HCI COMMUNICATION OVER SPI – USE CASE OF NVM PROGRAMMING

Product Family: **EM BLE System-On-Chip solution**

Part Number: EM9305
Keywords: SPI, HCI, NVM writing

PURPOSE

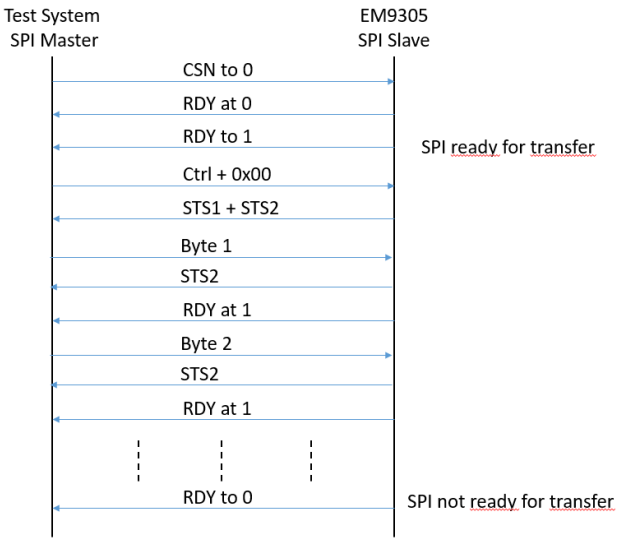
The purpose of this application note is to share information related to EM9305DVK interfacing with EM9305 device across HCI using SPI transport interface. The implementation details shared are focusing on NVM programming use case and associated performance level.

SCOPE

The EM9305 described in this application note has the ROM version 1.0 and the EM9305DVK Firmware version is 1.2.0 .
Above DVK configuration is available from EM9305 SDK.

EM9305 SPI Generic diagram

There are different ways to address the EM9305 via SPI. The most generic way is to use the RDY signal to know when the device is ready for a communication. Once the SPI FIFO RX is full, the RDY signal will be kept at "0".

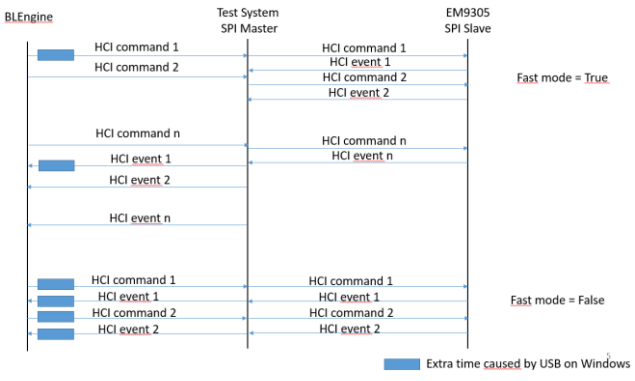


EM9305DVK SPI implementation

The EM9305DVK uses a slightly more complex way to communicate via SPI. To acknowledge the receipt of a command, the EM9305 sends back a status byte STS2. This byte contains the space left in the SPI FIFO RX and allows the DVK to know when it is full without the need to check the RDY signal. This flow enables to have a DMA transfer initiated by the SPI master as it knows the amount of data that can be received by the slave.

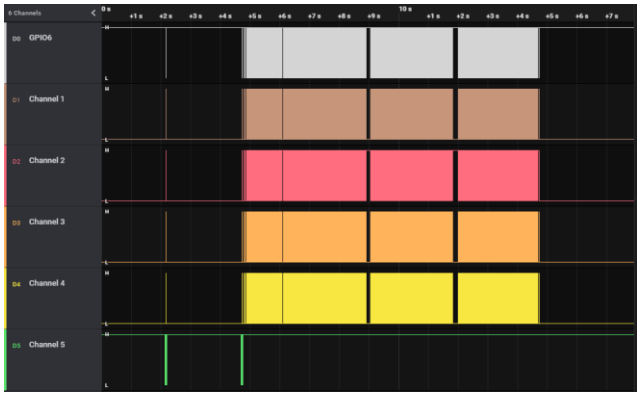
EM9305DVK Fast and Slow mode

When using the EM9305DVK and its associated software from the SDK, the user has the choice to use either a fast or a slow mode. In slow mode, every HCI event is brought back up to the scripting language before the next command is sent, while in fast mode, the HCI commands are chained as fast as possible while the events are kept at the EM9305DVK level until the end of the script. To be noted that the HCI commands are spread over multiple SPI events.



Below the traces captured by a logic analyser for fast and slow mode, when uploading the full NVM content through HCI commands over SPI. The duration indicated are the length of time to make this operation. The advantage of using the fast mode is clear.

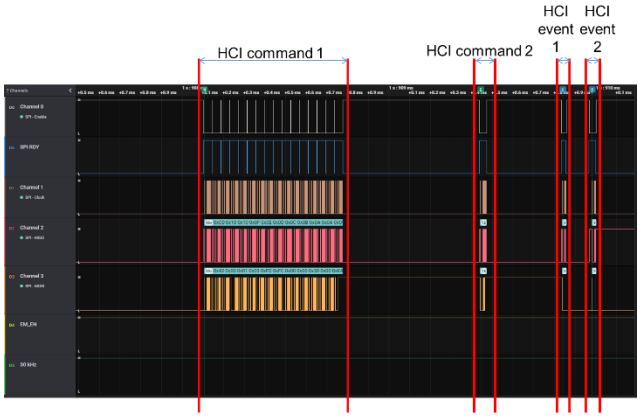
Fast mode: 10 seconds



Slow mode: 84 seconds

HCI over SPI "particular" case

In some particular cases, when sending HCI commands over SPI in fast mode, it can appear that two HCI events are following each other, rather than their respective commands. This behaviour is normal and is the result of the second command being sent while the first command is still being processed and the second command being small enough to fit entirely in the SPI FIFO RX



