International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-4, April 2019

Modelling and Formally Verifying Intel VT-x: Hardware Assistance for Processors Running Virtualization Platforms

Ram Chandra Bhushan, Dharmendra K Yadav

Abstract*: *The Virtualization in computer science is the process of creating a virtual replica of computer resources like processor, hardware platforms, network devices, etc. Hardware virtualization is the creation of Virtual Machines (VM) that acts like a real computer with an operating system. Virtualization is controlled by special software called hypervisor or Virtual Machine Monitor (VMM) which manages the resources for the running virtual machines. Virtual machines can't access hardware resources directly. The modern processors are enabled with the mechanisms to support the virtual machine environment. Intel provides hardware-assisted virtualization mechanisms for Intel processors. Intel VT-x provides virtualization mechanisms for processor virtualization. In this paper, we present a novel work of verifying the properties of Intel VT-x formally. The verification is carried out on the design level using the mCRL2 tool. The formal verification of Intel VT-x along with its code written in the mCRL2 modelling language is presented.

Index Terms*: *Formal Verification, Intel, mCRL2, Model Checkings.

I. INTRODUCTION

The concept of virtualization was first introduced in the 1960s by IBM which boosts the utilization of large and expensive mainframe systems. These systems got partitioned into logical, separate virtual machines that execute multiple applications and processes at the same time.

Virtualization, in general, refers to server virtualization, which means one physical server or machine gets partitioned into several virtual servers that can interact with other devices, applications, data, and users independently as it is in the case of a separate physical resource. Different virtual machines created on a single physical computer can run different OSs along with their multiple applications by sharing the shared resources. All virtual machine runs in isolation from each other and that way if one of the VM crashes, it doesn't affect the others.

There is a unique software which is responsible for providing virtualization known as Hypervisor or virtualization manager, acts as an interface between the hardware and the operating system. Hypervisor decouples both the OS and user applications from the resources of the machine. The hypervisor gives access of the processor and other resources to the operating systems and end applications. Virtualization technology can be used in reverse too, it means instead of partitioning one physical machine into several virtual machines, we can also combine multiple physical resources to form a bigger single virtual machine with more amount of resources. Virtualization used for storage purpose is an excellent example of reverse virtualization technology. In the case of network virtualization, the available bandwidth got split into different independent channels of different networks that are used by specific servers or devices. There is one more type of virtualization known as application virtualization which differentiates the applications according to the hardware and the OS by keeping them in a container which gets relocated without disrupting other systems. Several individual desktops are managed remotely with the help of desktop virtualization which enables a centralized server. Support staffs upgrade and patch in virtual desktops instead of the physical server. In industry, the virtualization is one of the hottest trends which helps organizations in increasing the utilization, flexibility, and cost-effectiveness in a distributed computing environment. Some of the leading vendor organizations that provide virtualization solutions are VMWare, Citrix, Microsoft, IBM, RedHat and many more.

II. BACKGROUND

There are usually two types of VMMs: Type 1 or bare metal and Type 2 or hosted VMM. Type 1 hypervisors doesn't require a host operating system instead it executes directly on the system hardware whereas type 2 also needs a host operating system. With the concept of virtualization, VMM multiplexes the hardware of the system and makes it possible for many virtual machines to use the system hardware simultaneously. The principle idea of virtualization is to run guest code on a (virtual) CPU and only intercept privileged operations accessing resources or system properties for which direct access is prohibited. The trap is the interception of the running guest. VMM then detects the reason for the trap and may react to the same.

Manuscript published on 30 June 2019.
* Correspondence Author (s)
Ram Chandra Bhushan*, Research Scholar, Department of Computer Science and Engineering, Motilal Nehru National Institute of Technology Allahabad, India.
Dharmendra K Yadav, Professor, Department of Computer Science and Engineering, Motilal Nehru National Institute of Technology Allahabad, India.

