White Paper

Real Time Applications
Precise Timing Solutions

Using IEEE-1588 Precise Time Protocol to Create a NOP_WAIT Instruction for the Nios® II Processor

Authors

Dan Biederman
Technical Leader
Intel Ethernet Products Group

Dennis Ejorh
Application Engineer
Intel Programmable Solutions Group

Table of Contents

Introduction

Precise time over networked systems through technologies like IEEE 1588 Precise Time Protocol (PTP), Precise Time Management (PTM), and Synchronous Ethernet create interesting possibilities and opportunities for real-time applications based on specialized microprocessors. One such opportunity is to incorporate precise time into a Nios® II processor custom NOP instruction, which we call NOP_WAIT. This custom instruction can use precise time information from an IEEE 1588 Grand Master or other source that allows a CPU or microcontroller to wait on a NOP instruction until it receives a predetermined precise time as shared by all devices in the system. Using this NOP_WAIT instruction, cameras, motors, energy sources, appliances, servers, networking schedulers, and so on can precisely synchronize across a network to perform coordinated operations, which improves a system's deterministic behavior.

This white paper details how you can incorporate precise time into the instruction set of a Nios II processor. The paper explains how to create a very simple precise time instruction, a specialized NOP, that simply waits for a precise time and then proceeds with the next instruction. This document provides the pseudo code for the instruction. Finally, this document reviews some potential applications and use cases for the new NOP-WAIT instruction.

One of the more interesting use case examples is the concept of a Time-Based Semaphore. Before delving into that topic however, here's a quick review of some important definitions.

Precise Time Accuracy and Precision

This paper discusses time precision and accuracy, which can be confusing. Arrows hitting a target is a great analogy to explain this concept. For precision, think of two arrows hitting precisely the same location on the target, with the second arrow splitting the first arrow. For accuracy, think of the two arrows hitting inside or near the center of the target. These analogies highlight the differences between precision and accuracy. Something can be precise but not accurate. For example: hitting the exact spot on a target far away from the center target many times in a row is precise, but not accurate. Similarly, something can be accurate but not precise. For example, an arrow hitting near the center target would be accurate, but the arrows could be apart from each other. Of course, the ideal situation is something that is both precise and accurate: that is all arrows hit in the same spot in the center of the target.

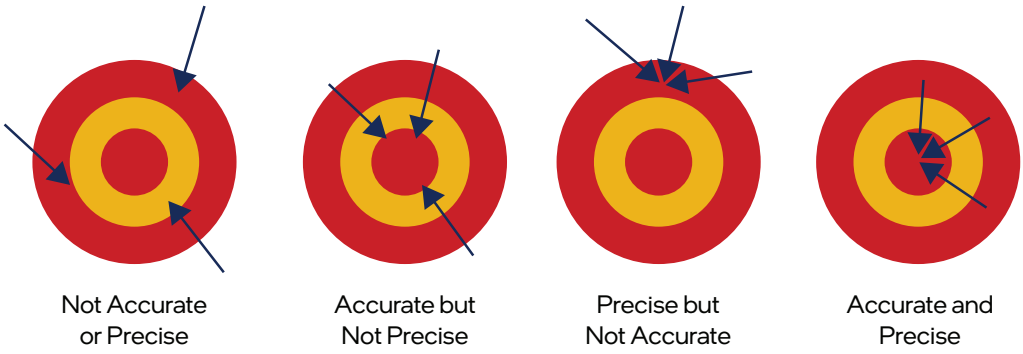

Figure 1. Different ways for arrows to hit a target illustrate the difference between accuracy and precision.

Time synchronization is very similar to the arrow analogy. Ideally, all devices in a network are synchronized precisely and accurately. That is, they are all synchronized to the same time, always. Unfortunately, in the real world, accuracy and precision can vary from the reference clock. This often happens because of slight frequency differences among the crystals and oscillators used by the different network devices.

In the practical world, networked devices need time-synchronization technologies to help reduce the real-world timing errors. Consequently, many organizations have developed synchronization methods to help align time precisely and accurately across network devices, synchronized to a reference clock value. These synchronization methods include Network Time Protocols (NTP), and IEEE 1588 PTP. However, even with tight synchronization, there will still be synchronization errors.

