ICE Debug Board FPGA Interface protocol

Ben Kempke, Pat Pannuto {bpkempke,ppannuto}@umich.edu

Revision 0.2.6 — October 17, 2013

Introduction

The ICE board is a dual-purpose development board, enabling both high-speed programming (via DMA) of M3 chips and interfacing with live M3 sessions and external peripherals.

This means that libraries must be able to gracefully handle asychronous, unexpected events. In particular, some semantics resemebling transactions would be desirable, as well as an effort to maintain event ordering over the serial communications link. The consequence is that this document should be viewed less as a protocol specification and more of a living document as we explore the best semantics for this domain.

At a high-level, every message is a composition of an *event id*, a *message type* and an optional *message*. Every message sent *to* the ICE board **MUST** be replied to by either an ACK or NAK response. Every message sent *from* the ICE board **MUST NOT** be acknowledged. The rationale for this asymmetry is to simplify the ICE hardware design. With this mechanism, the hardware is not obligated to preserve any communication state beyond the immediate running context. The event ids provide a total ordering of the event history, insofar as is possible.

Contents

1 Event Ids

1.1 Motivation

Some decisions in microcontrollers necessarily race. As contrived example, if two GPIO pins are defined as interrupts but are electrically connected in the external circuit, and then line is pulled high, which interrupt fires first? While the decision is arbitrary, there is motivation to define an ordering of events in the system. The events observed by ICE can be replayed in the M-ultaor for debugging, but the is only possible if they can be accurately re-created.

In practice, this diverges to two ideas:

1.1.1 Concurrent Events

Two events could be labeled with the same event id, indicating that they occurred too close together in time for the ICE to distinguish them. This would require divergence in the simulator for debugging runs; do-able, but not trivial.

1.1.2 I/O Pass-Through

Currently, the FPGA and the M3 share GPIO pins (Nx2 header):

```
          /-----\
FPGA ---- | o o | ----- M3
          \-----/
          GPIO N
```

Instead perhaps we can explicitly pass all I/O through the FPGA:

```
/---\
| o | ---- FPGA -- {TP} -- M3
\---/
GPIO N
```

This has the disadvantage of doubling the number of FPGA I/O pins consumed for each of the M3 I/O pins. It does enable the FPGA to strictly define an order of events. We can possibly explore this with the current board by mapping GPIOs 16-23 as FPGA-input only, jumpering over GPIOs 8-15 and having the FPGA drive those, leaving GPIOs 0-7 as the first one.

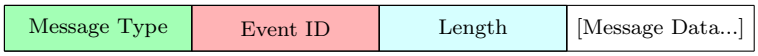
1.2 Implementation

Event IDs are an unsigned single byte number. They define a total ordering the events actually occurred in the system. In particular, if a command message is being sent to set GPIO 0 (an output) high at the same time that GPIO 1 (an input) goes high, the ordering will be `Event N: GPIO 1 --> High` then `Event N+1: GPIO 0 --> High`. With 1.1.2's pass through, perhaps the GPIO 1 transition could be delayed during command message reception, but this is for further exploration.

For message format consistency, event ids are included in both directions of communication, however the field may be safely ignored by the FPGA. The FPGA itself must by definition have some order of I/O events it processed, which are encapsulated by these event ids. The event id of a control message is assigned by the FPGA whenever it actually processes the event and is indicated to the controller via the ACK message.

2 Messages / Base Protocol

Messages are composed of a one-byte message type identifier followed by a one-byte event identifier followed by a one-byte unsigned length indicator followed by an optional message component. Visually:



Effort should be made to keep type identifiers within the ASCII range where reasonable, mapped to appropriate letters.

There are two types of transactions defined: *synchronous* and *asynchronous*. A synchronous message is one initiated by the controlling PC and must be responded to by a {N}ACK from the ICE board. An asychronous message is generated by the ICE board in response to a hardware event. Only one synchronous message is permitted to be live at any given time. The ICE board may send any number of asychronous messages before responding to the synchronous message.

The protocol in use is undefined until a version request is ACKed. The reception of a *NAK with reason* with a preferred version is **NOT** sufficient to establish a version, the controller **MUST** explicilty send another **v**ersion request and receive an ACK to establish the protocol version in use.

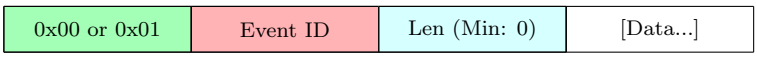
Only 'V' and 'v' messages are valid until a version has been negotiated. Any asynchronous hardware events may either be queued or discarded.

The base protocol defines the following immutable message types and their properties:

`0x00` ACK

`0x01` NAK

Synchronous Response



- **0:** ACK. Indicates success.
- **1:** NAK. Generic error code indicating failure.

– Unless otherwise specified...

- The remaining bytes shall be an ASCII-encoded string composed only of standard printable characters. The string shall not be NUL-terminated. The contents of this string is not well-defined and is expected to be something human-readable and useful.

– If specified...

- The response is permitted to be implementation defined.

`0x56 ('V')` – **Query Versions**

Synchronous Request

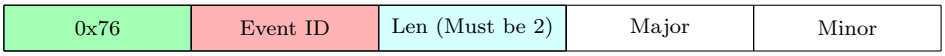
– This message queries the protocol version(s) this ICE implementation understands.

– If multiple versions are supported, they should be listed in order of version preference.

– Response:

`0x76 ('v')` – **Request to use version**

Synchronous Request



- The message shall be composed of exactly two bytes.
- Each byte shall be an unsigned number.
- The first byte shall be considered a major version number.
- The second byte shall be considered a minor version number.
- Response:

  - If the selected version is acceptable, an ACK shall be generated.

  - Otherwise a NAK shall be generated.
  - The NAK may optionally include a preferred version or list of versions in the same format as the 'V' response.
- This message is not queriable.

`0x58 ('X')` – **eXtension**

`0x78 ('x')` – **extension**

- These characters are reserved for future e**X**tentions.
- No further specification is defined here.

3 Protocols

The following protocols are currently well-defined:

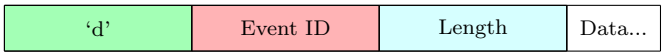
Contents

3.1 Version 0.1

Contents

3.1.1 0x64 'd' – Discrete interface I2C message

Synchronous Request, Asynchronous Message



The bytes in this message compose an I2C transaction.

The length field of this message necessarily limits the maximum message size to 255 bytes (addr + 254 data). Longer messages should be fragmented.

The sentinel value 255 for length indicates a *fragmented* message.

