Addicam V.Sanjay
Software Architect
Intel Corporation

Prashant Paliwal
Software Architect
Intel Corporation

Guidelines for migrating to Intel® Atom™ Processor from other Processor architecture

Architecting on ARM processor with the potential of future migration to Intel® Atom™ microarchitecture/ Intel Architecture

January 2010

323101

Executive Summary

There are many benefits to migrating from ARM* architecture to Intel® Atom™ microarchitecture, though there are challenges to overcome before the migration is complete. There are a number of decisions which need to be made related to the number of cores to use, whether to use a symmetric or asymmetric operating system, and most importantly will the real time requirements be met after the migration.

This white paper lists decisions that need to be made in relation to latencies, cores and the order in which the decisions need to be implemented. This paper also covers the experiments which need to be conducted to gather data related to these decisions. A flow chart is provided to help make a more intelligent and informed decision on the migration.

Contents

Business Challenge

Intel's move into embedded processors

Innovation in low power, high performance processors led to the birth of the Intel Atom processor, a low power and relatively high performance processor that revolutionized the computing industry powering the low cost laptops called "netbooks" and announced Intel's entry into embedded markets. The strong software ecosystem around established Intel® Architecture made it easier for customers and OEMs to adapt their designs to platforms built around the Intel Atom processor. Today, with continued innovation and a comprehensive roadmap for embedded devices, Intel Atom microarchitecture has significant advantages over other traditional embedded architecture. Intel Atom microarchitecture continues to find new applications ranging from industrial automation to in-vehicle infotainment. The Intel Atom Processor can be used in a low power high performance embedded device, such as a smart phone similar to the way ARM architecture has been used in such devices in the past.

Future growth of the Intel Atom microarchitecture into embedded markets requires a seamless transition for other processor based embedded software to Intel Architecture while maintaining the platform performance and compatibility. Winning designs by software is critical to encouraging designers to adopt Intel Atom microarchitecture for embedded markets. However, embedded systems pose a unique challenge. The nature of function specific embedded industry verticals like automotive, industrial automation and communications has resulted in proliferation of software solutions that have been very closely tied to the hardware capabilities of embedded platforms. This paper analyzes a case study with an Intel Atom processor-based solution that included migrating the ARM based software layers to Intel Architecture. The case study exposed the software challenges involved, and the conclusions outline a methodology that would facilitate the migration of ARM based software to Intel Architecture.

Intel® Atom™ Processor CE4100 - Cable Front End Pilot

Background on the Intel® Atom™ Processor CE4100

The Intel Atom Processor CE4100 is an Intel SOC solution for the set-top box market and other consumer electronic devices. The Intel Atom

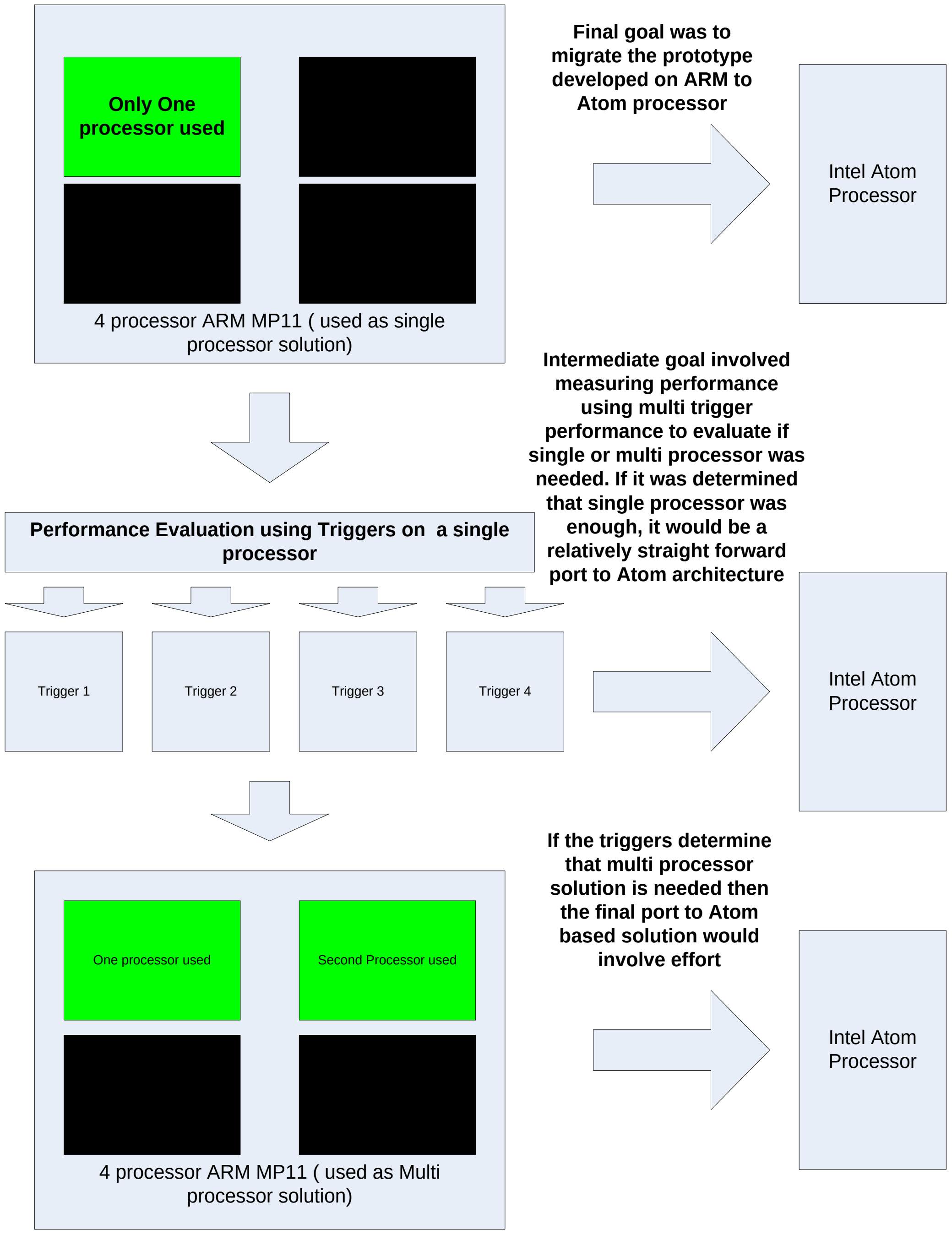
Processor CE4100 is based on a the Intel Atom core, designed using a 45 nm process. A Cable front end based on the Docsis 3.0 protocol was being designed to feed data to the Sodaville platform. The Cable front end was developed on an ARM MP11 processor (four cores). The architecture and software was designed on the ARM board with the knowledge that we will have to migrate this to Intel Architecture on the 32 nm process. All the data for this white paper has been derived from this experience.

Please read this section before proceeding

This section describes the ground conditions under which we conducted our experimentation and the various factors that are presented in this paper. This section will also describes the avenues, which this paper will not proceed into. We were tasked with providing a cable modem front end for the Intel® Atom™ Processor CE4100 media processor. The intent was that all of the processing needed by the cable modem would execute within the Intel Atom processor, which formed the core of the Intel Atom Processor CE4100. However because of logistic and administrative challenges, we decided to prototype the cable modem solution on a FPGA solution and eventually move the solution to the Intel Atom processor. We choose an ARM MP11 based FPGA solution, which includes four processors.

The general idea was to try to architect a solution on ARM MP11 which makes use of one processor to see if the required performance would be met on this one processor using variety of triggers. If the performance can be met using this one processor, then it would be very easy to migrate the architecture to an Intel Atom processor. If the performance requirements cannot be met using one processor, then this paper also presents a variety of multi processor solution which can be explored. There is no direct one to one correlation between the triggers and the variety of proposed multi processor solutions. All the triggers and the performance experimentation were conducted using one processor within the ARM MP11 core. The term processor and core is used interchangeably in this article.

