August 09, 2018

Extended SPI Bus

Leonid Lobachev

Extended SPI bus

ABSTRACT

Serial peripheral interface (SPI) is a protocol that enables low-level machine communication between electronics subsystems, e.g., microcontrollers, peripherals, etc. In applying SPI to communications between devices of substantially differing performance characteristics, e.g., clock frequency, memory capacity, or response time, the original SPI has some disadvantages. For example, due to the streaming nature of SPI communication, errors are not detected; due to differing clock frequencies at two ends, synchronization is difficult; due to differing memory capacity at two ends, the buffers of the end with less memory overflow; etc.

This disclosure describes an extension to SPI that features synchronization signaling and packet-splitting abilities to enable devices of differing abilities to communicate via SPI. The extension also enables asynchronous transaction execution over the SPI bus and adds error detection capabilities, both features that enable larger packet sizes and higher transmission/transaction rates.

KEYWORDS

Serial peripheral interface; SPI bus; serial communications; asynchronous transaction; I2C; packet splitting; packet synchronization; extended SPI

BACKGROUND

Serial peripheral interface (SPI) is four-wire, full duplex, serial interface bus that enables low-level machine communications between electronics subsystems, e.g., microcontrollers, peripherals, etc. Originally developed for relatively low-bandwidth communication, SPI remains in widespread use today due to its simplicity and robustness. SPI is a streaming protocol, e.g., it has no error detection or correction capabilities. If a bit is received in error or not received at all,

it is simply dropped. These qualities were acceptable for the original use-cases for which SPI was developed.

There is however a growing case for SPI (or other serial protocols, e.g., I2C) to be applied in situations other than the original use cases. For example, SPI is an economic choice for communications between relatively powerful computational devices, e.g., systems-on-chip (SoCs), application specific integrated circuits (ASICs), CPUs, etc., and relatively low-powered devices, e.g., microcontrollers, peripherals, etc. In such cases, e.g., communication between an SoC and a microcontroller, very high bandwidth is not necessary; however, data integrity is important. Thus, SPI can be used advantageously in this situation instead of more sophisticated and expensive protocols.

In applying SPI for communications between devices of substantially differing performance characteristics, e.g., clock frequency, memory capacity, response time, etc. the original SPI has certain shortcomings. For example, due to the streaming nature of SPI communication, errors are not detected; due to differing clock frequencies at two ends, synchronization is difficult; due to differing memory capacity at two ends, the buffers of the end with less memory overflow; etc.

DESCRIPTION

This disclosure describes an extension to SPI that features synchronization signaling and packet-splitting abilities to enable devices of differing abilities to communicate via SPI. The extension also enables asynchronous transaction execution over the SPI bus and adds error detection capabilities, both features that enable larger packet sizes and higher transmission/transaction rates.

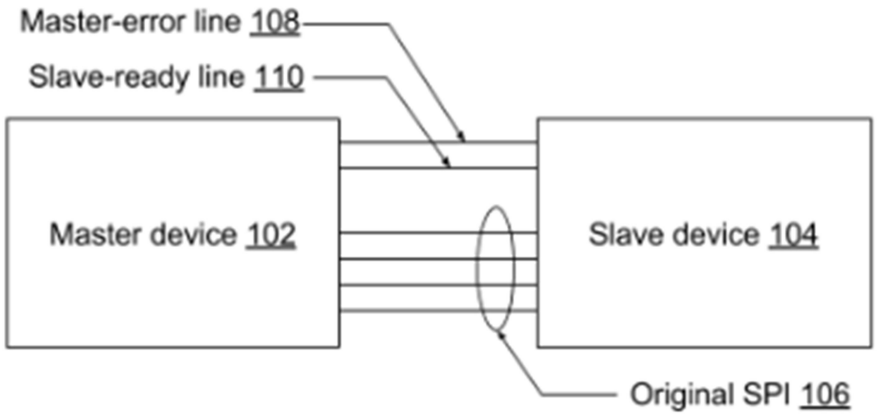

Fig. 1: Additional wires in extended SPI

In SPI, the bus is controlled by a master device that generates clocks on a serial-clock (SCK) line, selects a target slave device using a slave-select (SS) line, and simultaneously exchanges data in both directions using master-out-slave-in (MOSI) and the master-in-slave-out (MISO) lines. As explained before, SPI does not define a packet format, data integrity validation method, or synchronization mechanism.

This disclosure extends SPI by adding two interrupt lines between the master and slave devices. This is illustrated in Fig. 1, which shows original SPI lines (106) as well as the two additional lines, named master-error (ME) line (108) and slave-ready (SR) line (110) between master device (102) and slave device (104). The ME line is used by the master device to indicate the error status of a packet to the slave device. The SR line is used by the slave device to indicate to master device that it is ready to transmit or receive data. Typically, it is the slave device that requests a processing-time delay, e.g., the communication is controlled by the master device. If a master device needs a processing-time delay, it simply stops clocking the bus to get the requisite delay.