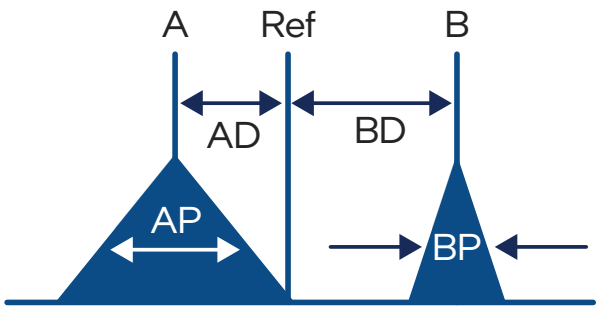

Figure 2. An ideal reference clock Ref, and two clocks, A and B, are synchronized and aligned to the reference clock with different mean values.

Figure 2 shows two clocks A and B trying to align to a reference clock, Ref. Clock A's mean value is a distance AD away from the reference clock, while Clock B's mean value is a distance of BD away. This distance or discrepency between a clock and its reference is related to the accuracy of the clock to the reference clock. In this case Clock A is closer to the reference, so it is more accrate. Similarly, AP is the width of all the samples in a histogram of Clock A versus the reference. BP is the widths of all samples in a histogram of Clock B versus the reference. In this figure one can see that AP is wider than BP. Hence Clock B is more precise than Clock A.

For there to be perfect precision, there would be no fluctuation in the samples resulting in a straight vertical line. Likewise for perfect accuracy, every sample should be centered perfectly around the reference clock. Finally to be both perfectly accurate and perfectly precise, every sample would line up exactly with each other and the reference clock.

However we do not live in a perfect world. We will have jitters that results in imperfect precision and accuracy.

A History of Time Synchronization

The Network Time Protocol, NTP, was invented in 1985 as a means of synchronizing time across computers and devices across the Internet. This protocol helped keep time values consistent across multiple devices in a network by reducing errors that could occur from inconsistent time values held within computers in the distributed system. NTP's time precision was in the millisecond or even microsecond range, which was quite good for 1985.

One of the issues with NTP was packet buffering in computers and networking equipment. Buffer delays often made the time synchronization very imprecise. Industrial and energy companies wanted to use Ethernet in their networks because they were less expensive than proprietary networks. However, the associated applications needed a protocol that was more precise than NTP, with precision in the sub-microsecond range. In 2002, the IEEE enacted PTP with the goal of improving the precision of the timing values. PTP also allowed for synchronizing various clocks and timers in measurement and control systems. These PTP systems can reside in a highly localized system, as in a factory, and can also synchronize devices across more distributed networks. The first version of this time precision protocol was called IEEE 1588 or IEEE 1588-2002.

After seeing the success of IEEE 1588-2002, the telecom industry became very interested in using the technology to synchronize cellular equipment. At the time, most cellular synchronization technology was based on the United States' Global Positioning System (GPS). However, the cellular industry wanted to be able to synchronize their cellular equipment without dependency on the GPS. For IEEE 1588 to work in cellular environments, PTP needed to be very precise across vast networks. To accomplish this, the IEEE introduced new technologies including the Transparent Clock to create a new version of the PTP standard called IEEE 1588-2008. These new technologies allow synchronization to occur over many nodes or hops in the cellular and service provider's networks.

The latest version of the PTP protocol is called IEEE 1588-2019. In this version of the standard, accommodations were also made for clocks in heterogeneous systems, which can have differing precision, resolution, and stability. Additional changes have been made for security, and virtualization.

The cellular companies have also introduced another synchronization technology called Synchronous-Ethernet, or Sync-E, technology. Sync-E is a physical-layer protocol. That means it works on the level of the actual electrical signals between devices for synchronizing the clock frequency instead of time. This technology allows the frequency of multiple devices in packet-based Ethernet networks to run at the same or extremely close to the same frequency. This technology allows the frequency of every device to be within a few PPM (parts per million) of each other. This allows the networked devices to implement synchronization, which means operation of the networked devices with the same frequencies. Such technologies had been used in telecommunication standards such as SONET and SDH. Now, these technologies were harnessed to make IEEE 1588 even more precise. Another way of implementing synchronization of PTP clocks is by using rateRatio. For more information on rateRatio, refer to Enhanced Synchronization Accuracy in IEEE1588.