Figure 1 Description of the Pilot


Only One processor used
4 processor ARM MP11 (used as single processor solution)
Final goal was to migrate the prototype developed on ARM to Atom processor
Intel Atom Processor
Intermediate goal involved measuring performance using multi trigger performance to evaluate if single or multi processor was needed. If it was determined that single processor was enough, it would be a relatively straight forward port to Atom architecture
Performance Evaluation using Triggers on a single processor
Trigger 1
Trigger 2
Trigger 3
Trigger 4
Intel Atom Processor
If the triggers determine that multi processor solution is needed then the final port to Atom based solution would involve effort
One processor used
Second Processor used
4 processor ARM MP11 (used as Multi processor solution)
Intel Atom Processor

Measurements on ARM

Some ARM processors have multiple cores. One of the design decisions we made was to determine whether we needed four cores (like some ARM processors) or one core (like some variants of the Intel Atom Processor). We defined a series of trigger points and measurement experiments to determine performance and possible alternative solutions based on one core. The measurement experiments involved:

- Kernel threads switching time
- Interrupt latency
- Scheduler latency
- Mutex/semaphore latency
- Processor latency (shared data)
- Actual code processing time

We also made extensive investigations into the choices of RTOS available on the various ARM platforms and their limitations. There was no RTOS available for the MP11 so we had to take a regular Linux version and spend time customizing it. We did not have a RTOS solution at all on ARM MP11.

We also ran experiments on the available memory speeds (single read/multiple reads) which showed that it was lower, and we had real time requirements from our software stack that we needed to meet. Based on this, we had to spend extra time in designing our software and come up with various alternatives such as:

- Multiple cores
- Symmetric operating systems
- Asymmetric operating systems
- Scenarios where in the code ran on a core without any operating system.
- Just firmware

Performance Tests that could be run on both Atom and ARM MP11 were not available. So we developed our own test.

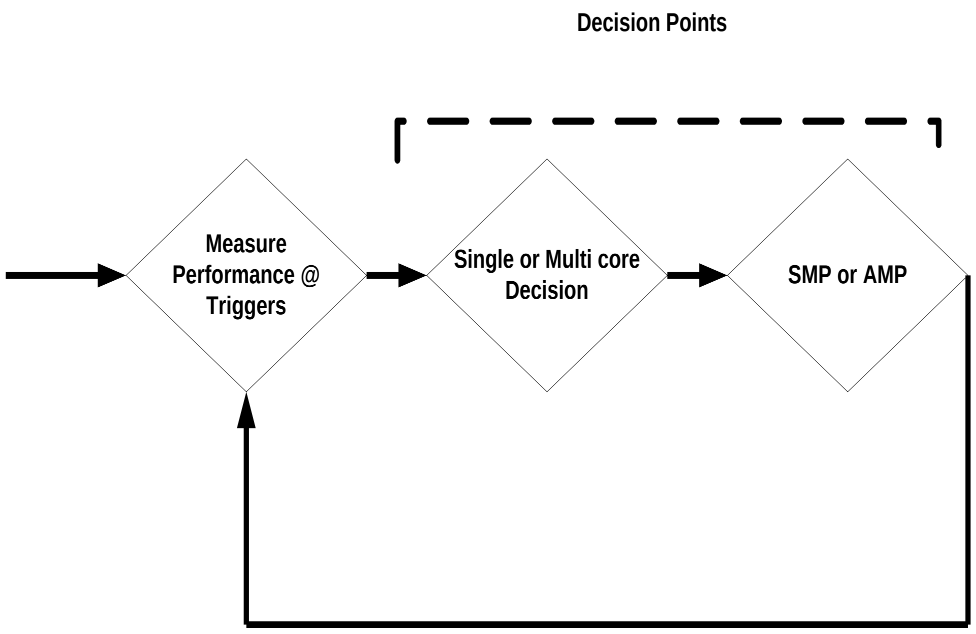
Decision Making Flow Chart for Moving From ARM to Intel® Atom™ Microarchitecture

Design Philosophy

Our basic philosophy has been to designing on a one processor system and moving to a multiprocessor only if performance tests warrant it. We believe that moving to a symmetric operating system leads us down a path where we will not have direct control on some of the most time critical aspects of a real time project. Development on a multiprocessor system on an embedded project is usually more complex and depends a great deal on the choices of RTOS available and on the symmetric/asymmetric choices that are available. Availability of development tools also play an important part in successfully moving from one processor to multiprocessor for an embedded project.

This section explains the decision making flowchart we used in determining if we need one processor or multiple processors. Figure 2 shows the basic decision making process.

Figure 2 Determining One Processor or Multiple Processors



The basic principle we followed in determining an ARM or Intel Atom microarchitecture solution involved measuring performance at various trigger points. Based on the results of these trigger points, an educated decision can be made on whether a single core is sufficient to meet the performance requirements, or if a multi core solution is needed. Further a decision needs to be made on what kind of operating system is needed (symmetric or asymmetric). These decisions will have a significant impact on the overall solution from a firmware, commercial OS choice, development cost and validation perspective.

Four triggers were used in our experiments. The number of triggers that are relevant to your individual experiment may vary. These triggers were run on a real time operating system. The triggers we experimented with were relevant from an embedded communication perspective (cable modem). A slight variation of these triggers will work for most of the embedded use cases:

- Trigger 1 is about meeting the most stringent real time requirement on your system on one channel
- Trigger 2 is about meeting the maximum data rate in one channel on your system

- Trigger 3 is about meeting the most stringent real time requirement on your system on multiple channels
- Trigger 4 is about meeting the maximum data rate on multiple channels on your system

In the cable modem front end project, these triggers translated themselves into

Trigger 1. Real time packet processing is taking more than 200 µs (P1).

Trigger 2. Increase the frequency/rate of the control packets to the maximum possible frequency (P1).

Trigger 3. Attempt to do real time packet processing for multiple channels within 600 µs (P2).

Trigger 4. Test whether the system can handle maximum data rates across four channels (P2).

All of these triggers were first performed on a single processor system. The results of these triggers allowed us to make an educated decision on the need for a multiprocessor system. At each of these trigger points, the following measurements were taken:

Key Measurements

- Kernel Thread Switching time
- Interrupt Latency
- Scheduler Latency
- Mutex, semaphore latency etc
- Processor latency (shared data)
- Actual Code processing time

The following sections provide details about the Triggers and how the key measurements will influence decisions.

Trigger 1: Meet the Most Stringent Real Time Requirement on One Channel

Value add of "if" statements

In the practical case we were dealing with, the goal was to process "map packets" within 200 µs. As we ran this experiment, we captured various performance measurements and built a list of IF/ELSE conditions to guide our decisions.

The value we are adding is in the IF statements. These were problems we encountered on a day to day basis. For example, we realized that we could not get any reference on these performance factors:

- Performance of actual kernel threads
- interrupt performance
- mutex measurements
- semaphore measurements
- latency measurements

For these factors, we had to conduct our own performance comparisons to measure it. This resulted in making kernel threads a key IF statement in our decision making flow chart.

We also realized that the actual memory bandwidth was lot less than what we had encountered in our research, so we more cognizant of that in our decision making.

IF/ELSE statement for decision making on Trigger 1

If Kernel Thread switch timing is more*,*

Try to merge thread functionality into single thread based on overall design

If threads cannot merged*,*

Move to SMP mode based on processor latency

If processor latency is too high,

Move to an Amp mode, and experiment and modify the data affinity to a processor

If interrupt latency is too high

Move to a preempt os

Move to an Amp mode, with map processing in firmware

If Scheduler latency is too high

Move to an Amp, where in we make sure that different high priority threads, ISR's, bottom halves are partitioned on different processors

Modify the scheduler switching functions (OS do Irq function) to our needs. Os Tuning

If Mutex latency, Semaphore latency, etc., is high

Change coding style

Optimize Kernel (OS) to reduce the latency

Get custom made patches from a third party source to address these problems

If Processor latency (communication, shared data, etc.) is too high

Partition data properly and fine tune communication mechanism (shared memory etc) between threads

If Actual code processing time is high

Optimize coding style, move to assembly if required

Move to a Smp based two processor solution

If more than one of the conditions above apply to your test, then a choose valid solution involving multiple processors, OS or no OS, assembly coding, and/or kernel optimization.

Trigger 2: Meet the Maximum Data Rate for One Channel

In the cable modem project, we increased the frequency of the control packets to the maximum possible frequency. As we started doing this experiment, we built a list of IF/ELSE conditions to guide our decisions. In the cable modem project, the data was either upstream or downstream. The control packets were packets like map processing, transmit scheduler, etc.

IF/ELSE statement for decision making on Trigger 2

If ISR's are coming at such a high rate that we do not have time for data processing

Find out in hardware if the packet is real time or not real time. For real time packets, raise an IRQ on a separate processor. For non real time packets, raise an IRQ on a different processor. For this we need an Amp solution. In this solution Real time control packets (map and TX scheduler) and upstream data path will be on one processor and the downstream data and other control packets on another processor.

If ISR's are coming at such a high rate that we are not able to process other control packets

Move to a Smp based mode where there are two processors to process interrupts

If this is not optimal and we see a lot of processor latency

Move to an Amp based two processor mode

If Threads are getting starved

Move to a Smp based two processor mode

Trigger 1 and 2 should be conducted in parallel and based on the data, an amalgamation of the listed possible solutions will be chosen.