In a typical use case, at one end of the extended SPI is a relatively powerful System-on-a-Chip (SoC), e.g., with a clock frequency of several hundred megahertz, while at the other end is a relatively low-powered microcontroller, e.g., with clock frequency in the tens of megahertz. In this case, original SPI offers no reliable way to synchronize the two devices, nor can it accommodate the delay needed by the slower end to prepare for reception of data. Extended SPI, as described herein, uses the SR line to enable inter-device synchronization. Since the SR signal is under the control of the slower end, it provides adequate time to the slower end to prepare for receipt of data.

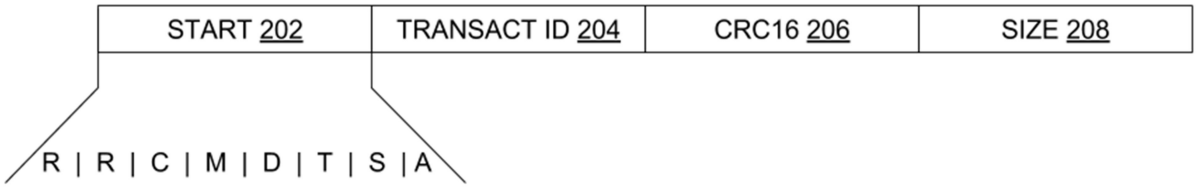

Fig. 2: Structure of packet header

Fig. 2 illustrates the structure of a packet header, per techniques of this disclosure. A packet header includes a number of fields, e.g., a start field (202), a transaction ID field (204), a cyclic redundancy check (error detection code) (206), and a size field (208). The start field (202) is one byte long and includes the following bits:

- R: Reserved bit (two R bits are shown in Fig. 2); set both to 0.
- C: Transaction completed bit.
- M: Master or slave bit; set to 1 for master-origin headers, 0 for slave-origin headers.
- D: Data follows bit; indicates data exchange to follow this packet header in the same direction.

- T: Transaction ID valid bit; set to 1 for master headers, set to 0 for slave headers with no transaction ID matching the one specified in master header. Slave aborts current exchange after sending such a header to master.
- S: Extended SPI protocol version supported bit.
- A: Acknowledgement bit; set to 1 for master packets, set to 0 or 1 for slave packets.

The transaction ID (204) is one byte long and serves a purpose similar to frame sequence number. The CRC16 field (206) is a two-byte error-detection code that protects the packet header. The size field (208) is a four-byte field used by the master to indicate the size of data packet about to follow the header, and by the slave to indicate maximum receivable packet size. If the size field in the response of a slave is smaller than the size field in the initial communication of the master, then the master breaks the (larger) packet into smaller sub-packets that are appropriate to the memory capacity of the slave. A relatively strong 16-bit CRC is used to protect the packet header, as the packet header comprises parameters of the data that follows it, and any error in receiving any packet header field can damage the subsequent data exchange. The fields of the packet header are designed to align to integral byte boundaries. Although the packet header has a 16-bit CRC, data packets might use CRCs of differing length, including stronger error-detection, e.g., 32-bit CRC, for longer data-packet sizes.

In a typical use case, one end of the (extended) SPI has sizeable memory capacity, e.g., several megabytes, while the other end has relatively low memory, e.g., in the range of kilobytes. The high-memory end initiates communication with a size field set, e.g., to 100 kilobytes. The low-memory end responds with a size field of, e.g., 4 kilobytes. The high-memory end breaks its 100-kilobyte packet into sub-packets that are no more than 4 kilobytes in size. The sub-packet length is not fixed and can vary over time. For example, if the low-memory end frees up

memory, it can signal that it is ready to receive data in larger, e.g., 16 kilobyte chunks. The splitting and reassembly of packets is transparent to a user or programmer of the extended SPI bus. For example, a programmer does not need to explicitly issue packet-splitting instructions. Rather, such packet-splitting requests are made automatically based on current memory availability.

The techniques of this disclosure enable transactions over the bus, e.g., multiple commands issued the master device can get queued at the slave device, and the commands do not have to complete synchronously or in order. There is provision for the master device to query slave device to find out if a response is ready. Likewise, there is provision for slave device to indicate receipt of command and status (complete/incomplete) of corresponding task. While awaiting completion of outstanding tasks, a master device can queue up additional transactions at the slave device. A representative use case is when the slave device is connected to an external device awaiting human user input. Another example is the execution of a computationally intensive command on a slave device while querying its thermal or power status.

