Barrelfish Project

ETH Zurich

HPET Driver Report

30.08.2018

By: **Rana Afifi**

Supervised by: **Lukas Humbel , Roni Hacki and Timothy Roscoe**

Contents

Introduction

Nowadays devices tend to provide more features and extra flexibility to the user but along with these options, the complexity is multiplied. An example of this is the High Precision Event Timer(HPET). This device provides three ways to route interrupts which can make it difficult for a programmer to make the correct configuration and proper routing. However, in 2017 an interrupt routing model has been proposed in the following paper [2]. Our goal is to find out whether this model could be applied to make the proper routing for the HPET or not.

Background

The High Precision Event Timer is a memory-mapped non-PCI device that was developed by Intel and Microsoft and released in 2005. It consists of a main 64-bit up counter and *Timer Blocks* that can have from 3 to 32 *Timers*. A single HPET chip can support up to 8 *Timer Blocks* which is equivalent to 256 *Timers*. Each *Timer* has a Comparator and a *Match Register* whose content is frequently compared against the Main Counter. If a timer is configured in *Periodic Mode*, an interrupt is triggered once the value of the Match Register is equal to the value of Main Counter . Some Timers could also be configured to generate interrupts when the Main Counter is equal to multiples of the Match Register value which is referred to as the *Periodic Mode* [1],[3].

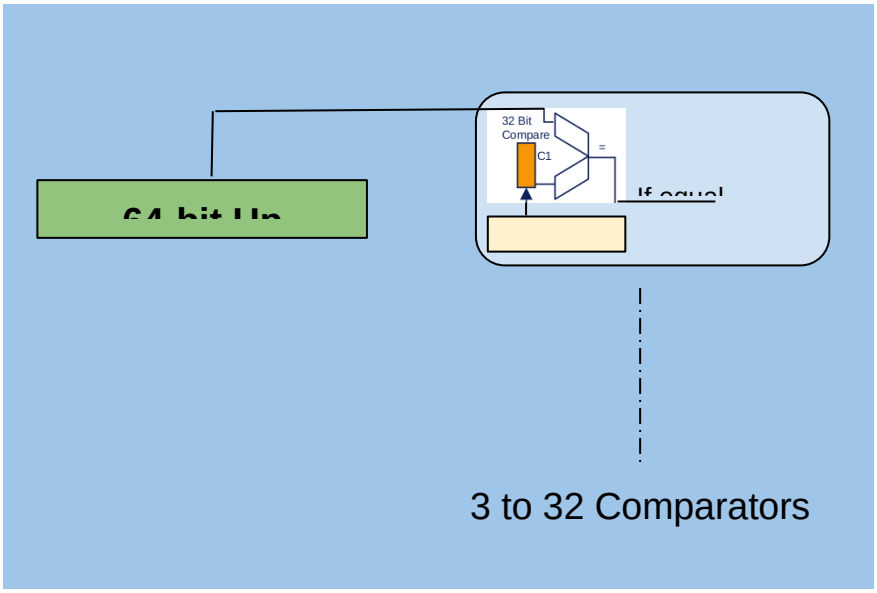

Fig1. Hardware Structure of the HPET

Currently, Barrelfish supports the the *LPC Timer* and the *Local APIC Timer*. The *LPC Timer* usually supports 4 timers which can be used to trigger interrupts[4]. The *Local APIC* Timer is mostly used for scheduling purposes since it is local to each CPU [5]. However, we found the HPET Timer of particular interest not only because it has more flexible features and configurations but because it has three different options for interrupt mapping; *Front Side Bus* mapping, *I/O APIC* mapping and the *Legacy* mapping which basically follows the conventional

8259 Timer Mapping. These three options provide a very good opportunity to test the Interrupt Routing Model proposed in [2].