This method is also used to implement a technique called" trap and emulate." For example, if a guest accesses a device which is emulated by the VMM, a trap into the VMM occurs. The VMM itself or a specialized VM monitor emulates the requested operations of the guest software by modifying the state of the virtual processor or memory of the trapping VM.

A. Intel VT

Hardware-assisted virtualization is a technique for improving the overall efficiency of virtualization. It involves CPUs that provide support for virtualization in hardware and other hardware components that help improve the performance of a guest environment. Intel provides hardware-assisted virtualization mechanisms for Intel processors.

B. Intel VT-x

Intel VT-x provides virtualization mechanisms for processor virtualization. The Intel VT-x extensions are probably the best-recognized extensions, adding migration, priority and memory handling capabilities to a wide range of Intel processors. There is a special sort of operations specially designed for virtual machine operations enabled by a Virtual Machine Extension (VMX) flag. This mode is enabled by a special command VMXON which has two operating modes: VMX root mode and VMX non-root mode. The hypervisor uses VMX root mode while the non-root mode is used by the virtual machines running on it. To go from VMX root mode to non-root mode, we use command VMENTRY, and for the reverse operation, we use the command VMEXIT.

The VMX non-root operations, the restricted environment is used to facilitate the virtualization. But still, for the critical tasks to be performed, we have to use VMEXIT. The commands like VMRESUME or VMLAUNCH do the initiation of the VM entry. Only a guest can be active or logical processor at one time in a VMM. The VMM regains the control with the command VMEXIT and then reacts to the cause of the trap. The VMM may provide the scheduled processor time to a different virtual machine and act accordingly. The command VMXOFF disables VMX operations.

A logical processor uses virtual machine control data structures (VMCSs) while it is in the VMX operation mode. VMCS manage the transition of states of a virtual machine in VMX non-root mode. This data structure is manipulated by different commands such as VMWRITE, VMREAD, VMPTRLD, and VMCLEAR. For each VM, a different VMCS is maintained. A logical processor may keep some active VMCSs. At any given time, the current VMCS can be any one of the active VMCS. There are some instructions like VMLAUNCH, VMREAD, VMRESUME, and VMWRITE that operates only on the current VMCS.

This is very important to know how the logical processor come to know that which VMCS is active and current. The address of a VMCS is the memory operand of the instruction VMPTRLD. Once the instruction executes that VMCS is both active and current on the logical processor. If another VMCS is there which has been active remains so, but no other VMCS is current. If VM entry operation is performed successfully, the VMCS, one who got referenced by the VMCS link pointer becomes active on the virtual processor. But, it would not change the current VMCS identity. After execution of the instruction VMCLEAR, that VMCS is neither active nor current on the logical processor.

C. Intel VT-d

Intel VT-d is the latest innovation of the Intel Virtualization Technology hardware architecture as on now. Intel VT-d helps the VMM better in utilizing the hardware. VT-d improves application compatibility and reliability and provides extra levels of manageability, I/O performance, security, and isolation. By using the VT-d hardware assistance built into Intel's chipsets, the VMM can achieve higher levels of performance, availability, reliability, security, and trust. Intel VT-d enables protection by restricting direct memory access (DMA) of the devices to pre-assigned domains or physical memory regions. A hardware capability known as DMA-remapping achieves this. The VT-d DMA-remapping hardware logic in the chipset sits between the DMA intelligent peripheral I/O devices and the computer's physical memory.

- Virtual Machine (Guest OS) requests peripheral devices in two forms. One is requesting without PASID another is requesting with PASID.
- The request from the peripheral device is transmitted to DMA remapping which intern converts VA (virtual address) of the peripheral device to HPA (Host Physical Address)
- The permission for accessing the peripheral device is first validated for any illegal access and then address of the allocated domain is mapped with peripheral device host physical address.
- Data from the peripheral device is sent to the host machine in the form of the data block.

