Graphics Demo's Communication Protocol

Introduction

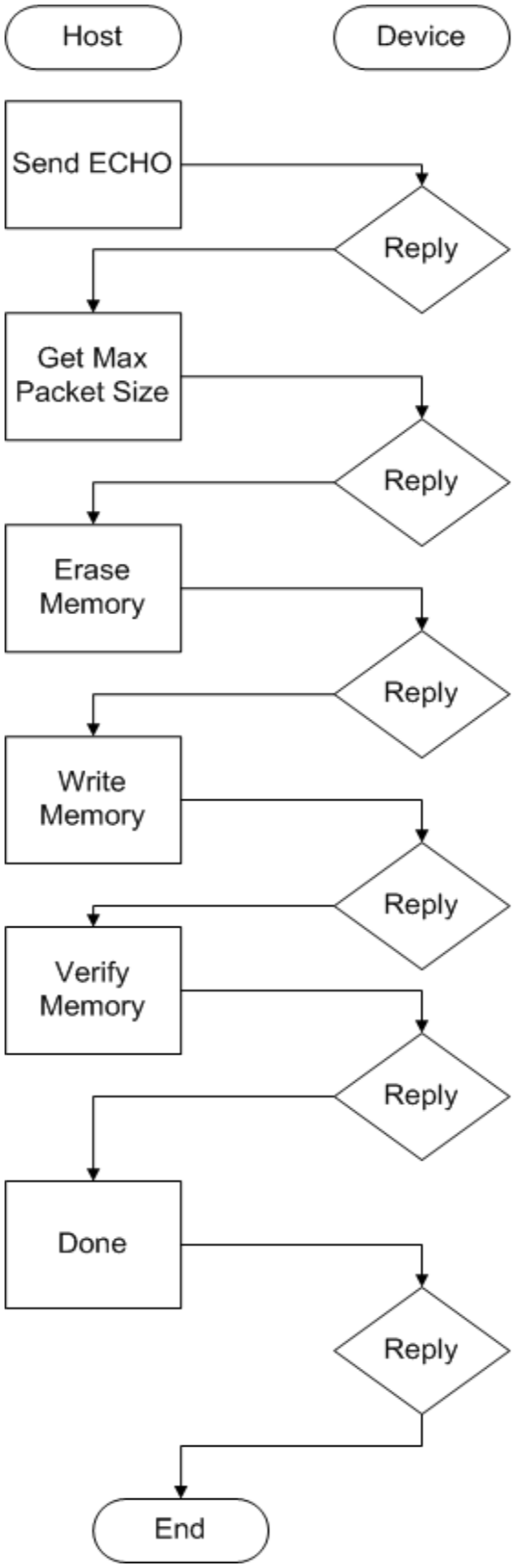
Graphics demos using the JAVA PC utility, `memory_programmer`, uses modules located in the directory, <MLA install path>/Graphics/Common, to receive, send, decode and program external memory flash devices. While the communication protocol is used exclusively by the `memory_programmer` and the firmware applications needed to upload data to an external memory source, it is not limited and can be used in a wide array of applications needing a communication protocol. The communication protocol has been abstracted from hardware as seen by the use of serial or USB communication mediums. This document will outline the topical communication flow, packet overview, currently supported commands and how to integrate the communication protocol into an application.

Throughout this document, the examples used will be from the communication scheme used by the memory programmer. This is only an example using this communication protocol.

Communication Flow

The communication flow is an ACK/NACK based protocol. The communication packets are divided into two types: command and response. Every command packet must be ACKed or NACKed by a response packet. The response packet containing the ACK/NACK can also contain a data payload. Response packets are not ACK/NACK.

This is an example of communication flow of the `memory_programmer` and the PIC device. Note that all command packets are ACK/NACKed. The Host in this case is the memory_programmer utility and the Device is a PIC microcontroller, however, these roles are not exclusive.

