A HW/SW Codesign Methodology based on UML

How to apply a model based UML design for an embedded system

By Senior Consultant Kim Bjerge (kim.bjerge@teknologisk.dk)

Preface

The Unified Modeling Language (UML) is a general purpose visual modeling language for system development. Although UML is most often associated with modeling of Object-Oriented software systems, it has a much wider scope such as being used for system design, including both hardware and software elements. This article will try to give a view of the areas for using UML in a HW/SW co-design methodology where focus is on the UML 2.1 standard and tools for real-time and embedded systems.

The steps in the UML based HW/SW co-design methodology are illustrated for a measurement system which is implemented on a Xilinx FPGA platform. It includes hardware IP blocks, a PowerPC running Linux covering driver and application software.

The audience of this article are software developers and system architects who wish to have an overview of how to use UML for modeling real-time and embedded systems covering both hardware and software elements.

The principle of Model-Driven Development (MDD) is a paradigm which promotes the use of models at different levels of abstraction and transformation between them in order to drive a concrete application implementation. MDD promotes the construction of models at a high level which can subsequently be transformed to the final implementation platform.

Our approach is Model-Driven Development based on UML which provides the developer with a high level of abstraction with focus being moved from coding, language and platform details to specification and design. It allows an iterative development and verification process on a higher abstraction level than pure coding.

The benefits will be an understandable, improved and robust design where focus is on the whole system design covering both hardware and software elements. By using a common visual UML, it is our experience that language improves communication and understanding of the system to be implemented. It becomes clear when discussing the diagrams where things are not understandable or where something is missing. It reduces the number of pitfalls and misunderstandings between project team members.

UML 2.1 diagrams

UML helps you specify, visualize and document models of embedded software systems - including their structure and design - in a way which fulfils the system requirements. It is possible to model just about any type of application, running on any type and combination of hardware, operating system, programming language or network.

It is a complex task to gather and analyze the requirements of an application and to incorporate these into a programme design. The industry currently supports many methodologies which define a way to do it. One characteristic UML feature is to strive at being methodology-independent. Regardless of the methodology used to perform analysis and design, UML can be used to express different views of the embedded system.

One of the benefits of using UML is that it is a standard maintained by the Object Management Group (OMG) and is widely supported by a range of UML tools. By using XMI (XML Metadata Interchange) it is possible to transfer an UML model from one tool into a repository or into another tool for refinement of the next step in the selected development process - like timing and performance analysis. These are some of the benefits of standardization.

UML 2.0 defines a number of diagram types that can be divided into three categories. Structure diagrams includes: class, structure, object, package, component and deployment diagram. They are all basically different, not with regard to the contents of the diagram but with regard to their purpose. Behaviour diagrams focus on specification of behaviour of individual elements covering state machine and activity diagrams. Use-case diagrams are used to emphasize functionality, identification of system interface (actors) and use-case scenarios. They focus on the specification of system usages. Interaction diagrams focus on how elements (objects) collaborate over time to achieve the functional goals.

Structure Diagrams

- Class and Object diagram
- Composite Structure diagram
- Package diagram
- Component diagram
- Deployment diagram

Behavior Diagrams

- Use-case diagram
- Activity diagram
- State Machine diagram

Interaction Diagrams

- Sequence diagram
- Timing diagram
- Communication diagram

UML usage levels

The use of UML diagrams for embedded system development can be divided into three levels depending on the ambition of usage and integration of automated code generation.

Informal level
UML diagrams are used in the analysis and design phases of the project to sketch a few diagrams from where the coding proceeds. In this approach the diagrams are used to get a common understanding of the system to be constructed. UML is only used in the analysis and design phase of the project and is not likely to be updated during the implementation phase.

Structural level
The UML diagrams are used together with a tool to generate code, and the developer fills in the rest using the model as a guide. This approach saves the programmer for manually creating all the classes and relations identified in the design model. A tool like Enterprise Architect[1] is able to support this development process. It is also possible to import existing code into the UML model. This could be used for reengineering and documentation of existing systems.

Executable level
This is the Model Driven Development approach. The model views in this approach and the generated source code have a close relation. Analysis, design, implementation and test are all integrated into the UML tool. Very early in the design, it is possible to actually execute and verify the model based on structural and behavioural UML diagrams. This kind of development changes the focus from the art of programming to the application development and abstracts the underlying hardware and operating system. A tool like Rhapsody from Telelogic supports this approach.

[1] Enterprise Architect from Sparx Systems, supports UML 2.1 diagrams and code generation

Getting started with UML

How do I best get started using UML in my project? Firstly, you have to decide on the UML usage level you wish to achieve (Informal, Structural or Executable). Secondly, you have to evaluate the level of skills in your team. Do you have sufficient knowledge and experience in object-oriented analyses, design (OOA/OOD) and UML? Does anybody in the team have sufficient experience with UML or model based development?

Get training

Based on team skills, basic training in UML is recommended. It can be a combination of reading books, articles and participating in a basic 3-5 days UML course. Some of the books and training material on which this article is based are listed in the last chapter.

Normally an UML course is based on a methodology since the UML diagrams have no information about how to use them in a development process. A general UML and method course for your team would also give a quick start and a common understanding of how to use a methodology together with the UML diagrams.

If you have very little experience and are aiming for a high level of modeling, I can recommend hiring external consultancy or employing an expert in the area of UML and modeling.

Select a methodology

There are a number of methodologies for using UML diagrams in a development process. You can choose to define your own or use some of the methodologies already described and used in the industry. Below is listed some of the most common methodologies for embedded systems. See references for books describing these methodologies.

- The **Unified Process** (UP)
- The **R**apid **O**bject-oriented **P**rocess for **E**mbedded **S**ystems (**ROPES**)
- The **Harmony** Process (Latest revision of ROPES)

It is recommended to adapt the chosen methodology to one's own needs. In the next chapter a model based HW/SW co-design methodology is described. This methodology is an adapted version of the "**Unified Process**"(UP) and **ROPES**.

Select a UML development tool

Depending on the UML usage level you wish to achieve, the selection of UML tool is important. Some tools forces you to use a certain methodology, others are purely drawing tools like Visio. In the last chapter you can find links to some of the UML tools available.