Triggers 3 and 4 details are similar to Triggers 1 and 2. Results obtained from Triggers 1 and 2 should be extrapolated to see if it works cross Trigger 3 and 4 as well.

In our practical example, we came up with the following choices:

Trigger 3:

Attempt real time processing for four channels within 600 µs

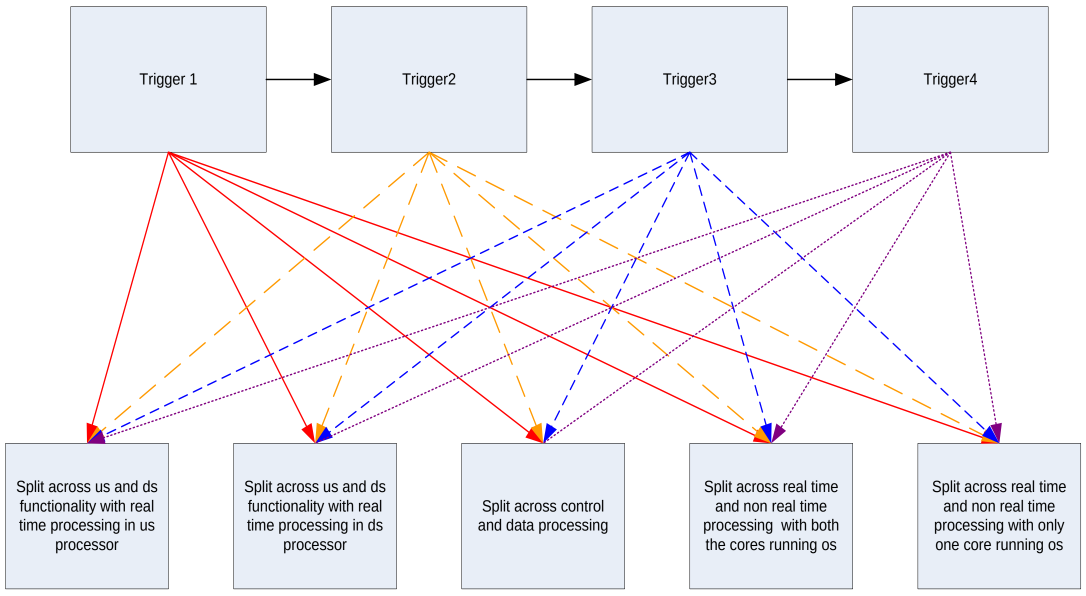
The problems and solutions are same as Trigger 1.

Trigger 4:

Attempt to handle maximum data rates across four channels (P2)

The problems and solutions are same as Trigger 2.

Figure 3 Relationship Between Triggers and Suggested MultiProcessor Solution


Trigger 1
Trigger2
Trigger3
Trigger4
Split across us and ds functionality with real time processing in us processor
Split across us and ds functionality with real time processing in ds processor
Split across control and data processing
Split across real time and non real time processing with both the cores running os
Split across real time and non real time processing with only one core running os

There is no 1:1 relationship between triggers and the suggested split of functionality across processors. All the triggers were carried out on one processor systems and depending on the performance results any one of the five suggested processor splits might work for the particular problem you are facing.

Split of Functionality across Processors, on the Basis of US/DS Functionality

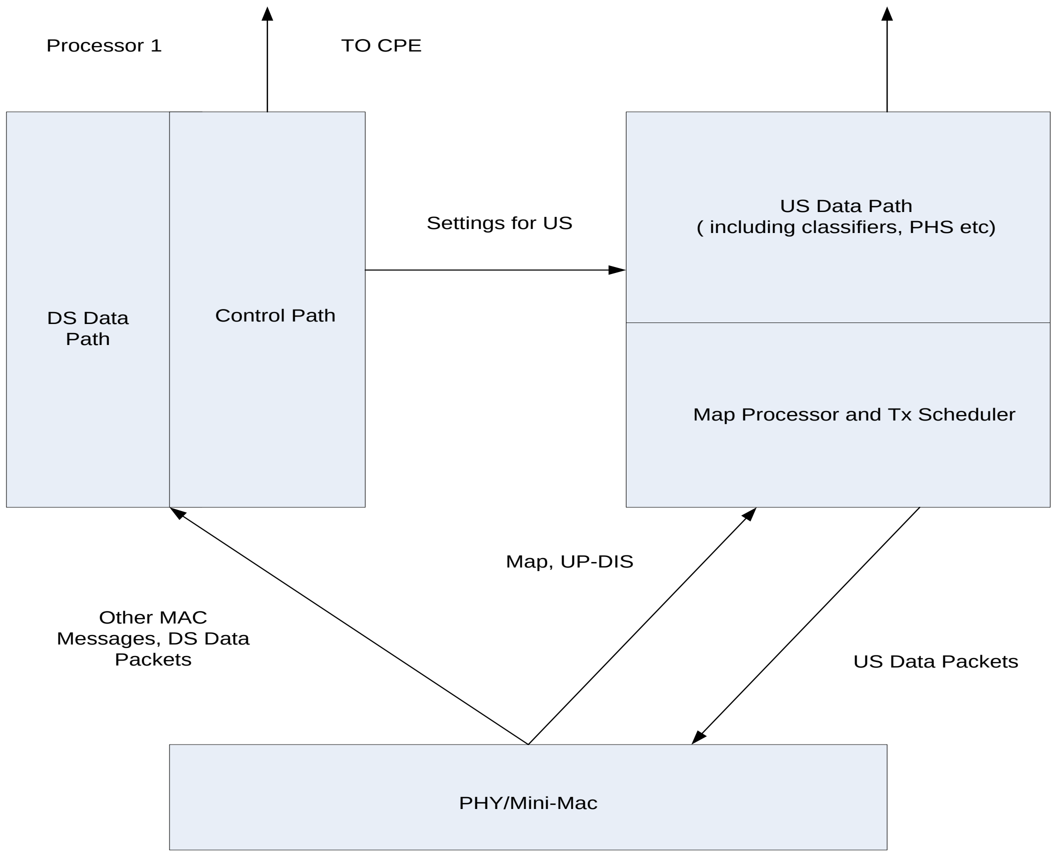
The next few sections give some example solutions based on cable modem functionality. Generic terms like upstream packets, downstream packets, control packets, map packets, and transmit scheduler are used in the following few sections. All of the block diagrams are designed to give the reader an idea of the splits possible. This paper does not go into detail on the cable modem solution, but rather to use the cable modem solution as an example.

Please read this section before proceeding

The following is a brief explanation of the overall cable modem design. This explanation will be relevant for the next four sections. The cable modem design has three distinct types of packets – upstream packets, downstream packets and control packets. There are time critical aspects in all of these packets. The most time critical packet of these are called map packets. Map packets come in the downstream direction and are control packets. Map packets specify the time instance at which the next upstream packet can be sent. In the map packet, the next upstream packet can be specified to be scheduled 200µs from the time the map packet has been received at the cable modem. This creates a real time requirement of 200 µs in which the map packet must be processed in the downstream direction so an upstream data packet must be prepared and sent out of the cable modem.

Most of this processing was executed using software. A component called Mini-MAC provided a basic level of filtering to figure out if a downstream packet is addressed to our cable modem. The rest of the processing, filtering, building, etc. of the packet was performed with software. The Phy was capable of both CDMA and TDMA transmissions.