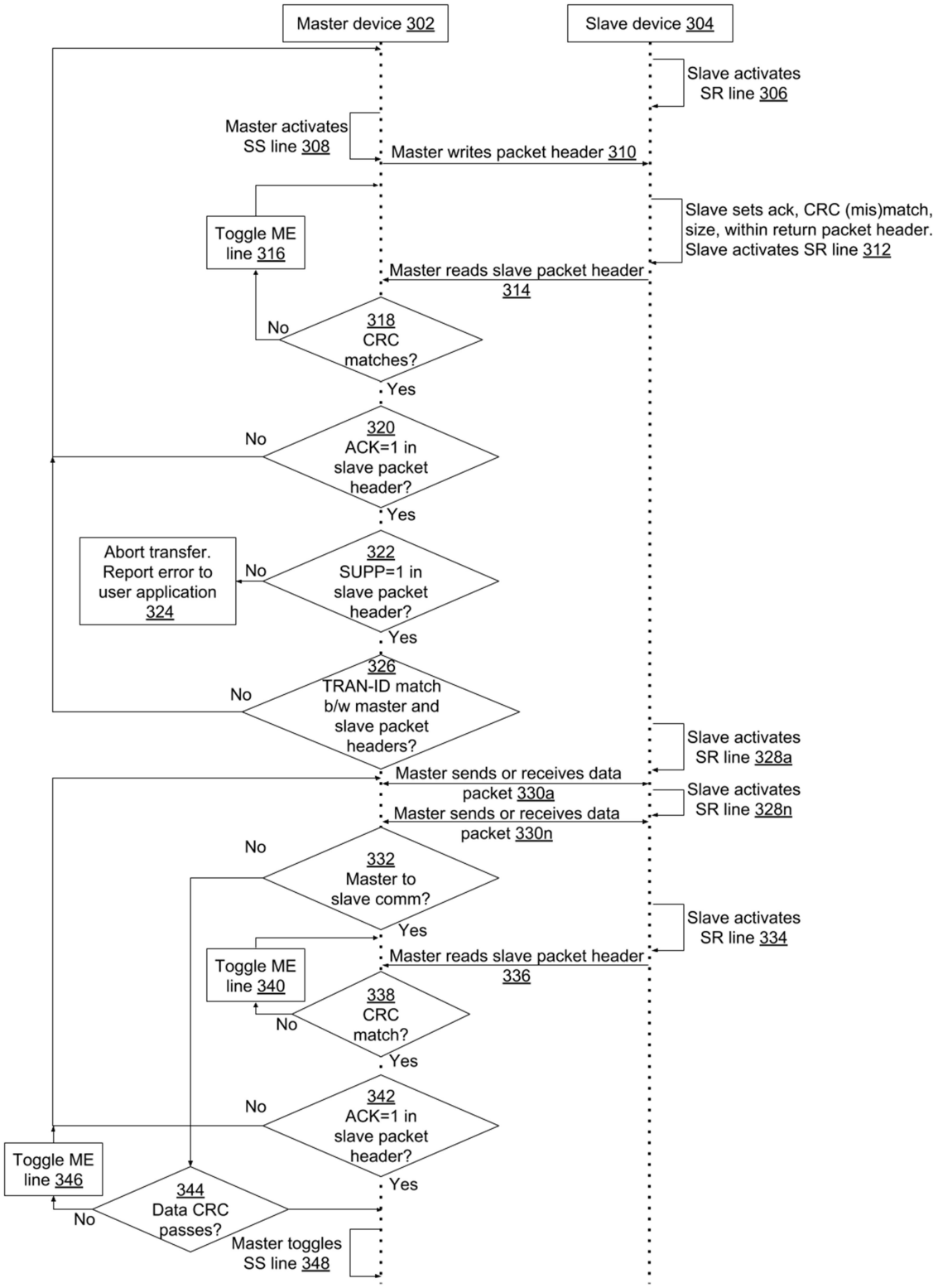

Fig. 3: Extended SPI

Fig. 3 illustrates an example of communications interchange between a master device (302) and a slave device (304), per techniques of this disclosure. Under extended SPI, packet exchange starts after the slave device is ready, indicated by the slave device activating the SR line (306). The master device activates the slave-select (SS) line (308) to indicate that it has data to transfer. As explained before, SR line activation, controlled by the slave device, enables synchronization between master and slave that have differing performance characteristics.