D. Intel VT-x Vs Intel VT-d

- VT-x stands for Intel VT Technology (Vander pool Technology) that enables you to install 64-Bit Guest Operating Systems within a Bare Metal Hypervisor.
- VT-d stands for an Input/output Memory Management Unit (IOMMU) enables guest virtual machines to use peripheral devices, such as Ethernet directly, accelerated graphics cards, and hard-drive controllers, through DMA and interrupt remapping

E. Formal Verification

In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics [1]. Formal Specification and Verification have been widely applied on separation kernels in recent years [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12]. An overview is available in [13].

III. Informal Description of the System

A. Proposed Model

The model has abstracted most of the Intel VT-x operation to provide a mechanism to prove the correctness of Intel VT-x virtualization mechanisms. The detailed model is shown in Figure 1. The below two flags should be set for the VMXON operation to be performed.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

(A) CPUID.1: ECX.VMX [bit 5]
(B) CR4.VMXE [bit 13]

When CPUID.1: ECX.VMX [bit 5] is set and CR4.VMXE [bit 13] is not set, the system throws 'Invalid opcode exception.' When CPUID.1: ECX.VMX [bit 5] is not set then also it throws an

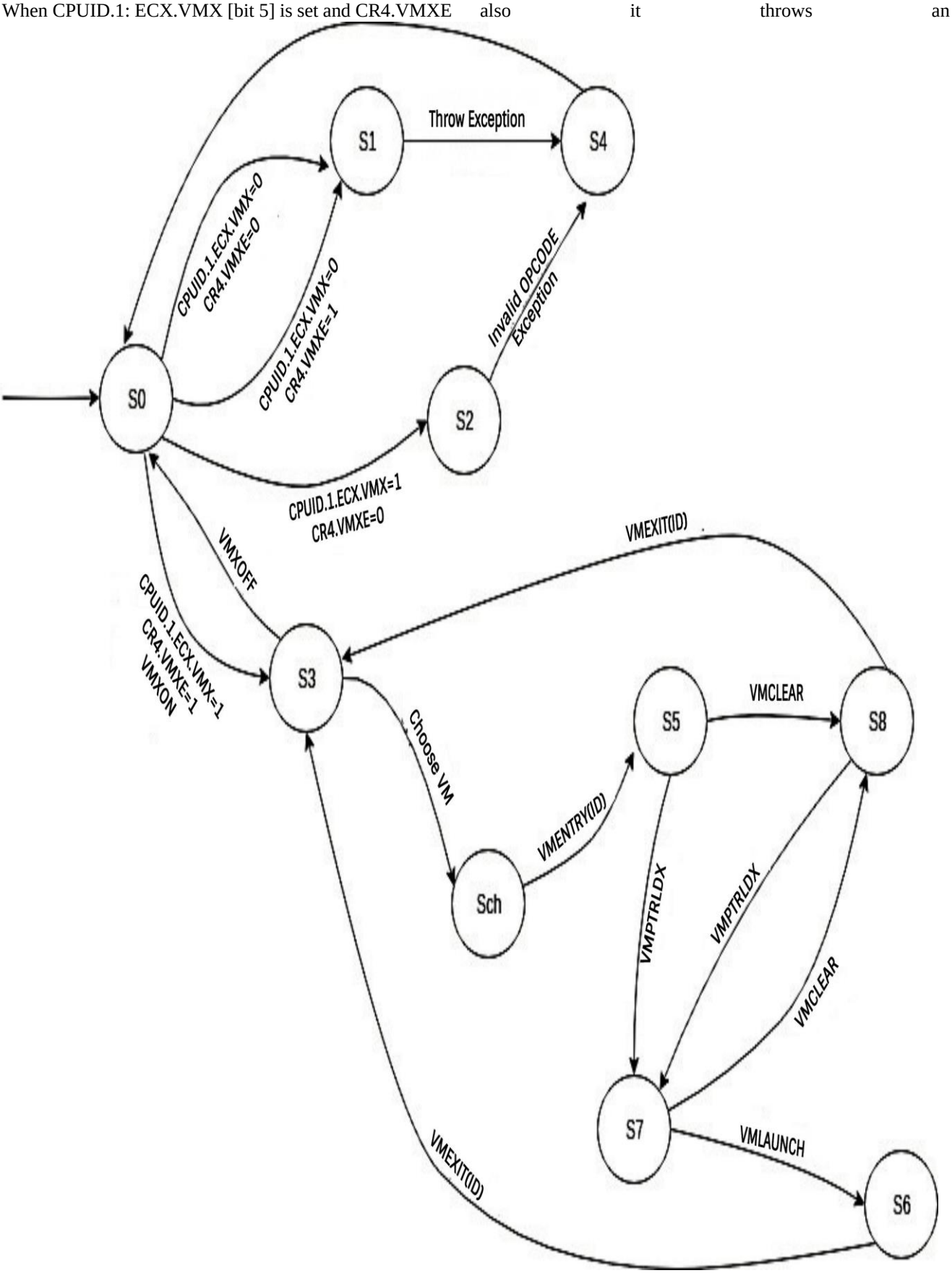

Fig. 1 The Proposed Model for Intel VT-x

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

exception. If both the flags are set, VMXON operation is performed, and VMX root mode is active. From this VMM state, we can enter in one of the many VMs by performing VMENTRY. By performing these operations, we enter the VMX non-root mode from VMX root mode. Every VM can have multiple VMCS used to store and load host and guest state. Initially, we reach a state in which our VM is active but not current and not launched. To make it current, we need to perform certain operations on VMCS. We need to make our VMCS current with VMPTRLD operation. To launch the current VM, we need to perform VMLAUNCH. Once it is launched, our virtual machine is enabled to use the logical processor. We can again make it not current and clear it's launch state with VMCLEAR command. Once we are done using the VM, we can perform VMEXIT operation to get back to the VMM. VMM identifies the reason for VMEXIT and acts accordingly. Now we are back to the VMX root mode. In this state, we can even shift from one VM to another one. Once all the required work is done, and we are no longer in need to use the VMs, we can exit the VMX operation mode with VMXOFF and reach the initial state. We can define each state as follows:

1. ***S0:*** Initial state through which we can enable VMX operation mode by changing the value of flags.
2. **S1:** It is a state in which value of flag CPUID.1: ECX.VMX [bit 5] is not set while the value of another flag (CR4.VMXE [bit 13]) can be set or not set.
3. **S2:** In this state flag CPUID.1: ECX.VMX [bit 5] is set but the other flag is not set, and when we try to execute VMXON operation in this state it throws an invalid opcode exception.
4. **S3:** In this state, we have successfully executed our VMXON operation, and as a result, all the VMX operations are enabled in this state. From this state, we can either choose to go back to the initial state S0 or move to a new state S4 after selecting a particular VMID.
5. **S4:** We reach this state once our system throws an exception due to incorrect values of flags and hence after this state we can go to initial state.
6. **S5:** In this state, we have entered a state in which our VMCS is active but not current and launched. To use the logical processor, VMCS should be current and launched.
7. **S6:** In this state, our VM is finally launched, and we can use a logical processor or perform VMEXIT operation to go back to VMX root mode in case we need to perform and critical operation.
8. **S7:** After execution of VMPTRLD command, we reach this state. VMPTRLD operation turns it's not current state to current, but still, it is required to be launched.
9. **S8:** In this state, our VM becomes inactive, not current and transparent, and hence we can use VMEXIT in case we want to switch to another VM.
10. **S9:** After we have chosen a particular VM to run on our hypervisor in the previous state, in this state we can enter to VMX non-root mode from the root mode with the command VMENTRY

B. Properties to be Verified

Formal verification is all about the verification of required properties of a system. The critical properties are listed below informally which has been converted into formal requirement in later section of the paper.

(a) When both the flags are set, then only VMENTRY should be possible.
(b) If CPUID.1:ECX.VMX[bit 5] is unset, VMXON should throw exception
(c) If CR4.VMXE flag is unset, VMXON should throw Invalid Opcode Exception
(d) If both flags are unset, VMXON should throw Exception
(e) Any VM should be able to be launched in future.
(f)Any launched VM should be able to exit to VMM.

