OPP Board Serial Interface
Interface Document

(Project: BrdSerIntf)

Project # : Not applicable
P/N: Not applicable
Rev 1.00

	PREPARED:	APPROVAL: LEAD ENGINEER	APPROVAL: ENGINEERING MANAGER
NAME	Hugh Spahr		
DATE			

Revision History

Version	Primary Author(s)	Description of Version	Date Completed
0.1	Hugh Spahr	Initial version	11/08/13
0.2	Hugh Spahr	Added save/erase configuration	12/21/13
0.3	Hugh Spahr	Added pictures of cards to locate headers. Added wiring diagram for solenoid card.	01/04/14
0.4	Hugh Spahr	Added new commands for Gen2 hardware	09/28/15
0.5	Hugh Spahr	Removed CRC from Inventory command	11/19/15
0.6	Hugh Spahr	Added Incandescent commands. Added num pixels to set neopixel color table command.	12/04/15
0.7	Hugh Spahr	Removed first OPP hardware commands to make the document more readable. (If need to see those commands, version 0.6 is the last version that includes them.) Fixed bit position description for Neopixel and incandescent commands to be more logical. Added wing board wiring pictures.	12/28/15
0.8	Hugh Spahr	Added INCAND_SET_ON_OFF command.	01/26/16
0.9	Hugh Spahr	Added CONFIG_IND_SOL, CONFIG_IND_INP, and SET_IND_NEO	02/02/16
0.10	Hugh Spahr	Added ON_OFF_SOL an DLY_KICK_SOL to solenoid configuration. Added SET_SOL_INPUT command.	08/06/16
0.11	Hugh Spahr	Fix solenoid wing pinout image	05/04/17
1.00	Hugh Spahr	Add RS232I_UPGRADE_OTHER_BRD and RS232I_READ_MATRIX_INP commands. Added CAN_CANCEL bit to solenoid configuration. Added WING_HI_SIDE_INCAND board type. Added section on upgrading multiple boards. Corresponds to version 1.0.0.0 of firmware.	11/03/18

Table of Contents

1 Purpose

The OPP Board Serial Interface provides a description of the serial interface for the Open Pinball Project board serial interface. The serial interface document includes the following wing boards:

- Gen2 Solenoid Driver Wing (Product ID 1013)
- Gen2 Incandescent Wing (Product ID 1014 and1017)
- Gen2 Input/Neo Wing for support of Neopixels (Product ID 1015)
- Gen2 Interface Wing (Product ID 1016)

2 Product Overview

BrdSerIntf contains the serial interface for OPP boards. This document describes the APIs

3 Applicable Documents

None

4 Terms, Definitions & Acronyms

API	Application Program Interface
EOM	End of Message (0xff)
ICD	Interface Control Document
OPP	Open Pinball Project
PWM	Pulse Width Modulation
MPF	Mission Pinball Framework

5 Serial Interface Overview

BrdSerIntf port for second generation cards is 115.2 kbps, 8, N, 1. Since the two generations of cards run at different baud rates, they can not be combined on a single serial chain. During the setup, a command is sent to get an inventory of cards that are on the serial interface chain. The cards are daisy chained together in the same way that FDDI or token ring boards are linked together. The host sends to the first card receiver which transmits to the second card's receiver. This continues until the last card where a jumper is installed so that the transmitter of the last card is connected to the host receiver.

Each card is given an address which is automatically discovered during the setup command. The host uses the address to send individual commands to cards.

All commands use card addresses to indicate the destination of the command, and a CRC8 at the end of the command. The exception to this is the inventory command where cards receive the command, and wait to see the end of message. The card then inserts its address before the transmitting the EOM.

After the inventory command is run, the cards are configured. The host then sends a Get Gen2 Cfg message to discover the wing board configuration for each card. The configuration of the boards is saved in non-volatile memory so they operate independently of the host. The cards do all of the real time processing themselves without the intervention of the host. The host only polls the cards to get status, such as if a solenoid input switch was activated.

Gen2 commands append a CRC8 to the end of transfers which matches the CCITT-CRC8. The generator polynomial is x^8+x^2+x+1 with an initial value of 0xff. This is true for all commands except for the initial inventory command.

5.1 Addressing

The most significant four bits of the address indicate card type, while the least significant four bits of the address byte indicate the instance of the card. The first card starts from 0, and each card increases its address by one.

Gen2 cards have the most significant nibble set to 2 (CARD_ID_GEN2_CARD).

