Linux VME Howto

Table of Contents

Table of Contents

Linux VME Howto

John Huggins and Michael Wyrick, `vmelinux@va.net`

$Revision: 1.2 $, $Date: 2000/08/17 15:45:30 $

This document came about to show the embedded system community how to run Linux on their VMEbus Pentium and other PCI local bus based VMEbus processor designs.

1.Introduction

2.Installation of the VMELinux Kernel Driver

3.How to talk to the VMEbus with the VMEUtils and the VMEShell Packages

4.How to talk to the Tundra Universe PCI−VME bridge using the devices drivers.

5.Advantages of the VMEbus, Linux and VMELinux

6.Current and planned Board Support

7.Conclusion

8.FAQ

1.Introduction

1.1 Knowledge Required

Using Linux on an embedded VMEbus processor board is not very difficult. However, more than fundamental knowledge is required. This document is not a primer on how to fully configure a Linux machine.

In order to understand this HOWTO document it is assumed that you are thoroughly familiar with the following:

- Configuring and compiling a Linux kernel to operate the various peripherals on your board. Kernel−HOWTO
- Setting up and configuring of network devices NET−3 HOWTO
- Setting up of inetd NET−3 HOWTO
- Setting up and use of the Tundra Universe PCI to VME Bridge Chip Tundra Universe. The new VMEUtils program makes knowledge of the Universe unnecessary for those who do not wish to deal with register level Universe access.
- Compiling and installing various network packages like Apache SiteWu−Ftpd FAQ
- The VMEbus Rev. D and VME64. Excellent information may be found at the VMEbus International Trade Association (VITA).

If you are uncertain of how to proceed with any of the above it is STRONGLY recommended that you use the links provided to familiarize yourself with all packages. We may not reply to any mail regarding any of the above. Please direct any questions to the appropriate author of the HOWTO or consult the respective hardware manufacturer.

This document describes the installation and use of VMELinux on a Xycom XVME−655 6U VME processor board. Other brands of VME boards that use a Pentium and the Tundra Universe chip should be capable of running VMELinux. Please consult the Board Support Section of the VMELinux web site for tested boards. VMELinux Project Web Site

1.2 Why use Linux on VMEbus systems?

Operating systems for VMEbus computers are usually Real−time Operating Systems (RTOS) which have high cost and a significant learning curve. In return the RTOS offers quick response to real world events for control of machinery or response to a process.

The VMEbus provides a rugged computer enclosure and interconnection system. Many system integrators require this ruggedness and also need very fast real−time response. However, there are many times when there is little need for real−time response, but the software still needs:

- networking capability,
- remote access via telnet or similar program,
- file transfer via FTP or similar programs,
- remote booting via BOOTP or similar method,
- a way to respond to system interrupts.

Linux has all these capabilities. Thus, the VMELinux Project exists.

1.3 Purpose

The purpose of VMELinux is to give the VME system integrator another choice in operating systems. Rich in features, high in reliability and low in cost, Linux offers benefits to the embedded computer industry. High cost operating systems economically prohibit the use of VME in many applications. With Linux and the VMELinux drivers, the rugged VMEbus has new possibilities.

The purpose of the VMELinux Project is to:

- Maintain and improve the free VMELinux Kernel Driver software,
- Offer added value software components such as the VMEUtils program and VMEShell utilities.
- Test the software on various makes and brands of manufacturer supplied VME processor boards,
- Maintain web based documentation on each tested brand and make of boards,
- Maintain this HOWTO.
- Integrate user suggested and user supplied improvements into the virgin code so we may all benefit from the programming talents of others.
- Become the original source for all the above software so VMELinux users can be assured of original code from the authors.

1.4 Feedback

As VMELinux is tested in the field, we encourage comments about how well or how bad it works. Please feel free to send comments to The VMELinux Project

As we get experience about each brand of VME CPU, we will list the different configurations in this HOWTO. For now we will describe only the Xycom board.