IV. The mCRL2 Specification

A. mCRL2 Toolset

mCRL2 facilitates us with many operators. Some of them are Initialization (Init), Communication (comm (C, P)), and Allow (allow (A, P)). The communication operator performs a renaming of multi-actions in which every action has identical parameters. The operator a|b is considered as a multi-action operator, if a and b are actions, then a|b represents the simultaneous execution of a and b. The operator Allow (allow (A, P)) removes all multi-actions from the transition system that do not occur in A. Any states that have become unreachable will also be removed by mCRL2, as the resulting system is smaller and bisimilar. The code written below shows the implementation of Intel VT-x with the help of several processes and actions discussed earlier.

B. The mCRL2 Source Code

```
act
vmxon, vmxoff, vmentry, vmclearx, vmptrldx, vmlaunch,
vmexit, setflag1, setflag2, unsetflag1, unsetflag2, restart;
proc
HOST = setflag1 . setflag2 . vmxon . VMM + unsetflag1 .
setflag2 . vmxon . EXCEPTION + unsetflag1 . unsetflag2 .
vmxon . EXCEPTION + setflag1 . unsetflag2 . vmxon .
INVALIDCODE ;
   VMM = vmentry . VMCS + vmxoff . HOST ;
VMCS = vmptrldx . vmlaunch . VM + vmclearx . VMCS2 +
vmptrldx . vmclearx . VMCS2 ;
VMCS2 = vmptrldx . vmlaunch . VM + vmptrldx . vmclearx
. VMCS2 + vmexit . VMM ;
   VM = vmexit . VMM ;
   EXCEPTION = restart . HOST;
   INVALIDCODE = restart . HOST ;
init
    allow(
          {vmxon, vmxoff, vmentry, vmclearx, vmptrldx,
vmlaunch, vmexit, setflag1, setflag2, unsetflag1,
unsetflag2},
    comm(
          {},
          HOST
          ) ) ;
```

V. The Resultant Model

The proposed model for Intel VT-x was shown in Fig. 1, which shows the working principle of the virtualization implementation in Intel VT-x processors. Next the model has been coded into mCRL2, which then generates the state space diagram with the help of LTSGraph tool supported by mCRL2 toolset. The resultant model has been shown in Fig. 2.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

VI. FORMAL VERIFICATION USING μ-CALCULUS

The μ-calculus used in mCRL2 is a first-order modal μ-calculus along with data-depended processes and regular formulas [14]. This section shows the verification of all the requirements mentioned above. The tool lps2pbes is used for verifying the formulas written in μ-calculus. Each occurrences of a variable are bound by the nearest quantifier in the scope of bound variable, which has the same name and the same number of arguments. Below in the table the informal requirements written above are converted formally in μ-calculus and has been checked on the model shown in Fig. 1 with the help of mCRL2 toolset, which either gives a Boolean Value True or False that is mentioned in the Table 1.

Table 1: Formal Requirements with their Expected and Actual Output

Formal Requirement	Expected Result	Actual Output
When both the flags are set, then only VMENTRY should be possible.	TRUE	TRUE
<setflag1.setflag2.vmxon.vmentry> true		
If CPUID.1:ECX.VMX[bit 5] is not set, VMXON should throw exception	FALSE	FALSE
<unsetflag1.setflag2.vmxon.vmentry> true		
If CR4.VMXE flag is not set, VMXON should throw Invalid Opcode Exception	FALSE	FALSE
<setflag1.unsetflag2.vmxon.vmentry> true		
If both flags are unset, VMXON should throw Exception	FALSE	FALSE
<unsetflag1.unsetflag2.vmxon.vmentry> true		
Any VM should be able to be launched in future.	FALSE	FALSE
<setflag1.setflag2.vmxon> <vmentry> <vmptrldx.vmlaunch> false		
Any launched VM should be able to exit to VMM.	TRUE	TRUE
<setflag1.setflag2.vmxon> <vmentry> <vmptrldx . vmlaunch> <vmexit> true	TRUE	TRUE

