# [v13,06/12] KVM: x86: Add Intel PT virtualization work mode

| | |
|---|---|
| **Message ID** | 1540368316-12998-7-git-send-email-luwei.kang@intel.com |
| **State** | New |
| **Headers** | show |
| **Series** | Intel Processor Trace virtualization enabling |
| **Related** | show |

| | | | |
|---|---|---|---|
| 10654365 | diff (/patch/10654365/raw/) | mbox (/patch/10654365/mbox/) | series (/series/34463/mbox/) |

## Commit Message

Kang, Luwei (/project/kvm/list/?submitter=168537) Oct. 24, 2018, 8:05 a.m. UTC

```
From: Chao Peng <chao.p.peng@linux.intel.com>

Intel Processor Trace virtualization can be work in one
of 2 possible modes:

a. System-Wide mode (default):
   When the host configures Intel PT to collect trace packets
   of the entire system, it can leave the relevant VMX controls
   clear to allow VMX-specific packets to provide information
   across VMX transitions.
   KVM guest will not aware this feature in this mode and both
   host and KVM guest trace will output to host buffer.

b. Host-Guest mode:
   Host can configure trace-packet generation while in
   VMX non-root operation for guests and root operation
   for native executing normally.
   Intel PT will be exposed to KVM guest in this mode, and
   the trace output to respective buffer of host and guest.
   In this mode, tht status of PT will be saved and disabled
   before VM-entry and restored after VM-exit if trace
   a virtual machine.

Signed-off-by: Chao Peng <chao.p.peng@linux.intel.com>
Signed-off-by: Luwei Kang <luwei.kang@intel.com>
---
 arch/x86/include/asm/intel_pt.h  |  3 ++
 arch/x86/include/asm/msr-index.h |  1 +
 arch/x86/include/asm/vmx.h       |  8 +++++
 arch/x86/kvm/vmx.c               | 68 ++++++++++++++++++++++++++++++++++++++++---
 4 files changed, 76 insertions(+), 4 deletions(-)
```

# Comments

Jim Mattson (/project/kvm/list/?submitter=169057) Oct. 24, 2018, 4:18 p.m. UTC | #1 (/comment/22283507/)

```
On Wed, Oct 24, 2018 at 1:05 AM, Luwei Kang <luwei.kang@intel.com> wrote:
> From: Chao Peng <chao.p.peng@linux.intel.com>
>
> Intel Processor Trace virtualization can be work in one
> of 2 possible modes:
>
> a. System-Wide mode (default):
>    When the host configures Intel PT to collect trace packets
>    of the entire system, it can leave the relevant VMX controls
>    clear to allow VMX-specific packets to provide information
>    across VMX transitions.
>    KVM guest will not aware this feature in this mode and both
>    host and KVM guest trace will output to host buffer.
>
> b. Host-Guest mode:
>    Host can configure trace-packet generation while in
>    VMX non-root operation for guests and root operation
>    for native executing normally.
>    Intel PT will be exposed to KVM guest in this mode, and
>    the trace output to respective buffer of host and guest.
>    In this mode, tht status of PT will be saved and disabled
>    before VM-entry and restored after VM-exit if trace
>    a virtual machine.
>
> Signed-off-by: Chao Peng <chao.p.peng@linux.intel.com>
> Signed-off-by: Luwei Kang <luwei.kang@intel.com>
> ---

> +#define SECONDARY_EXEC_PT_USE_GPA              0x01000000
> +#define VM_EXIT_CLEAR_IA32_RTIT_CTL            0x02000000
> +#define VM_ENTRY_LOAD_IA32_RTIT_CTL            0x00040000

Where are all of these bits documented? I'm looking at the latest SDM,
volume 3 (325384-067US), and none of these bits aredocumented there.

> +       GUEST_IA32_RTIT_CTL             = 0x00002814,
> +       GUEST_IA32_RTIT_CTL_HIGH        = 0x00002815,

Where is this VMCS field documented?

> +/* Default is SYSTEM mode. */
> +static int __read_mostly pt_mode = PT_MODE_SYSTEM;
> +module_param(pt_mode, int, S_IRUGO);

As a module parameter, this doesn't allow much flexibility. Is it
possible to make this decision per-VM, using a VM capability that can
be set by userspace? (In that case, it may make sense to have a module
parameter which allows/disallows the per-VM capability.)
```

```
> +static inline bool cpu_has_vmx_intel_pt(void)
> +{
> +       u64 vmx_msr;
> +
> +       rdmsrl(MSR_IA32_VMX_MISC, vmx_msr);
> +       return !!(vmx_msr & MSR_IA32_VMX_MISC_INTEL_PT);
> +}
```

Instead of the rdmsr here, wouldn't it be better to cache the
IA32_VMX_MISC MSR in vmcs_config?
Nit: throughout this change, the '!!' isn't necessary when casting an
integer type to bool.