1.5 VMELinux Revision History

Linux Kernel Driver

- November 1997, v0.2 – Initial version on Xycom Board
- December 1997, v0.3 – Useable version used for actual work with project.
- February 1998, v0.6 – DMA mode added to VME access modes.
- June, 1998, v0.8 – Fixed a few things to allow the new VMEUtils to work.
- June 24, 1998, v0.8a – Last version for the 2.0.x kernels
- April 2000, v0.95 – First version for the 2.2.x kernels

VMEUtils Program

- February, 1998, v0.6 – Created a command line interpreter to access the VMEbus
- June, 1998, v0.8 – Fixed several issues to allow VMEShell Utilities to function
- June 24, 1998, v0.8a – Previous working release.
- April 2000, v0.95 – Pretty much the same as before. Better install instructions.

VMEShell Utilities

- June, 1998, v0.8 – Created command line utilities that allow access to the VMEbus from the Linux shell prompt. These shell programs interface with the VMEUtils program.
- June 24, 1998, v0.8a – Changed the name of all the shell programs so they all begin with "vme." Current version made available on the website.
- April 2000, v 0.95 – Improved installation scripts.

1.6 Copyright/Distribution

2.Installation of the VMELinux Kernel Driver

2.1 Download the Source

Download the distribution from the VMELinux Web Site.

2.2 Install the source to the software

Place the file in your source directory; We suggest /usr/src. Untar the zipped/tarred file by typing...

```
tar -xzf VMELinux_0.95.tar.gz
```

Then:

```
cd vmelinux
```

You should see three directories:

```
ca91c042
vmeshell
vmeutils
```

In ca91c042 you should find:

```
ca91c042/
ca91c042/Makefile
ca91c042/ca91c042.c
ca91c042/ca91c042.h
ca91c042/README
ca91c042/e
ca91c042/ins
ca91c042/stat
ca91c042/uns
```

In vmeshell you should find:

```
vmeshell/vmer
vmeshell/README
vmeshell/vmeseek
vmeshell/cmd.vme
vmeshell/vmew
vmeshell/vmeregw
vmeshell/vmeregr
vmeshell/vmefa
vmeshell/vmecall
vmeshell/e
vmeshell/ec
vmeshell/fa.vme
vmeshell/map.vme
vmeshell/tmp.vme
```

```
vmeshell/vmedb
vmeshell/vmedl
vmeshell/vmedw
vmeshell/vmemap
vmeshell/vmerb
vmeshell/vmerf
vmeshell/vmerl
vmeshell/vmerw
vmeshell/vmewb
vmeshell/vmewf
vmeshell/vmewl
vmeshell/vmeww
vmeshell/makelinks
```

In the vmeutils directory you should find:

```
vmeutils/commands.cpp
vmeutils/commands.h
vmeutils/universe.h
vmeutils/Makefile
vmeutils/vmeutils.h
vmeutils/unilib.h
vmeutils/unilib.cpp
vmeutils/vmeutils.cpp
vmeutils/README
```

2.3 Compile the VMELinux components

Enter the "ca91c042" directory and make the VMELinux device driver module.

```
make
```

Now you must create the several /dev driver files. Type:

```
make devices
```

DON'T FORGET TO MAKE THE /dev/vme* DEVICES!!!

Once made, you should see the file "ca91c042.o" in the directory. This is a loadable module. See below for loading information. Plus, you should find several "vme..." files in the /dev directory. Here is how the files should look:

```
hostname:/dev# ls -l vme*
crw-rw-rw-   1 root     root      70,   8 Jul 30 10:51 vme_ctl
crw-rw-rw-   1 root     root      70,   0 Jul 30 10:51 vme_m0
crw-rw-rw-   1 root     root      70,   1 Jul 30 10:51 vme_m1
crw-rw-rw-   1 root     root      70,   2 Jul 30 10:51 vme_m2
crw-rw-rw-   1 root     root      70,   3 Jul 30 10:51 vme_m3
crw-rw-rw-   1 root     root      70,   4 Jul 30 10:51 vme_s0
crw-rw-rw-   1 root     root      70,   5 Jul 30 10:51 vme_s1
crw-rw-rw-   1 root     root      70,   6 Jul 30 10:51 vme_s2
crw-rw-rw-   1 root     root      70,   7 Jul 30 10:51 vme_s3
hostname:/dev#
```

Change to the "vmeutils" directory and type make there.

```
make
```

This will compile the "vmeutils" program. This program directly speaks to the kernel driver. It is a reference work for those of you who wish to write your own programs to directly speak with the driver.

Copy the program "vmeutils" to your user binary directory or let the makelinks script do this for your in the next step. On our system this is "/usr/local/bin." Alternatively, you can create a link in the user bin directory to the "vmeutils" program.