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

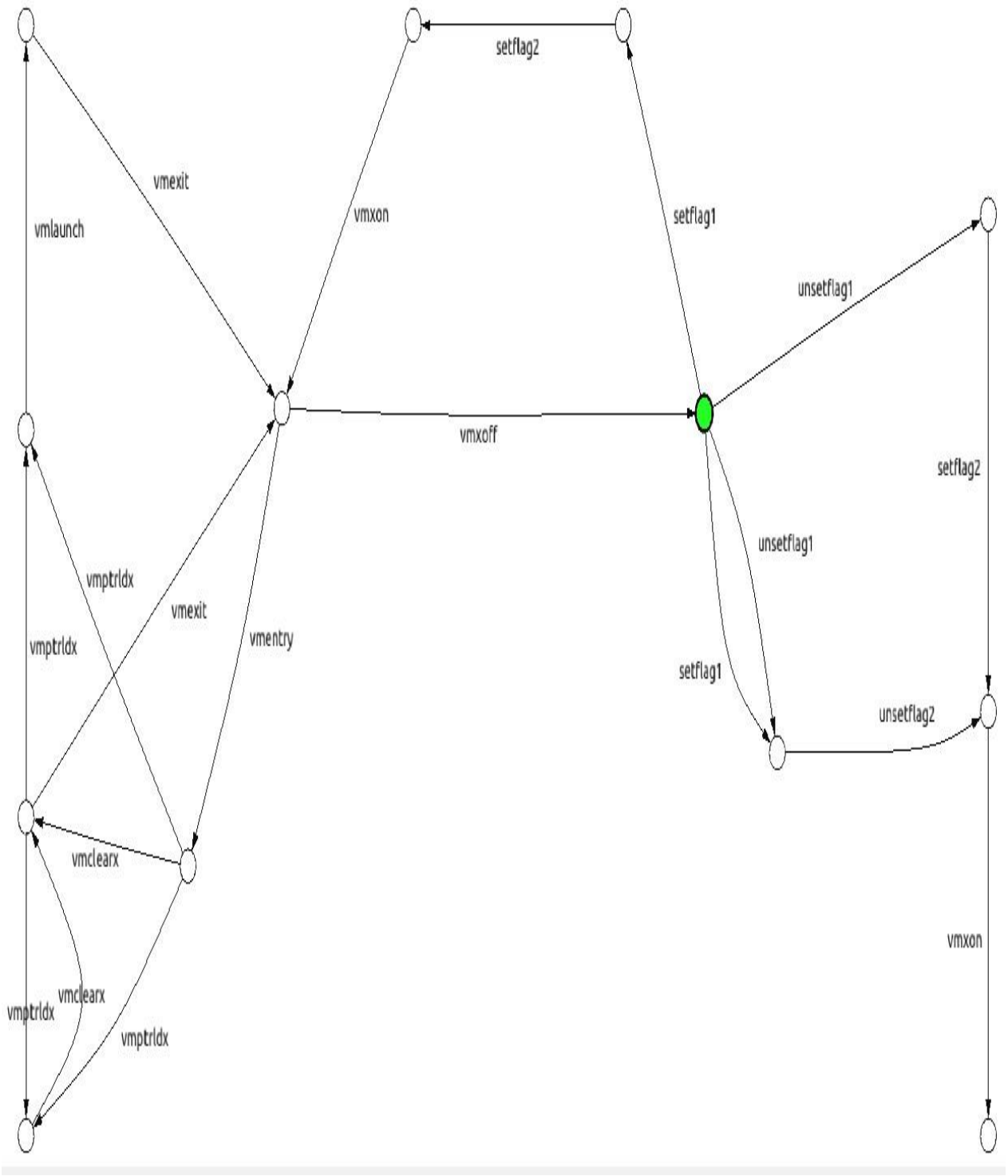

Fig. 2 State Space Diagram generated by mCRL2-LTSGRAPH for Intel VT-x

VII. CONCLUSION

The model shows the working of Intel-VTx processors for implementing virtualization among several virtual machines. The model checks that all the VMCS operations are correctly performed in a VM in VMX non-root mode. The generated model shows the working of a single VM having a single VMCS. In other words, only one VM is launched and current at a time. The model is verified for the specified requirements. The verification of model is done using the mCRL2 tool.

REFERENCES

1. Sanghavi, Alok (21 May 2010). "What is formal verification?". EE Times-Asia.
2. Craig, I.D.: Formal Refinement for Operating System Kernels, chap. 5. Springer (2007)
3. Dam, M., Guanciale, R., Khakpour, N., Nemati, H., Schwarz, O.: Formal verification of information flow security for a simple arm-based separation kernel. In: Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS'13). pp. 223–234. ACM, New York, NY, USA (2013).

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)

4. Freitas, L., McDermott, J.: Formal methods for security in the xenon hypervisor. International journal on software tools for technology transfer 13(5), 463–489 (2011)
5. Heitmeyer, C.L., Archer, M.M., Leonard, E.I., McLean, J.D.: Applying formal methods to a certifiably secure software system. IEEE Transactions on Software Engineering 34(1), 82–98 (2008)
6. T. C. Murray, D. Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao, and G. Klein. seL4: From general purpose to a proof of information flow enforcement. In IEEE Symposium on Security and Privacy, pages 415-429. IEEE Computer Society, 2013.
7. R. Richards. Modeling and security analysis of a commercial real-time operating system kernel. In D. S. Hardin, editor, Design and Verification of Microprocessor Systems for High-Assurance Applications, pages 301{322. Springer US, 2010}.