The master starts packet exchange by writing over the extended SPI bus a packet header (310). The packet header describes the direction and type of subsequent data transfer, the transaction ID and the transfer size, and is protected by CRC. In response, the slave device generates a return header packet comprising an acknowledgement bit, CRC match/mismatch, size, etc. As explained before, if the slave device sets in its response a size less than the size requested by the master device then the master device splits large packets into smaller packets of size specified by the slave device. The slave device activates the SR line (312) when ready. The master device reads the slave device packet header (314). The master device performs validation tests on received slave packet header, e.g.,

- CRC match (318): if this test fails, the master toggles the master-error (ME) line (316) and awaits a resend of packet header by the slave; if it passes, the master toggles the SS line to indicate successful reception.
- Acknowledgement test (320): if there is a NACK, e.g., acknowledgement bit set to 0, in the packet header sent by the slave, the master re-initiates packet exchange. A NACK event occurs when the CRC computed by the slave for the packet sent by the master does not match the CRC pattern embedded within the master packet.

- Extended SPI protocol version supported test (322): The reserved bits in the packet header (202) are nominally set to 0, but it is possible in a future revision of extended SPI to use these reserved bits for additional features, e.g., as CRC bits for each sub-packet of a larger (split) packet. The supported bit in the packet header (202) is used to signal such advanced versions of the extended SPI protocol, e.g., those that support CRC for split packets. If the protocol versions used by master and slave differ, e.g., as indicated by the supported bit, the master aborts transfer and reports an error to the user application (324).
- Transaction ID test (326): if the transaction ID fields don't match between slave and master packet headers, then the master generally reinitiates packet exchange. A special case, with transaction ID set to 0, is recognized as a non-error event as follows. A transaction ID set to 0 is used by the master device to poll the slave device to see if pending transactions, if any, are finished. In that case, a slave packet header might include a different transaction ID, e.g., the transaction ID of the completed transaction, which would not trigger packet exchange. If a pending transaction is signaled as becoming completed, it is followed by data packet exchange in either direction, similar to a regular transaction. The case of outstanding transactions is illustrated in Example 5 below.

If tests performed by the master device pass, then header exchange between master and slave is deemed to have successfully terminated, and the communication moves to data interchange. The master device waits for the slave device to activate the SR line (328a) and either transmits or receives data based on transfer direction from original master packet header (330a). If data is transmitted, it is protected by 32-bit CRC. As explained before, this data transfer may be conducted in sub-packets of size less than or equal to a size specified by the slave device. The

process of SR line activation (328a-n) followed by data transfer (330a-n) could be repeated multiple times in sequence if the data packet is split.

If the data transmission was from master to slave (332), the master device awaits the activation of the SR line (328) by the slave device and reads another slave packet header (336) to verify receipt of data by the slave. The master performs validation tests on the received slave packet header. If there are CRC mismatches (338), the master toggles the ME line and awaits re-transmit of the slave packet header. If the slave packet header includes a NACK (342), the master re-initiates data transfer. A NACK event occurs when the slave CRC check fails for the data packet. If validation tests pass, the master toggles the SS line to indicate successful data transfer (348).

If the data transmission was from slave to master (332), the master performs validation tests on the received data packet. If there are data packet CRC fails (344), or transaction ID mismatches between data and packet headers, the master toggles the ME line (346) and reinitiates data transfer. If the validation tests pass, the master toggles the SS line to indicate successful data transfer (348).

In this manner, extended SPI, as described herein, enables:

- the detection and recovery from physical bus transmission errors;
- the slave device to request the splitting of a transmission into multiple smaller transmissions commensurate with its current internal memory availability, e.g., due to SPI hardware FIFO or DMA engine limitations on memory;
- the slave device to indicate readiness using a dedicated line, such that the slave device has the necessary time to configure itself for communication, e.g., set up its internal SPI HW block, DMA engine, etc.;

- multiple transactions from master device to be executed simultaneously and asynchronously; etc.