- A message fragment **MUST** be exactly 255 bytes long.
- A fragment message **MUST** be followed by another fragment, or terminated by a regular **d**iscrete message.
  - The ICE board is permitted to interleave other, non-I2C related messages (e.g. GPIO events).
- A series of message fragments **MUST** always be terminated by a discrete with an explicit length. An I2C transaction of length 0 is permitted, e.g.:
  - An I2C transaction of length 510 (1 byte addr + 509 bytes of data) would be **three** messages. The first of length 255 (addr + bytes 0-253), the second of length 255 (bytes 254-508), and the third of length 0 (there is no more data, but the fragment series must be terminated).
- Fragments are treated as one logical message, but individual I2C bus transactions, by an ICE board. In practice this means:
  - Each fragment message must be individually ACK'd by ICE.
    → A NAK'd fragment message ends an I2C message.
    → The NAK offset is relative to the current fragment, not the whole I2C transaction.
  - Only the first fragment includes the I2C address.

- A stop bit should **NOT** be generated after a fragment, instead the I2C clock should be stretched until the next fragment has arrived.

ICE will respond with an ACK once every byte from an individual 'd' message has been ACK'd on the I2C bus.

If a byte is NAK'd on the I2C bus, ICE will respond with a NAK message of length 1 indicating the index of the first NAK'd byte (e.g. if the address is NAK'd, it will return 0).

NAK (0x01)	Event ID	Len (Must be 1)	Index of Byte NAK'd

3.1.2 `0x49 'I'` – Query I2C Configuration

Synchronous Request

0x49	Event ID	Len (Must be 1)	Parameter

These messages complement the set I2C messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

The following would query/response the address mask:

0x49	Event ID	0x01	0x61	
ACK (0x00)	Event ID	0x02	Ones Mask	Zeros Mask

3.1.3 `0x69 'i'` – Set I2C Configuration

Synchronous Request

The first byte of the message shall define which parameter is to be configured.

`0x69 0x63 'ic'` – Set I2C Clock Speed

'i'	Event ID	Len (Must be 2)	'c'	Clock Speed

- **Default:** `0x32` (50, 100 kHz)
- This shall be followed by a single byte valued N, where N * 2 kHz yeilds the desired clock speed. Values of N greater than 200 (400 kHz) exceed the I2C spec and may be rejected.

`0x69 0x61 'ia'` – Set ICE I2C Address

'i'	Event ID	Len (Must be 3)	'a'	Ones Mask	Zeros Mask

- **Default:** `0xff 0xff` (disabled)
- This shall be followed by two bytes, first the *ones mask* and then the *zeroes mask* as outlined below. The command sets the address mask that ICE board should pretend to be a device for. Coneceptually the mask is of the form `10xx010x`, where x's signify don't care. This is conveyed as a *ones mask* and a *zeroes mask*, where each mask defines the bits that must be a one or zero respectively. For the given example, the ones mask would be `10000100` and the zeroes mask `01001010`, generating a transaction of `0x61 0x84 0x4a`.
- To disable address-faking, set any bit as both required-one and required-zero. This impossible situation is a legal setting that will never match.
  - *Note:* While it is permissable to set the last bit must-be-zero (writeable-only) or must-be-one (readable-only), it is almost certainly an error to do so.

`0x69` Responses

- NAKs for this message shall be composed of an error code, followed by an optional explanitory string.

NAK (0x01)	Event ID	Len (Min: 1)	EINVAL (0x16)	["Out of Range"]

  - `EINVAL (22,0x16):` Invalid argument.

NAK (0x01)	Event ID	Len (Min: 1)	ENODEV (0x13)

  - `ENODEV (19,0x13):` The implementation does not support changing or querying this parameter. Unless otherwise specified, it **MUST** be hardcoded to the default.

3.1.4 `0x66 'f'` – FLOW (GOC) interface message

Synchronous Request

FLOW messages are formatted the exact same as 'd'iscrete messages.

3.1.5 `0x4f 'O'` – Query optical (FLOW (GOC)) Configuration

Synchronous Request

'O' (0x4f)	Event ID	Len (Must be 1)	Parameter

These messages complement the set I2C messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

3.1.6 `0x6f 'o'` – Set optical (FLOW (GOC)) Configuration

Synchronous Request

The first byte of the message shall define which parameter is to be configured.

`0x6f 0x63 'oc'`: Clock Speed (Divider)

'o'	Event ID	Len (Must be 4)	'c'	Clock Divider

- **Default:** `0x30D400` (2 MHz / 0x30D400 = .625 Hz)
- This shall be followed by a three byte value N (MSB-first), where 2 MHz / N yields the desired clock speed.

`0x6f` Responses

- NAKs for this message shall be composed of an error code, followed by an optional explanitory string.

NAK (0x01)	Event ID	Len (Min: 1)	EINVAL (0x16)	["Out of Range"]

  - `EINVAL (22,0x16):` Invalid argument.

3.1.7 0x47 'G' – Query GPIO State / Configuration

Synchronous Request

'G' (0x47)	Event ID	Len (Must be 2)	Parameter	GPIO IDX

These messages complement the set GPIO ('g') messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

3.1.8 0x67 'g' – Set / Configure GPIO

Synchronous Request, Asynchronous Message

The first byte of this message shall be a specificer, indicating what type of GPIO action is requested.

0x67 0x6c 'gl' – GPIO Level

- The first byte of the message shall be an integer indicating the GPIO index to set. The second byte shall be valued 0 or 1, depending on the desired GPIO state.

'o' (0x67)	Event ID	Len (Must be 3)	'l' (0x6c)	GPIO IDX	GPIO Val

0x67 0x64 'gd' – GPIO direction

- The first byte of the message shall be an integer indicating the GPIO index to set the direction of. The second byte shall be valued:
  - 0: Input
  - 1: Output
  - 2: TriState (DEFAULT)

'o' (0x67)	Event ID	Len (Must be 3))	'd' (0x64)	GPIO IDX	GPIO Direction

0x67 Responses

- ENODEV (19,0x13): The requested GPIO does not exist.

NAK (0x01)	Event ID	Len (Min: 1)	ENODEV (0x13)	["No such GPIO"]

- EINVAL (22,0x16): The requested GPIO exists, but cannot be configured this way at this time.

