

Vetus Add On Board Developer Guide

Hardware interface, identity storage, UART protocol, and firmware starting point

This guide explains what a Vetus add-on board must connect, store, receive, and return. It is intended to get a developer to a compatible starting point without prescribing what the add-on board itself must do.

An add-on board may perform any useful task. Compatibility depends on three things: safe power switching, correctly crossed UART connections, and predictable handling of the standard identity and result commands.

The supplied Vetus 30 Pin SIMM KiCad project is the mechanical and electrical starting template. It includes the tested board outline, connector placement, power-control circuit, ATmega4809 reference circuit, UPDI header, and the required Vetus connections.

At a glance	Requirement
Host transmit	Vetus pin 21 TX connects to the add-on MCU RX input
Host receive	Vetus pin 22 RX connects to the add-on MCU TX output
Power control	Vetus pin 20 is active high and opens the add-on board 5 V supply gate
Serial format	9600 baud, 8 data bits, no parity, 1 stop bit
Persistent identity	Internal EEPROM is recommended; provide an equivalent nonvolatile store if the MCU lacks it
Reference MCU	ATmega4809 or a similar MCU is recommended for the easiest implementation

Revision 1.0 | Add-on interface guide | Supplied with the Vetus 30 Pin SIMM KiCad project

1 Scope and design freedom

Vetus controls when the add-on board receives power and exchanges command data with the board over two one-way serial lines. Everything beyond that interface is the add-on developer's choice: sensors, adapters, signal generation, measurements, programming functions, storage, and custom test logic may all be implemented on the add-on board.

Use the ATmega4809 design and firmware as a known-working reference. Another MCU is acceptable if it can operate at the board's logic level, provide a dependable hardware UART, start cleanly when switched power is applied, and preserve board identity in nonvolatile storage.

The remaining ZIF pins may also be used for the add-on's documented test signals. Vetus may drive or read those pins when Vetus itself performs the test, and the add-on may send compatible signals through them when required. Signals presented to the Vetus ZIF interface must use 5 V logic levels and must have clearly defined direction and behavior.

Never use ZIF test pins as the add-on board's own power source or board ground return. The chip being tested may still use Vetus's normal parasitic-power method through the ZIF pins, including the VCC and ground assignments defined by its test. The add-on circuitry itself must be powered from the proper switched 5 V supply and must return its supply current through the dedicated main ground connection. Any function requiring another voltage, extra drive current, or specialized signal generation should be produced safely on the add-on board itself, with the resulting measurement or status returned to Vetus through the communications link.

2 Required electrical connections

Vetus pin	Direction as seen by Vetus	Connect on add-on	Purpose
20	Output	Power-gate control input	Active high request that enables the switched 5 V rail
21	Output TX	MCU RX	Commands sent from Vetus to the add-on
22	Input RX	MCU TX	Replies and result data sent from the add-on to Vetus

TX and RX must be crossed. Pin 21 is Vetus TX, so it goes to the add-on receiver. Pin 22 is Vetus RX, so it goes to the add-on transmitter. Label the nets from the viewpoint of the device driving them to avoid reversing the pair.

Pin 20 is a control signal, not the board's 5 V power feed. Use it as an active-high input to the switching circuit shown in the supplied KiCad project. The reference circuit uses that signal to operate a gate that connects the 5 V source to the add-on's switched supply rail.

Provide a common ground between Vetus and the add-on through the dedicated ground connection. Do not assign an ordinary ZIF test pin as the add-on's supply-current return. Place local bypass capacitors at every MCU supply pin and include bulk capacitance appropriate for the board load. The template shows the tested reference arrangement.

Use of other ZIF pins

Pins other than 20, 21, and 22 are available when the add-on's function needs them. This is hardcoded into Vetus Pars and therefore must always be the same. This means add-on boards can be more than 34 pins. They may carry documented 5 V test signals between Vetus and the add-on. If Vetus is performing the test, it can control and sample those pins using its normal test system. If the add-on is performing the work, it can place compatible 5 V signals on the required pins and report the outcome back to Vetus.

These pins must not provide power or the supply-current return for the add-on circuitry. Do not depend on parasitic power from a driven ZIF pin, combine pins to obtain board supply current, or feed the add-on supply rail from a test signal. This does not prevent the tested chip from using Vetus's normal parasitic-power arrangement through the VCC and ground pins assigned by its test definition. Use the dedicated switched 5 V path controlled by pin 20 and the

dedicated main ground connection for the add-on board itself. Generate non-5 V levels, higher-current outputs, analog stimulus, and other specialized conditions locally on the add-on board, protect the Vetus-facing pins, and send the final data or result back over UART.