Example 1: *Simple write transaction from master to slave*

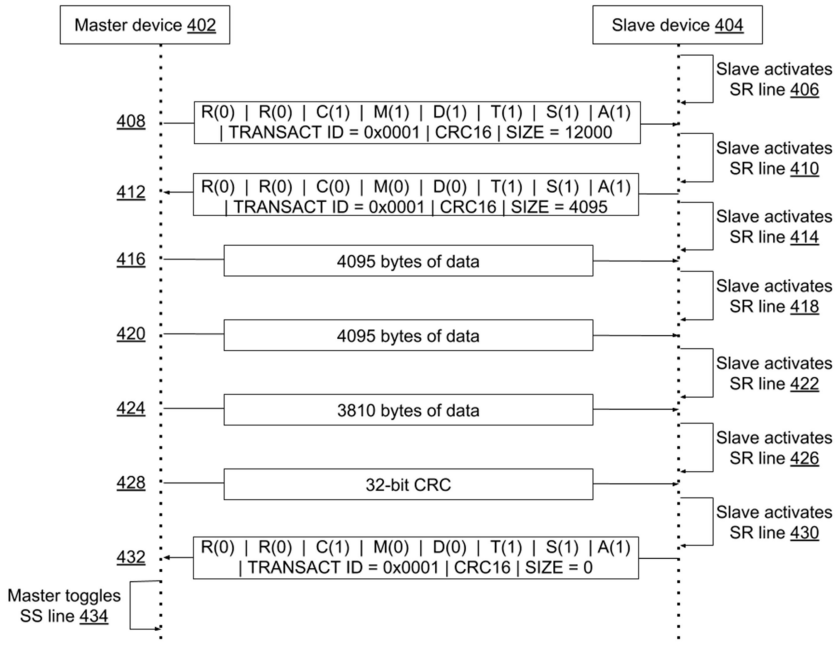

Fig. 4: Simple write transaction from master to slave

A simple write transaction from master device (402) to slave device (404) is illustrated in Fig. 4. The slave indicates readiness by activating the SR line (406). The master sends a packet header (408) with start-byte bits set as follows:

- R: reserved, 0;
- C: Transaction completed, 1;
- M: Master or slave bit, 1, since this is a master-origin header;
- D: Data follows bit, 1;
- T: Transaction valid bit, 1;
- S: Extended SPI supported, 1;
- A: Acknowledgement bit, 1.

The transaction ID byte is set to 0x0001, and the requested size is set to 12,000 bytes, with a 16-bit CRC protecting the master packet header.

The slave activates the SR line (410) to indicate that its return packet header is available for reading. The master reads the slave packet header (412) as follows:

- R: reserved, 0;
- C: Transaction completed, 0, since data transfer is yet to complete;
- M: Master or slave bit, 0, since this is a slave-origin header;
- D: Data follows bit, 0, since no data follows the header from the slave side;
- T: Transaction valid bit, 1;
- S: Extended SPI supported, 1;
- A: Acknowledgement bit, 1.

The transaction ID byte is set to 0x0001, and the requested size is set to 4095 bytes, with a 16-bit CRC protecting the slave packet header.

After each subsequent activation of the SR line (414, 418, 422), the master sends data packets respectively of 4095 bytes (416), 4095 bytes (420), and 3810 bytes (424), for a total of 12,000 bytes. The packets are protected by 32-bit CRC (428), the transmission of which is preceded by the activation of the SR line (426).

After the master-to-slave data completes transmission, the slave activates the SR line (430) and sends a packet header (432), read by the master as follows.

- R: reserved, 0;
- C: Transaction completed, 1, since the slave has received all 12,000 bytes;
- M: Master or slave bit, 0, since this is a slave-origin header;
- D: Data follows bit, 0, since no data follows this header;

- T: Transaction valid bit, 1;
- S: Extended SPI supported, 1;
- A: Acknowledgement bit, 1.

The transaction ID byte is set to 0x0001, and the requested size is set to 0 bytes (since this is a closing packet header), with a 16-bit CRC protecting the slave packet header. The master toggles the SS line (434) to indicate to slave that the write operation has terminated successfully.

Example 2: *Simple read transaction from slave to master*, illustrated in Fig. 5.

