S2W Command Response SPI Interface Definition 2.0

Version 0.2

Date: 06-Nov-2014

# Introduction

This document describes the S2W SPI Interface Version 2.0 designed to scale up for high throughput.

To achieve the high throughput the interface would use

1. Higher clock rate (10Mbps when running @120Mhz from PLL)
2. DMA Access for the Data Transfer

The new design uses command-response handling between GS2000 (always slave) and any MCU (Master MCU) through SPI interface.

The Master issues commands for read/write and waits for the response. If the response indicates

* Success: the action is taken
* Failure: the action is deferred and retried after some time or dropped.

Operation sequence: Command -> Response -> Data phase (If response success) -> Command -> Response -> Data phase (If response success) ->.

The HI Format is used for the message exchange. Refer to Appendix A for the details on the format.

## Transferring data from MCU to GS2000

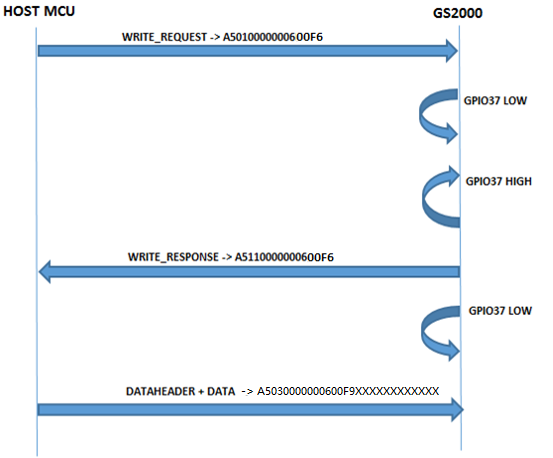


Figure : Transferring data from host to GS node

The step by step procedure involved in transferring data from HOST to GS node is as follows:

1. Host provides clock
2. Host sends a WRITE\_REQUEST to GS2000. It uses HI frame with,
   1. Class field -WRITE\_REQUEST,
   2. Length as the size of data to be transferred from host to GS2000

The maximum length allowed would be (2048 - 8 - 8 = 2032). The maximum DMA size on GS2000 is 1024. We make allowance of 8 bytes for the header of the HI frame carrying the data and 8 bytes for the Write Response HI frame.

1. After receiving WRITE\_REQUEST, GS2000 will pull downGPIO 37 (only if it is HIGH). This is to avoid a race condition when the GS2000 as well as host wants to send data.

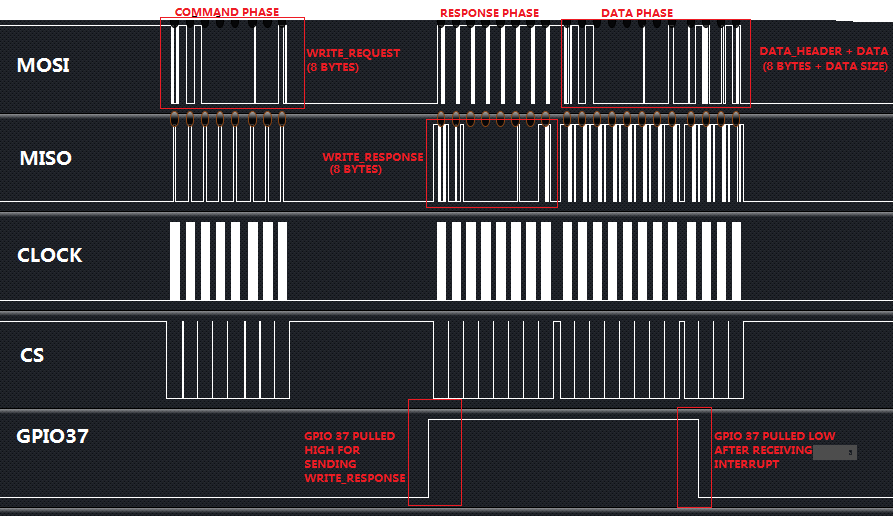
**NOTE:** If a race condition occurs when GS2000 has data to send to HOST and at the same time HOST also wants to send data to GS2000, first GS200 will respond for the WRITE\_REQUEST and will provide proper responses. So after HOST MCU WRITE is finished and MCU receives proper response, GPIO37 will be again made HIGH as GS200 has some pending data which MCU did not read. Now HOST MCU should issue READ\_REQUEST and read out the data present in GS2000.