8. Sanan, D., Butterfield, A., Hinchey, M.: Separation kernel verification: The xtratum case study. In: Verified Software: Theories, Tools and Experiments, pp. 133–149. Springer (2014)
9. M. M. Wilding, D. A. Greve, R. J. Richards, and D. S. Hardin. Formal verification of partition management for the AAMP7G microprocessor. In Design and Verification of Microprocessor Systems for High-Assurance Applications, pages 175-191. Springer US, 2010.
10. Velykis, A., Freitas, L.: Formal modelling of separation kernel components. In: Proceedings of the 7th International Colloquium Theoretical Aspects of Computing (ICTAC'10). pp. 230–244. Springer (2010)
11. Verbeek, F.F., Tverdyshev, S.S., etc.: Formal specification of a generic separation kernel. Archive of Formal Proofs (2014)
12. Verbeek, F., Havle, O., Schmaltz, J., Tverdyshev, S., Blasum, H., Langenstein, B., Stephan, W., Wolff, B., Nemouchi, Y.: Formal api specification of the pikeos separation kernel. In: NASA Formal Methods, pp. 375–389. Springer (2015)
13. Zhao, Y.: Formal specification and verification of separation kernels: An overview. ArXiv e-prints (2015), http://arxiv.org/abs/1508.07066
14. mCRL2 Analysing System Behaviour. http://mcrl2.org/web/user manual/index.html

AUTHORS PROFILE

Ram Chandra Bhushan is a research scholar, pursuing his Ph.D. from one of the most reputed institution of India Motilal Nehru National Institute of Technology Allahabad also known as MNNIT Allahabad. He passed his M.Tech from Indian Institute of Technology Kharagpur. His research area includes Formal Method, Verification, Automata, etc.

Dharmendra K Yadav is working as a Professor in Motilal Nehru National Institute of Technology Allahabad also known as MNNIT Allahabad. He has a vast experience of teaching and research. He acquired his Ph.D. from Indian Institute of Technology Bombay. His research area includes Formal Method, Verification, Automata, etc.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication (BEIESP)