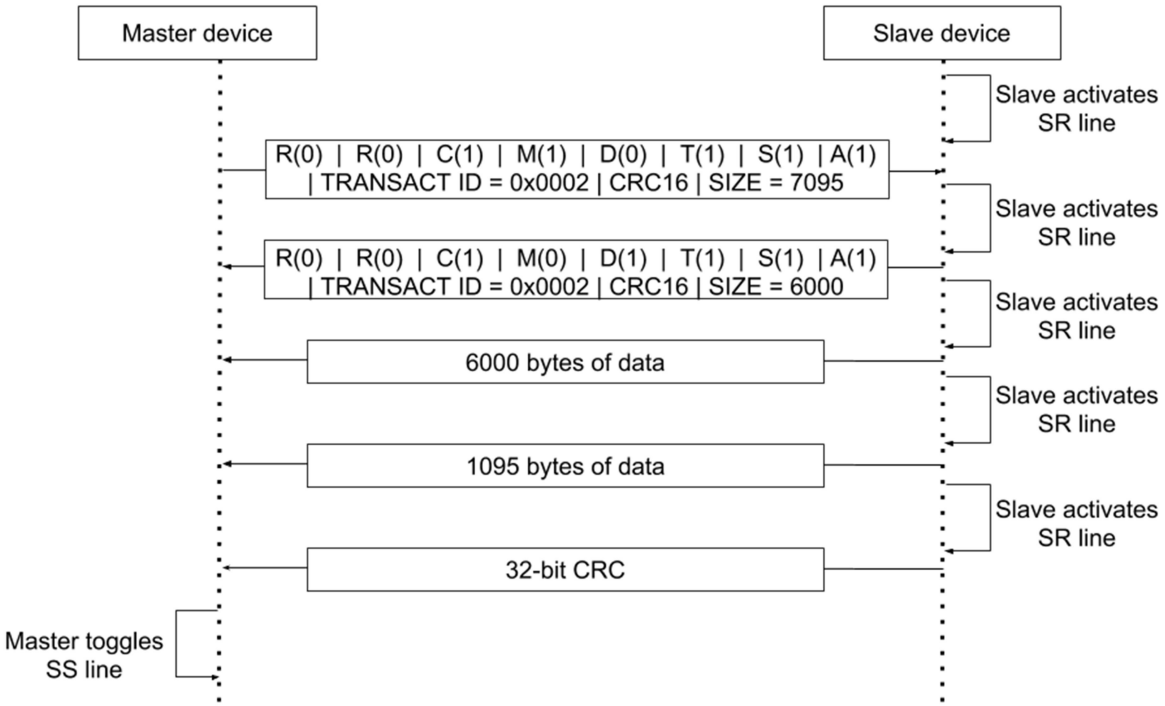

Fig. 5: Simple read transaction from slave to master

Example 3: *Write from master to slave with multiple errors*, illustrated in Fig. 6.

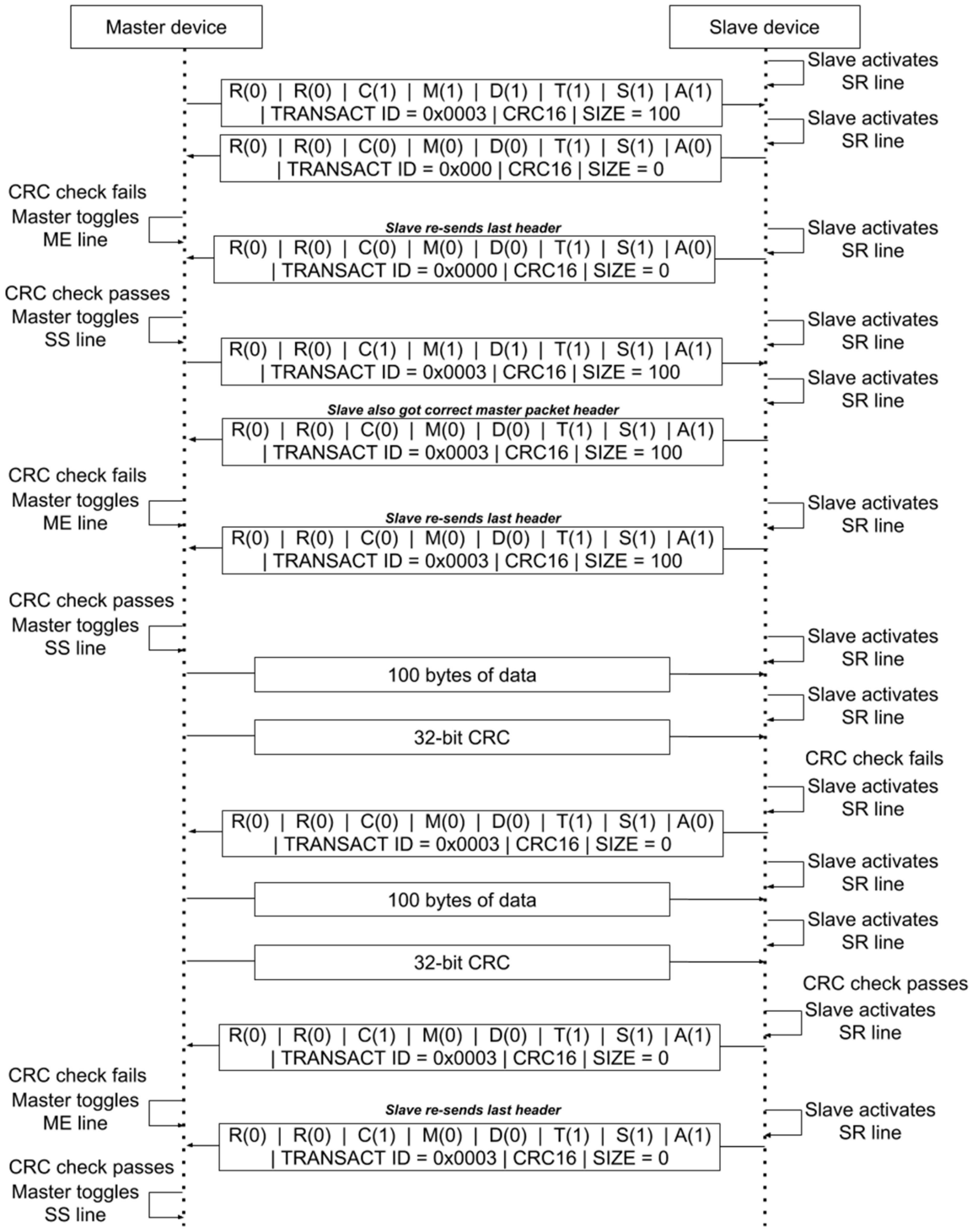

Fig. 6: Write from master to slave with multiple errors

Example 4: *Write-read transaction that does not complete immediately*, illustrated in Fig. 7, e.g., when a slave device performs a computationally intensive command.