NAK (0x01)	Event ID	Len (Min: 1)	EINVAL (0x16)	["GPIO is input"]

3.1.9 0x50 'P' – Query Power State

Synchronous Request

'P' (0x50)	Event ID	Len (Must be 1)	Parameter	PWR IDX

These messages complement the Set Power ('p') messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

3.1.10 `0x70 'p'` – Set Power State

Synchronous Request, Asynchronous Message

Set Power State messages allow direct control of set-point voltage and on/off states for various power domains on the ICE board. The first byte of this message shall be a specificer, indicating which parameter is requested. The second byte of the message shall be the power domain identifier. Currently implemented power domain identifiers are:

- 0: M3 0.6V (0.675V Default)
- 1: M3 1.2V (1.2V Default)
- 2: M3 VBatt (3.8V Default)

`0x70 0x76 'pv'` – Voltage State

- The first byte of the message shall be a single byte indicating the power domain identifier to set. The second byte (*v_set*) shall specify the voltage according to the equation:

$$V_{out} = (0.537 + 0.0185 * v\_set) * V_{default}$$

Valid values for *v_set* range from `0` to `31`

'p' (0x70)	Event ID	Len (Must be 3)	'v' (0x76)	PWR IDX	*v_set*

`0x70 0x6f 'po'` – On/Off State

- The first byte of the message shall be a single byte indicating the power domain identifier to set. The second byte shall be valued `0` or `1`, depending on the desired On/Off state.

'p' (0x70)	Event ID	Len (Must be 3)	'o' (0x6f)	PWR IDX	On/Off

3.2 Version 0.2

Version 0.2 adds the following new messages:

‘Oo’ and ‘oo’. These messages allow the default on/off state of the GOC light to be set.

‘??’. This message queries the capabilities of the ICE board.

‘?B’, ‘?b’. These messages get and set the ICE baud rate.

‘B’, ‘b’, ‘M’, and ‘m’. These messages control the MBus interface.

‘e’. This command injects messages on the EIN debug port.

Contents

3.2.1 0x3f '?' – Query ICE

Synchronous Request

The first byte of this message shall be a specificer, indicating what type of query is requested.

0x3f 0x3f '??' – Query capabilities

- This message shall query the capabilities of this ICE board. Not all boards have every hardware frontend. Use this message to query the capabilities of a given ICE board.

'?' (0x3f)	Event ID	Len (Must be 1)	'?' (0x3f)

- This message shall be responded to with a list of all top-level identifiers that the ICE board is capable of acting usefully upon. That is, if the ICE firmware understands 'd' messages but does not have an I2C frontend 'd' shall be omitted. Sub-types are not specified by this command. That is, if a version 0.2 ICE includes 'o' in its response it is assumed to understand both the 'oc' and 'oo' messages.
- Both set and query commands should be included in this list. That is, if a GOC frontend is present, both 'o' and 'O' should be included.
- As example, a version 0.2 ICE with no physical I2C frontend would respond:

ACK (0x00)	Event ID	Len (11)	'?_IifOoGgPp' (0x3f5f4969664f6f47675070)

0x3f 0x62 '?b' – Query baudrate

- This message shall mirror the '_b' message and report the currently set baud rate.

0x3f Responses

- NAKs for this message shall be composed of an error code, followed by an optional explanatory string.

NAK (0x01)	Event ID	Len (Min: 1)	EINVAL (0x16)	["Out of Range"]

  - EINVAL (22,0x16): Invalid argument.

3.2.2 0x5f '_' – Configure ICE

Synchronous Request

The first byte of this message shall be a specificer, indicating what type of query is requested.

'_' commands are setters for the '?' getters if appropriate.

`0x5f 0x62 '_b'` – **Set baudrate**

'_'	Event ID	Len (Must be 3)	'b'	Clock Divider

- This message shall set the baud rate for future messages. The new baud rate shall take effect after the ACK for this request.
- This shall be followed by a two byte value N (MSB-first), where $N = \lceil \frac{20\ MHz}{BaudRate} \rceil$.
- **Default:** `0x00AE` (20 MHz / 0x00AE ≈ 115200 Hz)
- The following message would set the ICE speed to 3 Megabaud:

'_' (0x5f)	Event ID	Length (3)	'b' (0x62)	0x0007

3.2.3 `0x64 'd'` – Discrete interface I2C message

Synchronous Request, Asynchronous Message

'd'	Event ID	Length	Data...

The bytes in this message compose an I2C transaction.

The length field of this message necessarily limits the maximum message size to 255 bytes (addr + 254 data). Longer messages should be fragmented.

The sentinel value `255` for length indicates a *fragmented* message.

- A message fragment **MUST** be exactly 255 bytes long.
- A fragment message **MUST** be followed by another fragment, or terminated by a regular **d**iscrete message.
  - The ICE board is permitted to interleave other, non-I2C related messages (e.g. GPIO events).
- A series of message fragments **MUST** always be terminated by a discrete with an explicit length. An I2C transaction of length 0 is permitted, e.g.:
  - An I2C transaction of length 510 (1 byte addr + 509 bytes of data) would be **three** messages. The first of length 255 (addr + bytes 0-253), the second of length 255 (bytes 254-508), and the third of length 0 (there is no more data, but the fragment series must be terminated).
- Fragments are treated as one logical message, but individual I2C bus transactions, by an ICE board. In practice this means:
  - Each fragment message must be individually ACK'd by ICE.
    → A NAK'd fragment message ends an I2C message.
    → The NAK offset is relative to the current fragment, not the whole I2C transaction.
  - Only the first fragment includes the I2C address.
  - A stop bit should **NOT** be generated after a fragment, instead the I2C clock should be stretched until the next fragment has arrived.

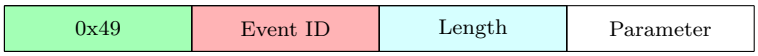
ICE will respond with an ACK once every byte from an individual 'd' message has been ACK'd on the I2C bus.

If a byte is NAK'd on the I2C bus, ICE will respond with a NAK message of length 1 indicating the index of the first NAK'd byte (e.g. if the address is NAK'd, it will return 0).

NAK (0x01)	Event ID	Len (Must be 1)	Index of Byte NAK'd

3.2.4 `0x49 'I'` – Query I2C Configuration

Synchronous Request

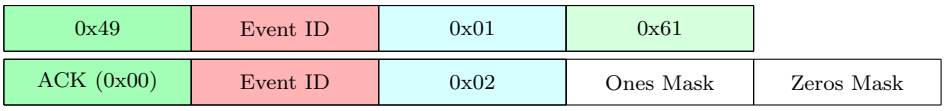


These messages complement the set I2C messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

The following would query/response the address mask:

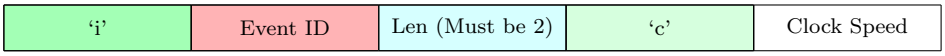


3.2.5 `0x69 'i'` – Set I2C Configuration

Synchronous Request

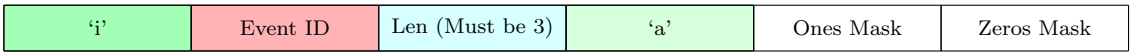
The first byte of the message shall define which parameter is to be configured.