Change to the "vmeshell" directory. There are no files to be compiled here. These are shell programs that use the "vmeutils" program to access the VMEbus. All the files beginning with "vme" should with have a link made or be copied to the "/usr/local/bin" directory.

just type:

```
./makelinks
```

You are now ready to try the driver.

2.4 Load the VMELinux Kernel Module

Make sure you are root and insert "load" the VMELinux Kernel Module for the Universe chip by typing...

```
insmod ca91c042
```

Or just type "./ins" to let the shell script do this for you. Once complete, type...

```
./stat
```

or

```
more /proc/ca91c042
```

You should see a list of registers displayed on your screen. Something like this...

```
Universe driver info:
  Control Pointer = 0000
  Stats  reads = 0  writes = 0  ioctls = 0
  LSI0_CTL = 00800000    LSI1_CTL = 00800000
  LSI0_BS  = C0000000    LSI1_BS  = 00000000
  LSI0_BD  = C0010000    LSI1_BD  = 00000000
  LSI0_TO  = 40009000    LSI1_TO  = 00000000
  LSI2_CTL = 00800000    LSI3_CTL = 00800000
  LSI2_BS  = 00000000    LSI3_BS  = 00000000
  LSI2_BD  = 00000000    LSI3_BD  = 00000000
  LSI2_TO  = 00000000    LSI3_TO  = 00000000
  image_va0   = 00000000     image_va1   = 00000000
  image_va2   = 00000000     image_va3   = 00000000

Driver Program Status:
  DMACTL 0    = 00000000 DMACTL 1    = 00000000
  DMACTL 2    = 00000000 DMACTL 3    = 00000000
  OkToWrite 0 = 0        OkToWrite 1 = 0
  OkToWrite 2 = 0        OkToWrite 3 = 0
  Mode 0      = 0        Mode 1      = 0
  Mode 2      = 0        Mode 3      = 0
```

If not, something went wrong.

2.5 Difficulties

The Universe driver does a good job of finding the Universe chip on a PCI bus, but differences in board design may prevent this. We tested all our routines on a Xycom XVME−655 board. There is little reason why this should not work on any other Intel board with a PCI bus and the Universe PCI−VME bridge chip. If you encounter problems, please let us know at the The VMELinux Project

---

3.How to talk to the VMEbus with the VMEUtils and the VMEShell Packages

3.1 What is the VMEUtils program

This program can be run as is. Once started, you will see a command prompt. Type ? And you will see a list of commands. While useful, I think you will find the VMEShell scripts a better way to go. They do use this program to speak with the kernel driver so it is necessary to have this program available in the current PATH.

The source code for "vmeutils" is also instruction on how to speak directly to the kernel driver. For those of you who wish to create programs that directly speak with the driver, these source files are good examples.

3.2 What are the VMEShell Scripts

The VMEShell programs are unix shell scripts. They offer the operator a simple way to access the data on a VMEbus. Using these commands creates temporary files in the user's working directory which store information on the last access you did. This is nice because it will be possible to log off the machine, log back in and proceed from where you left off without having to re−enter VMEbus information again. Plus, these files are stored in the current working directory, so you can have different VME access configuration just by setting up different directories for each VME board of interest.

Assuming you placed the shell programs and the "vmeutils" program in the /usr/local/bin directory, you should be able to log in as a regular user and run them. What follows assumes exactly this.

3.3 The "vmemap" command.

Login as a regular user and create a directory to experiment with. Once in this directory type:

```
vmemap
```

You should get a help screen like this...

```
Usage:  map address count space size type
  where address is VME Address to set Universe image to

    Space = 0 CR/CSR    Space = 1 A16
    Space = 2 A24       Space = 3 A32
```

```
Size  = 1 8 bit     Size  = 2 16 bit
Size  = 3 32 bit    Size  = 4 64 bit

Type  = 0 USR/DATA  Type  = 1 USR/PRG
Type  = 2 SUP/DATA  Type  = 3 SUP/PRG
```

This is where you tell VMELinux how you want to access the VMEbus. We assume you already know about the VMEbus' many modes of operation, but here is a short list to help you.