The observed SPI traffic matches the flow described below

- HCI command 1 is sent and processed, emptying the SPI FIFO RX but not the HCI "receive" buffer
- HCI command 2 is sent and kept in the SPI FIFO RX
- HCI command 1 processing is finished, emptying the HCI "receive" buffer
- HCI event 1 is sent, emptying the HCI "send" buffer
- HCI command 2 is processed, transferring the message from the SPI FIFO RX and emptying the HCI "receive" buffer
- HCI event 2 is sent, emptying the HCI "send" buffer

The first command is sent, and is being processed by the EM9305. While this "large" command is processed by the system, the SPI FIFO RX is emptied, freeing space for the next communication, but as the processing isn't finished yet, the HCI "receive" buffer isn't emptied and no event is ready to be sent. The DVK tries to send the second command in the meantime, and as the command is small and the SPI FIFO RX has free space, it is successful. Still this command isn't processed yet as the processing of the first command isn't finished yet. The processing of the first command is then completed, and the first event is sent. The processing of the second command is then started and completed and the second event is sent.

NVM programming using HCI commands over SPI

Programming time considerations

One of the principal use case of using the HCI commands over the SPI bus is to program the NVM. It uses consecutive write to NVM commands and should make the best possible usage of the SPI FIFO RX.

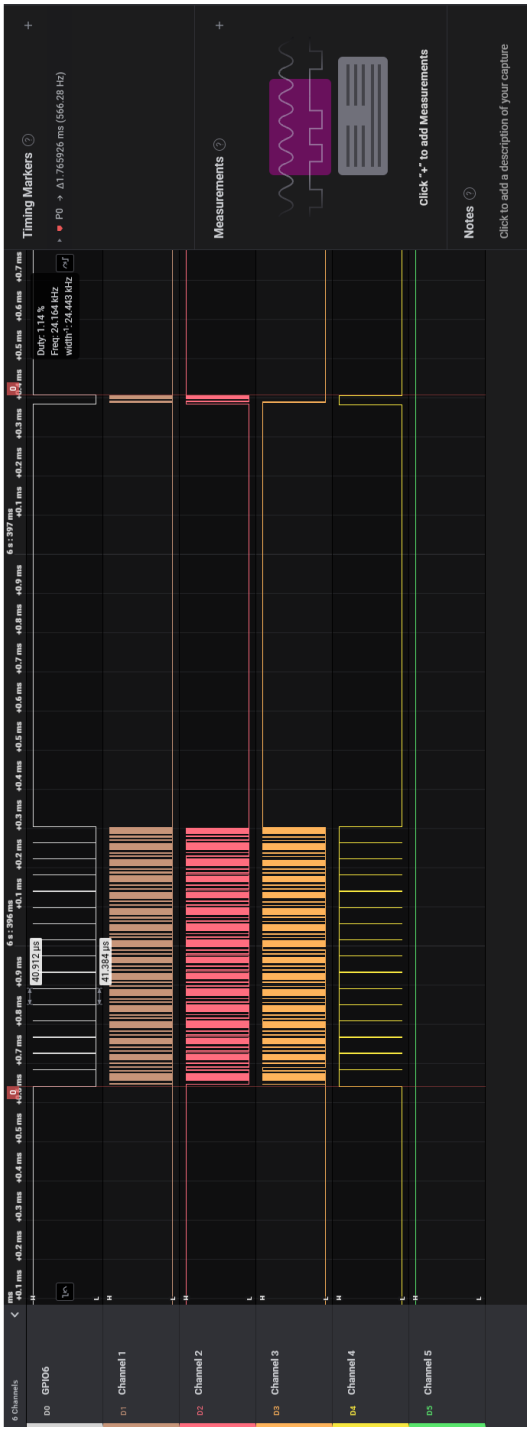
The times below are the times to program the full 512kB of NVM in different conditions. SPI is running at 7MHz on the DVK

Programming tool	Comments	time
Production tester	2.5MHz SPI	~15s
Production tester	Extrapolation at 8MHz	~5s
DVK	Fast mode	10s
DVK	Slow mode	84s
DVK	Extrapolation removing pause time between 248 useful Bytes writing bursts	**3.73s**

To minimize the writing time, the maximum length for an HCI command of 256bytes should be used. This would allow 248 bytes to be written in the memory and reduces the protocol overhead to its minimum.

Below, the times to physically write 248 Bytes in the NVM.

Programming tool	Comments	time
Theoretical	Best case - Physical write time, no control software overhead	496us
Theoretical	Worst case- Physical write time, no control software overhead	992us
DVK	Time between the end of the SPI write and the SPI event	~1100us



Power estimation consideration

To write 8kB of NVM, on a module supplied at 1.5V in Voltage Multiplier mode, 723uJ is necessary. An average of 2.22mA is measured during 217ms

Recommendation

To minimize the writing time of the NVM, while keeping a programming flow suited to simple programmers, EM recommends to

- Send a “write” command to the memory with the largest possible number of bytes
- Wait for the “event” to acknowledge the “write” was performed correctly
- Minimize the timing between receiving the “event” and sending the following “write” command