Start with a simple project

Start with a simple project. The best would be if you are planning on a new embedded platform using an object-oriented programming language like C++ or Java. It is recommended to be at least two persons involved in the project since modeling with UML is very much concerned with visual communication and information sharing in the development process.

A Model based HW/SW Codesign methodology

A process is the specification of a sequence set of activities performed by a collaborating set of workers resulting in a coherent set of project artifacts, one of which is the desired system.

A methodology consists of a language to specify elements and relations of interest and a process that tells the developer what parts of the language to use, how to use them, and when to use them.

"Bruce Powel Douglass"

The purpose of this chapter is to give an example of a methodology based on UML. The objective is not to turn you into an expert in using such a methodology but to give an overview of the steps that could be involved when proceeding from a specification to an UML implementation of an embedded system.

This is a HW/SW codesign methodology based on UML where both hardware and software elements of an embedded system are included. The methodology case example is targeting a FPGA platform where one part of the system will be implemented in software and another in hardware. The purpose is to describe the overall system with focus on the partitioning and interfacing between hardware and software elements. UML is used on the "informal level" in the case example which illustrates the development steps covering analysis and design.

The method consists of three main phases and is based on an iterative use-case driven process. The first step is a system specification that may be a pure text document or be based on interviews with the user and customer of the system to be developed. Based on this input, the use-case specification is developed. The system analysis, design and implementation iteration is repeated for a selected number of use cases. The end result is a delivery plan which includes a number of product releases during the development process. This approach ensures that you will be able to get early feedback from the user or customer before completing the entire development process.

The methodology includes the following phases for every iteration.

- **System Analysis - Domain and Application**
  - Focus on what to be made
- **System Design - Architecture and Interface**
  - Focus on how to make it
- **System Implementation and Validation**
  - Focus on making and verifying it

Only the analysis and design phases are described in this article.

The figure below illustrates the methodology including the UML diagrams created in each phase.

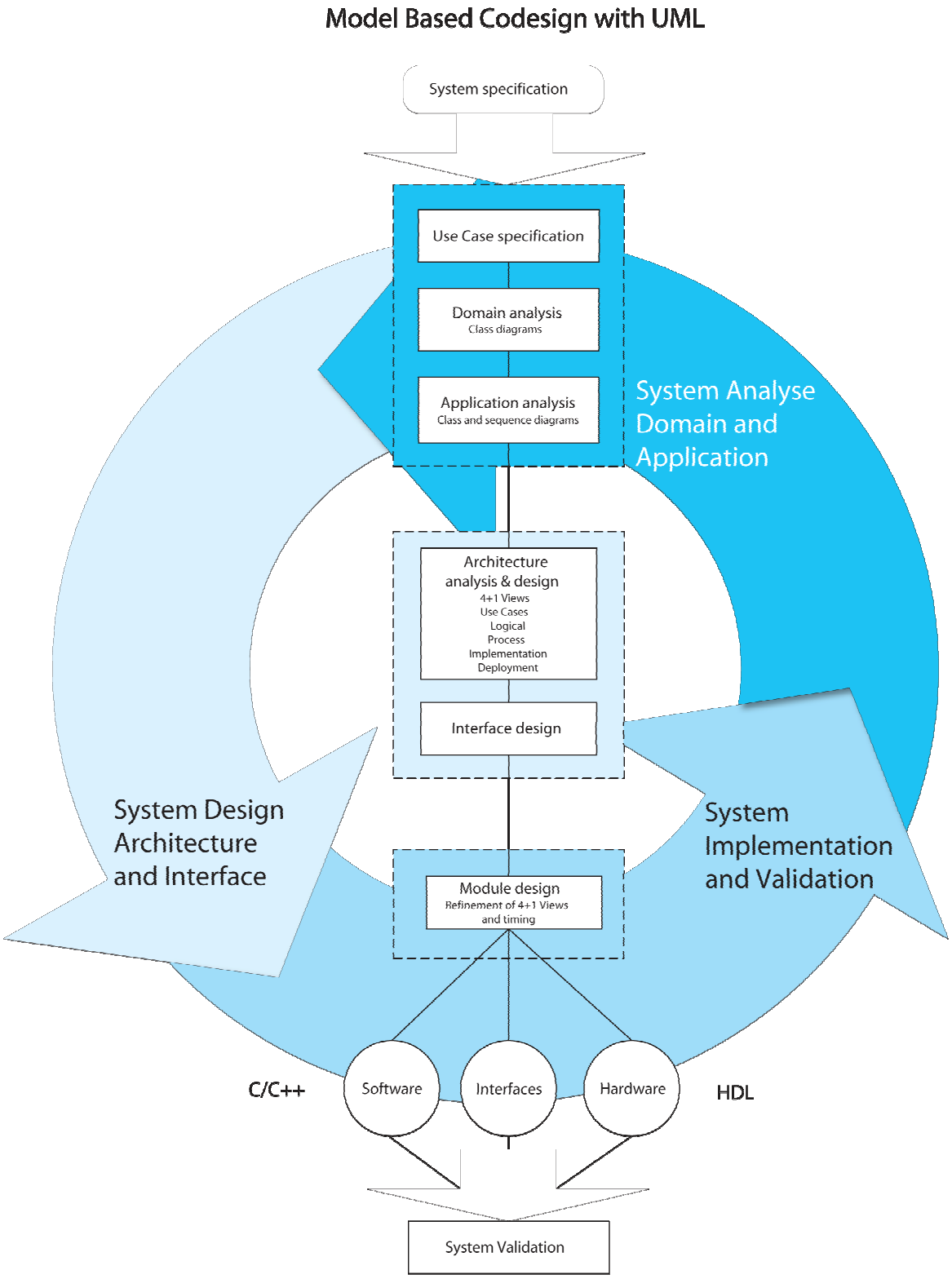

Figure 1 Model based Codesign with UML

System Analysis - Domain and Application

The goal of this phase is to make it clear "**what to be made"**. It covers analysis of the system domain including interaction with the environment and required functionality.