3 Power up and power down behavior

1. Vetus keeps pin 20 low while the add-on board is off.
2. Vetus drives pin 20 high. The add-on power gate opens and the switched 5 V rail rises.
3. The MCU resets, initializes its clock and UART, and becomes ready to receive. It must not transmit startup text or unsolicited bytes.
4. Vetus sends a framed command only after allowing the board time to start.
5. The add-on receives the full request, performs the operation, stores a numeric result if requested, and sends only the defined reply.
6. When the add-on is no longer needed, Vetus stops communications and drives pin 20 low to remove add-on power.

Design the board so a partially powered MCU cannot back-power the 5 V rail through TX, RX, or another signal. MCU pins should remain high impedance during reset, and any level translation or protection must also behave safely while the add-on supply is off.

4 MCU selection

The ATmega4809 is recommended because the supplied reference firmware and KiCad design already use it. Its internal oscillator, hardware USART, EEPROM, and UPDI programming interface make it a straightforward fit. It can work as a basic eeprom for simple boards or it can act as both an eeprom and do work when required. Therefore, this chip and other similar chips are ideal for this. No outside eeprom is ever needed. It is a cheap and convenient multi-use chip. However, any similar chip should work fine. When you read Atmega4809 you may assume that similar chips work as well. The Atmega4809 just happens to be on hand.

A different MCU is permitted. At minimum it should provide:

- A hardware UART capable of 9600 baud, 8 N 1 operation.
- Nonvolatile memory for board ID, name, manufacturer, and other persistent configuration. Internal EEPROM is preferred; otherwise emulate it safely in flash or add an external EEPROM.
- A clean reset and predictable pin state during the switched-power ramp.
- Enough RAM for receive buffering, a 16-byte command packet, and any result storage used by the board.
- A practical programming/debug connector that does not interfere with the add-on connector or enclosure.

If flash is used instead of EEPROM, implement wear management, integrity checking, and recovery from interrupted writes. Identity data should never be erased or rewritten on every startup.

5 Supplied KiCad starting project

The included Vetus 30 Pin SIMM project is a simple add-on board that preserves the tested dimensions and connector geometry. Begin with the newest top-level schematic and PCB files. Keep the board outline, mating connector, keying/orientation, mounting clearances, and required interface nets intact while adding the circuitry for your design.

Reference block	What it provides
30-pin SIMM edge	Physical mating interface and tested contact placement
40-pin FFC mapping	Adapter connection used by the reference board

Reference block	What it provides
ATmega4809	Known-working reference MCU footprint and supply connections
Power gate	Active-high pin 20 control of the switched 5 V rail
UART	Pin 21 to MCU RX and pin 22 to MCU TX
UPDI header	Programming connection, power reference, and ground
Decoupling	Local 10 uF and 0.1 uF reference capacitors

Before ordering a derivative board, confirm connector orientation from both sides, pin numbering, board thickness and edge geometry, component height beneath any stacked board, and that no copper or component violates the mechanical keep-out areas.

6 How Vetus places smaller packages in the 40 pin socket

This numbering rule is essential when laying out an add-on board. Vetus always controls a 40-position ZIF socket, but the device or adapter selected in a test may have fewer than 40 logical pins. Vetus uses the stored package pin count to center the two rows at the ends of the socket while leaving the unused positions in the middle.

For any supported package, the pin count must be even. Divide the pin count by two to obtain the number of pins on each side. Logical device pins 1 through half the pin count occupy ZIF positions beginning at ZIF 1. The remaining logical pins occupy the opposite end of the socket, ending at ZIF 40.

Logical device pin	Vetus ZIF position
1 through N divided by 2	The same number: ZIF 1 upward
N divided by 2 plus 1 through N	ZIF position = 40 minus N plus the logical pin number

For a 30-pin board, N is 30 and each side contains 15 pins. Device pins 1 through 15 map to ZIF 1 through 15. Device pin 16 maps to ZIF 26, pin 17 maps to ZIF 27, and the sequence continues until device pin 30 maps to ZIF 40. ZIF positions 16 through 25 are intentionally unused. This is why the supplied schematic connects the first 15 SIMM positions to ribbon/ZIF pins 1 through 15 and the other 15 positions to pins 26 through 40 rather than connecting the SIMM straight through as 1 through 30.