Kang, Luwei (/project/kvm/list/?submitter=168537) Oct. 25, 2018, 12:35 a.m. UTC | #2 (/comment/22284099/)

```
> > From: Chao Peng <chao.p.peng@linux.intel.com>
> >
> > Intel Processor Trace virtualization can be work in one of 2 possible
> > modes:
> >
> > a. System-Wide mode (default):
> >    When the host configures Intel PT to collect trace packets
> >    of the entire system, it can leave the relevant VMX controls
> >    clear to allow VMX-specific packets to provide information
> >    across VMX transitions.
> >    KVM guest will not aware this feature in this mode and both
> >    host and KVM guest trace will output to host buffer.
> >
> > b. Host-Guest mode:
> >    Host can configure trace-packet generation while in
> >    VMX non-root operation for guests and root operation
> >    for native executing normally.
> >    Intel PT will be exposed to KVM guest in this mode, and
> >    the trace output to respective buffer of host and guest.
> >    In this mode, tht status of PT will be saved and disabled
> >    before VM-entry and restored after VM-exit if trace
> >    a virtual machine.
> >
> > Signed-off-by: Chao Peng <chao.p.peng@linux.intel.com>
> > Signed-off-by: Luwei Kang <luwei.kang@intel.com>
> > ---
>
> > +#define SECONDARY_EXEC_PT_USE_GPA              0x01000000
> > +#define VM_EXIT_CLEAR_IA32_RTIT_CTL            0x02000000
> > +#define VM_ENTRY_LOAD_IA32_RTIT_CTL            0x00040000
>
> Where are all of these bits documented? I'm looking at the latest SDM, volume 3 (325384-067US), and none of these bits aredocumented
> there.

This part is in the " Intel® Architecture Instruction Set Extensions and Future Features Programming Reference"
https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf

>
> > +       GUEST_IA32_RTIT_CTL             = 0x00002814,
> > +       GUEST_IA32_RTIT_CTL_HIGH        = 0x00002815,
>
> Where is this VMCS field documented?
>
> > +/* Default is SYSTEM mode. */
> > +static int __read_mostly pt_mode = PT_MODE_SYSTEM;
> > +module_param(pt_mode, int, S_IRUGO);
>
> As a module parameter, this doesn't allow much flexibility. Is it possible to make this decision per-VM, using a VM capability that can be
> by userspace? (In that case, it may make sense to have a module parameter which allows/disallows the per-VM capability.)
```

```
It is a good idea from my point of view, I think it need more discussion and can be implement in next phase if have strong requirement.

>
>
> > +static inline bool cpu_has_vmx_intel_pt(void) {
> > +       u64 vmx_msr;
> > +
> > +       rdmsrl(MSR_IA32_VMX_MISC, vmx_msr);
> > +       return !!(vmx_msr & MSR_IA32_VMX_MISC_INTEL_PT); }
>
> Instead of the rdmsr here, wouldn't it be better to cache the IA32_VMX_MISC MSR in vmcs_config?
> Nit: throughout this change, the '!!' isn't necessary when casting an integer type to bool.

MSR_IA32_VMX_MISC is not read frequency and just read once in this patch set during initialization.

Thanks,
Luwei Kang
```

Thomas Gleixner (/project/kvm/list/?submitter=107) Oct. 30, 2018, 9:30 a.m. UTC | #3 (/comment/22290859/)

```
Kang,

On Thu, 25 Oct 2018, Kang, Luwei wrote:
> > > +#define SECONDARY_EXEC_PT_USE_GPA              0x01000000
> > > +#define VM_EXIT_CLEAR_IA32_RTIT_CTL            0x02000000
> > > +#define VM_ENTRY_LOAD_IA32_RTIT_CTL            0x00040000
> >
> > Where are all of these bits documented? I'm looking at the latest SDM, volume 3 (325384-067US), and none of these bits aredocumented
> > there.
>
> This part is in the " Intel® Architecture Instruction Set Extensions and Future Features Programming Reference"
> https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf
>

Yet another PDF which will change it's location sooner than later. Can you
please stick that into the kernel.org bugzilla and reference the BZ in the
change log, so we have something for posterity?

Thanks,

        tglx
```