This phase focuses on a use-case analysis of the application domain. The first step is to identify the external actors interacting with the system. This could be other communication devices, sensors, actuators or system users. The second step would be to identify use cases for each external actor and describe the use-case scenarios. Figure 2 Measurement System use-case diagram is a use-case analysis of a system measuring vibrations from a sensor and calculating a power spectrum (PSD). PSD are averages of sample blocks over a period of time. The PSD result is transmitted to a central server. Each use-case scenario is described in the use-case analysis document.

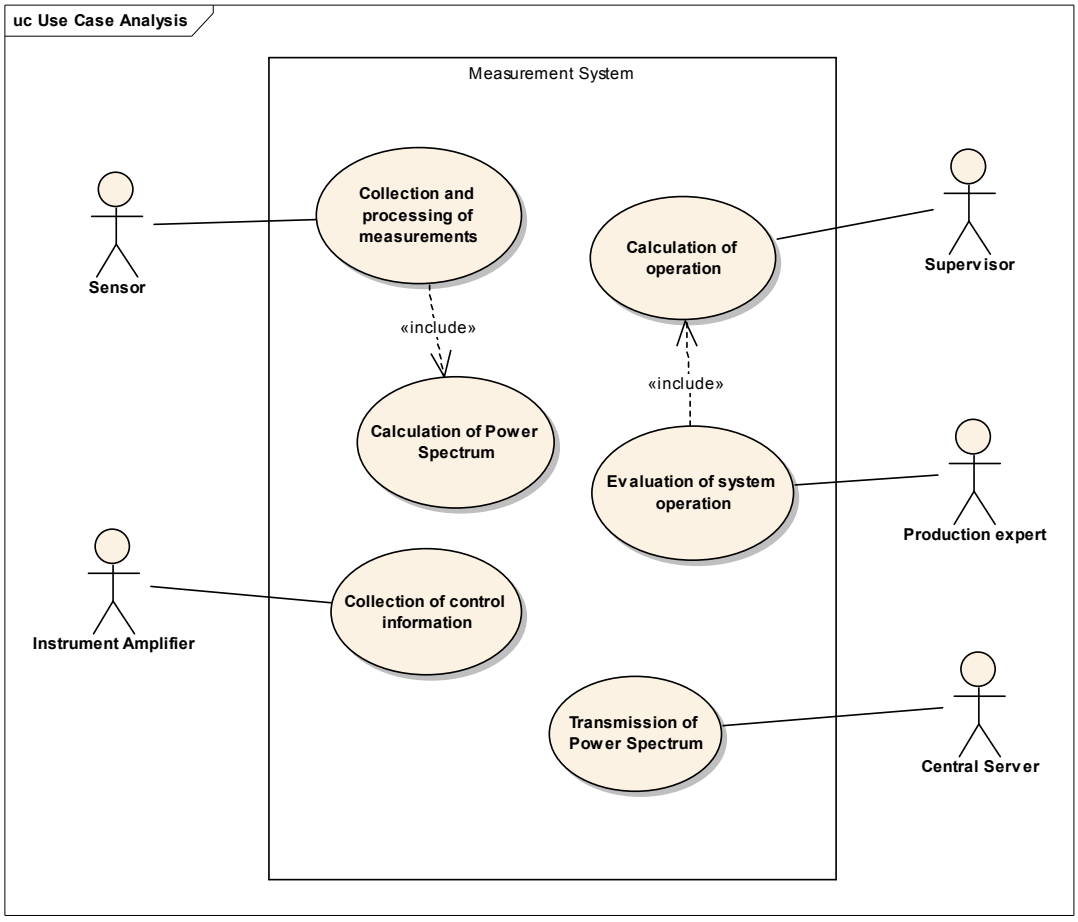

Figure 2 Measurement System use-case diagram

The domain analysis is based on the use-case analysis as a starting point. The domain model identifies the key concepts from the application domain necessary to realize the use-case requirements and to refine them. Classes and objects in the application domain are identified. These classes could be things like the sensor, sample block, power spectrum and instrument amplifier identified from the measurement use-case description. The next step would be to clarify the relations between the classes by examining how they relate to each other. In our case we have decided to use the signal type to describe associations between the identified classes. The result is an UML domain class diagram. See below.

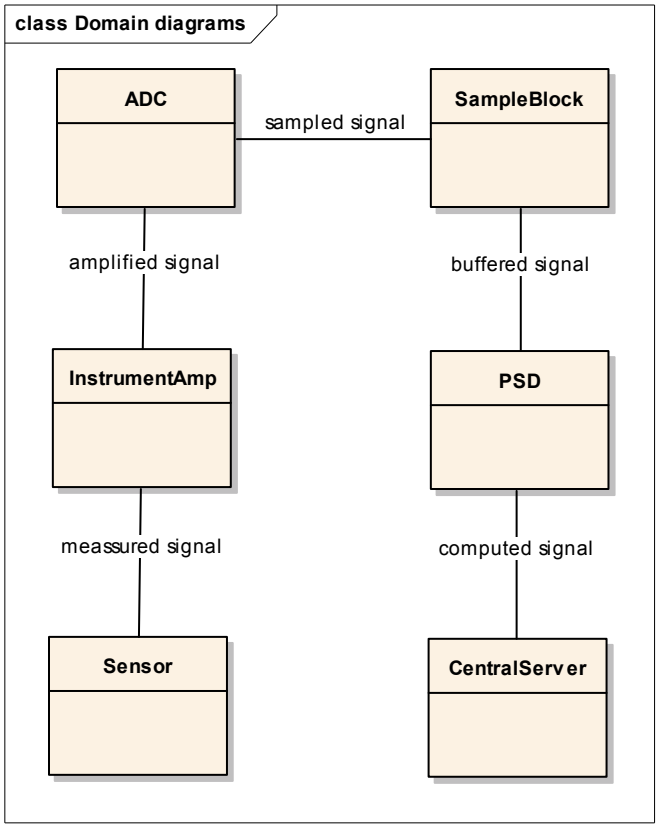

Figure 3 Measurement System domain class diagram

The final analysis step is to select a number of use-case scenarios for the first iteration of use cases to be implemented. For every use-case scenario a control class is added to the class diagram and for every actor an interface class is added. The result will be an updated class diagram which contains the domain, control and interface classes. See Figure 4 Measurement System application class diagram for an example. We have added direction of navigability to some of the associations between the application classes to indicate the direction of the message flow in processing the sampled data.