30-pin logical pin range	ZIF or ribbon range	Status
1 through 15	1 through 15	Connected in ascending order
No 30-pin device pins here	16 through 25	Deliberate center gap
16 through 30	26 through 40	Connected in ascending order

The a and b labels on J1 identify the two physical contact rows of the SIMM-style connector. The matching a and b contacts for each connector position are tied to the same Vetus logical signal in this template. Vetus does not treat the letters as additional device pins; it operates on the logical pin number after applying the pin-count translation.

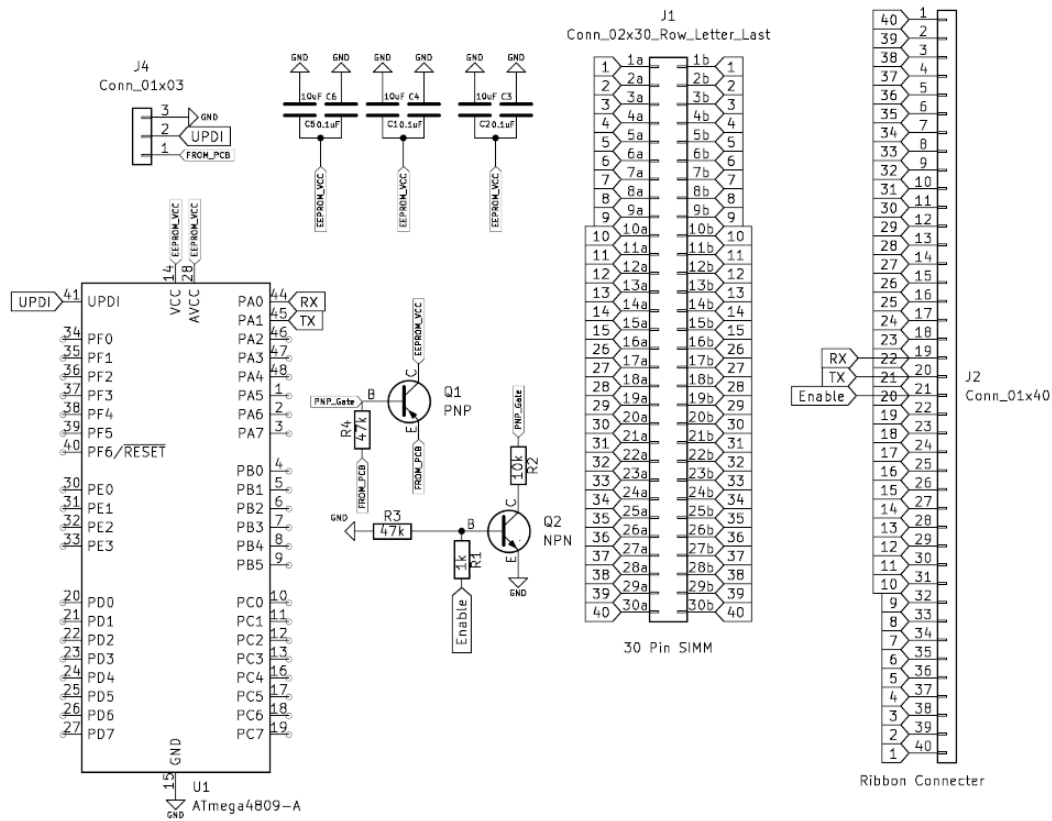


Figure 1 Thirty pin SIMM mapping to the forty pin Vetus ribbon and ZIF numbering

Board layout rule

Lay out and label the add-on connector according to the logical package pins, then connect those contacts to the corresponding Vetus ZIF numbers using the rule above. Do not assume that a 30-pin board uses ZIF 1 through 30. If pins 16 through 30 were connected to ZIF 16 through 30, Vetus would drive and read the wrong physical contacts because its firmware maps the upper half of a 30-pin package to ZIF 26 through 40.

The firmware performs this translation whenever it is given a chip pin number. Internally, ZIF 1 is index 0 and ZIF 40 is index 39. For logical pins on the first side it uses pin minus one. For pins on the second side it counts backward from index 39 according to the package total. Therefore the package pin count stored with the test definition must match the physical adapter. A wrong pin count changes the translation even when the logical pin numbers in the test code have not changed.

