Virtual Machines for embedded software development

7. April 2014

Authors: Tord Fauskanger and Herman Wintermark

Abstract

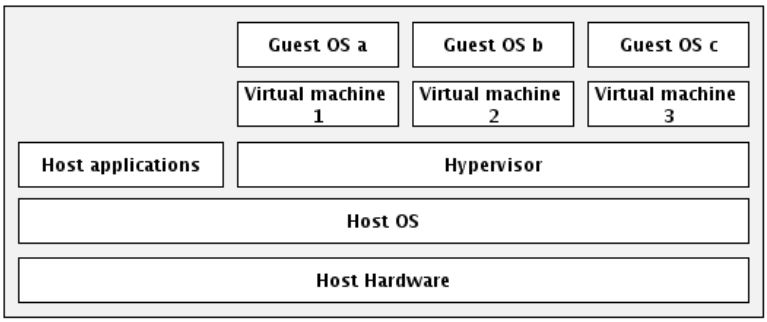
Using a virtual machine (VM) as a development platform for embedded software has many advantages. VMs enable better portability, greater redundancy, lower setup- and recovery times. The ability to archive the entire development environment allows better control over software packages and libraries needed for a specific project. Furthermore, you have the added benefit over having the ability to hand over the whole development environment to the customer after project completion in addition to the developed source code. The VMs host machine will need to have more memory than a machine that would run without a VM. When using a VM for development it is important to have a clear strategy in how licensing and source control should be handled.

What is a virtual machine?

A virtual machine (VM) is a software-layer abstraction of hardware that allows a process to execute in an emulated environment. Virtual machines can be divided into two categories; System VM and Process VM. This article only deals with System VMs, which run a guest OS in a virtual hardware environment. Process VMs, such as Java Virtual Machine, translate generic calls into platform specific calls.

The hypervisor, also called the virtual machine monitor, runs on the host OS and allocates resources to each VM. The VM typically emulates a physical machine, but requests for hardware resources like CPU, memory, hard disk, I/O devices and network are managed by hypervisor which translates these requests to the underlying physical hardware. The hypervisor can allow many individual, isolated VM environments to run in parallel. Virtual machines enables portable computing environments that is independent of a specific machine and can easily be moved or copied.

Most hypervisors support shared folder between guest OS and host OS and bi-directional clipboard. To enable these services the guest OS must install extra software. These utilities makes it simple to switch between working within the host OS and guest OS's.



How is it used and what does it solve?

At Bitvis we use VMs for our development environments as it gives us greater flexibility and mobility. With VMs we get a computer independent development environment that can easily be backed up and distributed to a new machine. This enables us to ensure that all developers on the project team have the same development environment. Having every developer working within identical environments have several benefits like; getting rid of all obscure compiling bugs caused by developers having different version of software and eliminating a large portion of “works on my machine” issues.

By having the development environments sandboxed within a virtual machines we gain additional benefits. First, we have an environment that can run Windows, Linux or Mac OS X. No more struggling with Cygwin or incompatibilities with Linux and OS X. On top of this you get all the sandboxing gains of virtualization. You can have multiple project with potentially conflicting dependencies running side by side.

In addition, when running the development environment within a VM, new developers on the team can start working efficiently in a matter of minutes instead of using days to install all the required software.

Maintenance of the development environment is also simpler since instead of having X number of environments to maintain there is now only one. Administrators can also take advantage of virtual environments to simply backups, disaster recovery, new deployments and basic system administration tasks. The use of virtual machines also comes with several important management considerations, many of which can be addressed through general systems administration best practices and tools that are designed to managed VMs.

Having backed-up and archived our VMs we are able to quickly recreate the complete environment that was used to build a specific binary and add debug information if needed. When a customer wants to continue the development of an embedded software solution, they just require a copy of the virtual machine that was used in the development, instead of spending a lot of time getting the project to work on their own computers.

Traditionally after a project completion, the customer would get a copy of the source code. With use of a Virtual Machine, we can now simply handover the whole development environment.

Things to consider before starting with VMs

Source control

Source control is essential in all software projects. With use of virtual machine for development, it might be tempting to just keep a backup of the virtual machine between releases, but this is not a good idea. It is hard to compare the content of one virtual machine with another one. Source code in a virtual machine should be under source control e.g. Git, like any other software development environments. The source control repository e.g. git master branch, should be stored on a server outside the virtual machine.

VM administration

When doing backups for virtual machines, one has to consider the tradeoff between simplicity and how much disk space one wants to use. If one simply stores copies of the virtual machines this will consume a lot of space but will require minimal effort. Installing complex backup solutions can reduce the disk footprint, which the VM backups consume, but this adds to the complexity. Disk space is cheap, and backups are cheap to do. Usually it is only necessary to take a snapshot of a virtual machine for each release. Storing VM images for each individual project and labeling them with release and date is a simple but effective solutions to keeping backups of the VMs. The backup VMs should be stored on a redundant medium.