1. GS2000 will pull GPIO 37 high to inform the host when it is ready with the WRITE\_RESPONSE.

**NOTE:** The host will wait for the GPIO 37 transition from Low to High before applying the clock.In case there is pending data to transfer from GS2000 to MCU, the GS2000 will indicate the same in the additional info in the response which the host can process.

1. Host will provide the clock to read WRITE\_RESPONSE
2. GS2000 will send WRITE\_RESPONSE to the host. It will use HI frame with,
   1. Class field – WRITE\_RESPONSE
   2. Length as the size of the data the GS2000 will be able to receive. This is 0 if GS2000 is unable to receive (flow control)
   3. The status field is WRITE\_OK if it is ready to receive the data and WRITE\_NOT\_OK if it is not ready to receive the data
   4. When the WRITE\_RESPONSE is sent, the GS2000 will trigger an interrupt to pull down Ready to Send Signal (GPIO low).
3. Host will send the data and data header using HI Frame with
   1. Class field – DATA\_FROM\_HOST
   2. Length as the size of the data (this length must be less than or equal to the length mentioned in the WRITE\_RESPONSE)
4. Once the entire process of WRITE is complete, MCUshould check the GPIO 37 for any pending data from GS2000 and GPIO is
   1. LOW, will stop the clock
   2. HIGH, will start the procedure for READ once it is ready to receive.

#### Timing Diagram for Sending Data from Host toGS Node



## Transferring data from GS2000 to the MCU

The following figure shows the procedure involved in transferring data from GS node to host.

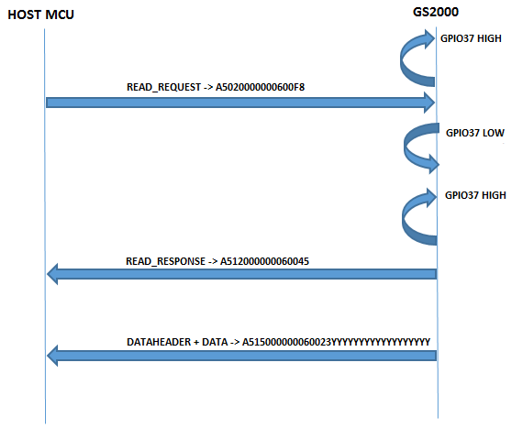


Figure : Transferring data from GS node to host

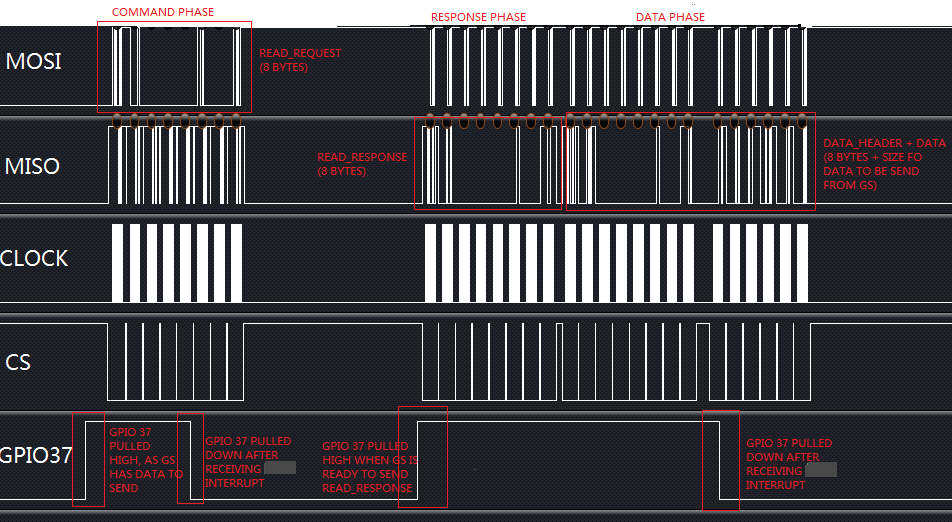
The step by step procedure involved in transferring data from GS node to host is as follows:

1. GS2000 will pull up GPIO 37 (HIGH) when there is data to send from GS2000 to the host.
2. Host provides clock when it is willing to receive
3. Host sends the command (READ\_REQUEST) to GS2000. It will use HI frame with,
   1. Class field -READ\_REQUEST,
   2. Length is the size of data that host is willing to receive from GS2000

The maximum length allowed would be (2048 - 8 - 8 = 2032). The maximum DMA size on GS2000 is 1024 (can use multiple). We make allowance of 8 bytes for the header of the HI frame carrying the data and 8 bytes for the Write Response HI frame.