Examples:
First Gen2 card: 0x20
Third Gen2 card: 0x22

6 Serintf.h

The serial interface file contains the structures used to interface with the solenoid driver boards and input boards. If programming using the C programming language, this file can be included to get the necessary defines.

```
/* Each command starts with the Card ID except for inventory and EOM.  Next comes
 *  the command, then any data.
 */
typedef enum
{
  RS232I_GET_SER_NUM        = 0x00,
  RS232I_GET_PROD_ID        = 0x01,
  RS232I_GET_VERS           = 0x02,
  RS232I_SET_SER_NUM        = 0x03,
  RS232I_RESET              = 0x04,
  RS232I_GO_BOOT            = 0x05,
  RS232I_CONFIG_SOL         = 0x06,     /* For each solenoid, CFG_SOL_TYPE,
                                           *  Initial kick, and Duty Cycle.
                                           */
  RS232I_KICK_SOL           = 0x07,     /* Value, mask */
  RS232I_READ_SOL_INP       = 0x08,     /* Data */
  RS232I_CONFIG_INP         = 0x09,     /* For each input, CFG_INP_TYPE */
  RS232I_GEN2_UNUSED        = 0x0a,     /* Data, data */
  RS232I_SAVE_CFG           = 0x0b,
  RS232I_ERASE_CFG          = 0x0c,
  RS232I_GET_GEN2_CFG       = 0x0d,     /* For each wing,  GEN2_WING_TYPE */
  RS232I_SET_GEN2_CFG       = 0x0e,     /* For each wing,  GEN2_WING_TYPE */
  RS232I_CHNG_NEO_CMD       = 0x0f,
  RS232I_CHNG_NEO_COLOR     = 0x10,
  RS232I_CHNG_NEO_COLOR_TBL = 0x11,
  RS232I_SET_NEO_COLOR_TBL  = 0x12,
  RS232I_INCAND_CMD         = 0x13,
  RS232I_CONFIG_IND_SOL     = 0x14,
  RS232I_CONFIG_IND_INP     = 0x15,
  RS232I_SET_IND_NEO        = 0x16,
  RS232I_SET_SOL_INPUT      = 0x17,
  RS232I_UPGRADE_OTHER_BRD  = 0x18,
  RS232I_READ_MATRIX_INP    = 0x19,
```

```
 RS232I_NUM_CMDS,

  RS232I_INVENTORY          = 0xf0,     /* Each card adds byte for card type */
  RS232I_EOM                = 0xff,
} __attribute__((packed)) RS232I_CMD_E;

#ifndef RS232I_INSTANTIATE
 extern
#endif
const U8                    CMD_LEN[RS232I_NUM_CMDS]
#ifdef RS232I_INSTANTIATE
 ={ 4,  /* RS232I_GET_SER_NUM */       4,  /* RS232I_GET_PROD_ID */
    4,  /* RS232I_GET_VERS */          4,  /* RS232I_SET_SER_NUM */
    0,  /* RS232I_RESET */             0,  /* RS232I_GO_BOOT */
    48, /* RS232I_CONFIG_SOL */        4,  /* RS232I_KICK_SOL */
    4,  /* RS232I_READ_SOL_INP */      32, /* RS232I_CONFIG_INP */
    2,  /* RS232I_GEN2_UNUSED */       0,  /* RS232I_SAVE_CFG */
    0,  /* RS232I_ERASE_CFG */         4,  /* RS232I_GET_GEN2_CFG */
    4,  /* RS232I_SET_GEN2_CFG */      6,  /* RS232I_CHNG_NEO_CMD */
    6,  /* RS232I_CHNG_NEO_COLOR */    4,  /* RS232I_CHNG_NEO_COLOR_TBL */
    97, /* RS232I_SET_NEO_COLOR_TBL /  5,  /* RS232I_INCAND_CMD */
    4,  /* RS232I_CONFIG_IND_SOL */    2,  /* RS232I_CONFIG_IND_INP */
    2,  /* RS232I_SET_IND_NEO */       2,  /* RS232I_SET_SOL_INPUT */
    0,  /* RS232I_UPGRADE_OTHER_BRD */ 8,  /* RS232I_READ_MATRIX_INP */
  }
#endif
;
/* Note:  This is the length of the cmd excluding card ID and cmd at the
 *  beginning and CRC8 at the end.
 */

typedef enum
{
  CARD_ID_CARD_NUM_MASK     = 0x0f,

  CARD_ID_TYPE_MASK         = 0xf0,
  CARD_ID_SOL_CARD          = 0x00,
  CARD_ID_INP_CARD          = 0x10,
  CARD_ID_GEN2_CARD         = 0x20,

} __attribute__((packed)) RS232I_CARD_ID_E;

typedef enum
{
  USE_SWITCH                = 0x01,
  AUTO_CLR                  = 0x02,
  ON_OFF_SOL                = 0x04,
  DLY_KICK_SOL              = 0x08,
  USE_MATRIX_INP            = 0x10,
  CAN_CANCEL                = 0x20,
} __attribute__((packed)) RS232I_CFG_SOL_TYPE_E;

typedef enum
{
  DUTY_CYCLE_MASK           = 0x0f,   /* lsb 4 bits are duty cycle */
  MIN_OFF_MASK              = 0x70,
} RS232I_DUTY_E;
/* Min off time is 0-7 times the initial kick time.  If initial kick
```

```
 * is 20 ms and min off is 5, the solenoid will be forced off for 100 ms
 */

typedef enum
{
  STATE_INPUT               = 0x00,
  FALL_EDGE                 = 0x01,
  RISE_EDGE                 = 0x02,
} __attribute__((packed)) RS232I_CFG_INP_TYPE_E;

typedef enum
{
  WING_UNUSED               = 0x00,
  WING_SOL                  = 0x01,
  WING_INP                  = 0x02,
  WING_INCAND               = 0x03,
  WING_SW_MATRIX_OUT        = 0x04,
  WING_SW_MATRIX_IN         = 0x05,
  WING_NEO                  = 0x06,
  WING_HI_SIDE_INCAND       = 0x07,
} __attribute__((packed)) RS232I_GEN2_WING_TYPE_E;

typedef enum
{
  NEOCMD_BLINK_SLOW         = 0x00,
  NEOCMD_BLINK_FAST         = 0x20,
  NEOCMD_FADE_SLOW          = 0x40,
  NEOCMD_FADE_FAST          = 0x60,
  NEOCMD_ON                 = 0x80,   /* overrides other cmds */
  NEOCMD_MASK               = 0xe0,
  NEOCMD_COLOR_TBL_MASK     = 0x1f,
} __attribute__((packed)) RS232I_GEN2_NEO_CMD_E;

typedef enum
{
  INCAND_ROT_LEFT           = 0x00,
  INCAND_ROT_RIGHT          = 0x01,
  INCAND_LED_ON             = 0x02,
  INCAND_LED_OFF            = 0x03,
  INCAND_LED_BLINK_SLOW     = 0x04,
  INCAND_LED_BLINK_FAST     = 0x05,
  INCAND_LED_BLINK_OFF      = 0x06,
  INCAND_LED_SET_ON_OFF     = 0x07,

  INCAND_SET                = 0x80,
  INCAND_SET_ON             = 0x01,
  INCAND_SET_BLINK_SLOW     = 0x02,
  INCAND_SET_BLINK_FAST     = 0x04,
} __attribute__((packed)) RS232I_GEN2_INCAND_CMD_E;

typedef enum
{
  SOL_INP_SOL_MASK          = 0x0f,

  SOL_INP_CLEAR_SOL         = 0x80,
} __attribute__((packed)) RS232I_SET_SOL_INP_E;
```

7 Commands

The following section describes all the commands and gives examples. All second generation a CRC8 to the end of the command/response except for the inventory command.

7.1 Get Serial Number, 0x00

CardAddr + 0x00 + Serial number (4 bytes) + CRC8

Example: Host queries the gen2 card (addr 0x22) for its serial number. The card responds with the serial number 0x01234567.

Gen2 Cmd: 0x22 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0xc6 | 0xff

Gen2 Resp: 0x22 | 0x00 | 0x01 | 0x23 | 0x45 | 0x67 | 0x06 | 0xff

7.2 Get Product ID, 0x01

CardAddr + 0x01 + Product ID (4 bytes) + CRC8

Note: Gen2 cards always return the configuration of the wing boards that identifies the capabilities of the board.

Example: Host queries the first Gen2 card (addr 0x20) for its product ID. Card responds the Port A has a NeoPixel board, Port B is an input board, and Port C and D have Solenoid boards.

Gen2 Cmd: 0x20 | 0x01 | 0x00 | 0x00 | 0x00 | 0x00 | 0xf6 | 0xff

Gen2 Resp: 0x20 | 0x01 | 0x06 | 0x02 | 0x01 | 0x01 | 0x46 | 0xff

7.3 Get Version, 0x02

CardAddr + 0x02 + Firmware Version (4 bytes) + CRC8

Get the firmware version that is running on a card. The firmware version is broken into a byte representing the major version, minor version, sub version and engineering version.

Example: Host queries the third Gen2 card (addr 0x22) for its version number. The card responds that it is running version 1.5.6.0 (major version 1, minor version 5, subversion 6, and engineering version 0).

Command: 0x22 | 0x02 | 0x00 | 0x00 | 0x00 | 0x00 | 0x50 | 0xff

Response: 0x22 | 0x02 | 0x01 | 0x05 | 0x06 | 0x00 | 0xf8 | 0xff.

7.4 Set Serial Number, 0x03

CardAddr + 0x03 + Serial number (4 bytes) + CRC8

Example: Host sets the serial number of the first Gen2 card (addr 0x20) to 32 (0x20 in hex). If the serial number has already been set, the card responds with the current serial number of the card. If the serial number can be programmed, it responds with the serial number found in the command.

Command: 0x20 | 0x03 | 0x00 | 0x00 | 0x00 | 0x20 | 0xd2 | 0xff

Response: 0x20 | 0x03 | 0x00 | 0x00 | 0x00 | 0x20 | 0xd2 | 0xff

7.5 Reset, 0x04

CardAddr + 0x04 + CRC8

Example: Reset the fifth Gen2 card (addr 0x24). This resets the card and an inventory command must be resent to reconfigure the card's address.

Command: 0x24 | 0x04 | 0x31

Response: No response since the card resets

7.6 Go Boot, 0x05

CardAddr + 0x05 + CRC8

Example: Reset the Gen2 card (addr 0x20). This resets the card and keeps the card in the bootloader so that the card firmware can be updated.

Command: 0x20 | 0x05 | 0x62

Response: No response since the card resets. After the reset the card remains in the bootloader. Cards should only be updated when connected directly to a host without any other cards on a chain.