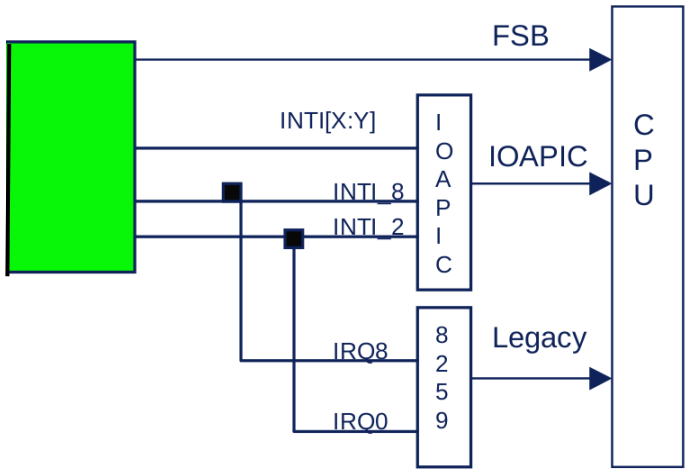

Fig2. HPET interrupt mapping

Adapted from *IA-PC HPET Specification*, by Intel , 2004

Interrupt Model

The interrupt model is written in Prolog, Eclipse CLP. It consists of interrupt sources, interrupt destinations and interrupt controllers[2]. Each controller has a set of *Inports* and *Outports*. The *Inports* are connected to interrupt sources while the *Outports* are connected to interrupt destinations and possibly other controllers; these ports are assigned unique numbers. We express the HPET controller in this model as follows: The *Inports* are connected to the timers themselves and a single *Outport* is connected to MSI Controller for the *FSB Mapping* and the rest are connected to the *I/O APIC* controller. Since Barrelfish runs in *I/O APIC* mode, the *Legacy Mapping* option is of no particular interest to us. Constraints of every controller are added to the model and the interrupt routing is treated as a constraint satisfaction problem. Typical constraints on the HPET Controller include timers support to *FSB mapping* and vector numbers that can be triggered by every timer. Fig(3) demonstrates how this interrupt model is applied on the HPET.

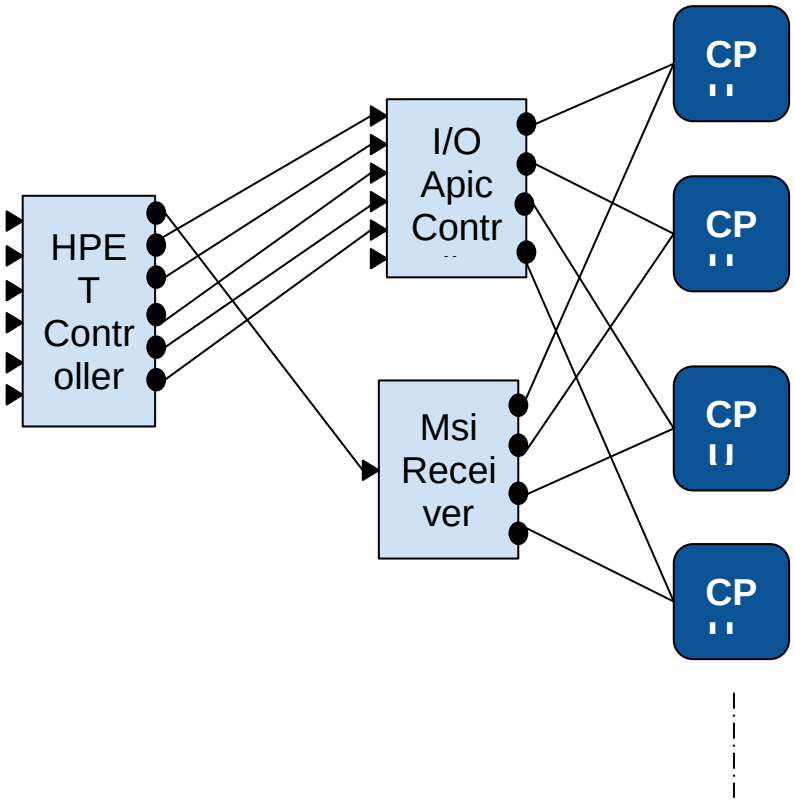

Fig3. Interrupt Model applied on the HPET Controller

Workflow

The HPET driver follows the same flow as any typical Barrelfish driver. When the ACPI discovers that the HPET is connected, it sends a message to *Kaluga*, the device manager in Barrelfish. *Kaluga* then requests HPET info from the ACPI table. The information that is recorded in the ACPI Table can be found in the IA-PC HPET Specification. For the Barrelfish driver, the information retrieved from this table was the base address of the memory mapped region, the size of ACPI Table and number of timers. Afterwards, the HPET controller is added to the *System Knowledge Base (SKB)* which in turn reserves unique identifiers for the ports and sends them back to *Kaluga* which creates the necessary capabilities and passes them along with the port range to the driver. Although there can be several timer blocks in the system, a driver is instantiated for a single timer block whose base address is found in the ACPI. If it is required that the driver works for more than a single timer block then the base addresses for timer blocks would vary by 1K memory size. Such support is not yet implemented into the driver.