1. While receiving READ\_REQUEST, the GS2000 will trigger an interrupt to pull down GPIO 37(low).
2. GS2000 will pull the GPIO 37 high to inform the host when it is ready with the response.
3. Host will provide the clock for reading READ\_RESPONSE
4. GS2000 will send a response to the host. It will use HI frame with,
   1. Class field – READ\_RESPONSE
   2. Length as the size of the data the GS2000 will be transmitting.
   3. The status field is READ\_OK if it is ready to receive the data and READ\_NOT\_OK if it is not ready to receive the data
   4. When the READ\_RESPONSE is sent, the GS2000 will trigger an interrupt to pull down Ready to Send Signal (GPIO low).
5. GS2000 will send the data header and data using HI Frame with,
   1. Class field – DATA
   2. Length as the size of the data (this length must be less than or equal to the length mentioned in the READ\_RESPONSE)
6. Host will stop the clock.

#### Timing Diagram for Sending Data from GS Node to Host



# Annexure

## 1.1 HI Frame Format [From Master Side]

All messages carried over the Host Interface have a common format. They are composedof a HI header, and parameters depending on the header. HI frames are composed, inaddition to the HI header and parameters, of a start delimiter and a HI HEADER checksum.

This format is defined in below Figure.

The Start-of-frame delimiter is the single-byte value 0xA5, used to ensure synchronizationat the frame level. The driver starts the reception process when it recognizes the delimiter.

The length of the delimiter has been reduced to 1 byte to avoid alignment problems when waiting for the start element. However, no provisions are made to ensure that the subsequent data stream does not contain a byte with value 0xA5, so it is possible for the driver to mistake a data byte for a delimiter. Therefore, a header checksum has been added to ensure correct synchronization. A single checksum byte is used, computed as the 1’s complement of the 8-bit long (modulo-256) sum of all the bytes of the HI HEADER (not including the Start delimiter). Note that each byte is independently added to the sum, as an integer between 0 and 255, without regard for its significance within its own data field.

HI Header

HI Parameters

Class

1 byte

Reserved

1 byte

Additional Info

2 bytes

Length

2 bytes (11 bits)

data whose format is depending on the class of the HI frame

0 to 1024 bytes

Start of Frame

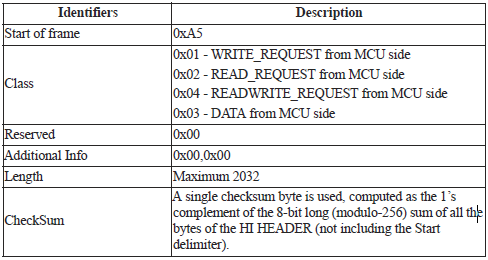
HI Header

Checksum

HI Parameters

**Figure HI Frame Format (From Host Side)**

The format of HI Parameters field is determined by the service class. The service class ofeach frame is signaled by the value of the first field. Available service class identifiers (seebelow Table).



## 1.2 HI Frame Response [From GS2000 Side]

HI Header

HI Parameters

Class

1 byte

Reserved

1 byte

Additional Info

2 bytes

Length

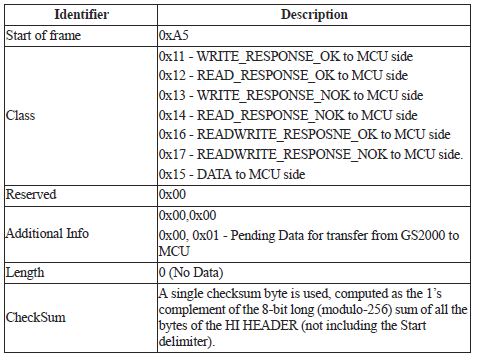
2 bytes (11 bits)

Start of Frame

HI Header

Checksum

HI Parameters

.

.