7.7 Configure Solenoid Board, 0x06

CardAddr + 0x06 + Solenoid Parameters (16 sets of parameters, 3 bytes each or 48 bytes total) + CRC8

Each solenoid is configured using three bytes. The first byte is the type of solenoid which is a bit field. (0x01 - USE_SWITCH, 0x02 - AUTO_CLR, 0x04 - ON_OFF_SOL, 0x08 - DLY_KICK_SOL, 0x10 – USE_MATRIX_INP, and 0x20 - CAN_CANCEL). If the solenoid is configured to use the switch input, when the switch input is closed, the solenoid fires. If the solenoid is configured to auto clear, the host can kick the solenoid using the kick which is automatically cleared, and the solenoid can be kicked another time without sending a command to clear the kick. If the solenoid is configured as an ON/OFF solenoid, it will be driven at 100% when the input is active (used for dual wound flippers). If the solenoid is configured as a delay kick solenoid, the third byte determines the delay after the switch is active before kicking the solenoid. If the solenoid is configured to use a matrix input to kick the solenoid, the the third byte is used to indicate the 8x8 switch matrix input on this board that activates the solenoid. If the solenoid is configured as can cancel, the initial kick can be canceled to provide a shorter pulse to enable things like flipper tap passing.

The second byte is the initial kick for the solenoid in milliseconds. If the initial kick is supposed to be 100 ms, the byte should be 0x64.

The third byte is broken into the duty cycle to "hold" the solenoid active, and the minimum off time. The period for the hold is 16 ms. The least significant four bits indicate the amount of "on" time during the 16 ms period. To keep the solenoid on for 50%, during the hold period, the four bits should be set to 0x08. This is 8 ms/16 ms or 50%. For 25% hold, set the value to 0x04. This is 4 ms/16 ms or 25%. The maximum hold strength is 87.5% unless using an on/off solenoid.

The minimum off time is calculated using the least significant three bits of the most significant nibble of the third byte. ((3rd byte & 0x70) >> 4). These three bits are multiplied by the initial kick time to determine the minimum off time for the solenoid. If the solenoid is using the switch contact, and the initial kick time is 105 ms, and the minimum off time is 315 ms, and the duty cycle is 75%, the configuration bytes would be 0x01 0x69 0x3c.

If the solenoid is configured as a delayed hold solenoid, the least significant nibble of the third byte is the delay after the switch occurs to kick the solenoid. The nibble is multiplied by two to calculate the delay. If the least significant nibble is 0x3, the code will wait for 6 ms before kicking the solenoid. This gives very fine granularity for supporting things like kickback solenoids in out lanes. Note: This means that delay kick solenoids can not be PWM'd.

Note: Unused solenoids should have their parameters configured to 0x00 | 0x00 | 0x00 to disable them. With Gen2, only wing boards configured/populated as solenoids will use the solenoid parameters. Solenoid parameters for wing boards that aren't solenoid wings will be ignored.

Example: Configure the first Gen2 card (addr 0x20). The first four solenoids are configured as flippers, the next four are configured as pop bumpers, the next four are not used, and the last four are configured as VUKs. The flippers are configured with an initial kick of 48 ms and 25% hold. The pop bumpers have initial kicks of 48 ms and no hold. The VUK have initial kicks of 100 ms and no hold.

Command: 0x20 | 0x06 | 0x01 | 0x30 | 0x04 | 0x01 | 0x30 | 0x04 | 0x01 | 0x30 | 0x04 | 0x01 | 0x30 | 0x04 | 0x01 | 0x30 | 0x00 | 0x01 | 0x30 | 0x00 | 0x01 | 0x30 | 0x00 | 0x01 | 0x30 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x01 | 0x64 | 0x00 | 0x01 | 0x64 | 0x00 | 0x01 | 0x64 | 0x00 | 0x01 | 0x64 | 0x00 | 0xc4

Response: The card removes the command from the communication stream.

7.8 Kick Solenoids, 0x07

CardAddr + 0x07 + Solenoids To Kick (2 bytes) + Mask (2 bytes) + CRC8

Command for the host to kick the solenoids. The solenoid bits are active high. The mask bits allow individual or groups of solenoids to be turned on and off. If the solenoid is configured to auto clear, after the solenoid kicks, the bit is automatically cleared so a second command isn't needed to kick it again. Solenoids to kick bytes and mask are16 bit quantities.

Example: Kick the third Gen2 card (addr 0x22). Solenoid wing cards are on Port A and Port D. Turn on first and last solenoids for Port A wing card. Turn off second solenoid on Port D wing board.

Command: 0x22 | 0x07 | 0x00 | 0x09 | 0x20 | 0x09 | 0x44

Response: The card removes the command from the communication stream.

7.9 Read Gen2 Inputs, 0x08

CardAddr + 0x08 + Data (4 bytes) + CRC8

Gen2 cards use a single command to read inputs. It returns the inputs from both solenoids and input wing boards. Solenoid wing boards only populate the low nibble of the port byte.

Each solenoid card returns a byte listing if any new inputs have occurred since the last read input command. The command contains a blank data byte to be filled out by the card. Bits are numbered from right to left with bit 0 being 0x01.

Example: Read first Gen2 card (addr 0x20). Card has solenoids on Port A and Port D, and input boards on Port B and Port C. Solenoid or Port A has bits 0, 1, 3, solenoid on Port D has bit 2 set, input card on port B has 0, 1, 4, and 5 set, and input card on port C has 0, 3, 4, and 7 set.

Command: 0x20 | 0x08 | 0x00 | 0x00 | 0x00 | 0x00 | 0x8d

Response: 0x20 | 0x08 | 0x04 | 0x99 | 0x33 | 0x0b | 0xb1

7.10 Configure Inputs Board, 0x09

CardAddr + 0x09 + Input Parameters (32 sets of parameters, 1 byte each or 32 bytes total) + CRC8

Each input is configured using one byte. Switches are typically active low since the input line has a pull up resistor, and the switch connects the input to ground. STATE_INPUT returns the current value of the input. FALL_EDGE returns a single '1' with the read input board command when the input changes from a high to low value. The '1' is held until the input is read. The next read input will return a '0' unless the input transitions from low to high, then low again. RISE_EDGE returns a single '1' with the read input board command when the input changes from a low to high value. The '1' is held until the input is read. The next read input will return a '0' unless the input transitions from high to low, then high again.

Note: Unused inputs should have their parameters configured to 0x00. With Gen2, only wing boards configured/populated as inputs will use the input parameters. Input parameters for wing boards that aren't input wings will be ignored.

Example: Configure the first Gen2 card (addr 0x20) with input wing boards on Port B and Port C. Port B 8 inputs are STATE_INPUTS, and Port C first 4 inputs are FALL_EDGE, and last four inputs are RISE_EDGE.

Command: 0x20 | 0x09 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x01 | 0x01 | 0x01 | 0x01 | 0x02 | 0x02 | 0x02 | 0x02 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x7f

Response: The card removes the command from the communication stream.

7.11 Save Cfg, 0x0b

CardAddr + 0x0b + CRC8

Save the current configuration to the card. The next time the card is powered, it will automatically be configured without need another configuration command.

Example: Save the configuration to the first Gen2 card (addr 0x20)

Command: 0x20 | 0x0b | 0x48

Response: The card removes the command from the communication stream.

7.12 Erase Cfg, 0x0c

CardAddr + 0x0c + [CRC8 (Gen2 only)]

Clear the saved configuration. The next time the card is powered, it will need to be configured. Note: This command does not disable the current configuration.

Example: Erase the configuration to the first Gen2 card (addr 0x20)

Command: 0x20 | 0x0c | 0x5d

Response: The card removes the command from the communication stream.

7.13 Get Gen2 Cfg, 0x0d

CardAddr + 0x0d + 0x00 + 0x00 + 0x00 + 0x00 + 0x49

Get the Gen2 configuration of the wing boards. Note: Wing boards are not discovered, but must be programmed using the Set Gen2 Cfg command. The configuration can be made persistent by using the Save Cfg command.