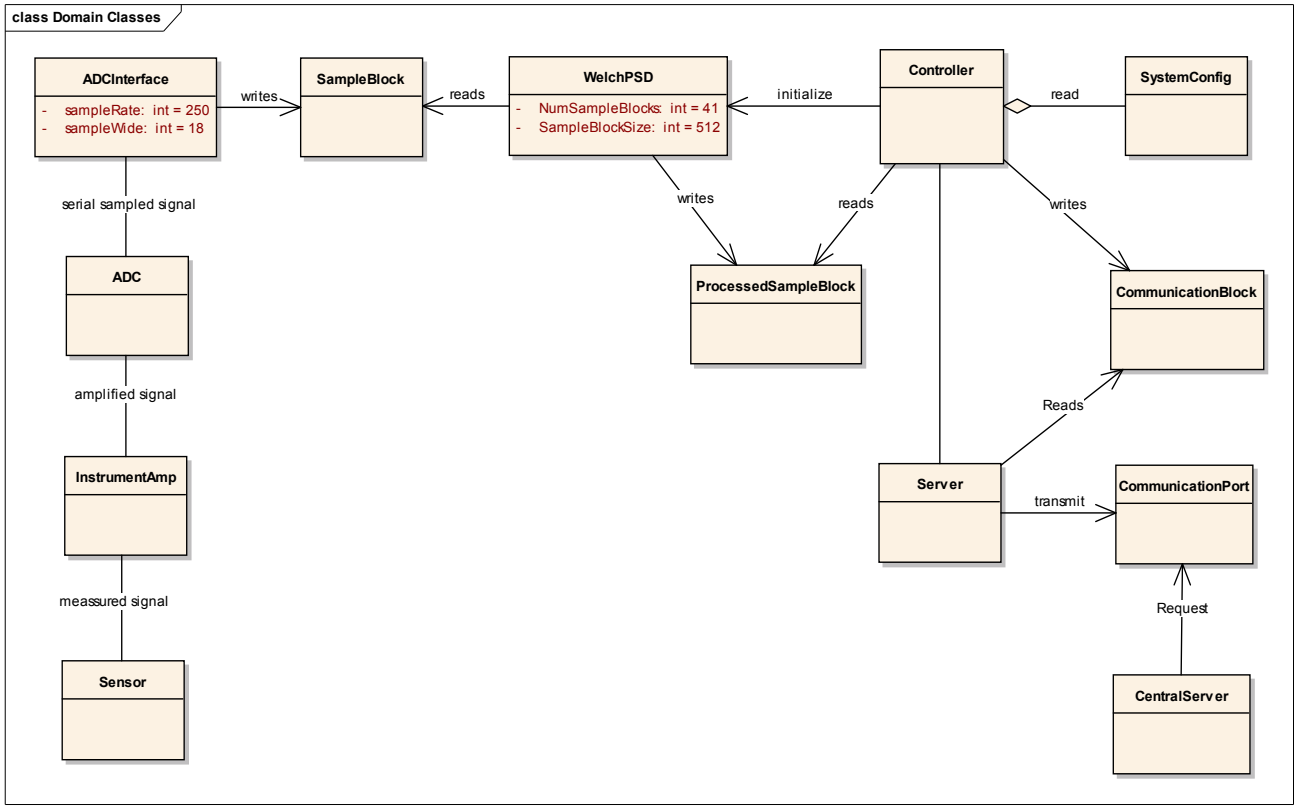

Figure 4 Measurement System application class diagram

The last step in the application analysis is to select a number of use-case scenarios and describe them in terms of sequence diagrams. The UML sequence diagram provides the behavioural view of the relations between the application objects. See Figure 5 Measurement System application sequence diagram for an example. This sequence diagram covers the use-case scenario “Collection of measurements” and “Calculation of Power Spectrum”.

Capturing of Quality of Service requirements like schedulability, performance and time is crucial for designing real-time systems. The most differentiating characteristics of real-time systems are their concern and treatment of time. In the analysis of real-time systems, it is important to define the external timeliness requirement to understand the problem. Response performance may be specified into a worst-case deadline and an average response time.

In the following case we have calculated the average update rate of the power spectrum (PSD) to 12 times per second. The calculation is based on the sample processing rate of 250 kSPS (kilo samples per second) for the measurement of sampled input data. The response of the measurement system is determined by the operating system and the load of the system. The PSD itself has little influence

due to the number of interrupts from the PSD compared with the transmission rate of the server connection.

