Designing VM2 Application Boards

This document lists some things to consider when designing a custom application board for the VM2 embedded controller. It is intended to complement the VM2 Datasheet.

A few of the items in this document are very important; you should definitely consider them before designing your board:

- Serial Ports – Serial Port 1 is essential for code development
- A Program mode switch or link is essential for code development. A Boot mode switch or link may also be very useful.
- USB Device port – A minimal USB device port is very useful for production programming and very simple to design in.

Mode pins

Given that you are likely to be developing your application code on the target hardware, your application board needs to at least have the ability to put the VM2 into Program Mode.

The full list of modes is:

Mode	***Method***	***Comments***
Program	Reset with **PROG** mode pin pulled to ground.	Used during code development.
Boot	Reset with **BOOT** pin pulled **high**, to Vcc (3.3V).	Used to reprogram the Venom Language/OS when OS is not present.
USB disable	Pull the **USB Control** pin low to disable the USB File system when starting in Program Mode.	Not usually needed.

Program mode

The most commonly used mode is Program Mode. Program Mode forces the VM2's on-board Venom compiler to communicate over the main serial port. You can include a DIP switch or a link on your application board to pull the Program mode pin low.

If you aren't using this pin just let it float high.

Boot mode

Boot mode is used to download the Venom OS into a VM2. Normally you would do download the Venom OS using a USB port as it is faster. However, if the VM2 has lost its OS then you will need to use Boot mode. You can design a DIP switch or a link into your application board to pull the

Boot mode pin high.

If you aren't using this pin just let it float low.

USB/CAN mode

The USB control pin is sometimes used as a mode pin by pulling it to ground. It is only necessary to use it as a mode pin if you will be using the VM2's CAN bus.

You can design a DIP switch or a link into your application board to control this pin. If you aren't using it just let it float high.

The state of the **USB/CAN mode** pin determines how the VM2's two **USB/CAN bus** pins are used when the VM2 is waiting at the *Clear Memory* prompt (i.e. when it starts in Program Mode).

If you let this pin float high (the default state), then the VM2 will use its two **USB/CANBus** pins as a USB bus to present its Flash file system as a USB Mass Storage device – i.e. just like a memory stick.

If it is pulled to ground then the VM2 will *not* attempt to use its two **USB/CANBus** pins as a USB bus. This is necessary if these pins are used in a CAN bus in your application.

USB/CAN mode and the LED

If you are intending to use the USB bus to production-program your VM2's (and this is the best way), then it is useful to be able to see the VM2's on-board LED, as this is used to indicate when USB file transfers are in progress, and therefore when it is time to reset the VM2 so as to initiate the self-reprogramming sequence. If the design of your equipment means that the LED is not visible then it may be possible to fit an additional LED (and ballast resistor) and drive it from the VM2's LED output pin.

If it is not possible to see the LED signal then it is still possible to use the USB file transfer – you will have to wait long enough after your PC has finished to be sure the file write sequence has fully completed on the VM2.

If the VM2 is connected to a Terminal (e.g. VenomIDE), answering the Venom's **Clear RAM?** Question with **Y** (or any character) will give you an indication of when any currently active USB transfer has finished. Then, to initiate the self-reprogramming sequence just type **Reset** or hit **F11** in VenomIDE.

Reset button

It is sometimes useful to incorporate a Reset push button on application boards, to reset the VM2 without having to power it off and on again. The button should pull the RESET signal to ground

while it is held down.

If you aren't using this pin just let it float.

Mode connector

When a VM2 and its application board are buried deep inside some equipment it is sometimes useful to connect the VM2's mode, Reset and LED signals to a pin header. You can then bring all these signals out to a more easily accessible place via a cable. It's also useful to bring out the main serial port and the USB port, if possible.

USB Device port

The simplest and fastest way to *production-program* VM2s with firmware and other application files requires a USB port. For this reason, if possible, it is good to build some kind of USB Device port into every VM2 application board. This can be very simple – just three header pins and a resistor – or it can be a full USB Device port. There is a circuit schematic on our website showing how to do each of these. The USB Device circuit schematic is called **Sch-USB-port.pdf**

If you bring out the USB signals to a pin header then you will need to make an adapter to convert to a standard USB connector.

Serial Ports

All code development is done using the VM2's main serial port (Serial Port 1) as a terminal to communicate with the VM2's on-board compiler. For this reason it is good to have a serial port built into every application board.

A serial port may use either **RS232 levels** or **logic levels**. The two options are examined below.

Using a built in RS232 level shifter

If design your serial port to use RS232 levels it can connect directly to a COM port on your PC. See our circuit schematic **Sch-serial-ports.pdf** for ideas about how to do this.

Using an external RS232 level shifter

If you don't want to build an RS232 level shifter circuit into your application board you can use our external level shifter (product code **5914**).

You will have to bring all the Serial Port 1 signals directly to a pin header. You will also have to bring out Ground and Vcc (3.3V) to power the level shifter.

It is useful to design your header pinout to mirror the signals on the **5914**'s header, so you can use a simple 0.1” ribbon cable between the two. The signal names on the **5914** describe the VM2 signal names they connect to – you should not cross them over.

I2C Port

It's usually a good idea to build in an I2CBus connector into every application board. This then allows you to extend the I/O capability of the board in the future, adding digital and analogue I/O without having to redesign the board. It's best to use our standard 5-pin Molex 6410 header (or an equivalent) so that our standard I2C cards can plug in directly.

You will also need to provide pull up resistors on the SCL and SDA signals. The normal value is 4K7.

An I2C port can also help with testing an application board, as the VM2 then has access to some external I/O pins it can use to interact with the application board.