Example: Get the configuration to the second Gen2 card wing boards (addr 0x21). Card responds the Port A has a NeoPixel board, Port B is an input board, and Port C and D have Solenoid boards.

Command: 0x21 | 0x0d | 0x00 | 0x00 | 0x00 | 0x00 | 0x49

Response: 0x21 | 0x0d | 0x06 | 0x02 | 0x01 | 0x01 | 0xf9

7.14 Set Gen2 Cfg, 0x0e

CardAddr + 0x0e + Input Parameters (4 sets of parameters, 1 byte each or 4 bytes total) + CRC8

Set the Gen2 configuration of the wing boards. Note: The configuration can be made persistent by using the Save Cfg command.

Example: Set the configuration to the second Gen2 card wing boards (addr 0x21) with Port A as a NeoPixel board, Port B as an input board, and Port C and D as Solenoid boards.

Command: 0x21 | 0x0e | 0x06 | 0x02 | 0x01 | 0x01 | 0x5f

Response: The card removes the command from the communication stream.

7.15 Change Neopixel Cmd, 0x0f

CardAddr + 0x0f + NeoCmd + Offset + Mask (4 bytes) + CRC8

Change the Neopixel command to the masked Neopixels. The offset is the number of Neopixels from the start of the chain to change the command, and the mask can change up to 32 contiguous Neopixels. To change the first 8 Neopixels, the offset equals 0, and the first byte of the mask would be 0x000000ff.

Example: Set the Neopixel command to fast blink (0x20) on the first Gen2 board to light the fifth through fifteenth Neopixel. This command would be written either using an 0 offset and using an offset within the mask, or starting with offset 4, and masking the first ten bits.

Command: 0x20 | 0x0f | 0x20 | 0x00 | 0x00 | 0x00 | 0x7f | 0xf0 | 0xc8 or
0x20 | 0x0f | 0x20 | 0x04 | 0x00 | 0x00 | 0x07 | 0xff | 0x60

Response: The card removes the command from the communication stream.

7.16 Change Neopixel Color, 0x10

CardAddr + 0x10 + NeoColor + Offset + Mask (4 bytes) + CRC8

Change the Neopixel color index to the masked Neopixels. Colors are indexed (0-31) into the color table to determine the actual color. The offset is the number of Neopixels from the start of the chain to change the command, and the mask can change up to 32 contiguous Neopixels. To change the first 8 Neopixels, the offset equals 0, and the first byte of the mask would be 0xff.

Example: Set the Neopixel color index to 5 on the first Gen2 board to light the fifth through fifteenth Neopixel. This command would be written either using an 0 offset and using an offset within the mask, or starting with offset 4, and masking the first ten bits.

Command: 0x20 | 0x10 | 0x05 | 0x00 | 0x00 | 0x00 | 0x7f | 0xf0 | 0x54 or
0x20 | 0x10 | 0x05 | 0x04 | 0x00 | 0x00 | 0x07 | 0xff | 0xfc

Response: The card removes the command from the communication stream.

7.17 Change Neopixel Color Table, 0x11

CardAddr + 0x11 + Index + ColorBytes (3 bytes) + CRC8

Change a single entry in the Neopixel color table. Neopixels color entries are ordered: green byte, red byte, blue byte.

Example: Change the Neopixel color table entry index 12 to have green full on (0xff), red half on (0x80), and blue quarter on (0x40).

Command: 0x20 | 0x11 | 0x0c | 0xff | 0x80 | 0x40 | 0x76

Response: The card removes the command from the communication stream.

7.18 Set Neopixel Color Table, 0x12

CardAddr + 0x12 + ColorBytes (32 sets of parameters, 3 bytes each or 96 bytes total) + numPixels + CRC8

Change all the Neopixel color table entries. I picked random color values that are all different. I have no idea what colors these are except 0x00 0x00 0x00 is off, and 0xff 0xff 0xff is white. The number of pixels is a single byte at the end of the command with 64 pixels set in this command. Note: This command takes nearly 10 ms to send, so it should be used sparingly or probably only when setting up the non-volatile memory configuration.

Command: 0x20 | 0x12 | 0x0c | 0x00 | 0x00 | 0x00 | 0xff | 0xff | 0xff | 0x40 | 0x00 | 0x00 | 0x80 | 0x00 | 0x00 | 0xc0 | 0x00 | 0x00 | 0xff | 0x00 | 0x00 | 0x00 | 0x40 | 0x00 | 0x00 | 0x80 | 0x00 | 0x00 | 0xc0 | 0x00 | 0x00 | 0xff | 0x00 | 0x00 | 0x00 | 0x40 | 0x00 | 0x00 | 0x80 | 0x00 | 0x00 | 0xc0 | 0x00 | 0x00 | 0xff | 0x40 | 0x40 | 0x00 | 0x80 | 0x80 | 0x00 | 0xc0 | 0xc0 | 0x00 | 0xff | 0xff | 0x00 | 0x00 | 0x40 | 0x40 | 0x00 | 0x80 | 0x80 | 0x00 | 0xc0 | 0xc0 | 0x00 | 0xff | 0xff | 0x40 | 0x00 | 0x40 | 0x80 | 0x00 | 0x80 | 0xc0 | 0x00 | 0xc0 | 0xff | 0x00 | 0xff | 0x40 | 0x40 | 0x40 | 0x80 | 0x80 | 0x80 | 0xc0 | 0xc0 | 0xc0 | 0xff | 0x80 | 0x40 | 0x80 | 0x40 | 0xff | 0x40 | 0xff | 0x80 | 0x40 | 0x7d

Response: The card removes the command from the communication stream.

7.19 Incandescent Command, 0x13

CardAddr + 0x13 + IncandCmd + Mask (4 bytes) + CRC8

Change the incandescent wing boards to turn on/off bulbs, make them blink slow/fast, or rotate the bulbs that are on. The mask contains the affected bulbs with the first byte received for wing 0. Blinking and on/off are independent so a bulb can be set to blink and on at the same time. The bulb being lit overrides the blinking. The rotate commands allow lane shifting to be easily implemented where all the lanes would be set to blink, and the completed lanes are set to lit. Pushing the flipper button would send a rotate command to rotate the on/off state of the inlane bulbs. The set on/off command clears all blinking masks, and uses the mask to turn bulbs on/off. (A one bit turns a bulb on while a zero bit turns a bulb off. This was implemented so MPF only needs to send a single command to turn set the state (on/off) of all the bulbs.)

The INCAND_SET sub-command sets the state (both the on/off and the blink/no blink) in one command. It uses the INCAND_SET bitfields to configure the bulb settings.

Example: On the first wing board light bulbs 0-3, and on the third wing board, light bulbs 0, 2, 4, 6.

Command: 0x20 | 0x13 | 0x02 | 0x00 | 0x55 | 0x00 | 0x0f | 0x0d

Example: Set all the bulbs on wing 3 to be on and blink rapidly. (In this case the bulbs will appear on, but if the bulb is commanded off using INCAND_LED_OFF, it will start blinking).

Command: 0x20 | 0x13 | 0x85 | 0xff | 0x00 | 0x00 | 0x00 | 0xab

Response: The card removes the command from the communication stream.

7.20 Configure Individual Solenoid, 0x14

CardAddr + 0x14 + SolIndex + SolenoidCfg (3 bytes) + CRC8

Change the configuration of a single solenoid. The index contains the index of the solenoid [0-15]. The solenoid configuration uses the same values as the configure solenoid command above.

Example: Configure the first Gen2 card (addr 0x20). Configure the third solenoid (wing 0, last solenoid) as a flipper.

Command: 0x20 | 0x14 | 0x03 | 0x01 | 0x30 | 0x04 | 0x9d

Response: The card removes the command from the communication stream.