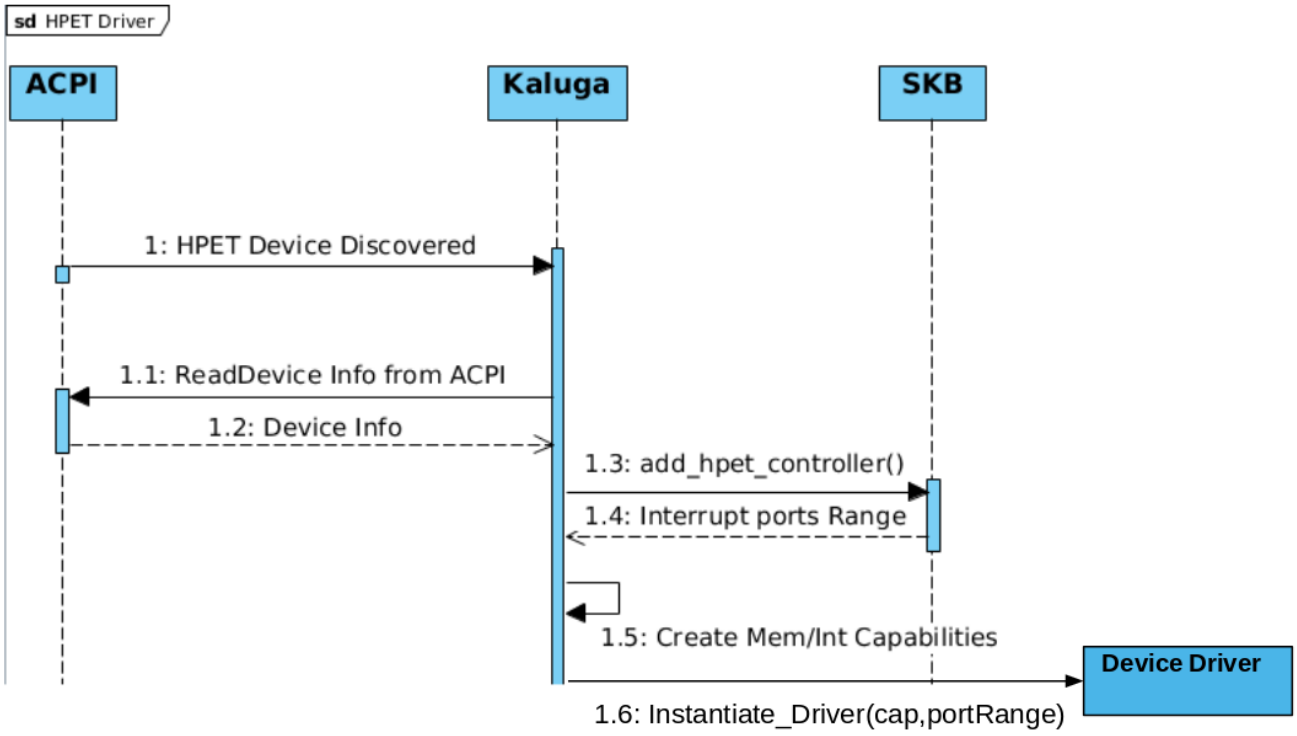

Fig4. Sequence Diagram of Driver Instantiation

Inside the driver, the memory capability is used to map the device register into the virtual address space; this

allows the driver to read the registers from the device. The *Mackerel* file written for the HPET allows easy manipulation of the device's registers.

By being able to access the device registers, the HPET timers and counters can be configured. In addition, the interrupt mapping capabilities are read from the registers of each timer and are inserted to the *SKB* as the validation predicate.

After the HPET connects to interrupt controller service, the service requests the routing from the *SKB* for specific timers. When the routing succeeds, the interrupt handler of the corresponding interrupt is added to the default waitset and the driver receives the destination interrupt capability in addition to the mapping data as it is required to be inserted to the HPET timers registers.

The sequence diagram of the workflow explained above is demonstrated in Fig(5).