There are two I2C ports available on the VM2. The port 1 is the best to use for general purpose as it interferes with fewest other VM2 IO functions.

You will need to decide what voltage to operate the I2C Bus at. The minimum voltage is a nominal 3.3V. The maximum voltage is a nominal 5.0V. You will need to ensure that the chosen bus voltage suits every device on the bus, including any future devices.

I2C Bus addresses

We have adopted a standard set of addresses for devices on the I2C Buses on our controllers. There is an application note on our website that lists these: **i2c_table_vm2.html**

If you stick to this standard it will help to keep your design consistent with our software developments.

Power supply

The VM2 controller needs a good quality, precise power rail at 3.3V. This is usually provided by a 3.3V linear regulator, though a switching regulator may also work well.

The VM2 itself is likely to draw around 50mA maximum, but you will also have to provide enough current for the VM2 IO pins when they drive high, any other components on the board, and anything off board that draws current from the board.

Low Power considerations

Some application boards need to consume a low power, either continuously or when in a low power mode. These are some of the considerations for designing a low power application board.

Power supply

A lot of power can be wasted in a linear regulator if the supply voltage is much higher than the VM2's 3.3V rail. You can use switch mode power supplies (SMPS) to improve the efficiency here. Some SMPS can operate with a quiescent current of less than 5uA. See the **LT8614** and similar.

It may sometimes be useful to use a SMPS to reduce the bulk of the voltage and then a linear supply to power the VM2, etc. This is what we do on some of our VM2 Application Boards.

Remove unnecessary systems

Take off-board those sub-systems that aren't actually part of your application.

For example:

The RS232 level shifter built into all our standard application boards is often only used during development. It can draw around 20-40mA. If you don't need it, don't put it into your board, but instead use an external level shifter, as detailed in the Serial Ports section.

Power zoning

It may be possible to design your application board so that sections of the board ('zones') are powered down when they are not needed. If you do this you will have to be very careful to control any VM2 IO signals that interface with the de-powered zones so they don't end up powering the de-powered zones through the interface signals. In general, just before powering a zone down, you should arrange for all the VM2 IO pins that interface with the zone to be pulled low. Usually the best ' digital attribute' value to use is *Input, Active high, Pulled to inactive state* which has the value **2** or **%010**.

One tricky area is power zoning devices that are connected to the SPI or I2C Buses. It is usually easiest to treat all the devices on a particular bus, and the bus itself, as one power zone.

Software

This isn’t strictly a hardware design issue, but software considerations can improve power consumption a lot.

You can use **Every** rather than **Forever** in loops, or use **Wait** statements within loops. This uses less power.

You can put the VM2 into a low power sleep mode using **RealTimeClock**.**TimeOut** message. In this mode, the VM2 isn't doing any processing, but will wake up when pre-defined IO channels change state, or the **RealTimeClock** alarm is triggered. In this mode the VM2 draws about 55uA, providing other power saving strategies are also employed.

You can use **OperatingSystem.Low** to pull all unused VM2 I/O pins into a defined logic state, which reduces the power lost in 'shoot through' current in the input circuit of each IO pin. This

should be called after you have defined all your I/O pins. See the software documentation for **OperatingSystem.Low** to make sure you comply with all the conditions required to make this work correctly.

If you aren't using the VM2's low power sleep mode you may want to slow down the VM2's master clock. The minimum clock speed is 16Mz (it is 72MHz by default). This reduces the power consumption almost linearly. This is done using the **OperatingSystem.Speed** message.

If you *are* using the VM2's low power sleep mode then it may be more power efficient to not slow down the clock.

Ethernet

The Ethernet IC can be put into a low power mode when it is not needed. See the Ethernet object documentation.

Display circuits

When using the VM2D to drive TFT displays you should make sure that all the display signals are routed over a ground plane. This is to prevent 'ground bounce' due to return currents from all the display signals. See the section on ground return paths, below.

Analogue inputs

If you are driving the VM2's on-board analogue inputs from resistor dividers, or other moderately high impedance sources, it seems to be necessary to include a capacitor to ground at each input, otherwise the inputs read incorrectly. This hasn't been investigated fully, but a capacitor of around 100nF seems to work well.

Other circuits

Other circuits may be copied from the schematics for our standard application boards, available on our website.

General guidelines to follow when designing circuits

Ground return paths

Good grounding is very important for fast signal buses such as the TFT Display, SPI bus, USB Device, USB Host, Ethernet and memory card.

In the case of fast signal buses, good grounding is largely about reducing the inductance of the ground-return path – mainly by minimising the area of any loops formed between the signal path and the ground return path.

The ideal solution is to use a four-layer PCB where two layers may be devoted to ground and power

planes.

Even when there is a whole ground plane for a signal's return current to flow through, the returning current will actually flow along the *path of least inductance*, which is the path that minimises the loop area relative to the outgoing signal path – i.e. the ground return current will try to follow the outgoing signal track.

Even if you don't have a four-layer PCB you can still provide a good ground return path by making sure a ground track closely follows the route of the signal tracks, typically on the other side of the PCB. Don't use a thin ground track as it will have a high self inductance. Use a relatively wide track, or several roughly parallel tracks as these configurations have a much lower inductance.

Even if you are using a four-layer PCB, it is still possible to have a *poor* ground return path if the ground plane has breaks that cross the signal path, or is not connected directly to the local ground at both ends of the signal path.

Please contact our engineers if you have any questions.

Testing

If you are making a large number of your application boards then the time taken to test them becomes important. For this reason we always try to use pluggable connectors whenever screw terminals are required. This allows the board to be plugged and unplugged from the test harness quickly, reducing the time to test. If you can't do this you might be able to design in cheap pin headers to facilitate testing.

