Packet-Level and Transport-Level Protocols

DIGILENT®
www.digilentinc.com
215 E Main Suite D | Pullman, WA 99163
(509) 334 6306 Voice and Fax

Revision: 2/16/06

Introduction

This document describes the protocols for constructing, sending, and receiving commands and data between master and slave devices using the SPI, TWI, or RS232 binary interface.

Packet-Level Protocol

The packet-level protocol provides a transparent means of sending various command and data payloads between a master and a slave device independent of the hardware interface. All that is required of the interface is the ability to transmit byte-aligned data and to identify boundaries between packets.

Fundamentally, the protocol consists of packets containing a simple header (typically one or two bytes) followed by an arbitrary amount of application-specific data. All packets conform to this format, whether they are sent from the master to the slave, or vice-versa. The packet header's upper three bits indicate the type of packet, and the lower five bits (and the second byte, if present) describe the sub-type.

The following is a brief summary of all packet types and pre-defined sub-types:

000x xxxx – *pktSync* This packet type includes various synchronization bytes used to implement specific transport-level protocols. The byte's specific meaning is interface-dependent, with some sub-types not being used by all interfaces.

- **00000 – *synIdle*** A filler byte sent by the slave or master.
- **00001 – *synReqWr*** Sent by the master to request to send a packet to the slave. The slave will not respond with meaningful data until it is ready to respond, at which point it will send an *rspPacketStart*. This is only used in SPI and binary RS232 mode.
- **00010 – *synReqRd*** Sent by the master to request to read a packet from the slave. The slave will not respond with meaningful data until it is ready to respond, at which point it will send an *rspPacketStart*, then a packet header followed by the packet's payload (if any). Should a master attempt to read from the slave when there is no data available for reading, an *rspNoData* packet will be received. This is only used in SPI and binary RS232 mode.
- **00011 – *synDone*** In binary RS232 mode, this is used to terminate every packet sent (by either the master or the slave). This must be transmitted immediately after the last data byte of the packet for it to be recognized as valid.

001c cccc – *pktCmdI* An immediate command byte. Immediate commands have no operands other than the five lower bits, which denote either an application-specific command (high command id 16 -31) to be executed or one of the mandatory immediate commands (low command id 0 - 15) that all slave devices must respect. All immediate commands may also be implemented in the short format with a single byte parameter.

- **00000 – *cmdReset*** Resets the device to its power-on defaults.
- **00001 – *cmdGetRsp*** Returns the response code generated by the reception and/or execution of the last packet in a single byte *pktRsp* packet. Upon sending the response code, the buffer will be reset to the default of *rspPacketOk*.
- **00010 – *cmdGetDevType*** Returns the device class type and firmware revision within the class in a 2-byte payload *pktDataS*. The first byte is the device class type, and the second byte is the specific device ID.
- **00011 – *cmdGetDevName*** Returns a null-terminated ASCII string identifying the device and firmware revision. It may be up to 30 characters long (31 with null) in a *pktDataS*.
- **00100 – *cmdGetDevCap*** Returns a 2-byte payload where each bit flags a device class capability.
- **00101 – *cmdGetFirmVer*** Returns the firmware revision in a 2-byte payload where the MSByte is a major revision number and the LSByte is a minor revision number. For example, Rev 1.0.
- **00110 – *cmdGetPacketSize*** Returns the maximum packet payload size supported by this device in a 2-byte payload.

010s ssss – *pktCmdS* A short-form command packet. The lower five bits in the header byte (shown here as S's) denote the length of the data contained in the rest of the packet. This payload data represents an application-specific command and operands. Typically, the first byte of the packet's payload will indicate the command to execute, with subsequent bytes representing command parameters.

011s ssss – *pktDataS* A short-form data packet. The lower five bits in the header byte (shown here as S's) denote the length of the data contained in the rest of the packet.

100h hhhh ssss ssss – *pktDataL* A long-form data packet. The lower five bits in the header byte (shown here as H's) are the lower five bits in the upper byte of the data payload's size, while the second byte (shown here as S's) is the lower byte of the data payload's size (representing a maximum payload size of 2^(5+8) = 8KB).

101x xxxx – *pktRsv1* This packet type is reserved for future uses.

110x xxxx – *pktRsv2* This packet type is reserved for future uses.

111r rrrr – *pktRsp* Response byte. A slave device may generate these bytes in response to a master. The lower five bits (shown here as R's) denote the response code. The following codes are currently established, with all other codes being reserved for future use:

- **00000 (pktRsp + 0x00) – *rspPacketStart*** Sent by the slave to the master to indicate that it is cleared to begin transmission of a packet. Also sent by the slave immediately prior to transmission of any response packet (*pktRsp*, *pktDataS*, etc.). This is only used in SPI and binary RS232 mode.
- **00001 (pktRsp + 0x01) – *rspPacketOk*** Result code generated by the slave to indicate that a packet was successfully received and processed. The response code buffer is populated with this at start-up and after a successful read of the buffer (through *cmdGetRsp*).
- **00010 (pktRsp + 0x02) – *rspNoData*** Byte sent in response to a master requesting to read data when none is available.

- **11011 (pktRsp + 0x1B) – *rspCmdFailure*** Indicates that an internal error unrelated to the communications protocol was encountered during execution of a command. Commands which generate this error will indicate in their description what conditions can cause such an error.
- **11100 (pktRsp + 0x1C) – *rspBadCommand*** Result code generated by the slave to indicate that it did not recognize a command specified in a command packet.
- **11101 (pktRsp + 0x1D) – *rspBadParameter*** Result code generated by the slave to indicate that a parameter for a valid command packet was invalid (in most cases, this means out-of-range).
- **11110 (pktRsp + 0x1E) – *rspBadType*** Result code generated by the slave to indicate that it does not respect the specified packet type.
- **11111 (pktRsp + 0x1F) – *rspBadPacket*** Result code generated by the slave to indicate that a packet was determined to be malformed. Generally, this is caused by an incorrect quantity of parameter data – either too much or too little – or by the slave having received a different amount of data than indicated by the header.

Transport-Level Protocols

The transport level is responsible for providing the interface-independent means of sending packets (as described previously). Currently, there are established protocols for sending packets over SPI, TWI, and RS232 binary interfaces.

Serial Peripheral Interface (SPI)

Generally, SPI uses mode 0 (sample on rising clock edge, setup on falling, and the clock line idles low), with the most significant bit (MSB) transferred first. Additionally, for multi-byte parameters, the least significant byte is sent first (little endian).

To initiate a transfer with SPI, the master first pulls the slave-select (SS) line low. The master then begins continuously transmitting *pktReqWr* bytes, while waiting for the slave to respond with an *rspPacketStart* byte. The first byte received by the master after pulling the SS line low should always be discarded (because there is no way the slave could initially transmit a valid byte). Subsequent bytes will echo what was sent by the master on the previous byte (*pktReqWr*), until the slave responds with *rspPacketStart*.

On the byte immediately following the reception of *rspPacketStart*, the master must begin transmitting a valid packet. Packets begin with at least a one-byte packet header identifying the type, length, and payload of a packet. The upper three bits of this first byte identify the packet type (as described in the previous section). During reception of a packet, the slave will transmit *synIdles*.

Once the packet header has been transmitted, the packet's payload is then sent. The slave will receive and store all bytes from the master (unless its internal buffers become full – a situation that the master will receive no immediate notification of; an *rspBadPacket* error will, however, be generated and stored). All bytes received are assumed to be part of the packet, and if too few or too many bytes are sent for a particular packet, an *rspBadPacket* error will be generated. To