`0x69 0x63 'ic'` – Set I2C Clock Speed



- **Default:** `0x32` (50, 100 kHz)
- This shall be followed by a single byte valued N, where N * 2 kHz yeilds the desired clock speed. Values of N greater than 200 (400 kHz) exceed the I2C spec and may be rejected.

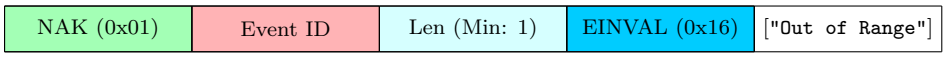
`0x69 0x61 'ia'` – Set ICE I2C Address



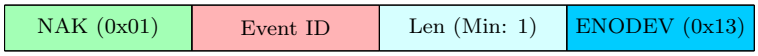
- **Default:** `0xff 0xff` (disabled)
- This shall be followed by two bytes, first the *ones mask* and then the *zeroes mask* as outlined below. The command sets the address mask that ICE board should pretend to be a device for. Coneceptually the mask is of the form `10xx010x`, where x's signify don't care. This is conveyed as a *ones mask* and a *zeroes mask*, where each mask defines the bits that must be a one or zero respectively. For the given example, the ones mask would be `10000100` and the zeroes mask `01001010`, generating a transaction of `0x61 0x84 0x4a`.
- To disable address-faking, set any bit as both required-one and required-zero. This impossible situation is a legal setting that will never match.
  - *Note:* While it is permissable to set the last bit must-be-zero (writeable-only) or must-be-one (readable-only), it is almost certainly an error to do so.

`0x69` **Responses**

- NAKs for this message shall be composed of an error code, followed by an optional explanitory string.



  - `EINVAL (22,0x16):` Invalid argument.



  - `ENODEV (19,0x13):` The implementation does not support changing or querying this parameter. Unless otherwise specified, it **MUST** be hardcoded to the default.

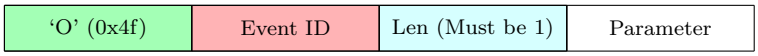
3.2.6 `0x66 'f'` – Debug (GOC/EIN) interface message

Synchronous Request

Debug interface messages are formatted the exact same as 'd'iscrete messages.

3.2.7 `0x4f 'O'` – Query debug (GOC/EIN) Configuration

Synchronous Request



These messages complement the set `'o'` messages.

The 'Parameter' field is the parameter specifier to query.

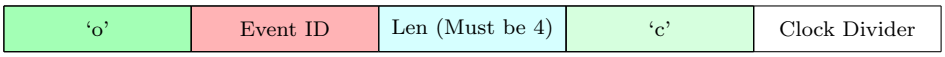
An ACK response should mimic the corresponding set message.

3.2.8 `0x6f 'o'` – Set debug (GOC/EIN) Configuration

Synchronous Request

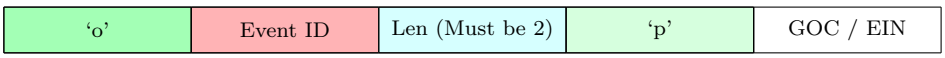
The first byte of the message shall define which parameter is to be configured.

`0x6f 0x63 'oc'`: Clock Speed (Divider)



- **Default:** `0x30D400` (2 MHz / 0x30D400 = .625 Hz)
- This shall be followed by a three byte value N (MSB-first), where 2 MHz / N yields the desired clock speed.

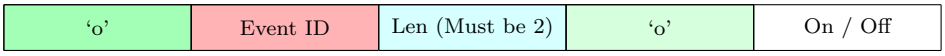
`0x6f 0x70 'op'`: Debug mode



- **Default:** `0x0` (EIN)
- Specifies which interface to use when transferring debug data. Possible values:

  0 EIN
  1 GOC

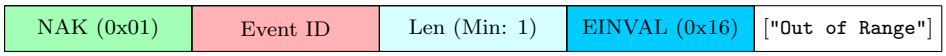
0x6f 0x6f 'oo': GOC light On/Off



- **Default:** 0x0 (Off)
- This shall be followed by either a 0 or a 1 indicating whether the default state of the light should be on or off. If the default is set to on, the light shall remain illuminated until set to off or a GOC pulse temporarily turns if off.

0x6f Responses

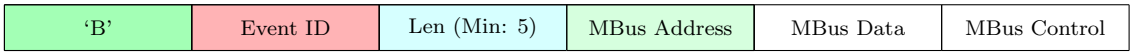
- NAKs for this message shall be composed of an error code, followed by an optional explanatory string.



  - EINVAL (22,0x16): Invalid argument.

3.2.9 0x42 'B' – MBus Snooped Message

Asynchronous Message



This shall be followed by a 32-bit MBus destination address as well as optional data. The current MBus implementation operates on 4-byte words, so the data field length must be a multiple of four. As with discrete ('d') messages, the length field limits the maximum message size to 255 bytes. Longer messages should be fragmented.

At the end of the message, snoop messages shall append one additional byte. This additional byte shall indicate the control bits of the snooped message. Control Bit 0 shall be mapped to bit 0 and Control Bit 1 shall be mapped to bit 1. The remaining bits are undefined.

This message is sent only from the ICE board to report messages it has snooped (but not ACK'd).

Messages that match both the ICE address and the snoop address (that is, would generate both 'b' and 'B' messages) shall not report snoop messages.

The ICE board shall not report snoop messages for message that it is sending.

3.2.10 0x62 'b' – MBus Message

Synchronous Request, Asynchronous Message

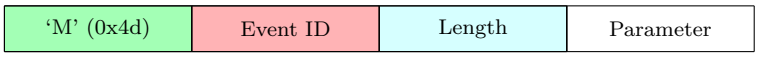
MBus messages are formatted the exact same as 'MBus Snooped' ('B') messages.

This command may be sent to the ICE board to send a message.

This message is sent from the ICE board to report messages sent to ICE. That is messages that matched the address masks assigned to ICE and that ICE has ACK'd on the bus.

3.2.11 0x4d 'M' – Query MBus Configuration

Synchronous Request



These messages complement the set of 'm' messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

3.2.12 `0x6d 'm'` – Set MBus Configuration

Synchronous Request

The first byte of the message shall define which parameter is to be configured.