This technology becomes more widely adopted as more and more engineers learn about the benefits of precise time.

A few years ago, data centers did not care about precise time, but today that is changing. Google has published papers on Carousel and Spanner that show that the importance of time is increasing in data center applications. Likewise, Facebook and the Open Compute Time Appliance Project (TAP) have just started using these technologies, with the goal of making precise time "better" in the data center. For more information on OCP TAP, visit their website on Time Appliances Project.

Currently there are proposals to use synchronization in other ways. Examples of such applications include Data-Plan Time Synchronization Protocols (DPTP) and HUYGENS. These protocols send thousands of packets to pass precise time to network devices including switches, routers, network interface cards (NICs), and processors. These protocols are easier to implement, as it does not involve the control plane. Data center architects and operators like this, because it reduces the data center "tax" that consumes processor cycles to implement synchronization protocols. However, these protocols may not be sufficiently precise for some of the latest IEEE 1588-2019 mobile backhaul requirements.

Additionally, Precise Time Synchronization technologies are a subset of the IEEE802.1 Time-Sensitive Networking (TSN). TSN consists of many standards for time synchronization (i.e. 802.1 AS), latency (i.e. 802.1Qbv, 802.1Qbu, etc.), reliability/reservation (i.e. 802.1CB, 802.1Qca, etc.) and management/configuration. TSN allows for more deterministic behaviors based on both precise and accurate time, as well as latency.

Finally, most of what has been discussed in this paper to this point describes precise time over wired interfaces like Ethernet. However today, precise time needs to travel over different connection technologies. One example of this is precise time over Wi-Fi. Example standards for this include IEEE 802.1AS with 802.11 TM (802.11-2012 specification) or FTM (in 802.11-2016 specification), which allow the benefits of time synchronization to be realized across wireless devices connected via Wi-Fi. Another example is PTM, which allows time synchronization over PCIe. PTM allows the benefits of time synchronization to be realized by a processor that is not directly connected to Ethernet or Wi-Fi.

Time Sync at the Processor

In today's world, we now make precise time available at the processor. Hence the processor can take advantage of this precise time to coordinate with other processors in a wired or wireless network, which allows all of these processors to work together more precisely and more efficiently. This white paper shows you how to implement time synchronization at the lowest level using a CPU NOP instruction. This paper shows:

1. How you can create a custom Precise Time NOP_WAIT instruction in a processor instantiated in an Intel® FPGA
2. An example of a Precise Time NOP_WAIT instruction
3. Potential use cases for the precise time using the Nios® II soft processor.

How to Create a NOP_Wait Precise Time Instruction Using Intel FPGAs

FPGAs allow for digital hardware designs to be implemented and changed simply by reprogramming them. Intel provides a soft-core processor, called the Nios II processor that can be modified to implement custom instructions. The Nios II processor is a small processor but a very powerful one. In some configurations, the Nios II processor can boot a version of Linux called ucLinux. The Nios II processor and the Intel® Quartus® Prime Software development tools allow designers to implement multiple custom instructions for the Nios II processor. This means designers can write and add instructions to help their implementation better meet design goals. For example, one can typically add a math function that allows the Nios II processor to execute one custom instruction that can replace tens or hundreds of instructions. This feature can greatly accelerate execution of some functions.

In this case, we have changed the processor's NOP instruction to wait for a specific time. That specific time could be based on an IEEE 1588 counter or some other internal timer. To keep the design simple, this example uses an internal counter.

Here is pseudo-code to explain at a high level what this new precise time instruction can do:

```
if op_code = NOP_WAIT begin
   if (counter >=  counter_value) begin
     Goto next instruction address
   end
   else being
     repeat the same instruction address
   end
 end
```

In this example, the operation waits at the same address until an internal "counter" value exceeds the specified "counter_value" needed to exit the loop. The "counter" in this pseudo code could be based on an IEEE 1588 clock counter, and the "counter_value" can be a precise time in the future when you want the processor to move on to the next instruction.