Example	Result
30-pin package, logical pin 1	ZIF 1
30-pin package, logical pin 15	ZIF 15
30-pin package, logical pin 16	ZIF 26
30-pin package, logical pin 21	ZIF 31
30-pin package, logical pin 30	ZIF 40

When validating a new board, check continuity from every logical connector contact to its expected ribbon/ZIF position, verify that positions 16 through 25 are not accidentally used for the 30-pin mapping, and test orientation from both the component side and solder side before powering the assembly.

7 Serial transport

Setting	Value
Baud rate	9600
Data bits	8
Parity	None
Stop bits	1
Idle state	High
Bit order	UART standard, least-significant bit first

Use a hardware UART whenever the selected MCU provides one. Receive bytes into a small queue in the UART interrupt, but parse and execute commands in the main loop. This keeps the receive interrupt short and prevents a lengthy board operation from losing later bytes.

Reject bytes with framing, parity, or overrun errors. On an error, clear the partial packet and wait for the next start byte. Do not let one damaged byte shift every later command.

8 Vetus request framing

Every request from Vetus starts with byte 0x7E. Bytes received before 0x7E while no request is active are ignored.

Normal requests contain 16 bytes after the start byte. Those 16 bytes are four unsigned 32-bit words, transmitted most-significant byte first:

Wire bytes after 0x7E	Meaning
0 to 3	Command word
4 to 7	Data 1
8 to 11	Data 2
12 to 15	Data 3

The command word has two accepted forms. A value from 0x00000000 through 0x000000FF is a command without a result slot. To request that a command place a value in a result slot, use the command byte in bits 15 through 8 and the slot number in bits 7 through 0. The reference firmware provides 64 result slots, numbered 0 through 63.

Example request	Meaning
7E 00 00 00 02 00 00 00 00 00 00 00 00 00 00 00	Run command 0x02 GET NAME; no result slot
7E 00 00 01 05 00 00 00 00 00 00 00 00 00 00 00	Run command 0x01 GET ID and store it in result slot 5

Spaces in the examples only show field boundaries. They are not transmitted.

9 Standard commands

Command	Name	Request	Add-on action and reply
0x01	Get board ID	Normal 16-byte request; include a valid result slot	Read the one-byte ID from EEPROM and store it as a 32-bit result in the requested slot. No immediate bytes are returned.
0x02	Get board name	Normal 16-byte request	Return 0x7E, one length byte, then exactly that many name bytes.

Command	Name	Request	Add-on action and reply
0x03	Get manufacturer	Normal 16-byte request	Return 0x7E, one length byte, then exactly that many manufacturer bytes.
0x21	Get result	Compact request: 0x7E 0x21 slot	Return exactly four bytes, most-significant byte first. This response has no 0x7E prefix.

10 Exact command sequences

Read the board ID

1. Vetus powers the add-on by asserting pin 20 high and waits for MCU startup.
2. Vetus chooses an unused result slot, for example slot 5 (You create these yourself in the add-on firmware).
3. Vetus sends 0x7E followed by the 16-byte request whose command word is 0x00000105. Data 1, Data 2, and Data 3 may be zero.
4. The add-on reads EEPROM address 0 and stores the ID in result slot 5.
5. Vetus sends the compact result request 0x7E 0x21 0x05.
6. The add-on returns four big-endian bytes. If the ID is 1, the reply is 00 00 00 01.

Read the board name

1. Vetus sends 0x7E followed by a normal 16-byte request with command word 0x00000002.
2. The add-on reads the stored name length, clamps it to 16, then reads that many bytes.
3. The add-on replies with 0x7E, the length byte, and the name bytes. The string is length-delimited and does not require a terminating zero.

Read the manufacturer

1. Vetus sends 0x7E followed by a normal 16-byte request with command word 0x00000003.
2. The add-on reads and clamps the manufacturer length to 16.
3. The add-on replies with 0x7E, the length byte, and exactly that many manufacturer bytes.

Run a custom command and retrieve a result

1. Assign the add-on command an unused one-byte opcode and document Data 1, Data 2, and Data 3.
2. Vetus chooses a result slot and sends a normal request with the opcode and slot packed into the command word.
3. The add-on validates the arguments, performs the work, and writes one unsigned 32-bit result to that slot.
4. Vetus requests the slot with 0x7E 0x21 slot and interprets the four returned bytes as a big-endian unsigned value.