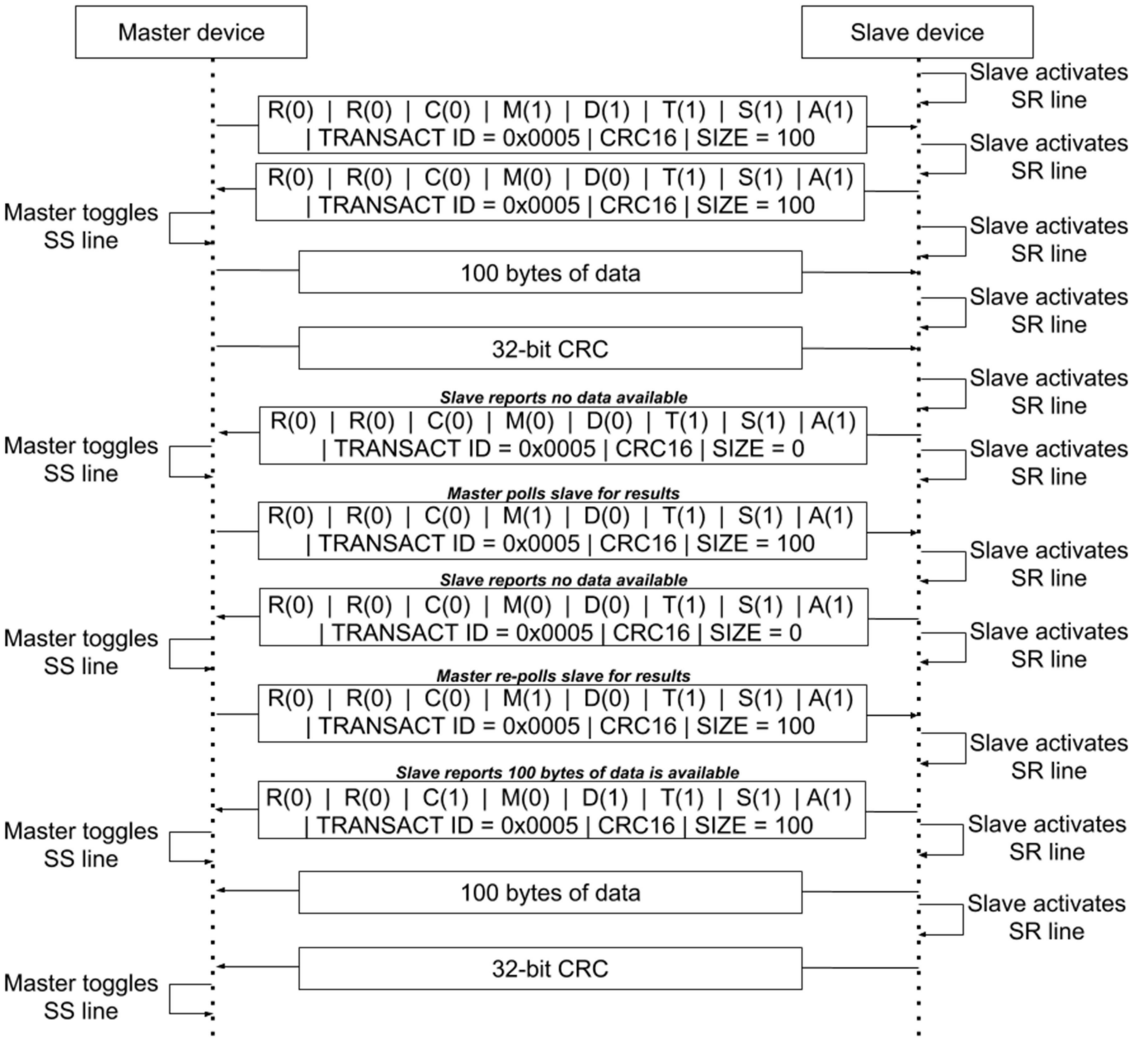

Fig. 7: Write-read transaction that does not complete immediately

Example 5: *Polling for outstanding transactions*, illustrated in Fig. 8. As mentioned above, extended SPI enables queueing and asynchronous completion of transactions. In the following example, transaction ID 0 is reserved, and transaction IDs 6 and 7 have not yet completed. The

master device specifically polls for incomplete transactions 6 and 7, and can poll for any incomplete tasks using transaction ID 0.