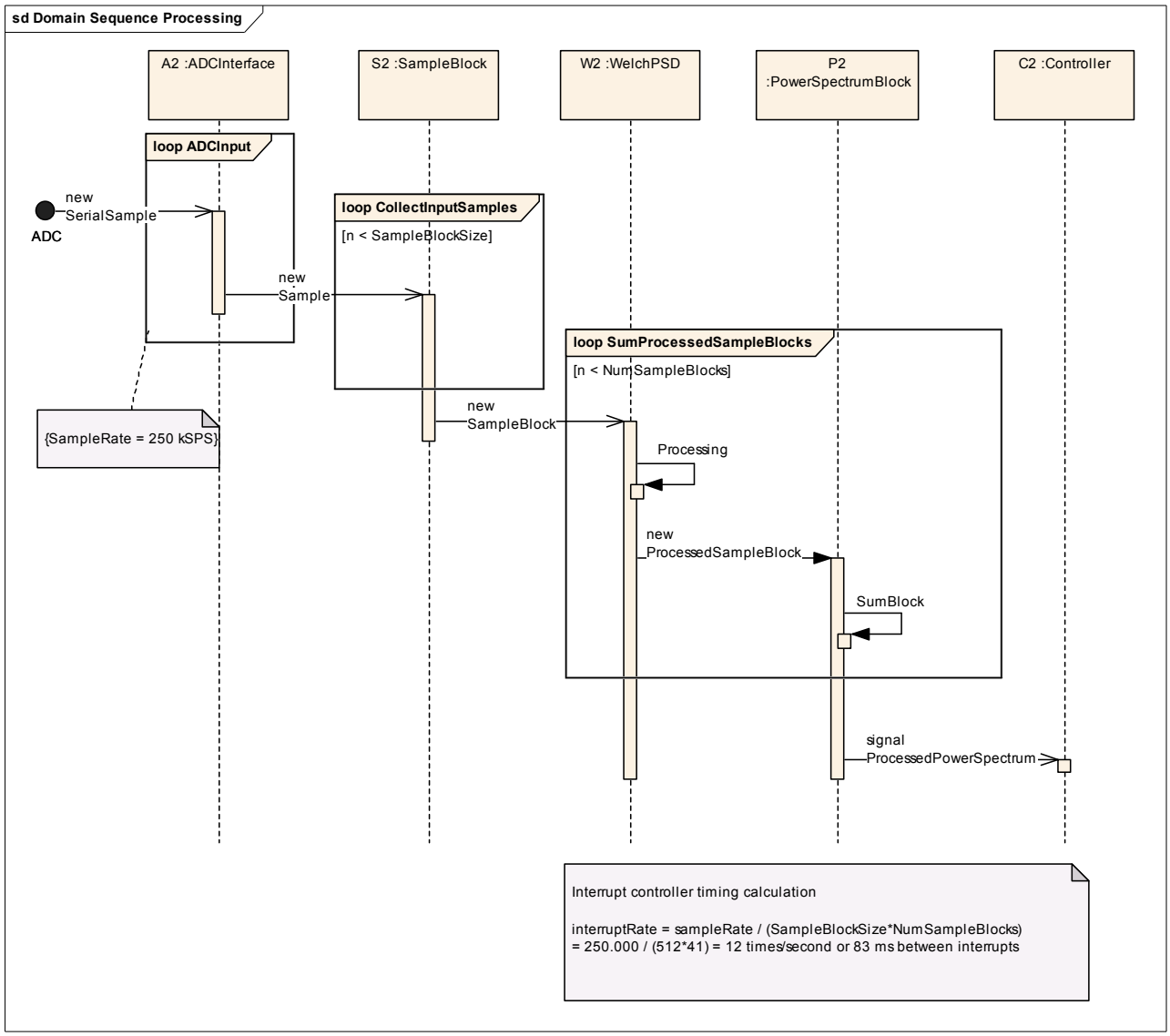

Figure 5 Measurement System application sequence diagram

System Analysis Activities

Below is a brief outline of the 3 analysis steps:

- Use-Case specification
  - Actor context description
  - Use cases
  - Scenarios

- Domain analysis
  - Classes and relations
  - Domain rules

- Application analysis
  - Control and interfaces
  - Quality of Service (QoS) requirements

System Analysis Artifacts

Below is listed the documents and UML diagrams produced in the analysis phase:

- Use-Case analysis document
- Domain and Application Class Diagrams (Static and relations)
- Sequence Diagrams (Dynamic and behavioural)

System Design - Architecture and Interface

The goal of this phase is "**how to make**" the architecture for the embedded system. For each iteration, the selected use cases are still the driving factor on where to focus design activities. In the design phase, we will create the HW/SW architecture that can realize the requested functionality based on the outcome of the analysis phase. The architecture is modelled in 4 different views: Process, Logical, Deployment and Implementation which are briefly described in the following case example.

In the process view we have added information to elements and classes which are executed concurrently. This view focuses on the management of resources and the concurrent aspects of system execution. By concurrent we mean that classes may execute in parallel rather than sequentially. A resource is an element that provides a limited service. For example, it may allow only one process at a time to access its internal data.

A decision has to be made based on experience and the chosen target platform for the partitioning of concurrent processing elements into hardware or software. Generally processing elements are suitable in hardware where there is a demand for high speed data processing and the computation is rather simple. Processing elements making complicated decisions or computes in sequence are more suitable in software implementation.

In our case study, the target is a FPGA platform including an embedded PowerPC (100 Mhz) running Linux. We have decided to implement the averaging of data blocks and computation of the power spectrum (PSD) in hardware to reduce the load of the embedded processor. In this case, the processor will be interrupted every 83 ms where it has to collect 512 samples for each sensor channel. The alternative would be to implement the PSD in software. However, this would require a much faster embedded processor calculating and collecting samples at a rate of 250 kSPS. Implementation in software would require a much faster CPU which would be more expensive than using the FPGA platform.

The stereo types <<hwProcess>> and <<swProcess>> are used to indicate processes implemented in hardware or software. The stereo types <<hwRessource>> and <<swRessource>> are used to indicate resources implemented in hardware or software. See Figure 6 Measurement System refined class diagram with concurrency.