`0x6d 0x6c 'ml'`: Set MBus Full Prefix

'm'	Event ID	Len (Must be 7)	'l'	Ones Mask	Zeros Mask

- **Default:** `0xffffff 0xffffff` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 6 bytes. The first three bytes shall be considered the ones mask the second three bytes shall be considered the zeros mask. The masks are 20 bits long, the bottom 4 bits of the transmitted masks shall be ignored.

`0x6d 0x73 'ms'`: Set MBus Short Prefix

'm'	Event ID	Len (Must be 3)	's'	Ones Mask	Zeros Mask

- **Default:** `0xf0 0xf0` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the bottom 4 bits of the transmitted masks shall be ignored.
- **NOTE:** Changing the short prefix after enumeration is a violation of the MBus protocol. The ICE board will permit this, but it may have unexpected consequences.

`0x6d 0x4c 'mL'`: Set MBus Snoop Full Prefix

'm'	Event ID	Len (Must be 7)	'L'	Ones Mask	Zeros Mask

- **Default:** `0xffffff 0xffffff` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 6 bytes. The first three bytes shall be considered the ones mask the second three bytes shall be considered the zeros mask. The masks are 20 bits long, the bottom 4 bits of the transmitted masks shall be ignored.
- Messages matching the snoop prefix shall be reported by a `'B'` message but will not be ACK'd on the physical bus by ICE.
- In the case of a conflict between a snoop and a match, the match takes precedence and ICE will ACK the message.

`0x6d 0x53 'mS'`: Set MBus Snoop Short Prefix

'm'	Event ID	Len (Must be 3)	'S'	Ones Mask	Zeros Mask

- **Default:** `0xf0 0xf0` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the bottom 4 bits of the transmitted masks shall be ignored.

- Messages matching the snoop prefix shall be reported by a `'B'` message but will not be ACK'd on the physical bus by ICE.
- In the case of a conflict between a snoop and a match, the match takes precedence and ICE will ACK the message.

`0x6d 0x73 'mb'`: Set MBus Broadcast Mask

'm'	Event ID	Len (Must be 1)	'b'	Ones Mask	Zeros Mask

- **Default:** `0x0f 0x0f` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the top 4 bits of the transmitted masks shall be ignored.
- These masks are the set of broadcast channels that the ICE board should ACK on the physical MBus.

`0x6d 0x53 'mB'`: Set MBus Snoop Broadcast Mask

'm'	Event ID	Len (Must be 1)	'B'	Ones Mask	Zeros Mask

- **Default:** `0x0f 0x0f` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the top 4 bits of the transmitted masks shall be ignored.
- These masks are the set of broadcast channels that the ICE board will not ACK on the physical MBus but will report as a snooped message (a `'B'` message).
- In the case of a conflict between a snoop and a match, the match takes precedence and ICE will ACK the message.

`0x6d 0x6d 'mm'`: Set master mode on/off

'm'	Event ID	Len (Must be 2)	'm'	`0x00` or `0x01`

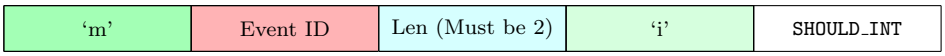
- **Default:** `0x0` (Off)
- Boolean whether ICE should act as the MBus master node (1 - on) or as a MBus member node (0 - off).

`0x6d 0x63 'mc'`: Clock Speed

'o'	Event ID	Len (4)	'c'	Clock Divider

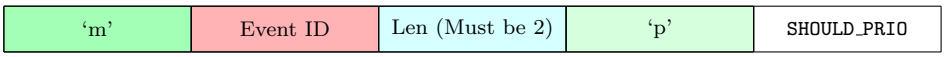
- **Default:** `0x000020` (10 MHz / 0x000020 = 312.5 kHz)
- This shall be followed by a three byte value N (MSB-first), where 10 MHz / N yields the desired clock speed.
- This configuration is meaningful only if ICE is configured as the MBus master node (see `'mm'`).

`0x6d 0x69 'mi'`: **Set should interrupt**



- **Default:** `0x0` (Off)
- NOTE: This command is not supported by the current MBus implementation.
- Configures whether ICE should interrupt the bus to transmit its next message. Possible values:
  - 0 Do not interrupt
  - 1 Interrupt (if necessary) for next message only. That is, this value is reset to 0 after the next `'b'` command sent to ICE is handled.
  - 2 Interrupt for all messages

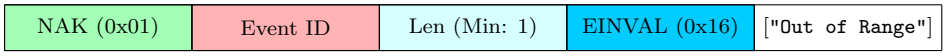
`0x6d 0x70 'mp'`: **Set should use priority arbitration**



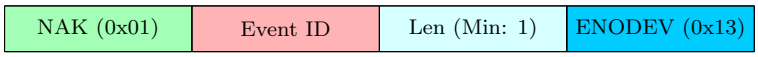
- **Default:** `0x0` (Off)
- Configures whether ICE should send high priority MBus messages. Possible values:
  - 0 Do not send priority message.
  - 1 Send all messages as high priority.

`0x6d` **Responses**

- NAKs for this message shall be composed of an error code, followed by an optional explanitory string.



  - `EINVAL (22,0x16)`: Invalid argument.



  - `ENODEV (19,0x13)`: The implementation does not support changing or querying this parameter. Unless otherwise specified, it **MUST** be hardcoded to the default.

3.2.13 `0x47 'G'` – Query GPIO State / Configuration

Synchronous Request

These messages complement the set GPIO ('g') messages.

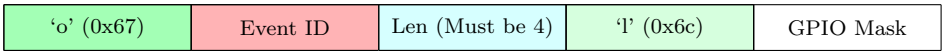
An ACK response should mimic the corresponding set message.

3.2.14 `0x67 'g'` – Set / Configure GPIO

Synchronous Request, Asynchronous Message

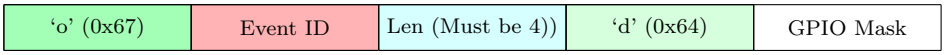
The first byte of this message shall be a specificer, indicating what type of GPIO action is requested.