- **address** is the actual VMEbus address you wish to see. This should be set to the lower most value of the address range of interest.
- **count** is the number of bytes you consider a valid range to view. This is the number of bytes starting at the address specified above.
- **space** is the addressing space (mode). For those of you who do not know what we are talking about here, the VMEbus has four overlapping address spaces that can be called independant of each other. A16 is a 64 KiloByte space. A24 is a 16 MegaByte space. A32 is a 4 GigaByte space. There is an A64 space defined the VME specification, but the Universe does not support it.
- **Size** refers to the maximum datawidth allowed for the VME board you are accessing. Some VMEbus board only handle 8 bit data paths. Others transfer 32 bits (four bytes) at a time. Some can handle a special VME block mode which can move 64 bits per transaction. The Universe can handle all these modes allowing you to mix inexpensive serial port boards with hugh memory arrays.
- **Type** is the type of VME transaction performed. Some VME boards make a distinction between "User" access (USR) and "Supervisior" access (SUP). Also, some boards allow access to two "pages" of memory: Program (PRG) and Data. The Universe supports all modes.

Typing...

```
vmemap 0x8000 0x100 1 2 0
```

sets up the VMELinux driver to access an A16 board at base address 8000 Hex with a range of 100H bytes with 16 bit data width and USR/DATA mode.

You will find two new files in your current directory.

- fa.vme
- map.vme

fa.vme stores a "fixed adder" value that will be added to all subsequent accesses with the programs below.

map.vme store the parameters above so you do not have to enter them every time.

All the following shell utilities read values from these two files to performs VME accesses.

3.4 Read Byte, Word or Long

Syntax:

- vmerb –[options] address size
- vmerw –[options] address size
- vmerl –[options] address size

3.5 Write Byte, Word or Long

Syntax:

- vmewb –[options] address value
- vmeww –[options] address value
- vmewl –[options] address value

3.6 Read the VMEbus to a file

Syntax:

- vmerf –[options] address size filename

3.7 Write a file to the VMEbus

Syntax:

- vmewf –[options] address filename

3.8 Parameters

There are several parameters used with these commands: address, size and filename.

- address − The actual hexadecimal VMEbus address you wish to read. If the map command is set to access A16 VME address space, the address should be 0xABCD. If the space is A24 then use 0xABCDEF. For A32 space use 0xABCDEFGH.
- size − The number of bytes to read. This value is always the number of bytes regardless of the data word size read. For example, if you want to read 16 bytes of information and use vmerl, the display will show 16 bytes displayed as 4 long words.
- filename − The name of the file to send "read" VMEbus data to or "write" VMEbus data from.
- value – a hex value written as "0xXXXX."

3.9 Options

Available options are defined with a single dash with the any combination of the following:

- q – Hides details on the access to the vmeutils program (default)
- Q – Shows details on the access to the vmeutils program
- p – Single access PCI addressing mode (opposite of d) (default)
- d – DMA access PCI addressing mode (opposite of p) (very fast access to the VMEbus)
- 0, 1, 2, or 3 – Which Universe chip "Image" to use (defaults to 0)
- b – binary mode off (default)
- B – binary mode on
- v – turn off verbose parameter printing (default)
- V – turn on verbose parameter printing to see how the driver is begin used

3.10 A Note about DMA mode.

VMELinux offers access to all the features of the Universe Chip. Especially useful is access to the DMA engine on the chip. With this feature the Universe chip transfers data on the PCI bus by becoming a PCI master. This is nice, but the real benefit comes from the VMEbus accesses. Even if the VMEbus interface is not using block mode transfers, the Universe chip can complete VMEbus transfers under 400 nanoseconds sustained. This is the direct result of the Universe taking complete control of both the PCI bus and the VMEbus. Thus, it is possible to access non block mode VMEbus peripherals much faster than older technologies.

4.How to talk to the Tundra Universe PCI−VME bridge using the devices drivers.

4.1 The device drivers used with VMELinux

- /dev/vme_ioctl
- /dev/vme_m0
- /dev/vme_m1
- /dev/vme_m2
- /dev/vme_m3
- /dev/vme_s0
- /dev/vme_s1
- /dev/vme_s2
- /dev/vme_s3

4.2 VMEMaster Device Drivers

/dev/vme_m* are drivers used to access the VMEbus as a bus master.