Paolo Bonzini (/project/kvm/list/?submitter=2536) Oct. 30, 2018, 9:49 a.m. UTC | #4 (/comment/22290911/)

```
On 30/10/2018 10:30, Thomas Gleixner wrote:
>> This part is in the " Intel® Architecture Instruction Set Extensions and Future Features Programming Reference"
>> https://software.intel.com/sites/default/files/managed/c5/15/architecture-instruction-set-extensions-programming-reference.pdf
>>
> Yet another PDF which will change it's location sooner than later. Can you
> please stick that into the kernel.org bugzilla and reference the BZ in the
> change log, so we have something for posterity?

Hopefully posterity will be able to read it in the SDM.  But I agree
it's a good idea to add it in the commit log.  Let's also wait for
Alexander to clarify what he thinks needs to be done.

Paolo
```

Kang, Luwei (/project/kvm/list/?submitter=168537) Oct. 30, 2018, 10:13 a.m. UTC | #5 (/comment/22290947/)

```
> >> This part is in the " Intel® Architecture Instruction Set Extensions and Future Features Programming Reference"
> >> https://software.intel.com/sites/default/files/managed/c5/15/architec
> >> ture-instruction-set-extensions-programming-reference.pdf
> >>
> > Yet another PDF which will change it's location sooner than later. Can
> > you please stick that into the kernel.org bugzilla and reference the
> > BZ in the change log, so we have something for posterity?
>
> Hopefully posterity will be able to read it in the SDM.  But I agree it's a good idea to add it in the commit log.  Let's also wait for Ale
> to clarify what he thinks needs to be done.

I will create a new bug for this feature in Bugzilla, please help confirm if can like this first (it my first time to create bug in kernel.or

Product: Virtualization
Component: KVM
Severity: normal   (option: high, normal, low, enhancement)
Hardware: i386
Kernel Version:  4.19
Summary: Intel Processor Trace enabling in KVM
Description:
Intel Processor Trace (Intel PT) is an extension of Intel Architecture that captures information about software execution using dedicated har

The suite of architecture changes serve to simplify the process of virtualizing Intel PT for use by a guest software. There are two primary e
1. Addition of a new guest IA32_RTIT_CTL value field to the VMCS.
  – This serves to speed and simplify the process of disabling trace on VM exit, and restoring it on VM entry.
2. Enabling use of EPT to redirect PT output.
  – This enables the VMM to elect to virtualize the PT output buffer using EPT. In this mode, the CPU will treat PT output addresses as Guest

Intel Processor Trace virtualization can be work in one of 2 possible modes by set new option "pt_mode". Default is System-Wide mode.
a. System-Wide mode (default):
   When the host configures Intel PT to collect trace packets of the entire system, it can leave the relevant VMX controls
   clear to allow VMX-specific packets to provide information across VMX transitions.
   KVM guest will not aware this feature in this mode and both host and KVM guest trace will output to host buffer.

b. Host-Guest mode:
   Host can configure trace-packet generation while in VMX non-root operation for guests and root operation
   for native executing normally.
   Intel PT will be exposed to KVM guest in this mode, and the trace output to respective buffer of host and guest.
   In this mode, the status of PT will be saved and disabled before VM-entry and restored after VM-exit if trace a virtual machine.

Attachment: < the PDF file >

Thanks,
Luwei Kang
```

Thomas Gleixner (/project/kvm/list/?submitter=107) Oct. 30, 2018, 10:23 a.m. UTC | #6 (/comment/22290959/)

```
On Tue, 30 Oct 2018, Kang, Luwei wrote:

> > >> This part is in the " Intel® Architecture Instruction Set Extensions and Future Features Programming Reference"
> > >> https://software.intel.com/sites/default/files/managed/c5/15/architec
> > >> ture-instruction-set-extensions-programming-reference.pdf
> > >>
> > > Yet another PDF which will change it's location sooner than later. Can
> > > you please stick that into the kernel.org bugzilla and reference the
> > > BZ in the change log, so we have something for posterity?
> >
> > Hopefully posterity will be able to read it in the SDM.  But I agree it's a good idea to add it in the commit log.  Let's also wait for A
> > to clarify what he thinks needs to be done.
>
> I will create a new bug for this feature in Bugzilla, please help confirm if can like this first (it my first time to create bug in kernel.
>

Looks good.

Thanks,

        tglx
```