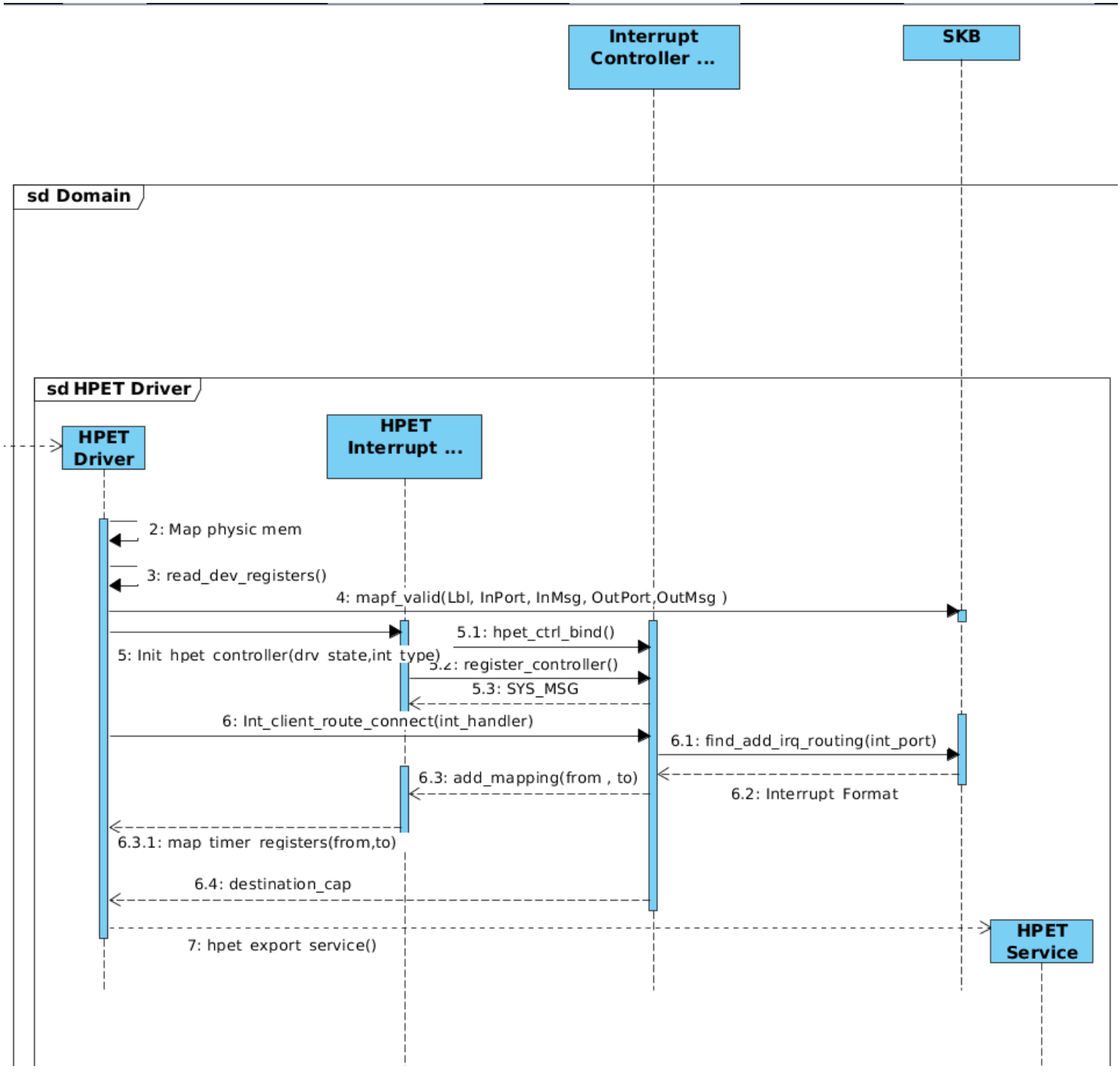

Fig5. Sequence Diagram of the driver's workflow

Modifications

A small modification has been introduced to the interrupt model which is the addition of connections between *Inports* and *Outports* of different controllers/devices. This has been done since the presence of global identifiers for the ports which could only be determined at runtime and which were shared between the ports of controllers that are connected to each other made it difficult to have an output range for the HPET in which part of it is the *MSI Inport* and the rest was the *I/O APIC Outport*. Thus, the solution was to change the way two controllers are connected and this change was that instead of sharing the same global port number if two ports are connected together, all ports have unique identifiers and a connection is inserted to the *SKB* to denote this connection.

Assumptions

One of the challenges faced was that there was no documentation of which *I/O APIC* is connected to the HPET, therefore an assumption has been made that the HPET is connected to the first *I/O APIC* in the system. This has been made because the HPET specification mentions that the interrupt numbers that are allowed for *I/O APIC* are GSI numbers and since they do not exceed 32, then it strong implies that it might be connected to the first *I/O APIC*. Also, in LINUX OS , the HPET driver is written such that it is connected to the first *I/O APIC*.

Conclusion and Future Work

Our work has proved that with the addition of connections - the modification mentioned previously- the interrupt model can be successfully applied to the HPET. As of the current time, the driver can trigger and receive interrupts from both the *FSB* and the *I/O APIC*. The next step to do is to export the HPET Timer as a service and start triggering interrupts at the intervals that the application requires; this would require wrapping the timers in an array of structs for better modularity and interface . Alternatively, a driver could be instantiated for each timer separately under the condition that they are all created in the same domain. This could even make timer interaction with other services more modular and more elegant.

References

[1] Wikipedia. *High Precision Event Timer*. Retrieved from https://en.wikipedia.org/wiki/High_Precision_Event_Timer

[2] Lukas Humbel,Reto Achermann,David Cock and Timothy Roscoe. 2017. Towards Correct-by-Construction Interrupt Routing on Real Hardware. In *Proceedings of the 9th Workshop on Programming Languages and Operating Systems , Shanghai, China , 8-14.*

[3] Intel. 2004. *IA-PC HPET (High Precision Event Timers) Specification*.

[4]Explore Embedded. *LPC Timers.* Retrieved from https://www.exploreembedded.com/wiki/LPC1768:_Timers

[5]OS Dev. *APIC Timer*. Retrieved from https://wiki.osdev.org/APIC_timer