`0x67 0x6c 'gl'` – GPIO Level



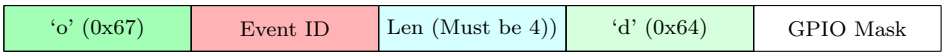
- **Default:** `0x000000` (All low)
- This shall be followed by a three byte bitmask indicating the desired output level of all 24 GPIO. This is only reflected in those GPIO which are configured as output. Each bit shall be valued:
  - 0: Low (0.0 V)
  - 1: High (1.2 V)

`0x67 0x64 'gd'` – GPIO Direction



- **Default:** `0x000000` (All input)
- This shall be followed by a three byte bitmask indicating the desired direction of all 24 GPIO. The output level of each GPIO which are configured as output are set via the 'Set GPIO Level' (`'gl'`) command. Each bit shall be valued:
  - 0: Input (DEFAULT)
  - 1: Output

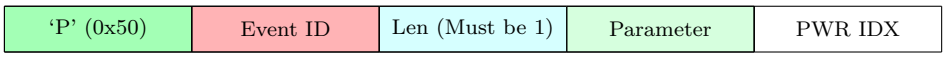
`0x67 0x69 'gi'` – GPIO Interrupt Enable



- **Default:** `0x000000` (All interrupts disabled)
- This shall be followed by a three byte bitmask indicating whether a change in each GPIO bit wil generate an GPIO level change message. Only those GPIO which are configured as input will generate interrupts. Each bit shall be valued:
  - 0: Interrupt Disabled (DEFAULT)
  - 1: Interrupt Enabled

3.2.15 `0x50 'P'` – Query Power State

Synchronous Request



These messages complement the Set Power ('p') messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

3.2.16 `0x70 'p'` – Set Power State

Synchronous Request, Asynchronous Message

Set Power State messages allow direct control of set-point voltage and on/off states for various power domains on the ICE board. The first byte of this message shall be a specificer, indicating which parameter is requested. The second byte of the message shall be the power domain identifier. Currently implemented power domain identifiers are:

- 0: M3 0.6V (0.675V Default)
- 1: M3 1.2V (1.2V Default)
- 2: M3 VBatt (3.8V Default)

`0x70 0x76 'pv'` – **Voltage State**

– The first byte of the message shall be a single byte indicating the power domain identifier to set. The second byte (v_set) shall specify the voltage according to the equation:

$$V_{out} = (0.537 + 0.0185 * v\_set) * V_{default}$$

Valid values for v_set range from `0` to `31`

'p' (0x70)	Event ID	Len (Must be 3)	'v' (0x76)	PWR IDX	v_set

`0x70 0x6f 'po'` – **On/Off State**

– The first byte of the message shall be a single byte indicating the power domain identifier to set. The second byte shall be valued `0` or `1`, depending on the desired On/Off state.

'p' (0x70)	Event ID	Len (Must be 3)	'o' (0x6f)	PWR IDX	On/Off

3.3 Version 0.3

Version 0.3 modifies the following messages:

- 'oc': EIN Clock Divider – argument increased to 3 bytes to allow for slower GOC/EIN speeds.
- 'p': Set Power State – Additional voltage domain added for USB GOC light.

Version 0.3 deprecates support for the following messages:

- 'd': Discrete interface I2C message
- 'I': Query I2C Configuration
- 'i': Set I2C Configuration

Contents

3.3.1 0x3f '?' – Query ICE

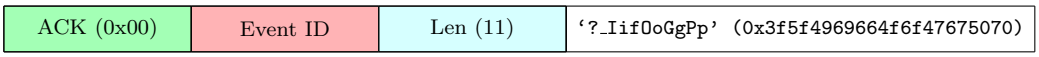
Synchronous Request

The first byte of this message shall be a specificer, indicating what type of query is requested.

0x3f 0x3f '??' – Query capabilities

- This message shall query the capabilities of this ICE board. Not all boards have every hardware frontend. Use this message to query the capabilities of a given ICE board.

- This message shall be responded to with a list of all top-level identifiers that the ICE board is capable of acting usefully upon. That is, if the ICE firmware understands 'd' messages but does not have an I2C frontend 'd' shall be omitted. Sub-types are not specified by this command. That is, if a version 0.2 ICE includes 'o' in its response it is assumed to understand both the 'oc' and 'oo' messages.
- Both set and query commands should be included in this list. That is, if a GOC frontend is present, both 'o' and 'O' should be included.
- As example, a version 0.2 ICE with no physical I2C frontend would respond:

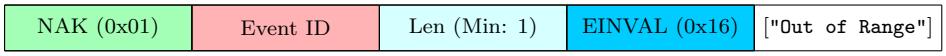


0x3f 0x62 '?b' – Query baudrate

- This message shall mirror the '_b' message and report the currently set baud rate.

0x3f Responses

- NAKs for this message shall be composed of an error code, followed by an optional explanatory string.



  - EINVAL (22,0x16): Invalid argument.

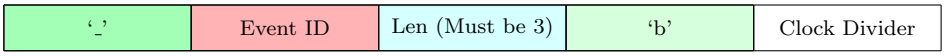
3.3.2 0x5f '_' – Configure ICE

Synchronous Request

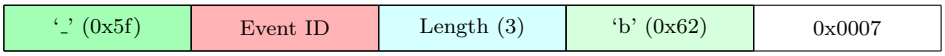
The first byte of this message shall be a specificer, indicating what type of query is requested.

'_' commands are setters for the '?' getters if appropriate.

0x5f 0x62 '_b' – Set baudrate



- This message shall set the baud rate for future messages. The new baud rate shall take effect after the ACK for this request.
- This shall be followed by a two byte value N (MSB-first), where $N = \lceil \frac{20\ MHz}{BaudRate} \rceil$.
- **Default:** 0x00AE (20 MHz / 0x00AE ≈ 115200 Hz)
- The following message would set the ICE speed to 3 Megabaud:



3.3.3 `0x66 'f'` – Debug (GOC/EIN) interface message

Synchronous Request

Debug interface messages are formatted the exact same as 'd'iscrete messages.

3.3.4 `0x4f 'O'` – Query debug (GOC/EIN) Configuration

Synchronous Request

These messages complement the set `'o'` messages.

The 'Parameter' field is the parameter specifier to query.

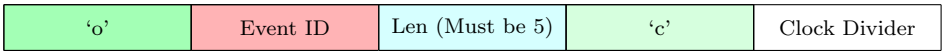
An ACK response should mimic the corresponding set message.

3.3.5 `0x6f 'o'` – Set debug (GOC/EIN) Configuration

Synchronous Request

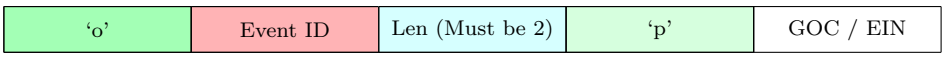
The first byte of the message shall define which parameter is to be configured.