The Universe chip offers the programmer four VMEMaster windows to the VMEbus. These windows are called Images. The details of the registers within these windows is beyond the scope of this Howto. Please refer to the Universe documentation for details. Tundra Universe

4.3 VMESlave Device Drivers

/dev/vme_s* are drivers used to allow another VMEbus master to access this device.

The Universe chip offers the programmer four VMESlave windows to the VMEbus. These windows are called Images. The details of the registers within these windows is beyond the scope of this Howto. Please refer to the Universe documentation for details. Tundra Universe

Slave VME modes are not yet supported by VMELinux.

4.4 Direct Control of the Universe Registers

/dev/vme_ioctl allows read and write access to the Universe registers.

For experienced users, this device allows direct access to the Universe chip's internal registers. Explanation of these registers and what they do is beyond the scope of this howto. Please consult the Universe documentation available from Tundra Universe

4.5 read()

n = read(vme_handle,buf,len);

Where:

- vme_handle = The value returned by "open,"
- buf = pointer to data block,
- len = number of bytes to read from the VMEbus.

4.6 write()

write(vme_handle,buf,len);

Where:

- vme_handle = The value returned by "open,"
- buf = pointer to data block,
- len = number of bytes to write to the VMEbus.

4.7 lseek()

lseek(vme_handle,vme_pnt,Seek_Type);

Where:

- vme_handle = The value returned by "open,"
- vme_pnt = The actual VME address to access,
- Seek_Type = SEEK_SET or SEEK_CUR

4.8 ioctl()

ioctl(vme_handle, command, argument);

Where:

- vme_handle = The value returned by "open,"
- command = IOCTL_SET_CTL or IOCTL_SET_MODE or IOCTL_SET_BS or IOCTL_SET_BD or IOCTL_SET_TO
- argument to be sent

And:

- IOCTL_SET_CTL = Sets the image CTL register to argument. Argument must be 32 bits.
- IOCTL_SET_MODE = "MODE_DMA" or "MODE_PROGRAMMED" − Sets the mode by which the Universe chips communicates to the PCI bus (Not VME Block Mode)
- IOCTL_SET_BS = Sets the image BS register to arguments. NOTE: The BD register must already be set prior to making this call.
- IOCTL_SET_BD = Sets the image BD register to argument.
- IOCTL_SET_TO = Set the image TO register to argument.

4.9 open() and close()

Here is where you open and close the four VMELinux Master or Slave devices plus the Control device. Slave images are not yet supported.

- vme_handle = open("//dev//vme_m0",O_RDWR,0);
- uni_handle = open("//dev//vme_ctl",O_RDWR,0);

- close(vme_handle);
- close(uni_handle);

5.Advantages of the VMEbus, Linux and VMELinux

5.1 Pin and socket connectors

The VMEbus standard uses pin and socket connectors. This is superior to edge connections in that the connection is not exposed to humidity and other environmental conditions. It is a more expensive way of doing things, but offers longer times before failure.

5.2 Eurocard assembly

A VMEboard is either a 3U (160 x 100 mm) or a 6U size (160 x 233.35 mm). These sizes correspond to the Eurocard standard for board modules and card cages. Eurocard is a popular format used by many different busses including CompactPCI. This popularity makes the materials needed for cage assembly inexpensive and easy to obtain.

5.3 Linux is Low Cost

The nature of Linux is in its user supported and freely available format. The number of people using Linux is growing. The number of people contributing to the continued development of the Linux software base is growing. It is unfair to state that Linux is a good value because it is available for little to no charge. Linux is a good value because it works.

5.4 Linux is Stable

There are those who say that Linux us an unstable operating system. It is true that the new Linux kernels in development are experimental and should not be relied on for critical applications. However, stable versions of the Linux OS are always available and provide very stable operation. VMELinux is always based on the stable versions of the kernel source; Today's stable kernels are the 2.0.X series.

5.5 Linux is Dynamic

Because so many people are developing Linux, you do not have to wait long for improvements, fixes or new features to become part of the Linux distribution.

6.Current and planned Board Support

While the VMELinux driver should work with any PCI based design, the following boards have actually run our software.

6.1 Xycom XVME655 Pentium VMEbus Board

- This XyCom board is compatible with the standard VMELinux kernel driver package from VMELinux Project
- A prepared kernel will be available soon. It will be based on the newest version of the Linux kernel and will include appropriate drivers for the onboard NE2100 Ethernet interface. Check the website for details.