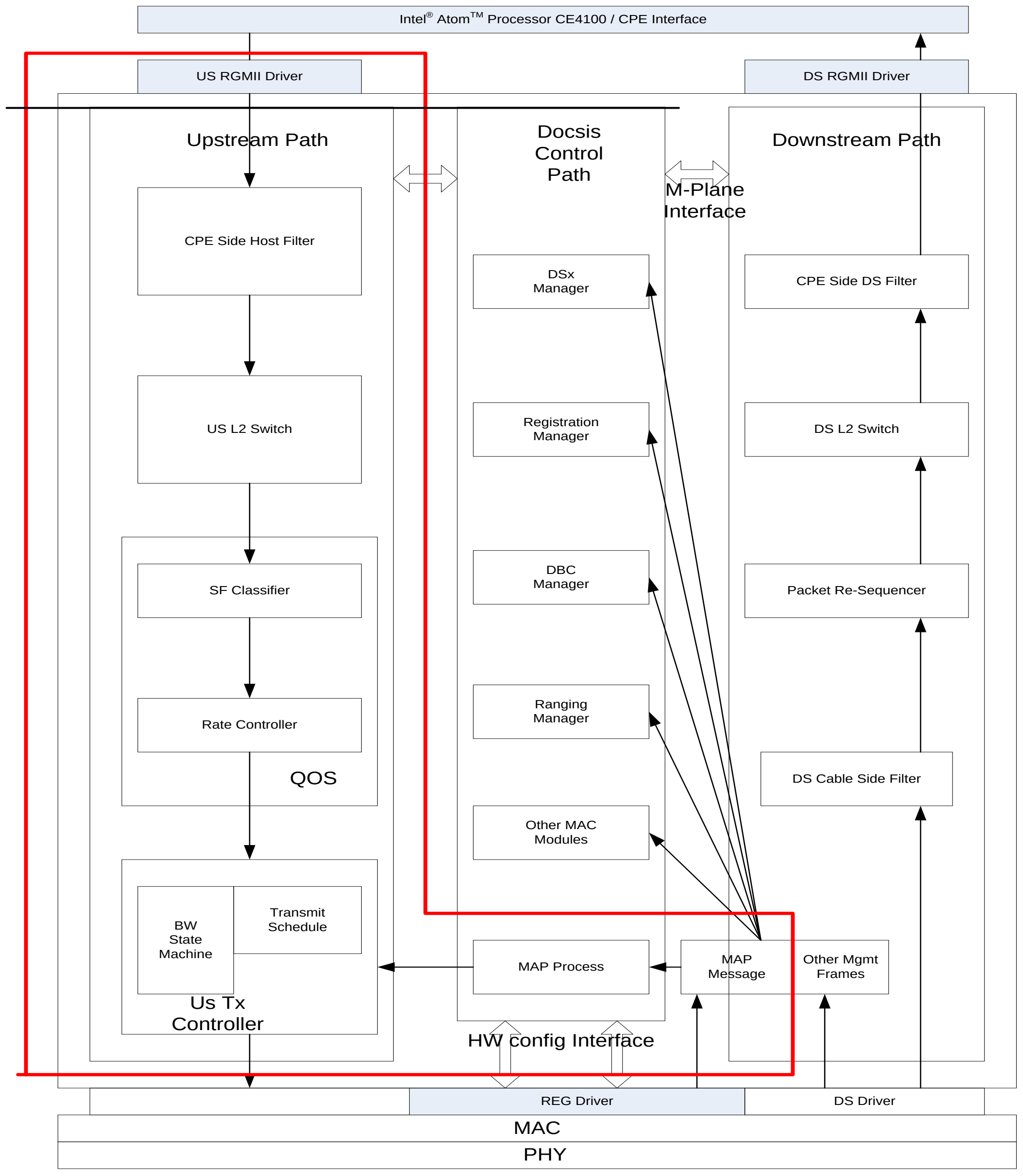
Figure 4 Conceptual Design


Processor 1
TO CPE
DS Data Path
Control Path
Settings for US
US Data Path (including classifiers, PHS etc)
Map Processor and Tx Scheduler
Map, UP-DIS
Other MAC Messages, DS Data Packets
US Data Packets
PHY/Mini-Mac

In this design, all of the upstream processing was executed on one processor. which also performed the real time map processing. All of the down steam processing is done on a different processor, making use of two processors in this design. We also made use of a symmetric operating system, where different threads were programmed for a particular core affinity.

Figure 5 shows the practical design of this two processor solution. Area marked in red encapsulates the processing performed on processor 2. The other area shows Processor 1.

Figure 5 Practical Design



Advantages

This approach is optimized across cores, as both processors split the work as well as the communication across the cores.

The communication across cores is minimal and also does not fall in a time critical path.

Disadvantage

This approach needs the Mini-MAC to process classification of MAC control packets. Our hardware was originally designed in a manner where the hardware component (Mini-MAC) would process only destination address filtering. In this design, a hardware change was needed so MAC control packets could be identified in hardware and transferred to a separate queue before the software on a different processor started handling them.

If we need to code MAP/TxScheduler in assembly, then it will get a little trickier. For performance reasons, it might be necessary to code some critical parts of map processing and transmit the scheduler. Writing code in assembly is error prone and development usually is more cumbersome.

Split of Functionality across Processors, on the Basis of Non Time Critical US/DS Functionality

This design is very similar to the earlier design where the functionality has been split on the basis of upstream/downstream functionality. The only difference here is the map processing (the most time critical functionality) is on the processor where downstream processing is done instead of the upstream processor. We again made use of symmetric operating system design with processor affinity fixed for key threads.

Figure 6 Conceptual Design of Split of Functionality across Processors, on the Basis of Non Time Critical US/DS Functionality

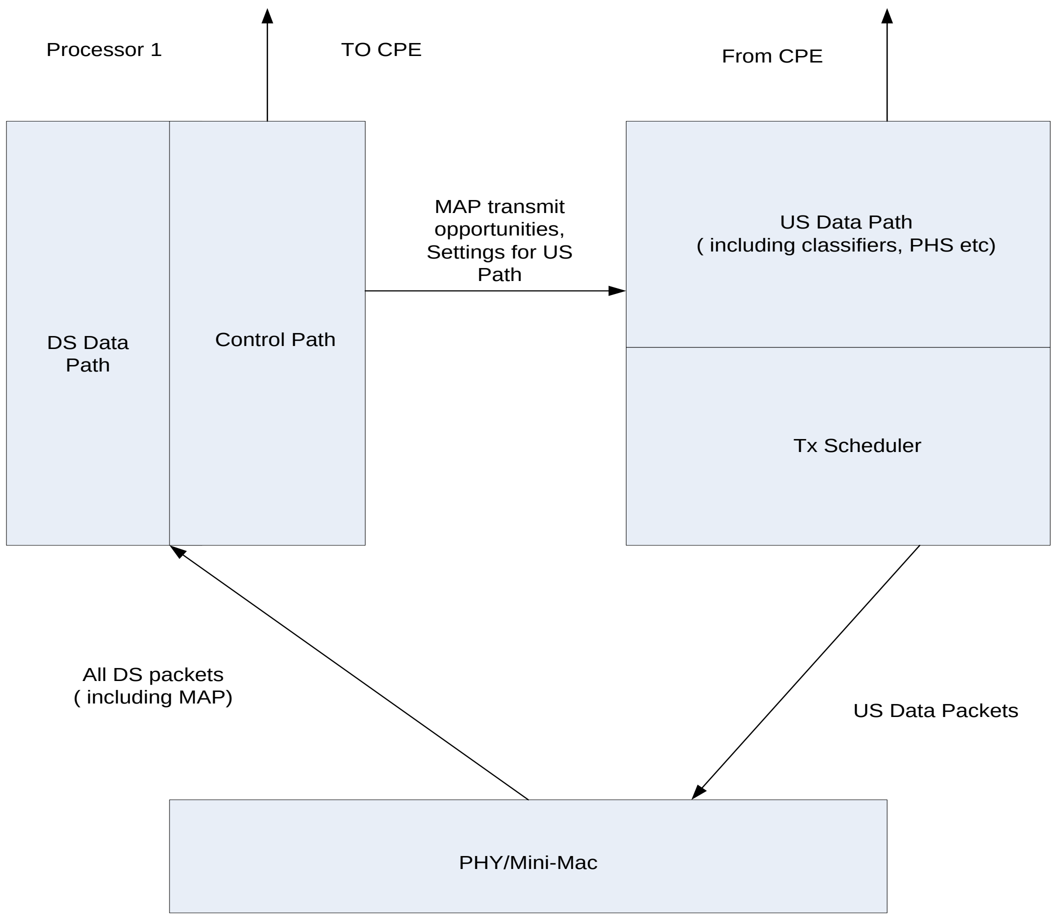
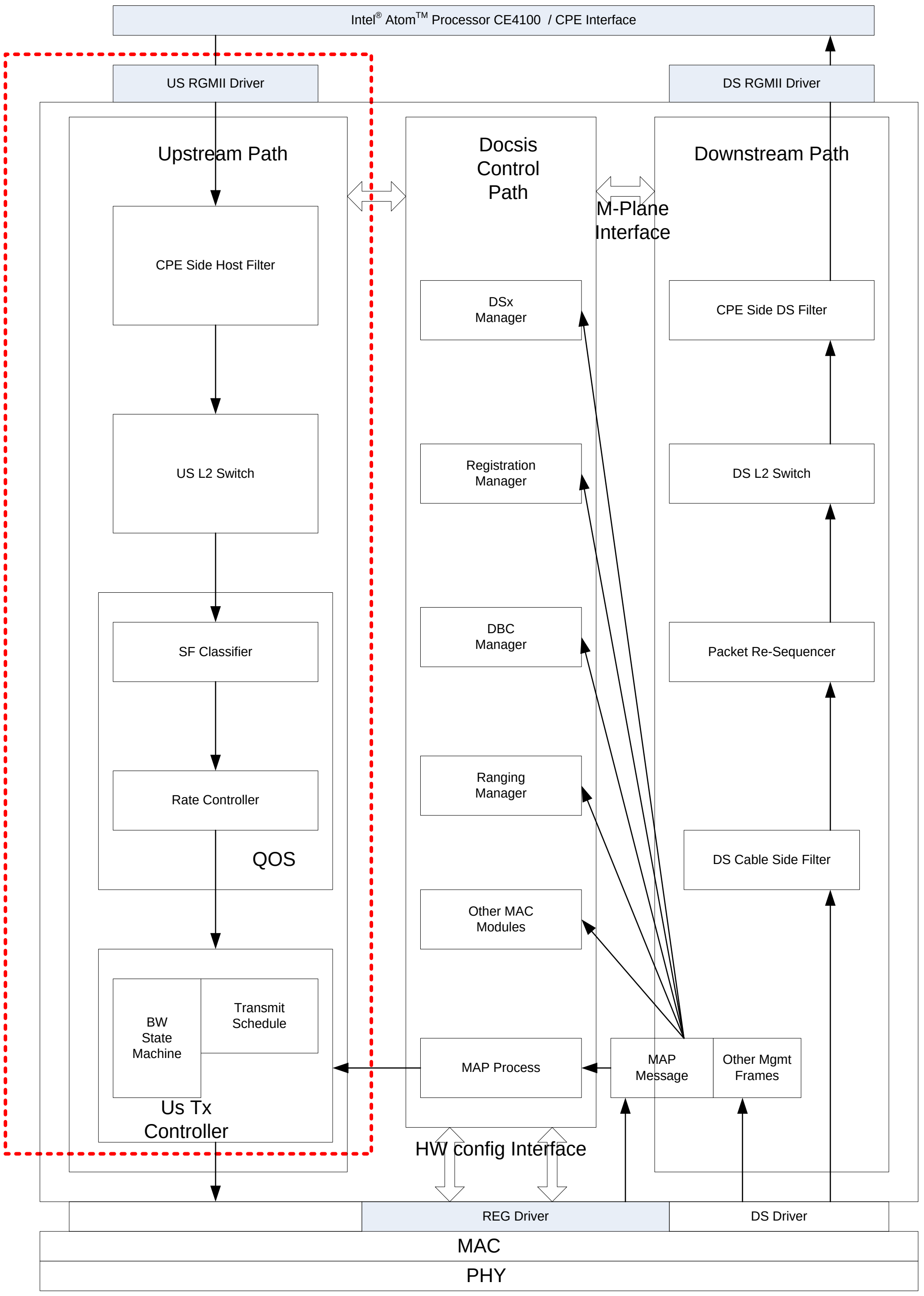