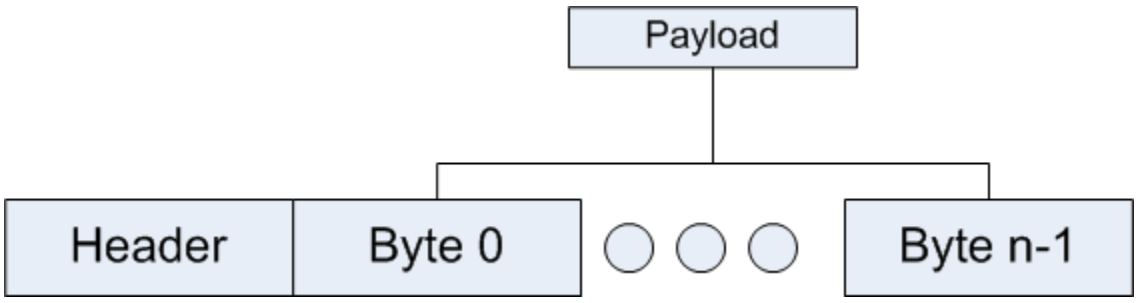

Host
Device
Send ECHO
Reply
Get Max Packet Size
Reply
Erase Memory
Reply
Write Memory
Reply
Verify Memory
Reply
Done
Reply
End

The following is a description of the communication method with uploading a HEX file. This description is to show an example of the flow (command – response) method of the communication protocol.

1. An echo command packet is sent. This packet has no payload and is used to establish communication between the host and device.
2. A command packet requesting the maximum packet size is sent. This is used by the host to ensure that the packet length is never larger than the device's resources. Devices are able to dictate the resource requirements for the packet.
3. A command packet requesting that the external memory source has been erased. The device will need to make sure that the external memory source is in the proper state to write data.
4. The memory write command packets are sent. The amount of command packets required depend on the HEX file and the maximum packet size.
5. The verify memory command packets are sent. The device will perform a checksum over a memory region.
6. The done command packet will be sent. The device is notified that all memory uploading and verifying are complete.

Packet Overview

All packets, command and response, have the same structure: header field followed by an optional payload. Zero length payloads are valid packets.



Header

The packet header is four bytes in length

ACK.NACK (1 bit)	Reply (1 bit)	Command (6-bits)	Checksum (8-bits)	Length (16-bits)

- **ACK/NACK** - This bit is a response to a command and the reply bit must be set.
- **Reply** - This bit is set when a response to a command is processed. Every command packet must have a reply
- **Command** - The command of the packet. All reply packets will reply with the same command.
- **Checksum** - The checksum of the entire packet. If the calculated checksum does not match the header's checksum, the packet must be NACKed.
    - The checksum is calculated by

        - seeding the 8-bit value with 0xFF
        - adding all of the payload byte values
        - multiplying the result by -1
    - A zero length payload will have a checksum of 1.
- **Length** - The length of the payload, ranging from 0 - 65,000.

Payload

The packet payload is variable length. The length member of the header will indicate the size of the payload in bytes.

Supported Commands

The modules in the common directory support a limited range of commands, specific to uploading data to external memory. The commands are not limited to those defined in the `comm_pkt` header file. An application is free to add or delete commands. If the application is going to use the `memory_programmer` JAVA application, any commands added should not conflict with the currently supported commands.

The following commands are supported:

- **ECHO** - A zero payload command. Used to establish communication with a device.
- **MAX PAYLOAD SIZE** - requesting the maximum payload size accepted by the device
- **MEMORY ERASE** - requesting that the memory be prepared to be written to.
- **MEMORY WRITE** - write the data in the payload to a memory section
- **MEMORY VERIFY** - requesting a checksum over a memory range
- **DONE** - all memory programming and verifying are done

The values and symbol names of these commands can be found in the `comm_pkt` header file.

Integration Into Other Applications

To integrate the communication protocol used by the `memory_programmer` PC utility and firmware, the application must contain the following files:

- **`comm_pkt`** – Receive, send and decode packets
- **`comm_pkt_callback`** – Provides the communication medium routines needed by the `comm_pkt` module.

The application will need to provide the hardware and communication functionality to the `comm_pkt_callback` routines. An example can be seen in the common directory of the Graphics demos. The application will also need to define the set of commands used. It is the application's responsibility to handle the commands and responses.