Kang, Luwei (/project/kvm/list/?submitter=168537) Oct. 31, 2018, 12:36 a.m. UTC | #7 (/comment/22292499/)

```
> > > >> This part is in the " Intel® Architecture Instruction Set Extensions and Future Features Programming Reference"
> > > >> https://software.intel.com/sites/default/files/managed/c5/15/arch
> > > >> itec ture-instruction-set-extensions-programming-reference.pdf
> > > >>
> > > > Yet another PDF which will change it's location sooner than later.
> > > > Can you please stick that into the kernel.org bugzilla and
> > > > reference the BZ in the change log, so we have something for posterity?
> > >
> > > Hopefully posterity will be able to read it in the SDM.  But I agree
> > > it's a good idea to add it in the commit log.  Let's also wait for Alexander to clarify what he thinks needs to be done.
> >
> > I will create a new bug for this feature in Bugzilla, please help
> > confirm if can like this first (it my first time to create bug in
> > kernel.org :) )
> >
>
> Looks good.
>

Have done.
https://bugzilla.kernel.org/show_bug.cgi?id=201565


Thanks,
Luwei Kang
```

# Patch

| 10654365 | diff (/patch/10654365/raw/) | mbox (/patch/10654365/mbox/) | series (/series/34463/mbox/) |
|---|---|---|---|

```
diff --git a/arch/x86/include/asm/intel_pt.h b/arch/x86/include/asm/intel_pt.h
index 634f99b..4727584 100644
--- a/arch/x86/include/asm/intel_pt.h
+++ b/arch/x86/include/asm/intel_pt.h
@@ -5,6 +5,9 @@
 #define PT_CPUID_LEAVES                2
 #define PT_CPUID_REGS_NUM      4 /* number of regsters (eax, ebx, ecx, edx) */

+#define PT_MODE_SYSTEM         0
+#define PT_MODE_HOST_GUEST     1
+
 enum pt_capabilities {
        PT_CAP_max_subleaf = 0,
        PT_CAP_cr3_filtering,
diff --git a/arch/x86/include/asm/msr-index.h b/arch/x86/include/asm/msr-index.h
index 107818e3..f51579d 100644
--- a/arch/x86/include/asm/msr-index.h
+++ b/arch/x86/include/asm/msr-index.h
@@ -805,6 +805,7 @@
 #define VMX_BASIC_INOUT                0x0040000000000000LLU

 /* MSR_IA32_VMX_MISC bits */
+#define MSR_IA32_VMX_MISC_INTEL_PT                 (1ULL << 14)
 #define MSR_IA32_VMX_MISC_VMWRITE_SHADOW_RO_FIELDS (1ULL << 29)
 #define MSR_IA32_VMX_MISC_PREEMPTION_TIMER_SCALE   0x1F
 /* AMD-V MSRs */
diff --git a/arch/x86/include/asm/vmx.h b/arch/x86/include/asm/vmx.h
index ade0f15..b99710c 100644
--- a/arch/x86/include/asm/vmx.h
+++ b/arch/x86/include/asm/vmx.h
@@ -77,7 +77,9 @@
 #define SECONDARY_EXEC_ENCLS_EXITING           0x00008000
 #define SECONDARY_EXEC_RDSEED_EXITING          0x00010000
 #define SECONDARY_EXEC_ENABLE_PML               0x00020000
+#define SECONDARY_EXEC_PT_CONCEAL_VMX          0x00080000
 #define SECONDARY_EXEC_XSAVES                  0x00100000
+#define SECONDARY_EXEC_PT_USE_GPA              0x01000000
 #define SECONDARY_EXEC_TSC_SCALING              0x02000000

 #define PIN_BASED_EXT_INTR_MASK                 0x00000001
@@ -98,6 +100,8 @@
 #define VM_EXIT_LOAD_IA32_EFER                  0x00200000
 #define VM_EXIT_SAVE_VMX_PREEMPTION_TIMER       0x00400000
 #define VM_EXIT_CLEAR_BNDCFGS                   0x00800000
+#define VM_EXIT_PT_CONCEAL_PIP                 0x01000000