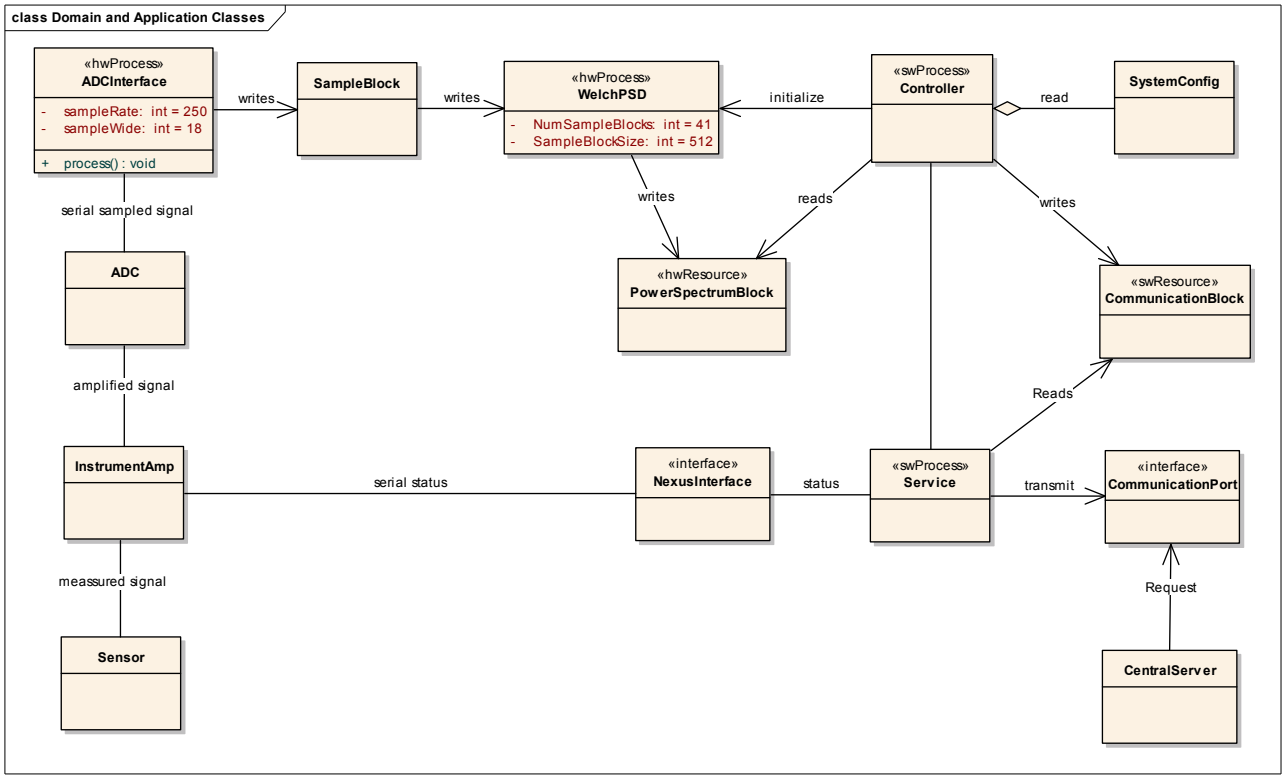

Figure 6 Measurement System refined class diagram with concurrency

The logical architectural view is concerned with models and how they are organized. This organization can be simple or very complex, depending on the needs and structure of the team using it. In this logical view all classes are divided into abstract packages with a typically layered architecture. Each layer in the logical view of the measurement system addresses the application, driver and real-time layer shown in Figure 7 Measurement System Package diagram.

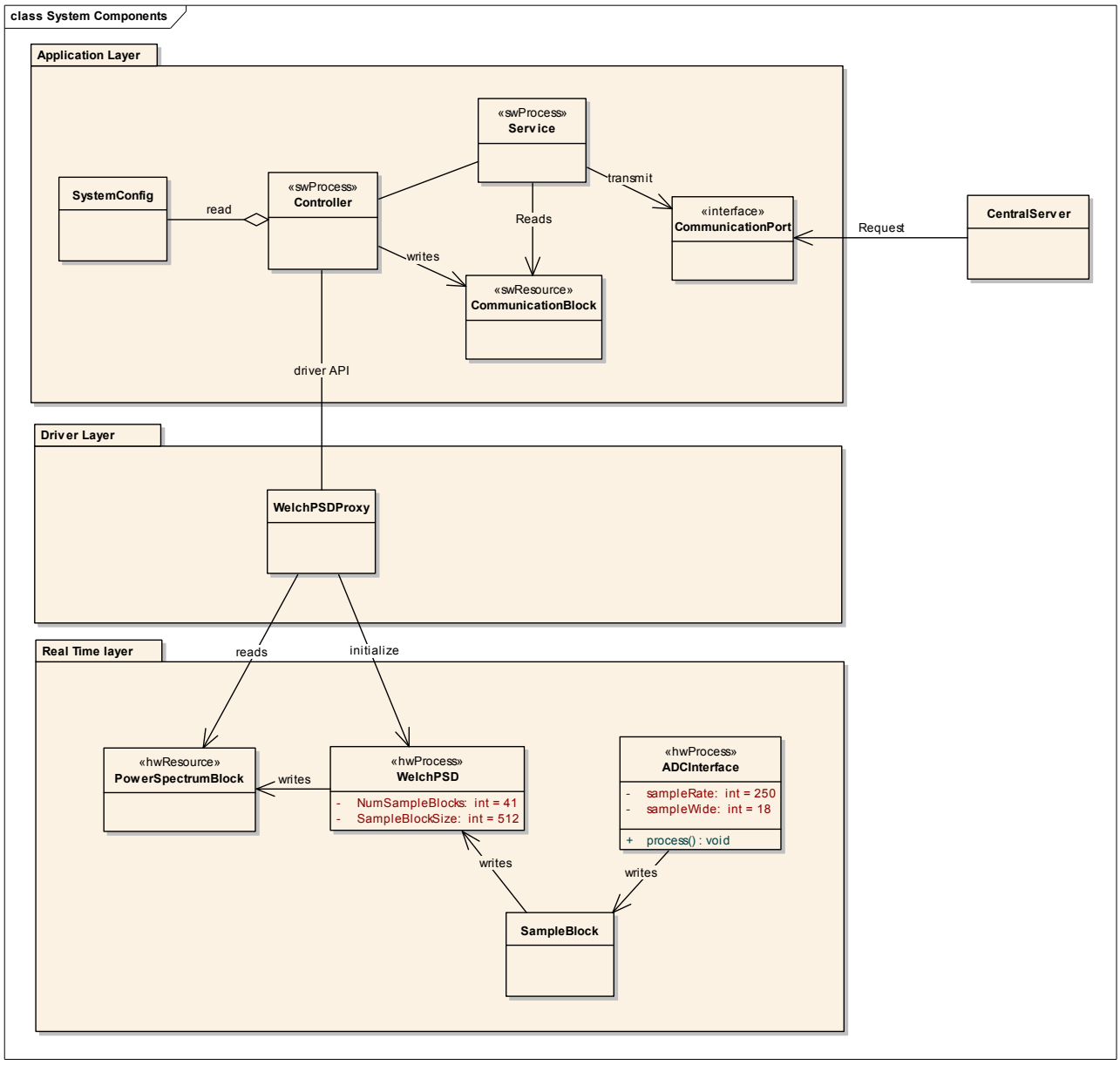

Figure 7 Measurement System Package diagram

The deployment view is documenting the physical devices in the system. This view focuses on how the software and hardware architecture maps onto the physical devices such as FPGA, processors, active sensors/actuators, controllers, displays, I/O devices or central servers. UML uses the concept of a node to represent a physical device.