Processor 1
TO CPE
From CPE
DS Data Path
Control Path
MAP transmit opportunities, Settings for US Path
US Data Path (including classifiers, PHS etc)
Tx Scheduler
All DS packets (including MAP)
US Data Packets
PHY/Mini-Mac

Figure 7 Practical Design of Split of Functionality across Processors, on the Basis of Non Time Critical US/DS Functionality



Advantages

This approach does not need a modification in Mini-MAC, while still achieving the split across processors. This is because the MAP packets which come in the downstream direction are all processed on the same processor where the MAP processing is done. The filtering of the MAP control packets can be done in software and not in hardware as the earlier design would have warranted.

Disadvantages

If we need to code MAP/TxScheduler in assembly, then it will get a little trickier. If real time map processing performance on one channel without the full frequency of packets bordering around the 200 µs, then it becomes extremely tricky with the full frequency of packets. To increase performance in this case, we might have to program some critical aspects of map processing and transmit scheduler options in assembly. However programming in assembly increases the total development time, debugging and maintenance and upgrade effort.

Communication across processors is time critical, as MAP information is passed from one processor to other processor. This is a key disadvantage of using multiprocessor systems. Extra effort is needed to make sure that time critical information is passed in an efficient manner between processors. This is easier said than done because it involves software implications, such as balancing real time requirements from processor latencies, interrupt latencies and operating system latencies as well as queue handling procedures.

Split of Functionality across Processors, on the Basis of Control and Data Processing

This design is based on separating Data and Control paths on two different processors.

Figure 8 Conceptual Design Split of Functionality across Processors, on the Basis of Control and Data Processing

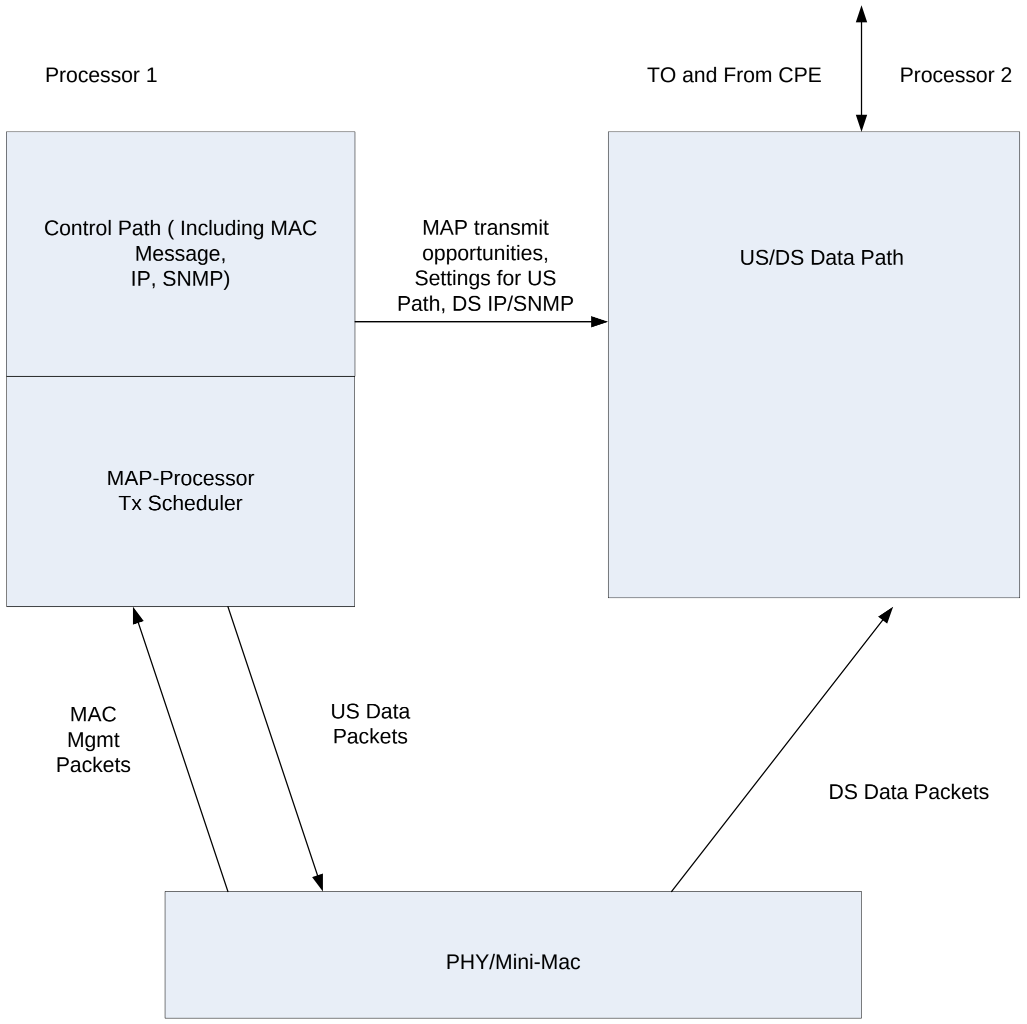
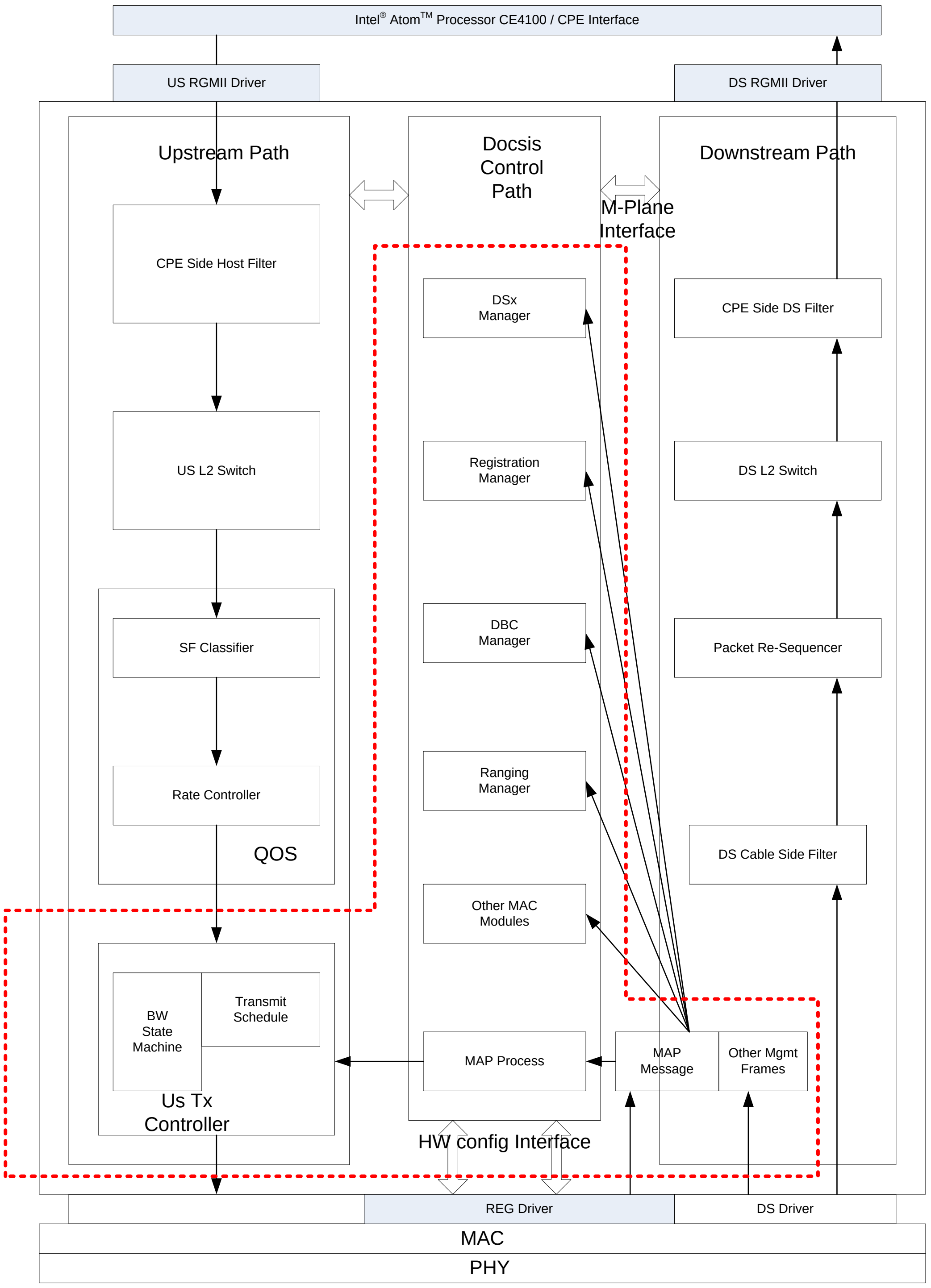