+#define VM_EXIT_CLEAR_IA32_RTIT_CTL            0x02000000

 #define VM_EXIT_ALWAYSON_WITHOUT_TRUE_MSR      0x00036dff

@@ -109,6 +113,8 @@
```

```
 #define VM_ENTRY_LOAD_IA32_PAT                  0x00004000
 #define VM_ENTRY_LOAD_IA32_EFER                 0x00008000
 #define VM_ENTRY_LOAD_BNDCFGS                   0x00010000
+#define VM_ENTRY_PT_CONCEAL_PIP                         0x00020000
+#define VM_ENTRY_LOAD_IA32_RTIT_CTL             0x00040000

 #define VM_ENTRY_ALWAYSON_WITHOUT_TRUE_MSR      0x000011ff

@@ -240,6 +246,8 @@  enum vmcs_field {
        GUEST_PDPTR3_HIGH               = 0x00002811,
        GUEST_BNDCFGS                   = 0x00002812,
        GUEST_BNDCFGS_HIGH              = 0x00002813,
+       GUEST_IA32_RTIT_CTL             = 0x00002814,
+       GUEST_IA32_RTIT_CTL_HIGH        = 0x00002815,
        HOST_IA32_PAT                   = 0x00002c00,
        HOST_IA32_PAT_HIGH              = 0x00002c01,
        HOST_IA32_EFER                  = 0x00002c02,
diff --git a/arch/x86/kvm/vmx.c b/arch/x86/kvm/vmx.c
index 641a65b..c4c4b76 100644
--- a/arch/x86/kvm/vmx.c
+++ b/arch/x86/kvm/vmx.c
@@ -55,6 +55,7 @@
 #include <asm/mmu_context.h>
 #include <asm/spec-ctrl.h>
 #include <asm/mshyperv.h>
+#include <asm/intel_pt.h>

 #include "trace.h"
 #include "pmu.h"
@@ -190,6 +191,10 @@
 static unsigned int ple_window_max        = KVM_VMX_DEFAULT_PLE_WINDOW_MAX;
 module_param(ple_window_max, uint, 0444);

+/* Default is SYSTEM mode. */
+static int __read_mostly pt_mode = PT_MODE_SYSTEM;
+module_param(pt_mode, int, S_IRUGO);
+
 extern const ulong vmx_return;
 extern const ulong vmx_early_consistency_check_return;

@@ -1955,6 +1960,20 @@  static bool vmx_umip_emulated(void)
                SECONDARY_EXEC_DESC;
 }

+static inline bool cpu_has_vmx_intel_pt(void)
+{
+       u64 vmx_msr;
+
+       rdmsrl(MSR_IA32_VMX_MISC, vmx_msr);
+       return !!(vmx_msr & MSR_IA32_VMX_MISC_INTEL_PT);
+}
```

```
+
+static inline bool cpu_has_vmx_pt_use_gpa(void)
+{
+       return !!(vmcs_config.cpu_based_2nd_exec_ctrl &
+                                SECONDARY_EXEC_PT_USE_GPA);
+}
+
 static inline bool report_flexpriority(void)
 {
        return flexpriority_enabled;
@@ -4580,6 +4599,8 @@  static __init int setup_vmcs_config(struct vmcs_config *vmcs_conf)
                        SECONDARY_EXEC_RDRAND_EXITING |
                        SECONDARY_EXEC_ENABLE_PML |
                        SECONDARY_EXEC_TSC_SCALING |
+                       SECONDARY_EXEC_PT_USE_GPA |
+                       SECONDARY_EXEC_PT_CONCEAL_VMX |
                        SECONDARY_EXEC_ENABLE_VMFUNC |
                        SECONDARY_EXEC_ENCLS_EXITING;
                if (adjust_vmx_controls(min2, opt2,
@@ -4625,7 +4646,8 @@  static __init int setup_vmcs_config(struct vmcs_config *vmcs_conf)
        min |= VM_EXIT_HOST_ADDR_SPACE_SIZE;
 #endif
        opt = VM_EXIT_SAVE_IA32_PAT | VM_EXIT_LOAD_IA32_PAT |
-               VM_EXIT_CLEAR_BNDCFGS;
+               VM_EXIT_CLEAR_BNDCFGS | VM_EXIT_PT_CONCEAL_PIP |
+               VM_EXIT_CLEAR_IA32_RTIT_CTL;
        if (adjust_vmx_controls(min, opt, MSR_IA32_VMX_EXIT_CTLS,
                                &_vmexit_control) < 0)