While a NOP may seem to be a very "boring" instruction, adding the concept of precise time opens new possibilities for this instruction. More relevant to this example, the NOP_WAIT instruction can force another set of instructions to execute at a precise time. Use case examples for this feature include:

- Starting code at precise times
- Precisely sharing a resource
- Preparing a unit to begin processing
- Coordinating bandwidth
- Coordinating power modes and activities
- Precise security updates
- Precisely capture events

General Use Case Example: Time-Based Semaphore

Now that we have a precise time NOP_WAIT instruction, let's see how it can be used. The time-based NOP_WAIT instruction is the basis for a powerful new concept called the Time-Based Semaphore. A semaphore is a means to control access to a common resource. A common semaphore used in industry today is a lock. That is, if one CPU has a lock on a resource, all other resources must wait until the lock is removed. When the CPU with the lock finishes, it frees the lock for the next CPU to use. Using the NOP_WAIT instruction, a Time-Based Semaphore serves as the means of access. For example, a developer's code could restrict requesting access to a shared resource until a precise time or time window.

In this case, the NOP_WAIT command can be used to force multiple processors to wait for a time to access a shared resource such as a memory. This situation is like freeway metering lights that reduce congestion and help traffic flow smoothly. However, in the example, one could coordinate the actions of tens, hundreds, or even thousands of processing elements. Each processing element is expected to get through a certain workload in a precise timeframe. If a processor gets ahead of its workload, it pauses using the NOP_WAIT command to wait before processing the next dataset. Thus, the shared resource does not receive requests that are very early, allowing the resource to be used by data for the given time window. This process also has the effect of making the resource accesses over a period more deterministic.

As shown in the following figure, the CPUs could be set to all process at the same time. However, the times used by the NOP_WAIT for each CPU could be staggered. This staggering method could be used to account for shared resource processing/response time, latency to/from the shared resource, and/or other metrics.

Precise Time Nios® II NOP_WAIT Industrial Use Cases

Now that we know how to create a precise time NOP instruction, let's see how industry could use this new instruction.

Robotics and Industrial Automation

In assembly lines, precise movement of equipment is needed. A precise time NOP_WAIT instruction could be used to delay the start of a program or assembly line operation until the precise time it is needed. Using a Nios II processor for such an operation would mean the Nios II processor is in a "wait" state and will start execution of the instructions after the precise time has occurred. Such a wait state would keep the Nios II processor 100% occupied and ready to precisely implement the code required by the device on the assembly line.

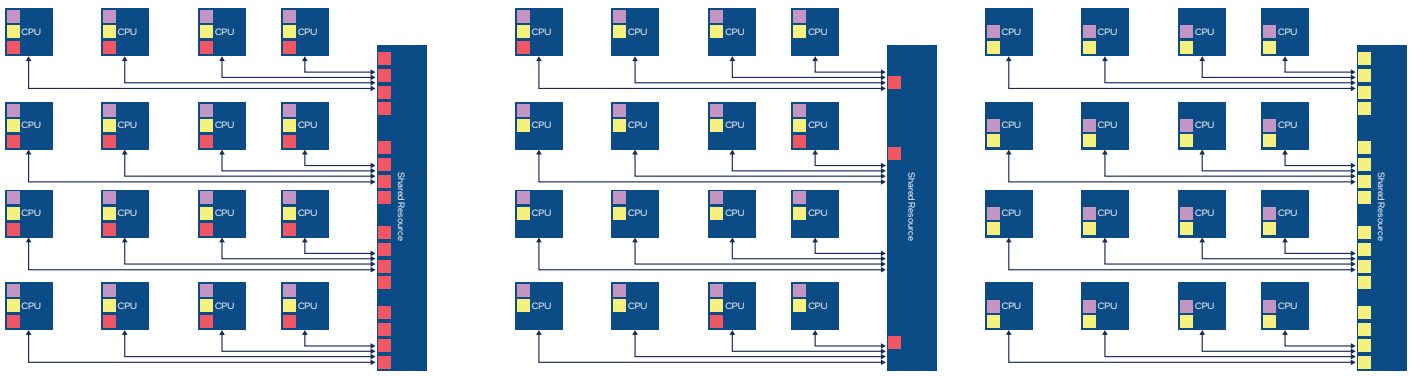

Figure 3. A group of 16 CPUs or processing elements that process three groups of data—red, yellow, and purple in order. The red data is handled first (left). The CPUs hold the yellow data using the NOP_WAIT instruction, allowing the red data to complete (middle). Then the yellow data is processed. When the yellow data has been processed, the NOP_WAIT instruction could be used to hold back the purple data for a precise time until the yellow data is complete first.