`0x6f 0x63 'oc'`: Clock Speed (Divider)



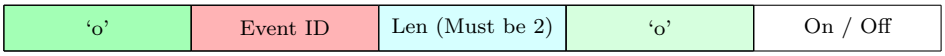
- **Default:** `0x0061A800` (4 MHz / 0x0061A800 = .625 Hz)
- This shall be followed by a bour byte value N (MSB-first), where 4 MHz / N yields the desired clock speed.

`0x6f 0x70 'op'`: Debug mode



- **Default:** `0x0` (EIN)
- Specifies which interface to use when transferring debug data. Possible values:
  0 EIN
  1 GOC

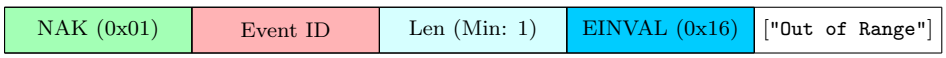
`0x6f 0x6f 'oo'`: GOC light On/Off



- **Default:** `0x0` (Off)
- This shall be followed by either a `0` or a `1` indicating whether the default state of the light should be on or off. If the default is set to on, the light shall remain illuminated until set to off or a GOC pulse temporarily turns if off.

`0x6f` Responses

- NAKs for this message shall be composed of an error code, followed by an optional explanatory string.



  - `EINVAL (22,0x16)`: Invalid argument.

3.3.6 0x42 'B' – MBus Snooped Message

Asynchronous Message

'B'	Event ID	Len (Min: 5)	MBus Address	MBus Data	MBus Control

This shall be followed by a 32-bit MBus destination address as well as optional data. The current MBus implementation operates on 4-byte words, so the data field length must be a multiple of four. As with discrete ('d') messages, the length field limits the maximum message size to 255 bytes. Longer messages should be fragmented.

At the end of the message, snoop messages shall append one additional byte. This additional byte shall indicate the control bits of the snooped message. Control Bit 0 shall be mapped to bit 0 and Control Bit 1 shall be mapped to bit 1. The remaining bits are undefined.

This message is sent only from the ICE board to report messages it has snooped (but not ACK'd).

Messages that match both the ICE address and the snoop address (that is, would generate both 'b' and 'B' messages) shall not report snoop messages.

The ICE board shall not report snoop messages for message that it is sending.

3.3.7 0x62 'b' – MBus Message

Synchronous Request, Asynchronous Message

MBus messages are formatted the exact same as 'MBus Snooped' ('B') messages.

This command may be sent to the ICE board to send a message.

This message is sent from the ICE board to report messages sent to ICE. That is messages that matched the address masks assigned to ICE and that ICE has ACK'd on the bus.

3.3.8 0x4d 'M' – Query MBus Configuration

Synchronous Request

'M' (0x4d)	Event ID	Length	Parameter

These messages complement the set of 'm' messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

3.3.9 0x6d 'm' – Set MBus Configuration

Synchronous Request

The first byte of the message shall define which parameter is to be configured.

0x6d 0x6c 'ml': Set MBus Full Prefix

'm'	Event ID	Len (Must be 7)	'l'	Ones Mask	Zeros Mask

- **Default:** 0xfffff0 0xfffff0 (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 6 bytes. The first three bytes shall be considered the ones mask the second three bytes shall be considered the zeros mask. The masks are 20 bits long, the bottom 4 bits of the transmitted masks shall be ignored.

`0x6d 0x73 'ms'`: **Set MBus Short Prefix**

'm'	Event ID	Len (Must be 3)	's'	Ones Mask	Zeros Mask

- **Default:** `0xf0 0xf0` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the bottom 4 bits of the transmitted masks shall be ignored.
- **NOTE:** Changing the short prefix after enumeration is a violation of the MBus protocol. The ICE board will permit this, but it may have unexpected consequences.

`0x6d 0x4c 'mL'`: **Set MBus Snoop Full Prefix**

'm'	Event ID	Len (Must be 7)	'L'	Ones Mask	Zeros Mask

- **Default:** `0xfffff0 0xfffff0` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 6 bytes. The first three bytes shall be considered the ones mask the second three bytes shall be considered the zeros mask. The masks are 20 bits long, the bottom 4 bits of the transmitted masks shall be ignored.
- Messages matching the snoop prefix shall be reported by a `'B'` message but will not be ACK'd on the physical bus by ICE.
- In the case of a conflict between a snoop and a match, the match takes precedence and ICE will ACK the message.

`0x6d 0x53 'mS'`: **Set MBus Snoop Short Prefix**

'm'	Event ID	Len (Must be 3)	'S'	Ones Mask	Zeros Mask

- **Default:** `0xf0 0xf0` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the bottom 4 bits of the transmitted masks shall be ignored.
- Messages matching the snoop prefix shall be reported by a `'B'` message but will not be ACK'd on the physical bus by ICE.
- In the case of a conflict between a snoop and a match, the match takes precedence and ICE will ACK the message.

`0x6d 0x73 'mb'`: **Set MBus Broadcast Mask**

'm'	Event ID	Len (Must be 1)	'b'	Ones Mask	Zeros Mask

- **Default:** `0x0f 0x0f` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the top 4 bits of the transmitted masks shall be ignored.
- These masks are the set of broadcast channels that the ICE board should ACK on the physical MBus.

`0x6d 0x53 'mB'`: Set MBus Snoop Broadcast Mask

'm'	Event ID	Len (Must be 1)	'B'	Ones Mask	Zeros Mask

- **Default:** `0x0f 0x0f` (Disabled)
- NOTE: This command is not supported by the current MBus implementation.
- This shall be followed by 2 bytes. The first byte shall be considered the ones mask the second byte shall be considered the zeros mask. The masks are 4 bits long, the top 4 bits of the transmitted masks shall be ignored.
- These masks are the set of broadcast channels that the ICE board will not ACK on the physical MBus but will report as a snooped message (a `'B'` message).
- In the case of a conflict between a snoop and a match, the match takes precedence and ICE will ACK the message.

`0x6d 0x6d 'mm'`: Set master mode on/off

'm'	Event ID	Len (Must be 2)	'm'	`0x00` or `0x01`

- **Default:** `0x0` (Off)
- Boolean whether ICE should act as the MBus master node (1 - on) or as a MBus member node (0 - off).

`0x6d 0x63 'mc'`: Clock Speed

'o'	Event ID	Len (4)	'c'	Clock Divider

- **Default:** `0x000020` (10 MHz / 0x000020 = 312.5 kHz)
- This shall be followed by a three byte value N (MSB-first), where 10 MHz / N yields the desired clock speed.
- This configuration is meaningful only if ICE is configured as the MBus master node (see `'mm'`).