The combination of UML components and deployment nodes are shown in the same diagram. A component is an artifact of development existing at runtime. In the measurement system we have an

application program (WelchPSD_DataProvider), a linux driver (WelchPSD_Driver) and two HW IP-blocks which are all components deployed on the embedded platform.

Our focus it to visualize where the logical concepts, in terms of components, are mapped to the physical architecture. See Figure 8 Measurement System deployment diagram which illustrates a deployment view for creating the partition of HW/SW components on our target platform which is a Xilinx Virtex4 FPGA with an embedded PowerPC processor running Linux 2.6.

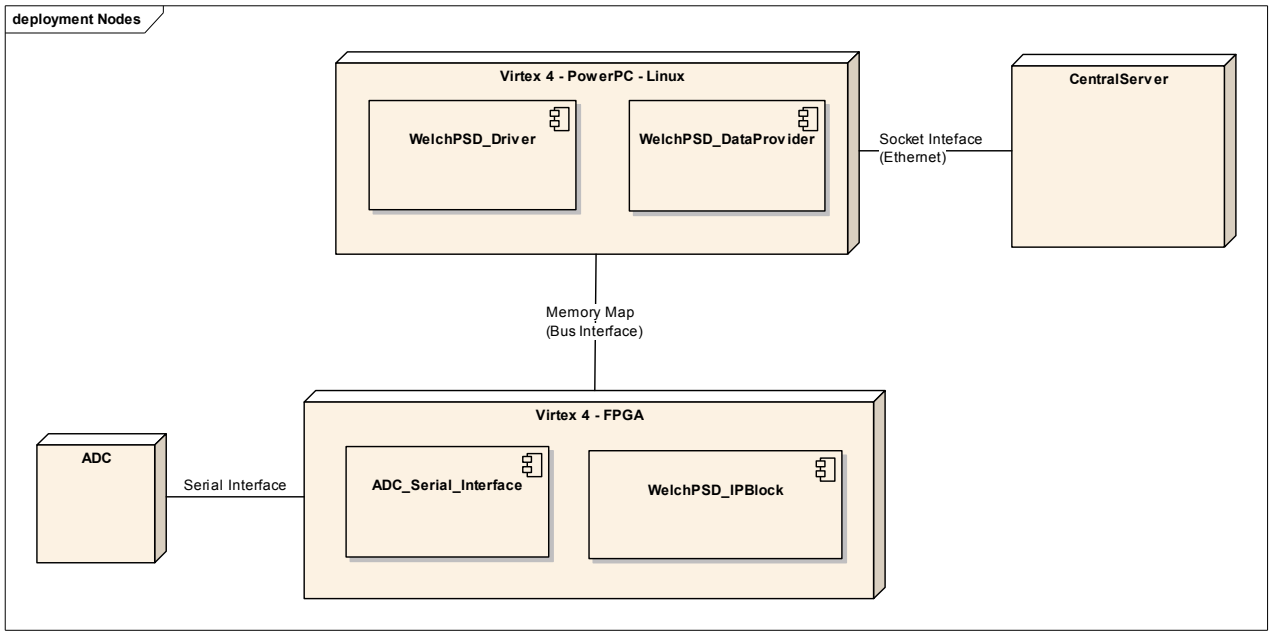

Figure 8 Measurement System deployment diagram

The last step is the implementation view where we are looking at the interface specification between components. It covers interface specification to external devices and between hardware and software objects inside the system.

The internal object behaviour in our system is described in this view using state or activity diagrams. A state diagram describes a finite state machine (FSM). A FSM is the case where an object possesses a finite set of conditions of states. A state is defined to be a mutually exclusive condition of existence defined by the set of events it processes and the actions it performs. Objects with state machines react to events in well-defined ways. Incoming events can induce transactions between states.

The difference between activity and state diagrams is that state diagrams are used when the classifier progresses primarily from state to state upon receipt of events, whereas activity diagrams are used when the classifier progresses from state to state, primarily when the actions performed in the predecessor state are completed. In practice, state diagrams are used to model reactive behaviour of classes when they proceed via the reception of events. Activity diagrams are used to model control flow behaviour of operations and activities on states. The common use of activity diagrams in real-time and embedded domain is to show computation of algorithms[2].

[2] Taken from ”Real Time UML Third Edition” by Bruce Powel Douglass

In the implementation view we use a combination of component and timing diagrams to describe interfaces. Ports are used to connect components together by using hardware signals. The use of navigation arrows for the connection between ports are used to indicate the direction of the signals.

In our measurement case study we will examine one of the hardware components in our design. In this case it is the serial interface that converts an 18 bit serial sample value from the ADC to a parallel sample which is processed by the hardware IP block (WelchPSD-IPBlock). This example illustrates how UML diagrams could be used for a hardware design at a very detailed level. See Figure 9 Analog to Digital Converter (ADC) serial input, combined component and timings diagram as an example of how such an interface specification could be described using UML.