6.2 XyCom XVME656 Pentium VMEBus Board

- This XyCom board is compatible with the standard VMELinux kernel driver package from VMELinux Project
- A prepared kernel will be available soon. It will be based on the newest version of the Linux kernel and will include appropriate drivers for the onboard AHA2940/AIC7000 SCSI and 82558 Intel EtherExpress Ethernet peripherals. Check the website for details.

6.3 Dynatem DPC1−0367

- This board is compatible with the standard VMELinux kernel driver package from VMELinux Project
- A prepared kernel will be available soon. It will be based on the newest version of the Linux kernel and will include appropriate drivers for the onboard SCSI and Tulip Ethernet peripherals. Check the website for details.

6.4 SBS/Or Computer VP7

- This board is compatible with the standard VMELinux kernel driver package from VMELinux Project
- A prepared kernel will be available soon. It will be based on the newest version of the Linux kernel and will include appropriate drivers for PCNET Ethernet peripheral. Check the website for details.
- The VP7 has a nice feature which performs the BOOTP protocol without need of a bootrom or similar modification. However, you must ask SBS for an updated BIOS with this modification.

6.5 DY4 179, A Power PC board

- An independant engineer finds this board is compatible with the standard VMELinux kernel driver package from VMELinux Project

6.6 Planned Board Support

If you do not see VMELinux support for your board let us know. Maybe the manufacture will lend us a board for development.

7.Conclusion

VMELinux offers the user a low cost way to implement a VMEbus system quickly, reliably and with all the advantages of a unix environment. We are using VMELinux in our projects so you can be sure future developments will come quick. On the drawing board for this year are:

- Porting to other brands of Intel VMEbus boards,
- Porting of VMELinux to other processors that use the Universe chip,
- A study of running the VMELinux kernel driver module as a RT−Linux task.

This document outlines the steps you need to install the VMELinux Kernel Driver into the example Xycom XVME−655 Pentium VME board. It is our hope that others will attempt installation of VMELinux into other boards and let us know their success.

Mail any responses to The VMELinux Project. If you have a question or an update to the document let us know and we will add it.

8.FAQ

8.1 The Shell utilities return a bunch of stars (*) when I access a board I know is there. What gives?

Check to be sure the /dev/vme... files have their permissions set to 666. If not, the shell utilities will return a * in place of data to indicate an error condition similar to a VME bus error.

8.2 The Shell utilities still return a bunch of stars (*) when I access a board I know is there. Now what?

It is possible the ca91c042 Linux kernel module has been compromised. Get root access and type "lsmod" to review the loaded modules. Do you see the ca91c042? If yes, try removing it and reinstalling it with "rmmod ca91c042" and then "insmod /path/to/the/ca91c042.o" to get things up again. If it is not there check to see if you are loading the module when you boot the machine, etc.

8.3 The Shell utilities still return a bunch of stars (*) when I access a board I know is there. HELP?

Time to get a VMetro board into the VME cage and see if any accesses are occuring. Also look at the /proc/ca91c042 file to see if the read and write counters are incrementing.

8.4 How does VMELinux handle interrupts?

The driver does handle interrupts, but if you compile your interrupt handler program as a Linux loadable module, that program can handle the interrupts directly. Examples of this will be available soon. It is important to note that user level program can be made to handle interrupts, but it is a much better idea to have your interrupt handlers as part of the Linux kernel via loadable modules. Yes, you can totally hose the kernel if you do something wrong, but that is the trade off between safety and performance.

8.5 I have RedHat 5.1 and can't get VMELinux programs to compile.

RedHat 5.1 includes a new compiler. If you manually edit the Makefile in each directory to call up the new egcs compiler, things should compile. We fully intend to support RedHat 5.1 installations, but for now I suggest using 5.0 or Slackware.

8.6 I have RedHat 6.x so I assume the above issue is fixed. Right?

Maybe. RedHat threw us, and many other kernel module driver writers, a curveball with the move to the egcs compiler. Thankfully, the two compiler camps, GCC and egcs, have united their efforts. All this incompatibility should just go away. For the moment, however, VMELinux will only be tested with GCC 2.95.x so that is what we suggest you use for a compiler. If you type "gcc −−version" at your prompt and get an "egcs..." back then we cannot say it will work for you.