Network Packet Processing

One of the most popular uses of FPGAs in the last few decades has been in networking equipment. The network equipment use case tends to focus more on processing incoming and outgoing packets. However, with new technologies being developed around Software Defined Networks (SDN) and Network Function Virtualization (NFV), the concept of time becomes more critical. One piece of networking equipment could be used to control functions of many other devices. A simple operation, like statistics collection from other devices, could lead to all devices responding at the same time, causing burstiness on the network. It could also cause burstiness at the processor for the incoming packets. Using a precise NOP_WAIT command to send requests and commands at precise times can reduce the chances of this burstiness, making both the processing and the network bandwidth more efficient and more deterministic.

Autonomous Vehicles

One popular use case for small processors is to control a small block in an SoC. An example of these blocks could be a memory controller, network controller, or data controller. In many real-time use cases such as autonomous vehicles, sensor data, such as a video frame or LIDAR data is very time critical. An autonomous vehicle should not process a stop sign after the vehicle has passed it. Hence, a Nios II processor could be used to preload the frame data at precise times to allow the sensor data to be processed at the appropriate time. For example, a frame from one camera could be loaded at the precise time that the previous camera's frame is estimated to complete. This method can reduce bottlenecks in the pipeline into those processing units.

Cellular and the Internet of Things (IoT)

The cellular industry has been championing precise time in many ways because precise time windows are used for communications, data transfers, tower handoffs, and across a range of wireless devices. If time is not managed correctly, it can have a negative effect on the overall system, as well as degrading the quality of the user experiences. No one likes their cellular calls dropped in the middle of a conversation. An Intel® FPGA incorporating a Nios II processor with the precise time NOP_WAIT custom instruction can be used to precisely coordinate events inside and outside a device so that all required events can occur deterministically. Such events include powering on radios and equipment just in time to start transmission and/or reception.

AI-Assisted Medical Implants

Artificial Intelligence and Machine Learning is a hot topic these days. These are applications that can take advantage of a precise time NOP instruction. For example, in areas such as AI-assisted medical implant technology for spinal injuries, the need for localized precise time synchronization would be important. If the damaged nerves in the spine were routed through a novel bio-compatible routing/processing device that included an Intel FPGA, it would be paramount that all clocks in such a device were highly precise in their time representation to allow for proper processing and routing of incoming signals to fulfill the desired motor function. The addition of precise time could allow for a patient to operate the body more deterministically, as evidenced by how smoothly a patient walks or how precisely a patient can feed themselves.

Conclusion

In conclusion, this document shows how architects, designers, and engineers can implement a custom instruction for the Intel FPGA Nios II processor, called the NOP_WAIT instruction, that provides access to precise and accurate time. This custom instruction causes the Nios II CPU to stall on a NOP_WAIT instruction until a precise time. At that precise time the Nios II processor moves on to the next instruction. This paper also discussed the powerful concept of a Time-Based Semaphore, and how a NOP_WAIT instruction could be used to implement such a semaphore. Finally, we discussed several industrial use cases, that show how industry might use the NOP_WAIT in everyday applications.

References

1. Nios® II Software Developer's Handbook
   www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/nios2/archives/n2sw_nii5v2gen2-18-1.pdf
2. Intel® Quartus® Prime Software
   www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html
3. Altera IEEE1588 System Solution
   www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an739.pdf
4. IEEE 1588 V2 Test: Intel® FPGA Programmable Acceleration Card N3000Z
   www.intel.com/content/www/us/en/programmable/documentation/ycv1586051310815.html
5. TPAUSE Instruction - Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4
   https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html