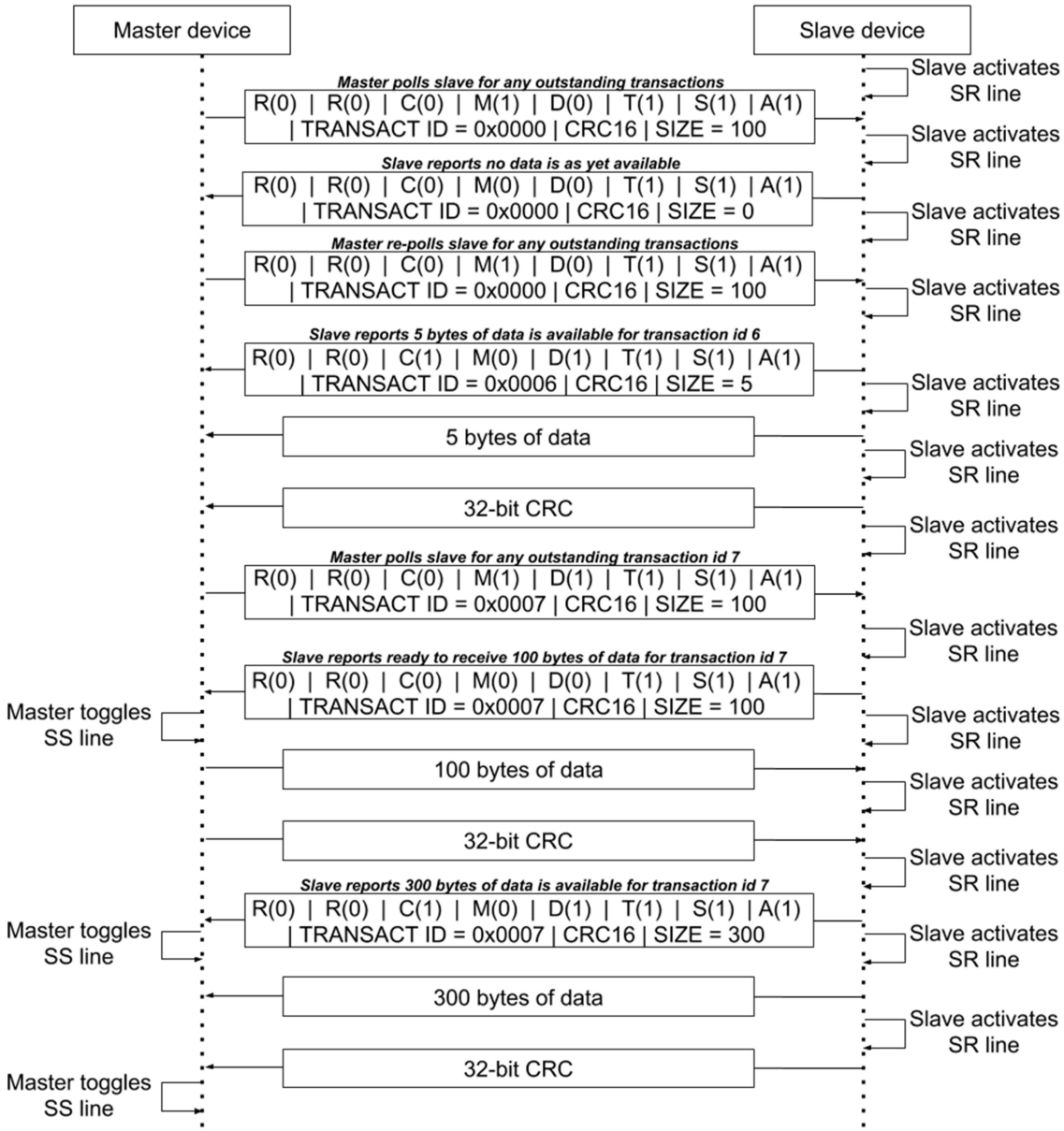

Fig. 8: Polling for outstanding transactions

CONCLUSION

In applying SPI to communications between devices of substantially differing performance characteristics, e.g., clock frequency, memory capacity, or response time, the original SPI has some disadvantages. For example, due to the streaming nature of SPI communication, errors are not detected; due to differing clock frequencies at two ends, synchronization is difficult; due to differing memory capacity at two ends, the buffers of the end with less memory overflow; etc.

This disclosure describes an extension to SPI that features synchronization signaling and packet-splitting abilities to enable devices of differing abilities to communicate via SPI. The extension also enables asynchronous transaction execution over the SPI bus and adds error detection capabilities, both features that enable larger packet sizes and higher transmission/transaction rates.