7.21 Configure Individual Input, 0x15

CardAddr + 0x15 + InputIndex + InputCfg (1 byte) + CRC8

Change the configuration of a single input. The index contains the index of the input [0-31]. The input configuration uses the same values as the configure input command above.

Example: Configure the first Gen2 card (addr 0x20). Configure the ninth input (wing 1, 2nd input, or P1[1]) for falling edges.

Command: 0x20 | 0x15 | 0x08 | 0x01 | 0xd2

Response: The card removes the command from the communication stream.

7.22 Set Individual Neopixel, 0x16

CardAddr + 0x15 + PixelIndex + ColorTblIndex (1 byte) + CRC8

Change the color of a single Neopixel. The pixel index is the index of the neopixel. The color table index [0-31] is the index in the color table of the color to set the Neopixel.

Example: Configure the first Gen2 card (addr 0x20). Configure the third neopixel, to use the fifth color table entry.

Command: 0x20 | 0x16 | 0x02 | 0x04 | 0xf6

Response: The card removes the command from the communication stream.

7.23 Set Solenoid Input, 0x17

CardAddr + 0x17 + InputIndex + SolIndex (1 byte) + CRC8

Change a solenoid to be triggered or not triggered by an input. The input index contains the index of the input [0-31]. The solenoid index contains the index of the solenoid [0 – 15]. If the msb of the SolIndex is set (0x80), the solenoid will stop using the input switch.

Example: Configure the first Gen2 card (addr 0x20). Configure the sixth solenoid to use the fourth input to trigger the solenoid.

Command: 0x20 | 0x17 | 0x03 | 0x05 | 0x8f

Response: The card removes the command from the communication stream.

Example: Configure the third Gen2 card (addr 0x22). Configure the eighth solenoid [wing 1] to stop using the default input which is the twelfth input.

Command: 0x22 | 0x17 | 0x0b | 0x87 | 0x8c

Response: The card removes the command from the communication stream.

7.24 Upgrade Other Board, 0x18

CardAddr + 0x18 + CRC8

Used when multiple cards are updated one at a time in a chain of cards. This command configures a card to ignore all data that it receives and simply pass it through to the next card. This is required because the Cypress bootloader which is used to upgrade the firmware must think that it is only talking to the one card that it is trying to upgrade. A string of 65 EOM bytes (0xff) will forward those bytes to the next card and reset the card to return to normal operation.

Example: Configure the second Gen2 card (addr 0x21) as a pass-thru.

Command: 0x21 | 0x18 | 0x24

Response: The card removes the command from the communication stream.

7.25 Read Matrix Input, 0x19

CardAddr + 0x19 + Data (8 bytes) + CRC8

The card returns the current state of each bit in the switch matrix. The command contains eight blank data bytes to be filled out by the card. Switch matrix column 0 data is returned as the first byte. Note: active switches (closed) are returned as a “1” bit. Active switches are aggregated and not cleared until the host reads the matrix inputs. This guarantees that an active switch will not be missed.

Example: Read first Gen2 card (addr 0x20) which is configured as a switch matrix. Column 0 reports all the switches active, while column 2 reports the least significant 4 bits as active.

Command: 0x20 | 0x19 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x33

Response: 0x20 | 0x19 | 0xff | 0x00 | 0x0f | 0x00 | 0x00 | 0x00 | 0x00 | 0x00 | 0x78

7.26 Inventory Command, 0xf0

The inventory command is used to figure out the number of cards and the position of the cards. When a card receives the inventory command, it listens and remembers the card addresses which match its card type. When it receives the end of message, it inserts its card address and increments the last matching card type address that it saw.

Example: Send the inventory command and find out that there are three Gen2 cards.

Command: 0xf0 | 0xff

Response: 0xf0 | 0x20 | 0x21 | 0x22 | 0xff

7.27 End of Message, 0xff

End of message is needed at the end of an inventory command. It can also be used at the end of any command chain.

8 CRC 8 Generation

Gen2 commands append a CRC8 to the end of transfers which matches the CCITT-CRC8. The generator polynomial is x^8+x^2+x+1 with an initial value of 0xff. The following lists Python code can also be found in the repository at Python\CalcCrc8\CalcCrc8.py

```
#!/usr/bin/env python
#
#============================================================================
## @mainpage
#
#                           OOOO
#                         OOOOOOOO
#         PPPPPPPPPPPPP   OOO    OOO   PPPPPPPPPPPPP
#       PPPPPPPPPPPPPP   OOO      OOO   PPPPPPPPPPPPPP
#      PPP          PPP   OOO      OOO   PPP          PPP
#     PPP           PPP   OOO      OOO   PPP           PPP
#     PPP           PPP   OOO      OOO   PPP           PPP
#     PPP           PPP   OOO      OOO   PPP           PPP
#      PPP          PPP   OOO      OOO   PPP          PPP
#       PPPPPPPPPPPPPP   OOO      OOO   PPPPPPPPPPPPPP
#        PPPPPPPPPPPPP   OOO      OOO   PPP
#                  PPP   OOO      OOO   PPP
#                  PPP   OOO      OOO   PPP
#                  PPP   OOO      OOO   PPP
#                  PPP    OOO    OOO    PPP
#                  PPP     OOOOOOOO     PPP
#                PPPPP       OOOO      PPPPP
#
#  This program is free software: you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation, either version 3 of the License, or
```

```
#  (at your option) any later version.
#
#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.
#
#  You should have received a copy of the GNU General Public License
#  along with this program.  If not, see <http://www.gnu.org/licenses/>.

#===============================================================================
##
# @file    CalcCrc8.py
# @author  Hugh Spahr
# @date    6/19/2014
#
# @note    Open Pinball Project
# @note    Copyright 2015, Hugh Spahr
#
# @brief Calculate the CRC8 of a message.  Calculates using nibbles and
#  bytes.

#===============================================================================

import sys
import os

## Main
#
#  Read passed in arguments.
#
#  @param  argv          [in]   Passed in arguments
#  @return None
def main(argv=None):

    end = False

    if argv is None:
        argv = sys.argv
    for arg in argv:
        if arg.startswith('-?'):
            print "python CalcCrc8.py [OPTIONS]"
            print "    -?                Options Help"
            end = True
    if end:
        return 0

    CRC8Lookup = [0x00, 0x07, 0x0e, 0x09, 0x1c, 0x1b, 0x12, 0x15, \
                  0x38, 0x3f, 0x36, 0x31, 0x24, 0x23, 0x2a, 0x2d]

    CRC8ByteLookup = \
        [ 0x00, 0x07, 0x0e, 0x09, 0x1c, 0x1b, 0x12, 0x15, \
          0x38, 0x3f, 0x36, 0x31, 0x24, 0x23, 0x2a, 0x2d, \
          0x70, 0x77, 0x7e, 0x79, 0x6c, 0x6b, 0x62, 0x65, \
          0x48, 0x4f, 0x46, 0x41, 0x54, 0x53, 0x5a, 0x5d, \
          0xe0, 0xe7, 0xee, 0xe9, 0xfc, 0xfb, 0xf2, 0xf5, \
          0xd8, 0xdf, 0xd6, 0xd1, 0xc4, 0xc3, 0xca, 0xcd, \
          0x90, 0x97, 0x9e, 0x99, 0x8c, 0x8b, 0x82, 0x85, \
```

```
            0xa8, 0xaf, 0xa6, 0xa1, 0xb4, 0xb3, 0xba, 0xbd, \
            0xc7, 0xc0, 0xc9, 0xce, 0xdb, 0xdc, 0xd5, 0xd2, \
            0xff, 0xf8, 0xf1, 0xf6, 0xe3, 0xe4, 0xed, 0xea, \
            0xb7, 0xb0, 0xb9, 0xbe, 0xab, 0xac, 0xa5, 0xa2, \
            0x8f, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9d, 0x9a, \
            0x27, 0x20, 0x29, 0x2e, 0x3b, 0x3c, 0x35, 0x32, \
            0x1f, 0x18, 0x11, 0x16, 0x03, 0x04, 0x0d, 0x0a, \
            0x57, 0x50, 0x59, 0x5e, 0x4b, 0x4c, 0x45, 0x42, \
            0x6f, 0x68, 0x61, 0x66, 0x73, 0x74, 0x7d, 0x7a, \
            0x89, 0x8e, 0x87, 0x80, 0x95, 0x92, 0x9b, 0x9c, \
            0xb1, 0xb6, 0xbf, 0xb8, 0xad, 0xaa, 0xa3, 0xa4, \
            0xf9, 0xfe, 0xf7, 0xf0, 0xe5, 0xe2, 0xeb, 0xec, \
            0xc1, 0xc6, 0xcf, 0xc8, 0xdd, 0xda, 0xd3, 0xd4, \
            0x69, 0x6e, 0x67, 0x60, 0x75, 0x72, 0x7b, 0x7c, \
            0x51, 0x56, 0x5f, 0x58, 0x4d, 0x4a, 0x43, 0x44, \
            0x19, 0x1e, 0x17, 0x10, 0x05, 0x02, 0x0b, 0x0c, \
            0x21, 0x26, 0x2f, 0x28, 0x3d, 0x3a, 0x33, 0x34, \
            0x4e, 0x49, 0x40, 0x47, 0x52, 0x55, 0x5c, 0x5b, \
            0x76, 0x71, 0x78, 0x7f, 0x6a, 0x6d, 0x64, 0x63, \
            0x3e, 0x39, 0x30, 0x37, 0x22, 0x25, 0x2c, 0x2b, \
            0x06, 0x01, 0x08, 0x0f, 0x1a, 0x1d, 0x14, 0x13, \
            0xae, 0xa9, 0xa0, 0xa7, 0xb2, 0xb5, 0xbc, 0xbb, \
            0x96, 0x91, 0x98, 0x9f, 0x8a, 0x8d, 0x84, 0x83, \
            0xde, 0xd9, 0xd0, 0xd7, 0xc2, 0xc5, 0xcc, 0xcb, \
            0xe6, 0xe1, 0xe8, 0xef, 0xfa, 0xfd, 0xf4, 0xf3 ]

    while (not end):
        print "Enter message [ex:  0x11 0x22 0x33]:"
        msg = sys.stdin.readline()
        msgBytes = msg.split()
        if (len(msgBytes) != 0):
            msgInts = []
            for indByte in msgBytes:
                # If separators used from brdIntf document, ignore them
                if indByte != '|':
                    msgInts.append(int(indByte, 16))
            crc8 = 0xff
            crc8Byte = 0xff
            for indInt in msgInts:
                crc8 = (((crc8 << 4) & 0xf0) ^ CRC8Lookup[ \
                    (((crc8) ^ (indInt)) >> 4) & 0x0f])
                crc8 = (((crc8 << 4) & 0xf0) ^ CRC8Lookup[ \
                    (((crc8 >> 4) & 0x0f) ^ (indInt)) & 0x0f])
                crc8Byte = CRC8ByteLookup[crc8Byte ^ indInt];
            print "CRC8 = 0x%02x" % crc8
            print "CRC8Byte = 0x%02x" % crc8Byte
        else:
            end = True
    return (0)

if __name__ == "__main__":
    sys.exit(main())
```