Processor 1
TO and From CPE
Processor 2
Control Path (Including MAC Message, IP, SNMP)
MAP transmit opportunities, Settings for US Path, DS IP/SNMP
US/DS Data Path
MAP-Processor Tx Scheduler
MAC Mgmt Packets
US Data Packets
DS Data Packets
PHY/Mini-Mac

Figure 9 Practical Design Split of Functionality across Processors, on the Basis of Control and Data Processing



We made use of a symmetric operating system with fixed processor affinity for key threads.

Advantages

This approach takes advantage of the fact that the control messages (and their processing) do not happen very frequently. So the remaining time of the processor can be used for MAP-TX scheduler.

Communication across the processor is not in the time critical path.

This approach takes advantage of a separate DMA context for MAC-MGMT and packet data. In our architecture, we had separate DMA contexts for management and regular data packets.

Disadvantages

The Mini-MAC needs to forward the messages received from the MAC-MGMT DMA to Processor 1, and messages received from DATA-DMA to Processor 2. This translated to some extra functionality on the hardware based Mini-MAC, but this is just incremental functionality on the Mini-MAC.

MAP-Processor/TX-Scheduler on a Separate Processor, with OS

This design is based on having all the real time processing on one processor and the rest of the processing on a different processor. This design also works on the assumption that there is an asymmetric operating system where the load balancing between the two processors is not chosen by the operating system.

Figure 10 Conceptual Design MAP-Processor/TX-Scheduler on a Separate Processor, with OS

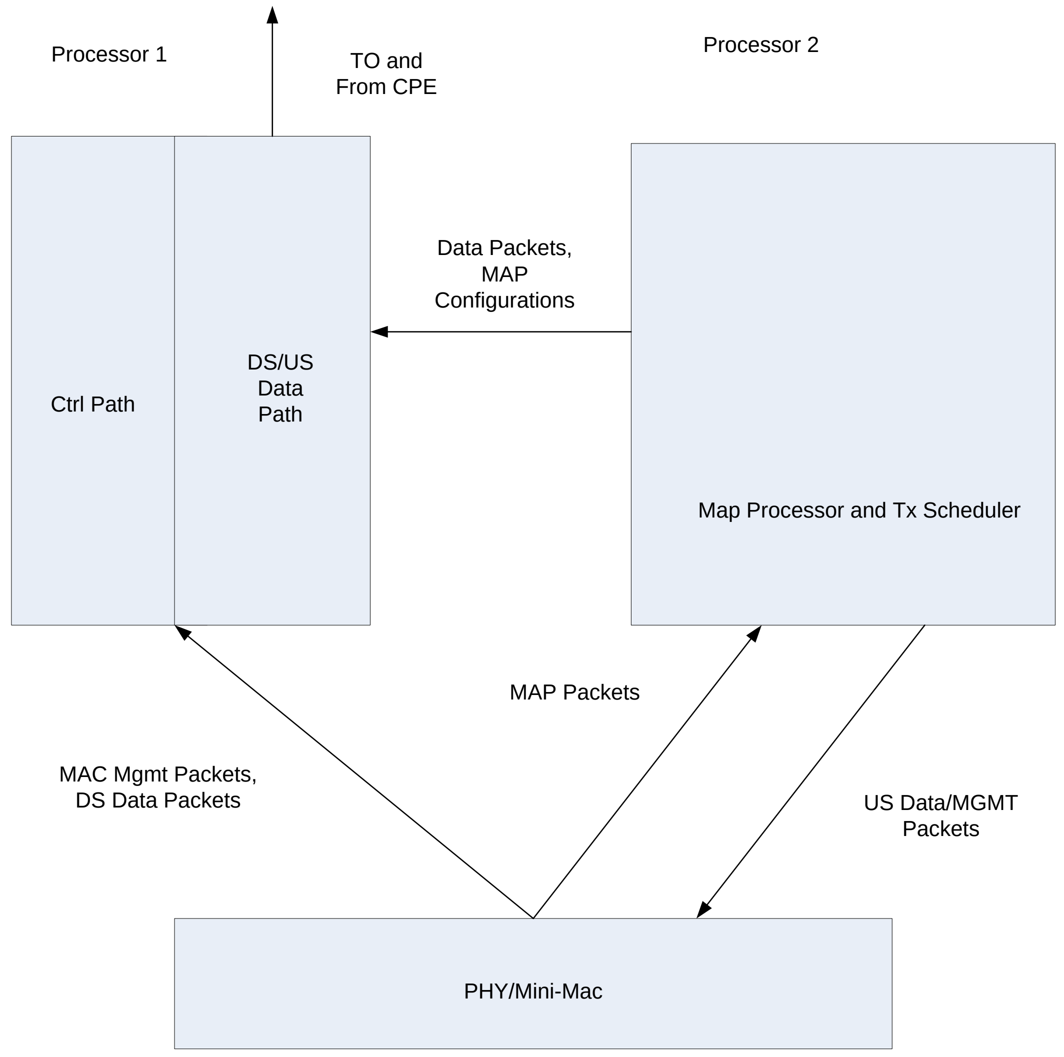
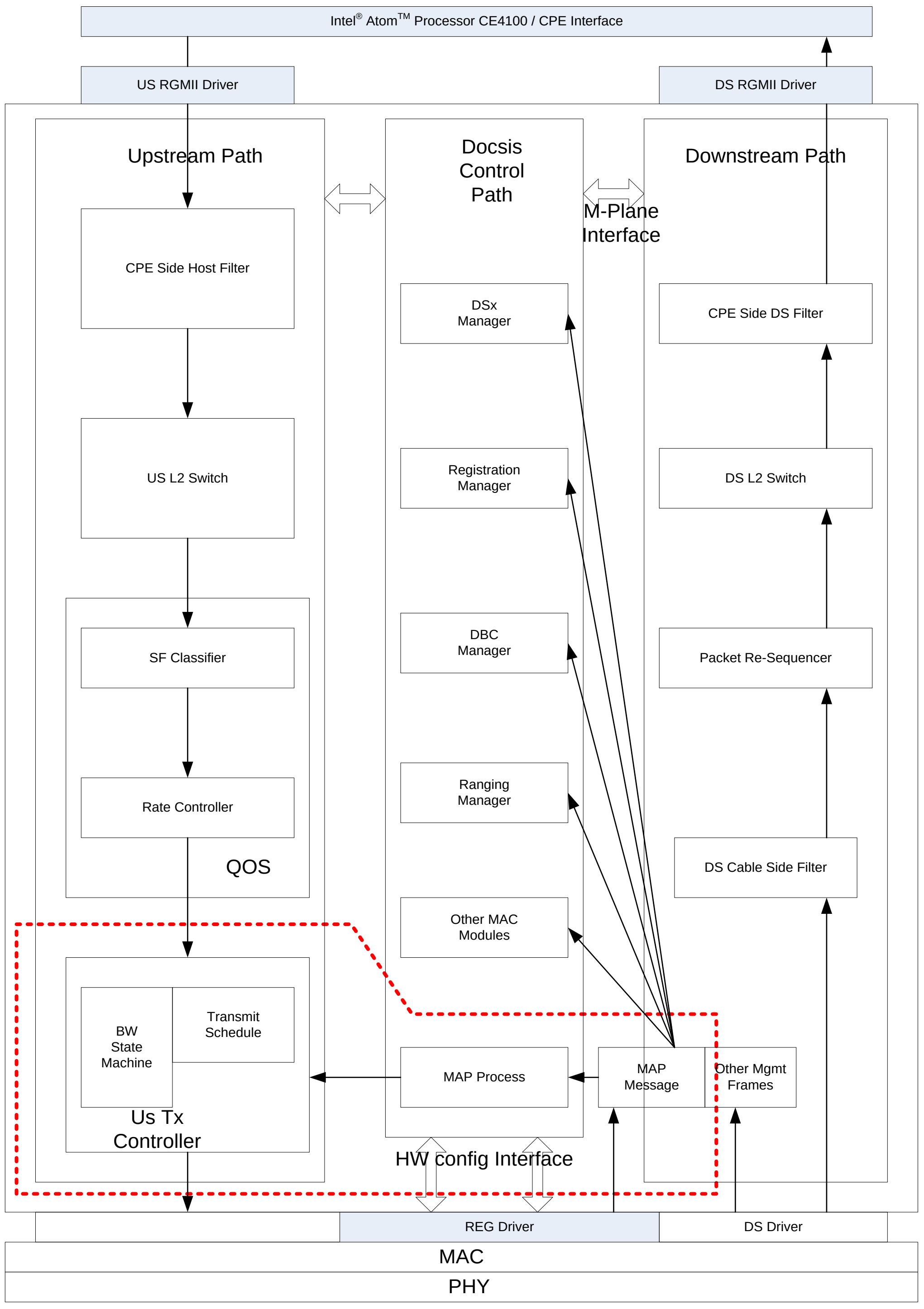


