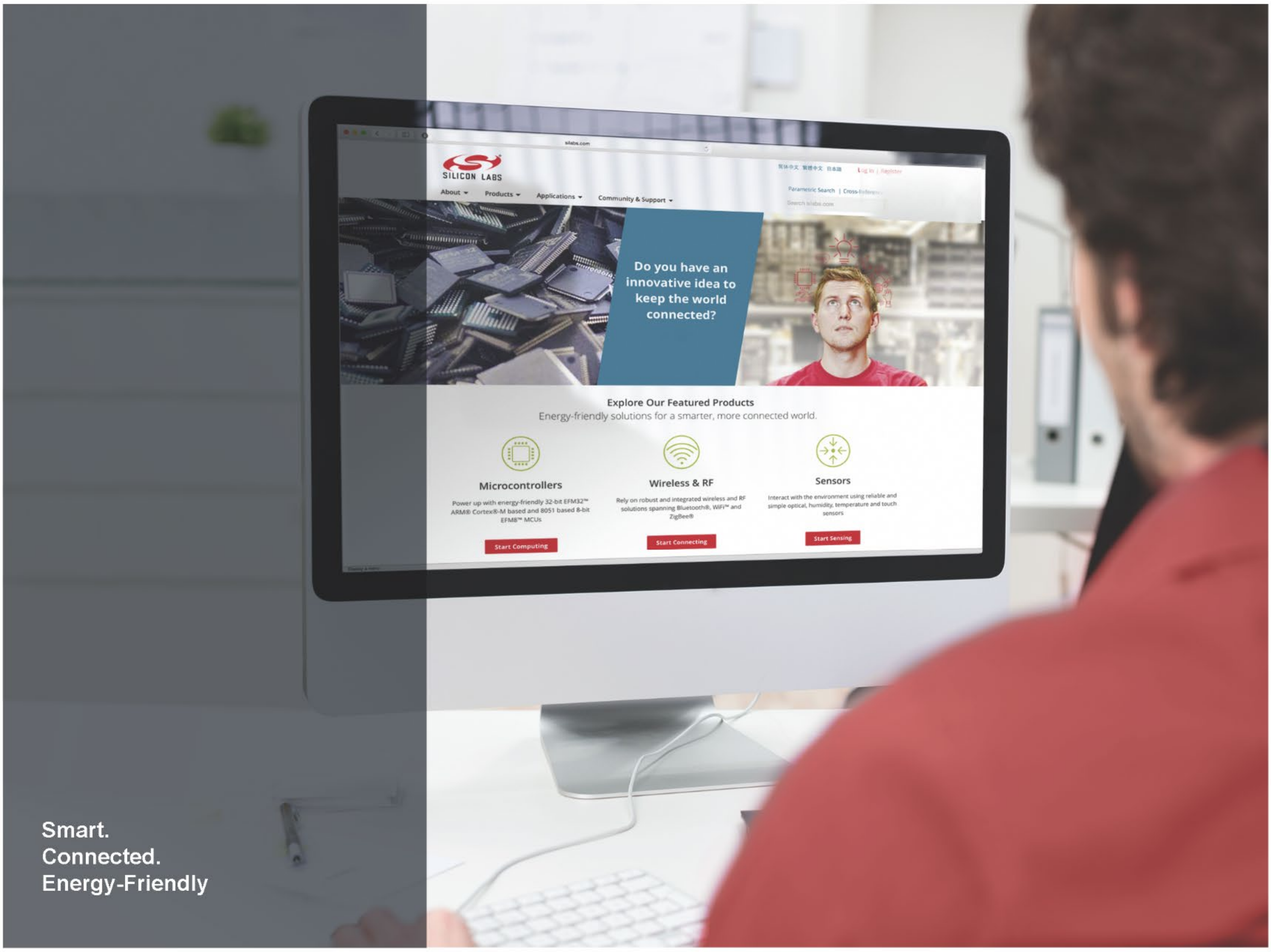
- Check all input supply voltages and ensure they are meeting the specifications.
- Check all output voltages and ensure they are meeting the specifications.
- Ensure the Power-up sequence (Input power supply, POC_IN, RESET_N) requirements are met.

11.2 Software Perspective

- Mode usage across the application is as configured while initializing SPI.
- The clock at which the SPI interface is working.
- SPI Pins connections from MCU to an external device(Module/chip).
- Check the SPI outputs on the Logic analyzer.

12 Revision History

Revision No.	Version No	Date	Changes
1	1.1	May, 2020	Initial version
2	1.2	Oct, 2020	Updated the SPI Header Images and added table for SPI Pins details Updated links and document names



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701

http://www.silabs.com