9 Updating multiple boards

Commands have been added to allow multiple OPP boards to be updated in a chain of boards. Since updating boards requires only a single board to be talking to the host computer, the other boards must

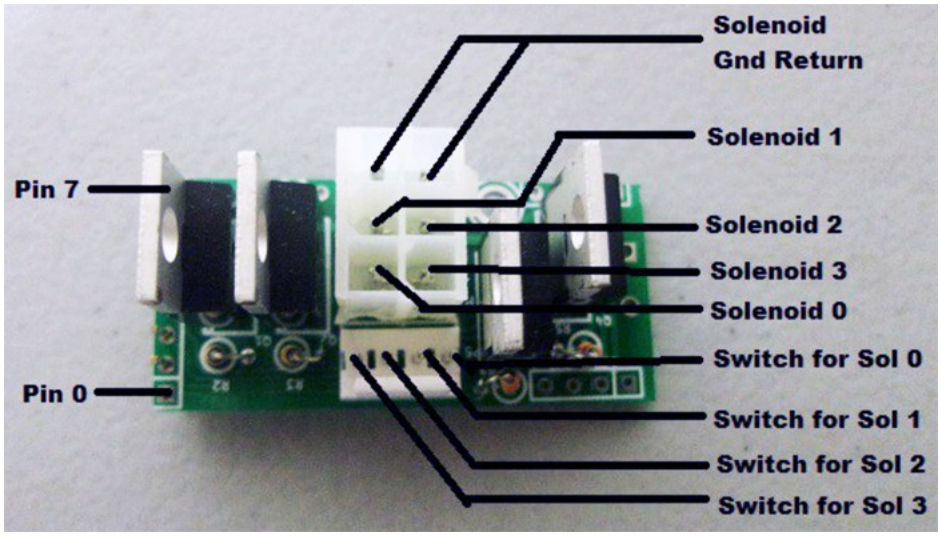
be placed in pass-thru mode. (Simply receive data bytes and transmit them to the next card without interpreting them.) To exit pass-thru mode, 65 EOM bytes (0xff) must be received. The reason for 65 bytes is because the bootloader uses a maximum command length of 64 bytes, so a string of 65 EOM bytes should never occur when communicating with the bootloader.

The following sequence needs to be followed to program a board that is part of a chain of boards:

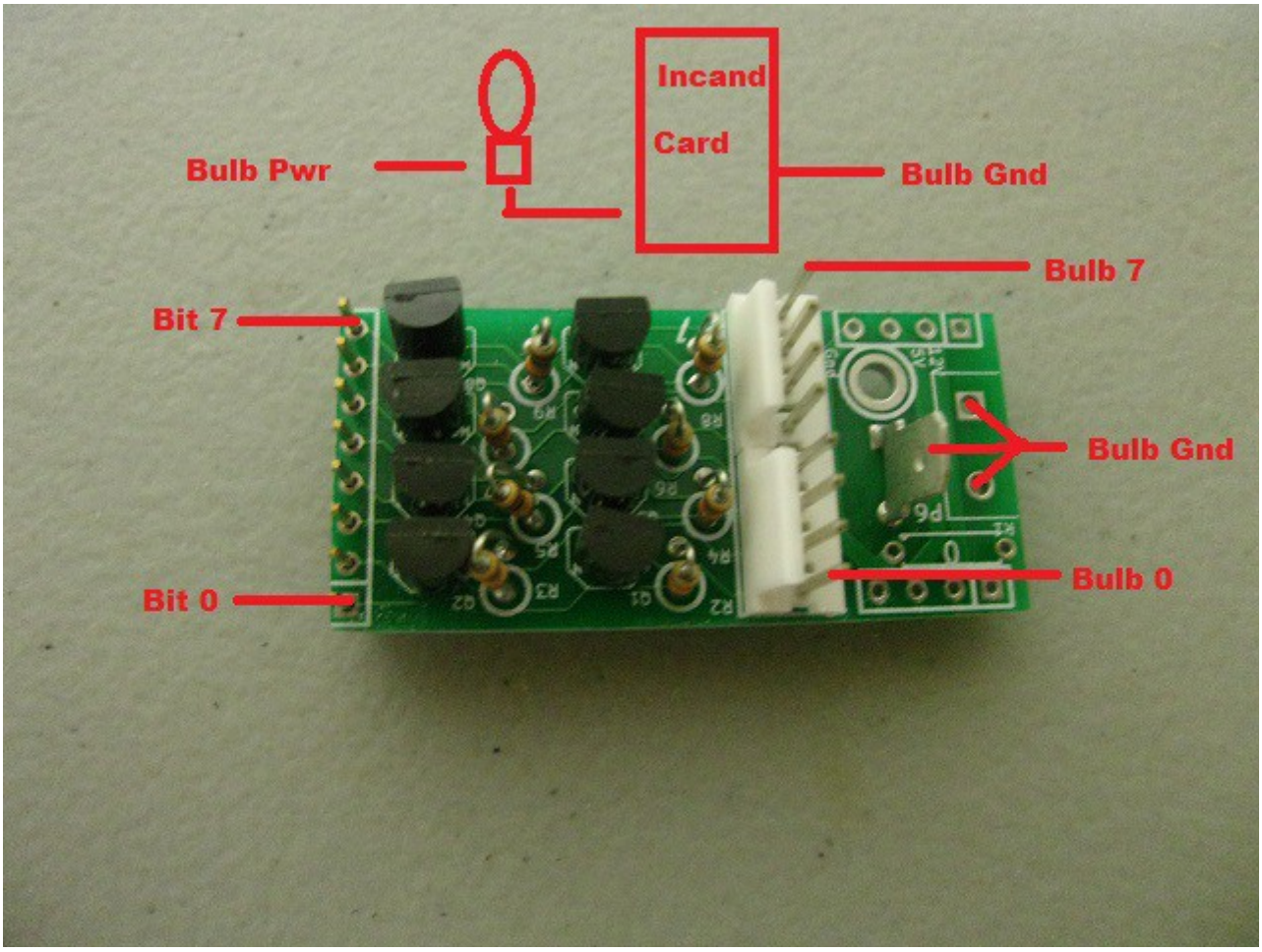
1. Send RS232I_GET_SER_NUM to get the serial number (it will be erased, so it must be reprogrammed)
2. Send RS232I_UPGRADE_OTHER_BRD to all the boards that aren't being upgraded.
3. Send RS232I_GO_BOOT to the board that is being upgraded.
4. Disconnect from the serial port
5. Run the cyflash command to upgrade one board. In windows an example of a valid cyflash command is "c:\Python27\Python.exe -m cyflash.__main__ --serial COM[serial-device] --serial_baudrate 115200 ..\..\Creator\Gen2Images\Gen2.rev0.3.0.0.cyacd"
6. Reconnect to the serial port
7. Send 65 EOM bytes for each card that is in pass-thru mode (probably not necessary)
8. Send RS232I_INVENTORY to re-assign addresses to cards
9. Send RS232I_SET_SER_NUM with the serial number read in step 1 to reprogram the serial number for the updated card.
10. Repeat above 9 steps for each card in the chain.