                return -EIO;
@@ -4644,11 +4666,20 @@  static __init int setup_vmcs_config(struct vmcs_config *vmcs_conf)
                _pin_based_exec_control &= ~PIN_BASED_POSTED_INTR;

        min = VM_ENTRY_LOAD_DEBUG_CONTROLS;
-       opt = VM_ENTRY_LOAD_IA32_PAT | VM_ENTRY_LOAD_BNDCFGS;
+       opt = VM_ENTRY_LOAD_IA32_PAT | VM_ENTRY_LOAD_BNDCFGS |
+               VM_ENTRY_PT_CONCEAL_PIP | VM_ENTRY_LOAD_IA32_RTIT_CTL;
        if (adjust_vmx_controls(min, opt, MSR_IA32_VMX_ENTRY_CTLS,
                                &_vmentry_control) < 0)
                return -EIO;

+       if (!(_cpu_based_2nd_exec_control & SECONDARY_EXEC_PT_USE_GPA) ||
+               !(_vmexit_control & VM_EXIT_CLEAR_IA32_RTIT_CTL) ||
+               !(_vmentry_control & VM_ENTRY_LOAD_IA32_RTIT_CTL)) {
+               _cpu_based_2nd_exec_control &= ~SECONDARY_EXEC_PT_USE_GPA;
+               _vmexit_control &= ~VM_EXIT_CLEAR_IA32_RTIT_CTL;
+               _vmentry_control &= ~VM_ENTRY_LOAD_IA32_RTIT_CTL;
+       }
+
        rdmsr(MSR_IA32_VMX_BASIC, vmx_msr_low, vmx_msr_high);

        /* IA-32 SDM Vol 3B: VMCS size is never greater than 4kB. */
```

```
@@ -6433,6 +6464,28 @@  static u32 vmx_exec_control(struct vcpu_vmx *vmx)
        return exec_control;
 }

+static u32 vmx_vmexit_control(struct vcpu_vmx *vmx)
+{
+       u32 vmexit_control = vmcs_config.vmexit_ctrl;
+
+       if (pt_mode == PT_MODE_SYSTEM)
+               vmexit_control &= ~(VM_EXIT_CLEAR_IA32_RTIT_CTL |
+                                   VM_EXIT_PT_CONCEAL_PIP);
+
+       return vmexit_control;
+}
+
+static u32 vmx_vmentry_control(struct vcpu_vmx *vmx)
+{
+       u32 vmentry_control = vmcs_config.vmentry_ctrl;
+
+       if (pt_mode == PT_MODE_SYSTEM)
+               vmentry_control &= ~(VM_ENTRY_PT_CONCEAL_PIP |
+                                    VM_ENTRY_LOAD_IA32_RTIT_CTL);
+
+       return vmentry_control;
+}
+
 static bool vmx_rdrand_supported(void)
 {
        return vmcs_config.cpu_based_2nd_exec_ctrl &
@@ -6567,6 +6620,10 @@  static void vmx_compute_secondary_exec_control(struct vcpu_vmx *vmx)
                }
        }

+       if (pt_mode == PT_MODE_SYSTEM)
+               exec_control &= ~(SECONDARY_EXEC_PT_USE_GPA |
+                                 SECONDARY_EXEC_PT_CONCEAL_VMX);
+
        vmx->secondary_exec_control = exec_control;
 }

@@ -6672,10 +6729,10 @@  static void vmx_vcpu_setup(struct vcpu_vmx *vmx)

        vmx->arch_capabilities = kvm_get_arch_capabilities();

-       vm_exit_controls_init(vmx, vmcs_config.vmexit_ctrl);
+       vm_exit_controls_init(vmx, vmx_vmexit_control(vmx));

        /* 22.2.1, 20.8.1 */
-       vm_entry_controls_init(vmx, vmcs_config.vmentry_ctrl);
+       vm_entry_controls_init(vmx, vmx_vmentry_control(vmx));
```

```
        vmx->vcpu.arch.cr0_guest_owned_bits = X86_CR0_TS;
        vmcs_writel(CR0_GUEST_HOST_MASK, ~X86_CR0_TS);
@@ -8018,6 +8075,9 @@  static __init int hardware_setup(void)

        kvm_mce_cap_supported |= MCG_LMCE_P;

+       if (!enable_ept || !cpu_has_vmx_intel_pt() || !cpu_has_vmx_pt_use_gpa())
+               pt_mode = PT_MODE_SYSTEM;
+
        return alloc_kvm_area();

 out:
```