If a custom operation needs more than one result, assign multiple result slots by convention or define a documented custom reply. Never send unsolicited progress text on the protocol UART.

11 EEPROM identity layout

Address	Size	Contents
0	1 byte	Board ID
1	1 byte	Board-name length, 0 to 16
2 to 17	16 bytes	Board name
18	1 byte	Manufacturer length, 0 to 16
19 to 34	16 bytes	Manufacturer
35	1 byte	Number of standard board-information commands; reference value is 3

Unused bytes in the fixed name and manufacturer fields may remain zero. Length values are authoritative. Clamp lengths before reading or transmitting so corrupt nonvolatile data cannot cause an out-of-bounds read.

The reference project places the identity object into the ELF and HEX EEPROM image, so MPLAB programs it with the device. Firmware does not rewrite the identity at startup.

12 Exact ATmega4809 reference snippets

The following excerpts are taken from the tested ATmega4809 board firmware. They are the compatibility core, not a complete add-on application.

Protocol and EEPROM definitions

```
#define CMD_GET_ID          0x01u
#define CMD_GET_NAME       0x02u
#define CMD_GET_MANUFACTURER 0x03u
#define CMD_GET_RESULT     0x21u

#define COMMAND_PACKET_SIZE 16u
#define RESULT_SLOT_COUNT  64u
#define NO_RESULT_SLOT     0xFFu
#define PACKET_START       0x7Eu

#define BOARD_INFO_SIZE    36u
#define BOARD_ID_ADDRESS   0u
#define BOARD_NAME_LENGTH_ADDRESS 1u
#define BOARD_NAME_ADDRESS 2u
#define BOARD_NAME_MAX_LENGTH 16u
#define BOARD_MANUFACTURER_LENGTH_ADDRESS 18u
#define BOARD_MANUFACTURER_ADDRESS 19u
#define BOARD_MANUFACTURER_MAX_LENGTH 16u
#define BOARD_COMMAND_COUNT_ADDRESS 35u
```

UART setup

```
#ifndef F_CPU
#define F_CPU 2000000UL
#endif

#define MAIN_BAUD 9600
#define USART0_BAUD_VALUE ((4 * F_CPU) / MAIN_BAUD)

void USART0_init(void)
{
    PORTA.DIRSET = PIN0_bm;
    PORTA.DIRCLR = PIN1_bm;
    PORTA.PIN1CTRL = PORT_PULLUPEN_bm;
    USART0.BAUD = (uint16_t)USART0_BAUD_VALUE;
    USART0.CTRLA = USART_CHSIZE_8BIT_gc;
    USART0.CTRLB = USART_RXCIE_bm;
```

```

    USART0.CTRLB = USART_TXEN_bm | USART_RXEN_bm;
}

```

In the ATmega4809 reference, PA0 is the MCU TX output and PA1 is its RX input. Therefore PA1 receives Vetus pin 21, and PA0 drives Vetus pin 22.

Big endian conversion and byte transmit

```

static uint32_t read_u32_be(const uint8_t *bytes)
{
    return ((uint32_t)bytes[0] << 24)
        | ((uint32_t)bytes[1] << 16)
        | ((uint32_t)bytes[2] << 8)
        | (uint32_t)bytes[3];
}

void USART0_sendByte(uint8_t data)
{
    while (!(USART0.STATUS & USART_DREIF_bm));
    USART0.TXDATAL = data;
}

```

Safe string reply

```

static void send_eeprom_string(uint8_t length_address,
                               uint8_t data_address,
                               uint8_t maximum_length)
{
    uint8_t length = eeprom_read_byte(
        &board_info_eeprom[length_address]);

    if (length > maximum_length)
    {
        length = maximum_length;
    }

    USART0_sendByte(PACKET_START);
    USART0_sendByte(length);

    for (uint8_t i = 0u; i < length; i++)
    {
        USART0_sendByte(eeprom_read_byte(
            &board_info_eeprom[data_address + i]));
    }
}

```

Result reply

```

uint32_t value = results[result_slot];
USART0_sendByte((uint8_t)((value >> 24) & 0xFFu));
USART0_sendByte((uint8_t)((value >> 16) & 0xFFu));
USART0_sendByte((uint8_t)((value >> 8) & 0xFFu));
USART0_sendByte((uint8_t)(value & 0xFFu));

```

13 Adding board specific commands

Keep the four standard commands unchanged. Add each board-specific opcode to the command switch and decide whether it acts immediately, stores a result, or returns a documented custom payload.

```

case CMD_YOUR_FUNCTION:
    if (arguments_are_valid(data1, data2, data3))
    {
        uint32_t value = perform_your_function(data1, data2, data3);
        if (result_slot < RESULT_SLOT_COUNT)
        {
            results[result_slot] = value;
        }
    }
    break;

```

The sample above shows the required mechanics but intentionally leaves the board's function to the developer. Validate every argument before changing hardware. Bound loops and waits, and return a defined error value rather than hanging indefinitely.