10 Board Connector Positions

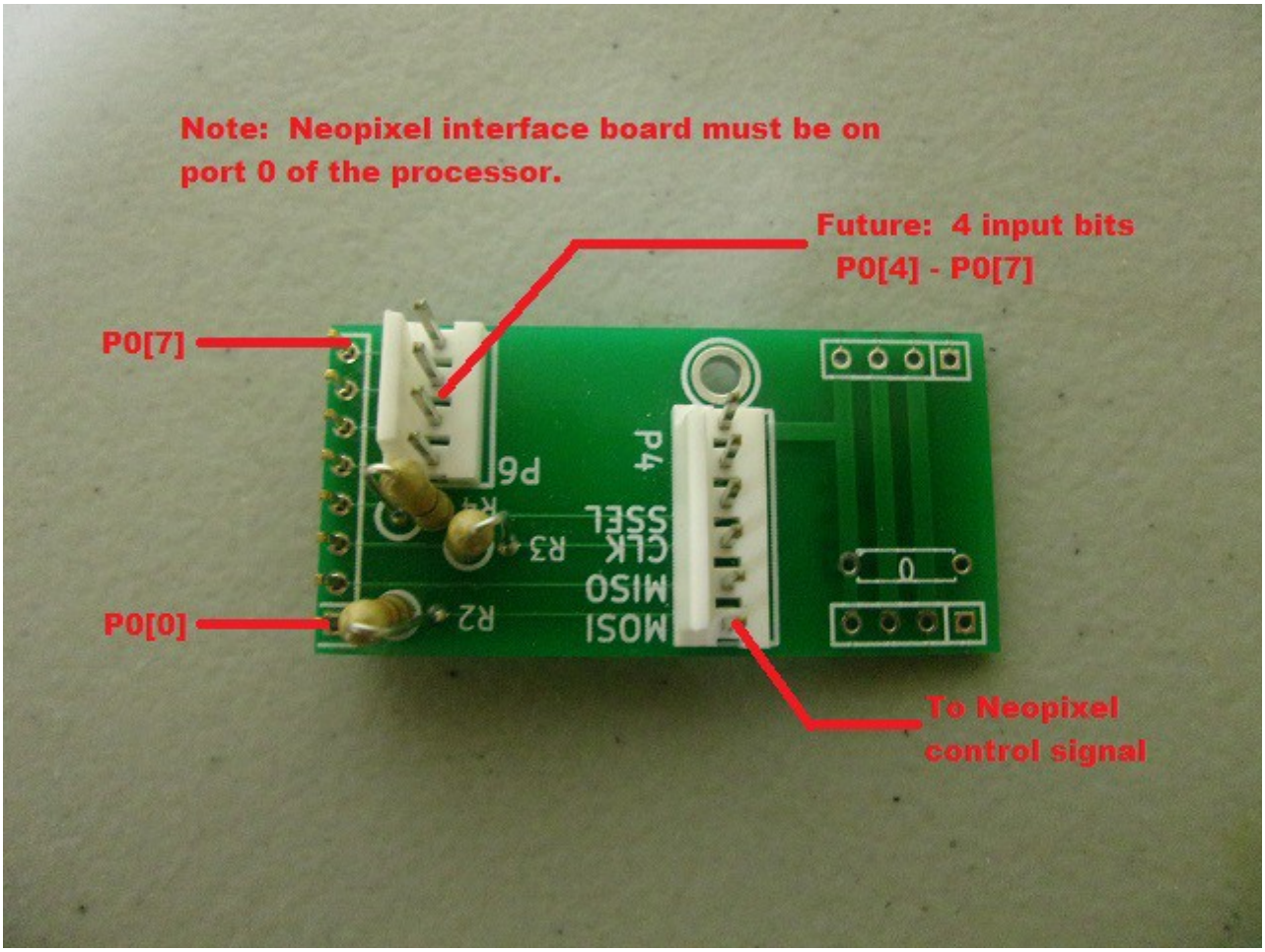
10.1 Solenoid Wing



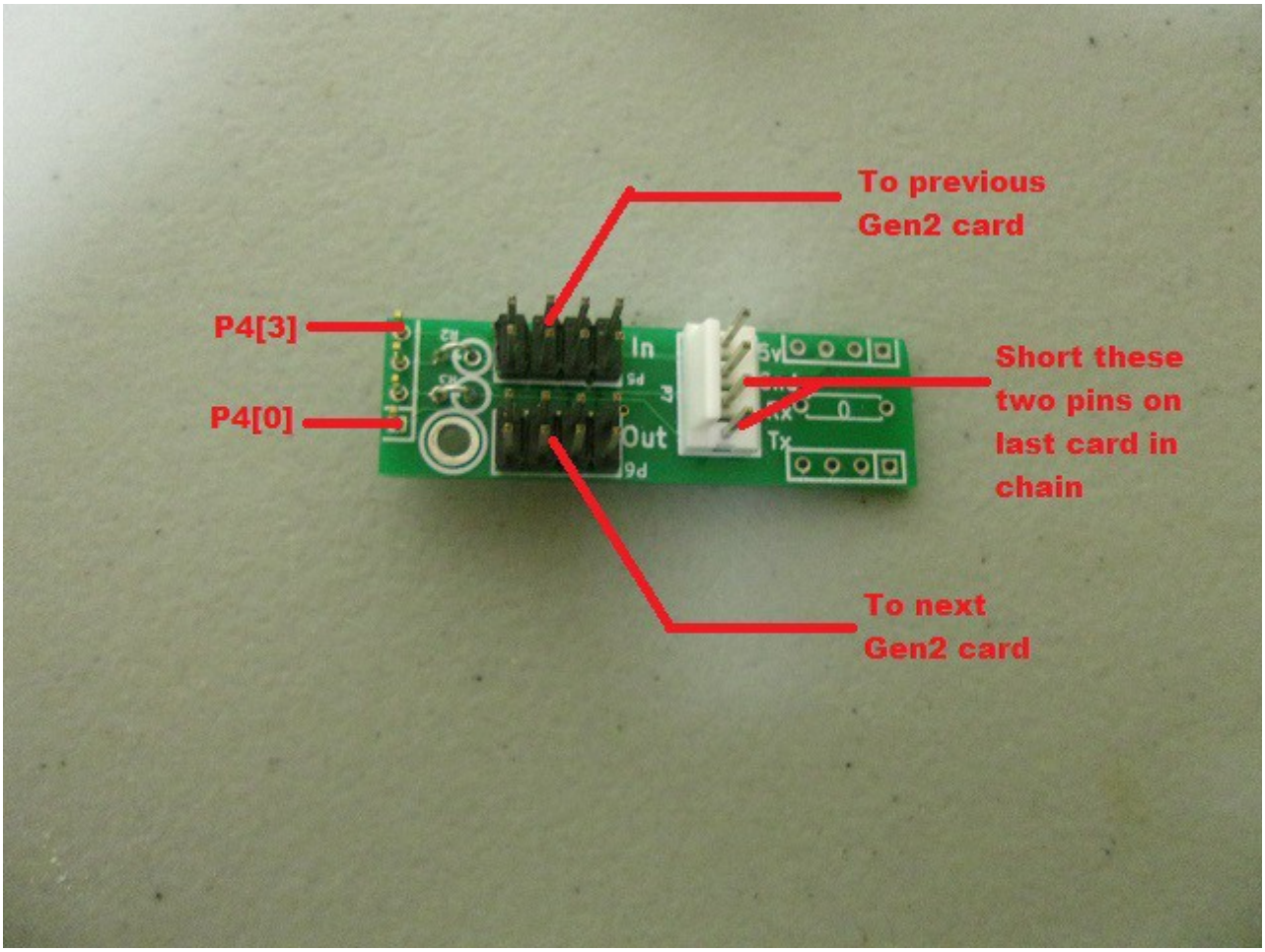
10.2 Incandescent Wing



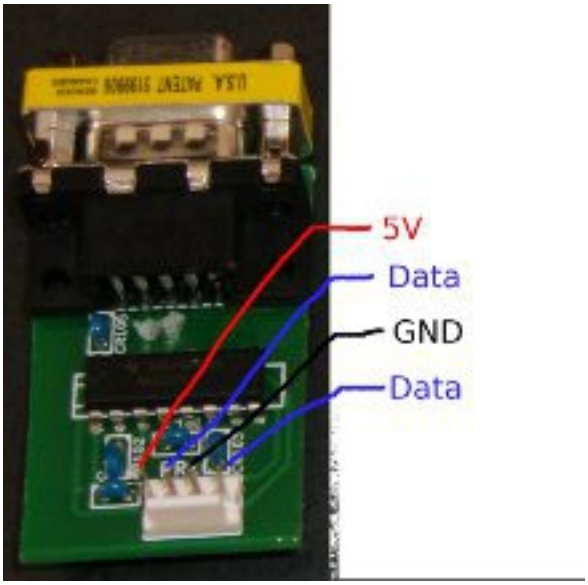
10.3 Input/Neo Wing



10.4 Interface Wing

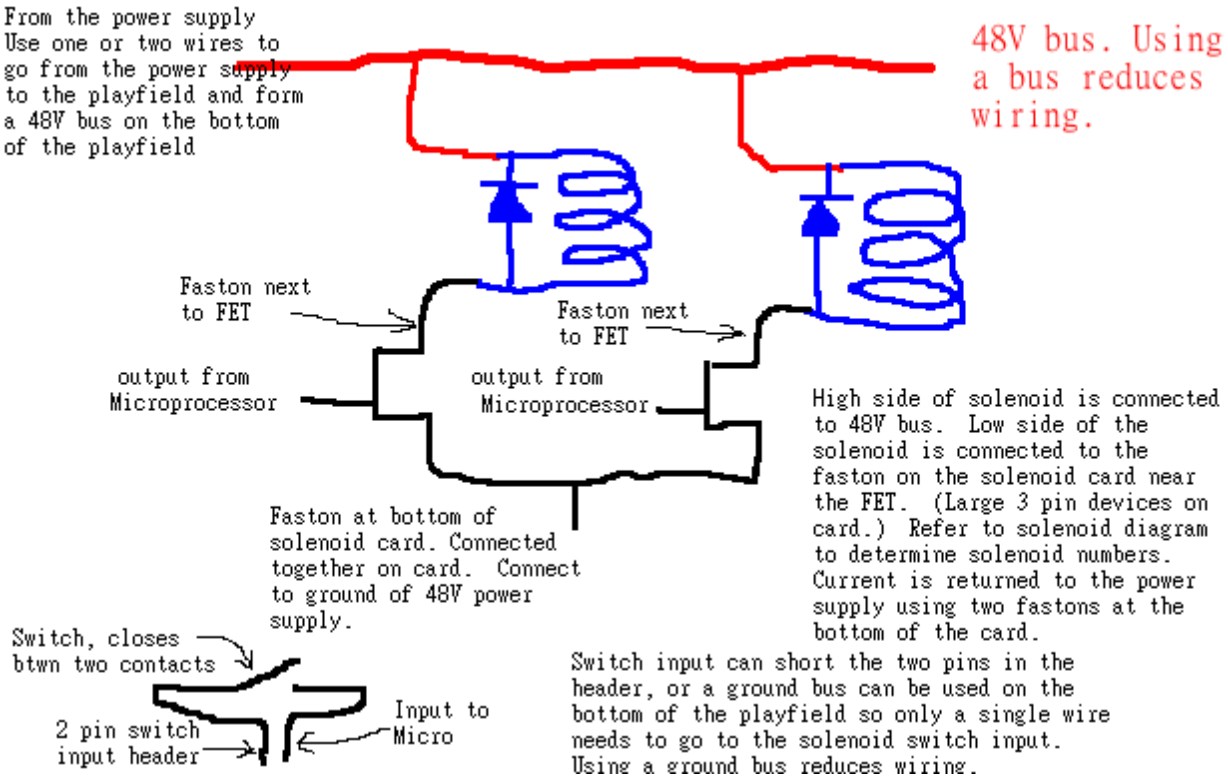


10.5 RS232 Interface



11 Wiring

11.1 Solenoid Card Wiring



11.2 Input Card Wiring

Switch input wiring is the same as the solenoid switch input wiring. Each connector has a ground pin and a pin that is an input to the processor. The processor detects that the switch is closed by looking for a low (grounded) input. The processor has an internal pull up so when the pin is not connected, it is seen as high.