Figure 11 Practical Design MAP-Processor/TX-Scheduler on a Separate Processor, with OS



Advantages

This option is viable if the MAP-TX scheduler (real time processing) did not meet 200 µs in any of the previous approaches. If the real time processing is not able to meet the 200 µs target, then it might be worthwhile to have only the real time processing on one processor and the rest of the processing on a different processor.

Disadvantages

The Mini-MAC (hardware) needs to forward the MAP messages to Processor 2. This means that there may be some changes in the hardware functionality.

This also opens the door to move from a two processor solution to a three processor solution. If we cannot fit control and data processing in one processor, we may need a third processor.

This also needs an asymmetric operating system executing all real time functionality on one processor and the non real time functionality on another processor. The operating system does not perform any real time load balancing as this might affect the overall design. With asymmetric operating systems, it is a responsibility of the software developers to coordinate communication between the two cores. This increases software development time, as well as debug and maintenance efforts.

MAP-Processor/TX-Scheduler on a separate processor, without OS

Figure 12 Conceptual Design MAP-Processor/TX-Scheduler on a separate processor, without OS

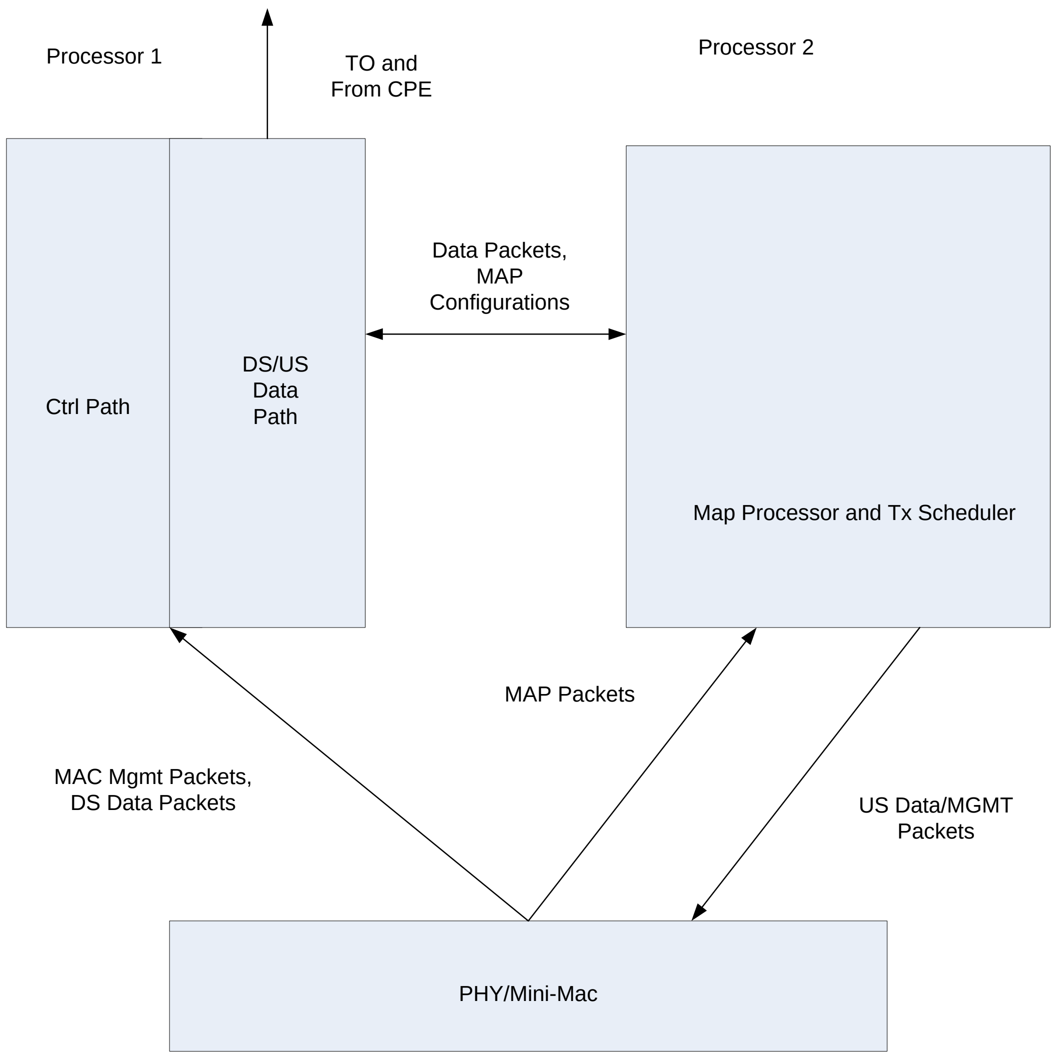
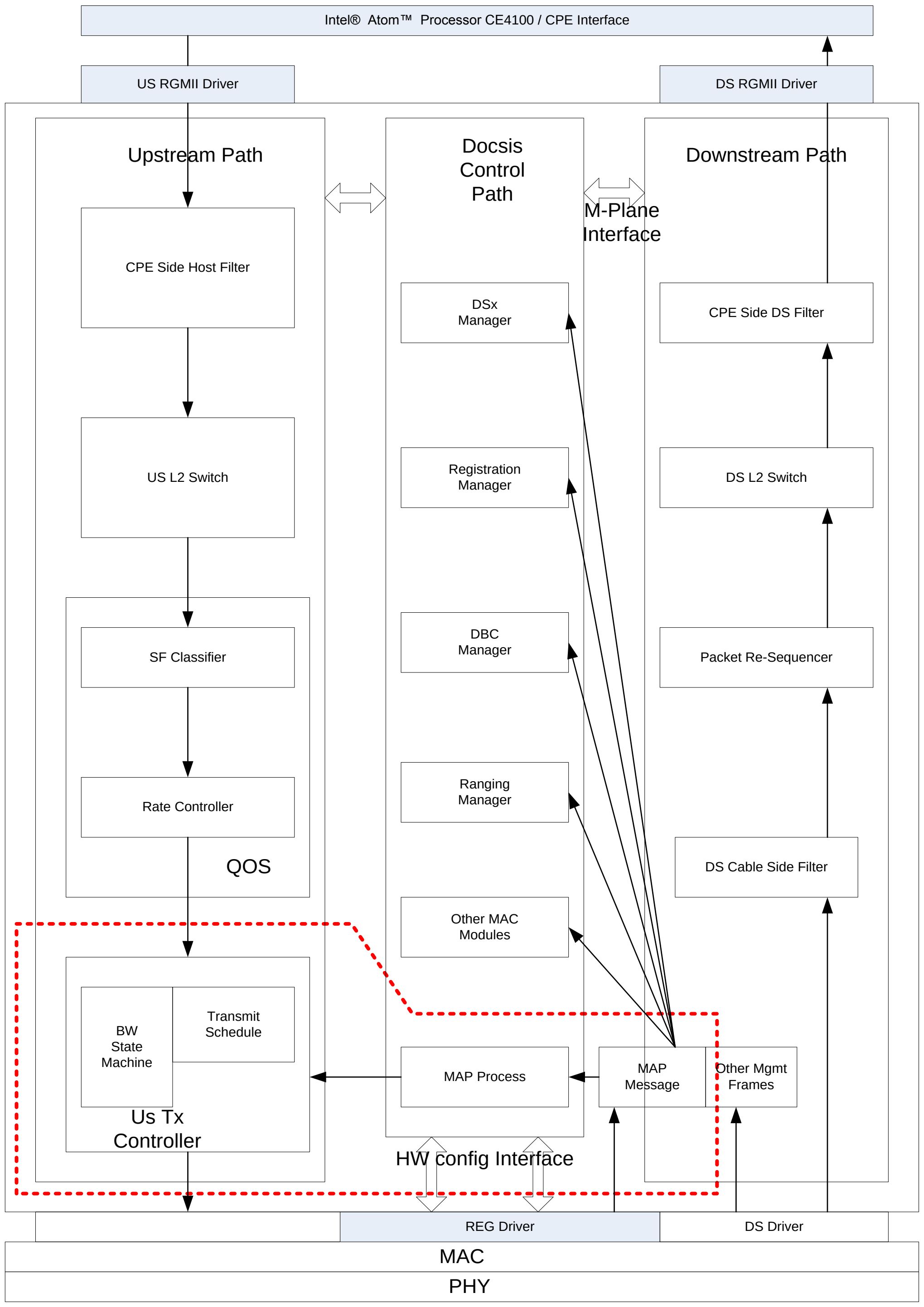