14 Receive parser behavior

1. While idle, discard bytes until 0x7E is received.
2. After 0x7E, collect the first payload byte.
3. If that byte is 0x21, collect one additional slot byte and process the two-byte compact request.
4. For every other first byte, collect a total of 16 payload bytes.
5. Decode all four 32-bit words as big endian.
6. Execute the command only after the complete expected request has arrived.
7. Reset the parser to idle after processing or after any receive error.

A start byte that happens to occur inside a 16-byte payload is ordinary data. It does not restart the parser. Resynchronization occurs only while idle or after the implementation deliberately abandons a damaged/incomplete packet.

15 Timing and response rules

- Do not transmit a boot banner, debug text, or periodic status bytes on the Vetus UART.
- Send a reply only when the command defines one.
- Finish receiving the request before beginning the reply.
- Keep replies to their exact documented length.
- Do not assume Vetus continuously listens after every command. Store numeric results and return them when 0x21 requests the selected slot.
- For a slow operation, define a busy/result convention rather than blocking forever.
- Clear volatile result slots on every power-up unless the board specification explicitly requires persistence.

16 Bring up checklist

Check	Expected result
Power off	Pin 20 low; switched 5 V absent; UART lines do not back-power the MCU
Power on	Pin 20 high opens the gate and the switched 5 V rail reaches the board
UART idle	Both serial lines idle high; no unsolicited bytes
Identity	0x01 result matches EEPROM address 0
Name	0x02 returns 0x7E, valid length, exact name bytes
Manufacturer	0x03 returns 0x7E, valid length, exact manufacturer bytes
Result	0x21 returns exactly four big-endian bytes
Error recovery	A framing/overrun error clears the partial request and later commands still work
Power cycle	Identity remains correct and volatile state resets
Mechanical	Board mates freely and respects the template outline and clearances

17 Common mistakes

Mistake	Correction
Connecting TX to TX	Cross the pair: Vetus pin 21 TX to MCU RX; Vetus pin 22 RX to MCU TX
Treating pin 20 as 5 V	Use pin 20 only as the active-high gate-control signal
Using a ZIF pin as board ground	Return add-on circuitry supply current only through the dedicated main ground connection
Sending ASCII commands	Commands are binary bytes, not text
Omitting 0x7E	Every request begins with the packet-start byte
Using little-endian words	All 32-bit request fields and result replies are big endian
Returning ID immediately	GET ID stores the value in a result slot; Vetus then uses GET RESULT
Null-terminated identity strings	Replies use an explicit one-byte length and exactly that many bytes
Writing identity at startup	Program identity into EEPROM/nonvolatile storage and read it at runtime
Debug prints on protocol UART	Use a separate debug connection or disable prints in production

18 Final compatibility contract

A compatible add-on board uses Vetus pin 20 as an active-high request to enable its switched 5 V supply, receives Vetus commands through pin 21 on the MCU RX input, and returns data through pin 22 from the MCU TX output. It communicates at 9600 baud using 8 N 1, recognizes 0x7E-framed requests, supports the standard identity and result commands, stores identity in EEPROM or equivalent nonvolatile memory, and remains silent except for defined replies.

Other ZIF pins may carry documented 5 V test signals in either direction as required by the test. The tested chip may use Vetus's normal parasitic-power method through the ZIF pins, including its test-defined VCC and ground pins. Those pins must never power the add-on circuitry or serve as the add-on board's own ground return. Use only the dedicated switched 5 V supply and dedicated main ground connection for board power; specialized voltages, drive functions, and test conditions belong on the add-on itself, with results returned to Vetus.

Once those requirements are met, the add-on board may implement any safe function appropriate to Vetus. Document every custom opcode, the meaning and valid range of Data 1 through Data 3, the result-slot behavior, expected execution time, electrical use of other connector pins, and any condition that can change the board's outputs.