`0x6d 0x69 'mi'`: Set should interrupt

'm'	Event ID	Len (Must be 2)	'i'	`SHOULD_INT`

- **Default:** `0x0` (Off)
- NOTE: This command is not supported by the current MBus implementation.
- Configures whether ICE should interrupt the bus to transmit its next message. Possible values:
  - 0 Do not interrupt
  - 1 Interrupt (if necessary) for next message only. That is, this value is reset to 0 after the next `'b'` command sent to ICE is handled.
  - 2 Interrupt for all messages

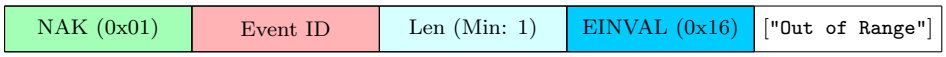
`0x6d 0x70 'mp'`: Set should use priority arbitration

'm'	Event ID	Len (Must be 2)	'p'	`SHOULD_PRIO`

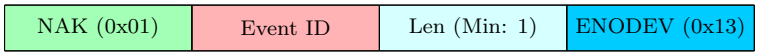
- **Default:** `0x0` (Off)
- Configures whether ICE should send high priority MBus messages. Possible values:
  - 0 Do not send priority message.
  - 1 Send all messages as high priority.

`0x6d` Responses

– NAKs for this message shall be composed of an error code, followed by an optional explanitory string.



○ `EINVAL (22,0x16):` Invalid argument.



○ `ENODEV (19,0x13):` The implementation does not support changing or querying this parameter. Unless otherwise specified, it **MUST** be hardcoded to the default.

3.3.10 `0x47 'G'` – Query GPIO State / Configuration

Synchronous Request

These messages complement the set GPIO ('g') messages.

An ACK response should mimic the corresponding set message.

3.3.11 `0x67 'g'` – Set / Configure GPIO

Synchronous Request, Asynchronous Message

The first byte of this message shall be a specificer, indicating what type of GPIO action is requested.

`0x67 0x6c 'gl'` – GPIO Level

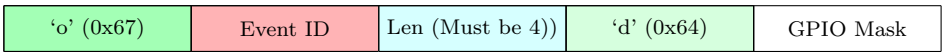
– **Default:** `0x000000` (All low)

– This shall be followed by a three byte bitmask indicating the desired output level of all 24 GPIO. This is only reflected in those GPIO which are configured as output. Each bit shall be valued:

○ 0: Low (0.0 V)

○ 1: High (1.2 V)

`0x67 0x64 'gd'` – GPIO Direction



– **Default:** `0x000000` (All input)

– This shall be followed by a three byte bitmask indicating the desired direction of all 24 GPIO. The output level of each GPIO which are configured as output are set via the 'Set GPIO Level' (`'gl'`) command. Each bit shall be valued:

○ 0: Input (DEFAULT)

○ 1: Output

`0x67 0x69 'gi'` – **GPIO Interrupt Enable**

'o' (0x67)	Event ID	Len (Must be 4))	'd' (0x64)	GPIO Mask

- **Default:** `0x000000` (All interrupts disabled)
- This shall be followed by a three byte bitmask indicating whether a change in each GPIO bit wil generate an GPIO level change message. Only those GPIO which are configured as input will generate interrupts. Each bit shall be valued:
  - 0: Interrupt Disabled (DEFAULT)
  - 1: Interrupt Enabled

3.3.12 `0x50 'P'` – Query Power State

Synchronous Request

'P' (0x50)	Event ID	Len (Must be 1)	Parameter	PWR IDX

These messages complement the Set Power ('p') messages.

The 'Parameter' field is the parameter specifier to query.

An ACK response should mimic the corresponding set message.

3.3.13 `0x70 'p'` – Set Power State

Synchronous Request, Asynchronous Message

Set Power State messages allow direct control of set-point voltage and on/off states for various power domains on the ICE board. The first byte of this message shall be a specificer, indicating which parameter is requested. The second byte of the message shall be the power domain identifier. Currently implemented power domain identifiers are:

- 0: M3 0.6V (0.675V Default)
- 1: M3 1.2V (1.2V Default)
- 2: M3 VBatt (3.8V Default)
- 3: GOC USB Voltage (5.0V Default)

`0x70 0x76 'pv'` – **Voltage State**

- The first byte of the message shall be a single byte indicating the power domain identifier to set. The second byte (v_set) shall specify the voltage according to the equation:

$$V_{out} = (0.537 + 0.0185 * v\_set) * V_{default}$$

Valid values for v_set range from `0` to `31`

'p' (0x70)	Event ID	Len (Must be 3)	'v' (0x76)	PWR IDX	v_set

`0x70 0x6f 'po'` – **On/Off State**

- The first byte of the message shall be a single byte indicating the power domain identifier to set. The second byte shall be valued `0` or `1`, depending on the desired On/Off state.

'p' (0x70)	Event ID	Len (Must be 3)	'o' (0x6f)	PWR IDX	On/Off

4 Document Revision History

Revision 0.2.7 – Oct 29, 2013

- Add ‘?c’ message
- Add ‘B’, ‘b’, ‘M’, and ‘m’ messages
- Add ‘e’ message
- Change ‘?c’ to ‘??’
- Add ‘_b’ and ‘?b’ messages

Revision 0.2.6 – Oct 17, 2013

- Add ‘Oo’ and ‘oo’ messages

Revision 0.2.5 – Jan 14, 2013

- Add ‘p’ and ‘P’ messages

Revision 0.2.4 – Dec 20, 2012

- Add ‘g’ and ‘G’ messages

Revision 0.2.3 – Dec 18, 2012

- Add ‘f’, ‘o’, and ‘O’ messages

Revision 0.2.2 – Dec 15, 2012

- ‘d’ fragments are 255 bytes long so that a final fragment of maximum length can be distinguished
- Make explicit that each ‘d’ fragment message is ACK/NAK’d
- When a ‘d’ fragment is NAK’d, that ends the transaction

Revision 0.2.1 – Nov 29, 2012

- 0.1’s ‘d’ → ‘e’
- Add fragmented ‘d’ messages

Revision 0.2 – Nov 27, 2012

- Largely redefine the protocol (ish)

Revision 0.1.2 – Nov 26, 2012

- Restrict Version to ‘V’ only
- Restrict eXtension to ‘X’ only

Revision 0.1.1 – Nov 16, 2012

- Add visualized packets

Revision 0.1 – Nov 14, 2012

- Initial revision