With use of Vagrant and Provisioning tools virtual machines can be setup with a simple configuration files. These files can be stored with the rest of the product sources.

Licensing

Virtual machine with a guest OS that needs a commercial license e.g. Windows, needs to activate the license like normal installation. Several of the virtual machine monitors have tools for handling licensing issues related to the guest OS.

When multiple developers are going to use node locked license files, these licenses should be stored outside the virtual machine e.g. on a shared folder. This allows developers to use their own license.

Some software vendors have restrictions in the licensing agreement related to installing their software on a virtual machine. However, newer licensing agreements usually allow either installation on a host machine or a VM.

Hardware

The primary hardware requirement when running a virtual machine is memory. The host machine needs to be able to have enough resources to run two operating systems so if you need 4GB ram to run one

instance of your operating system your VM host machine will need 8GB. Modern computers have CPUs with mutiple cores, this enables the VM to permanently occupy one or more of these cores which normally gives acceptable performance as most application are only able to use one core anyways.

When doing debugging and programming of embedded software on a target device it is common to use tools like JTAG probes or in-circuit debugger. These tools are normally connected to the development machine by USB. The hypervisor will forward the USB device from the host machine to the guest OS.

What software is used?

VirtualBox

At Bitvis we have good experience with using Oracle VM VirtualBox. The core package is free software released under GNU General Public License version 2 (GPLv2).

VirtualBox can load multiple guest OSs under a single host operating-system (host OS). Each guest OS can be started, paused and stopped independently within its own virtual machine (VM). The user can independently configure each VM and run it under a choice of software-based virtualization or hardware assisted virtualization if the underlying host hardware supports this. The host OS and guest OSs and applications can communicate with each other through a number of mechanisms including a common clipboard, virtualized network facility and shared folders.

Vagrant

Vagrant is a tool that is used for configuration of a VM and can be used with VirtualBox virtualization platform. Vagrant introduces a common configuration format and workflow for describing and creating development environments repeatably across Linux, Windows and Mac OS X. In vagrant a single file “Vagrantfile” is used to describe the VM. The file describes the type of machine, the software that it will install, and the way you can access the machine. The configuration file is created at project start and should be committed into version control along with the project.

With vagrant you get a persistant workflow to manage your VMs. Vagrant handles importing, creating, destroying, pausing and remote logon on your VMs and you don't have to think about the actual hypervisior running the VM.

Then a single command “vagrant up” is all that is needed to start the machine.

Instead of installing software on a VM by hand, you can use provisioners in Vagrant to do this automatically when the VM is created. Vagrant supports several methods for provisioning; Basic shell scripts, Chef or Puppet.

Example of a basic Vagrantfile:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.box = "precise64"
  config.vm.box_url = "http://files.vagrantup.com/precise64.box"
```

```
  config.vm.hostname = "DevelopmentBox"
  config.ssh.username  = "vagrant"
  config.vm.define "development" do |development|
    development.vm.network "private_network", ip: "192.168.3.2"
    development.vm.provider :virtualbox do |vb|
      vb.customize ["modifyvm", :id, "--memory", "2048", "--ioapic", "on"]
    end
  end

 config.vm.provision :chef_solo do |chef|
    chef.log_level = :debug
    chef.json = {
      "build_essential" => {
        "compileTime" => true
      }
    }

    chef.run_list = %w{
      recipe[build-essential]
      recipe[vim]
    }
end
```

This Vagrantfile configures a Ubuntu precise64 machine with memory of 2 Gb, fixed ip address 192.168.3.2 and with host name "DevelopmentBox". In addition it illustrates a basic example of using chef solo to install *build essential* and *vim* on the machine.

Use cases

At Bitvis we do a lot of FPGA development and these FPGAs have hard or soft CPU cores. We have developed on several occasions software for the Xilinx Zynq 7000 System on Chip devices. Development for Zynq devices requires a development application suite that is time consuming to install and configure. To minimize the development time we installed the software on a virtual machine. This way all the software developers received a working system without having to install all the software on multiple machines. This saves time and enable us to use our time writing software instead of configuring and installing software.

Another scenario where we have had good use of VMs is when setting up a continuous integration (CI) server for a project. When you have a working system that is able to build your project it is simple to use the same VM in a CI server, The CI server will then use this VM to build and run unit tests on the sources every time new code is committed. This enables us to more efficiently setup the CI environment and not duplicate the work of setting up a build environment for the CI server.

Many embedded software developers need to use specific operating systems like Linux for development while also needing another OS (Windows) for general day-to-day office applications. Some companies mandate a particular suite of applications for writing documents etc. and usually those applications does not run on Linux.

Embedded systems usually have a long lifespan. After the initial release of a product, it can take some time before new releases. In the meantime, there might be new versions of libraries, compilers and development tools installed on the development machines. This can cause incompatibility issues when there is a need for fixing bugs or adding new features to the product. By archiving and sandboxing the initial development environment in a VM, these potential issues are easy to handle.