Figure 13 Practical Design MAP-Processor/TX-Scheduler on a separate processor, without OS



This design is very similar to the earlier designs except that all the real time processing is executed on a dedicated core which does not have an operating system. This means that all the real time processing will be written as firmware code.

Advantages

This option is viable if the MAP-TX scheduler (real time processing) did not meet 200 µs in any of the previous approaches. In this approach, there won't be any operating system delays and there won't be any inherent code related latencies because the real time processing is in firmware code.

Disadvantages

Architecting complex and huge real time processing task in firmware is very difficult making software development a challenge.

Mini-MAC (hardware) needs to forward the MAP messages to Processor 2 and will need some modification.

If we cannot fit control and data processing in one processor, we may need a third processor. This may also require more firmware code.

Performance Measurement

One of the most important challenges we faced was the inability to procure real time performance data on various aspects of the Linux operating system on the ARM platform. For this, we conducted our own experiments and came up with data similar to what is listed in the table below. The exact code and the procedures we followed will be presented in a different white paper to be published in the future.

Table 1 Performance Tests

Details:	**Status**	
Measure using test programs - task switching time using	All measurements are using Modified Kernel with preemption	
	Kernel Mode	*User Mode*
Semaphore method	Switch time – 109 micro seconds	Switch time – 132 micro seconds
Mutex method	Switch time – 106 micro seconds	<Not done>
Wait queues method	Switch time – 109 micro seconds	<Not done>
Tasklet method	Creation time – 306 micro seconds	<Not done>
Interrupt Latency	15 micro seconds	N.A.
Interrupt Latency Measurement	ARM Timer is used to measure the interrupt latency.	
Kernel optimization – Identify patches available		
- Remove all modules that are not required	Almost all unwanted modules removed. Preemptive support added. Will do further optimization after test programs are done.	
- Identify and do all other finer optimizations possible		

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, Go to: http://www.intel.com/performance/resources/benchmark_limitations.htm

Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

Results have been simulated and are provided for informational purposes only. Results were derived using simulations run on an architecture simulator or model. Any difference in system hardware or software design or configuration may affect actual performance.

Interrupt Latency: Interrupt Latency is the time from when hardware event is triggered until a thread waiting for that event is unblocked and continues processing based on the interrupt.

§

Results

The results for our performance work regarding the cable modem front end for the Intel® Atom™ Processor CE4100 indicated that the Atom™ would satisfy our performance needs. Our chief concern of meeting the real time performance requirement of 200 µs was met. In our case it would be relatively straightforward to port our code and architecture to the main Intel Atom processor within the Intel® Atom™ Processor CE4100 which was handling the media processing as well. The same may or may not be true for the particular case you are dealing with. This paper is an attempt to provide a framework of performance measurement data to help guide you in making an intelligent decision on what works for you.

Conclusion

The conclusions are discussed within the following sub sections:

Design Flexibility

During the course of our experiments, one thing became increasingly clear was that it is critical to keep the options open and flexible when designing a comprehensive solution. Designing with flexibility ultimately avoids rework and saves crucial resources in terms of time and money. Our iterative approach of performing experiments with various triggers on ARM based FPGA helped us narrow down on the final design criteria. By adopting four-core architecture for experiments, we kept our options open when it came to testing the real time channel requirements for the communications use case.

Real-time Requirements and Intel Atom Microarchitecture

Though the ultimate aim for doing this exercise was to port our solution to Intel Atom microarchitecture, the experiments did reveal fundamental real-time requirements for a communications solution. Our analysis showed that the cable-modem front end could very well be implemented with an Atom processor. The migration experiments clearly outline that an Intel Atom processor is capable of handling real-time requirements posed by communication workloads.

Early Prototyping

One of the key learning's of this exercise is that early prototyping is necessary for any complex solution that has real-time requirements. By choosing to do experimental analysis using triggers for various cores, it gave us a good idea of the final solution once it would be ported to a single core. Tying our results and analysis with a flow chart based decision making aided us to come up with decisions early during the project.

IA Ecosystem (Tools and Software)

Choosing to work with an ARM FPGA based prototype gave us good insight into the availability of tools and software for our work. Our experiences and analyses clearly show that when it comes to tools and software availability, Intel Architecture is well supported. From the availability of the core operating system to the supporting tools around it, Intel Architecture presents various choices to OEMs and design vendors with its legacy of scalable solutions.

Application to Other ARM Based Communication Architectures

We started out with a thorough exercise analyzing real time requirements of our communications solution on an ARM based solution. Our results showed that most stringent real-time requirements posed by a communications system can actually be addressed by an Intel Atom solution. Intel Atom microarchitecture thus could apply very well to other legacy ARM-based solutions in the industry. With a strong software and tools ecosystem around Intel Architecture, Intel Atom microarchitecture is positioned to meet and exceed the needs of the embedded design industry.

Authors

Addicam V. Sanjay is a Software Architect with Embedded and Communications Group at Intel Corporation.

Prashant Paliwal is a Software Architect with Embedded and Communications Group at Intel Corporation.