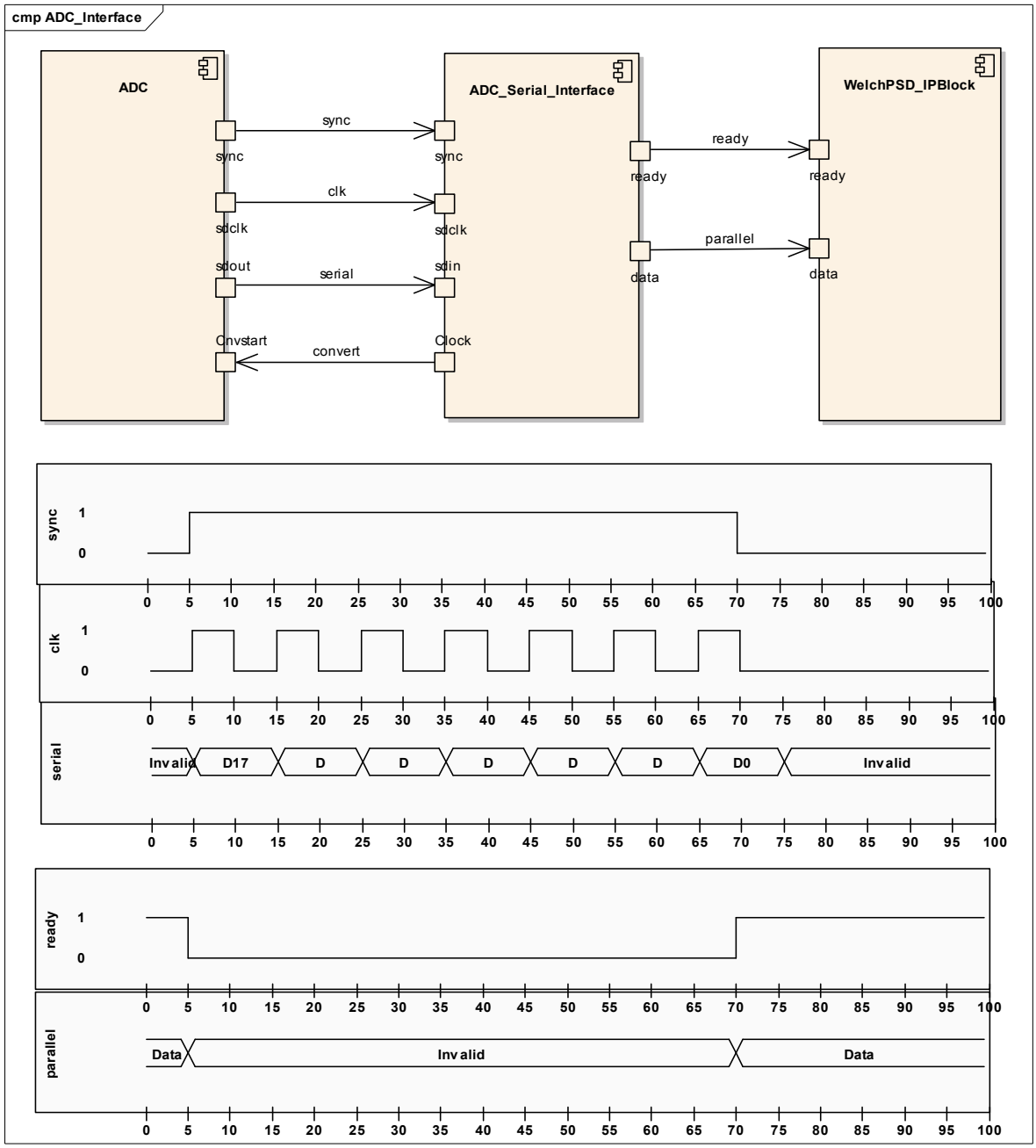

Figure 9 Analog to Digital Converter (ADC) serial input, combined component and timings diagram

The internal behaviour of components and classes is described with UML activity or state diagrams. In the following example you can see how the ADC serial to parallel interface behaviour is described with an UML state diagram. This state diagram concerns the behaviour of the ADC serial interface hardware block. This state machine could be implemented in VHDL or SystemC source code.

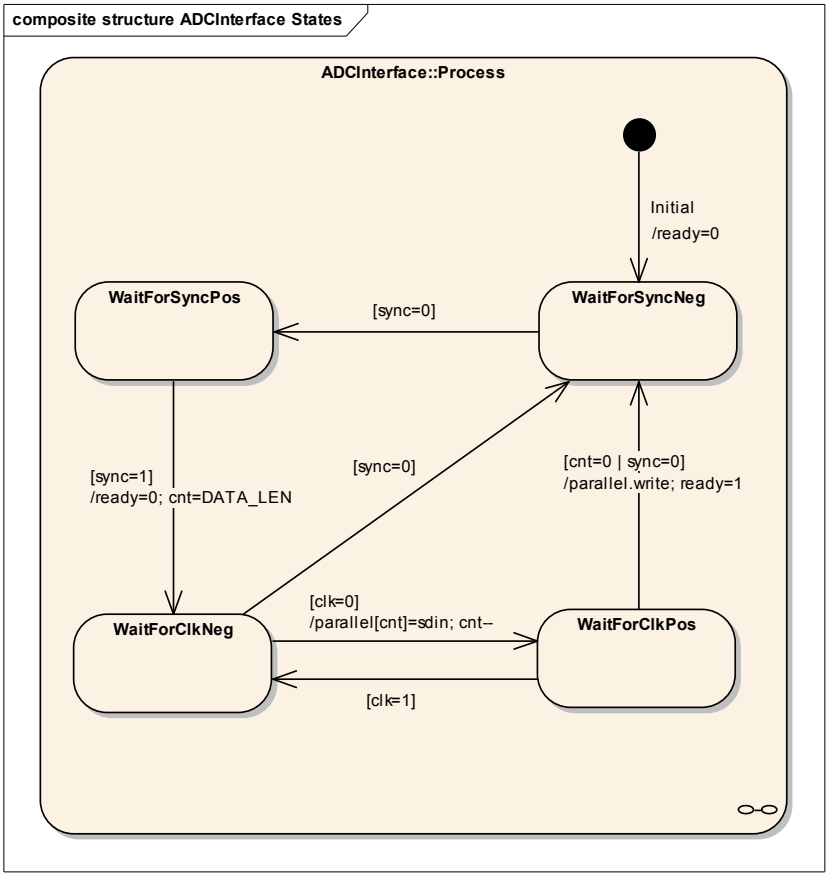

Figure 10 ADC internal state diagram

The final step is to define a memory map describing the data structure to be used as the interface description between the HW block and driver software. In our measurement system we have used a combination of a UML sequence diagram and a textual description to design the interface between the WelchPSD_IPBlock (HW block) and the WelchPSD_Driver software.

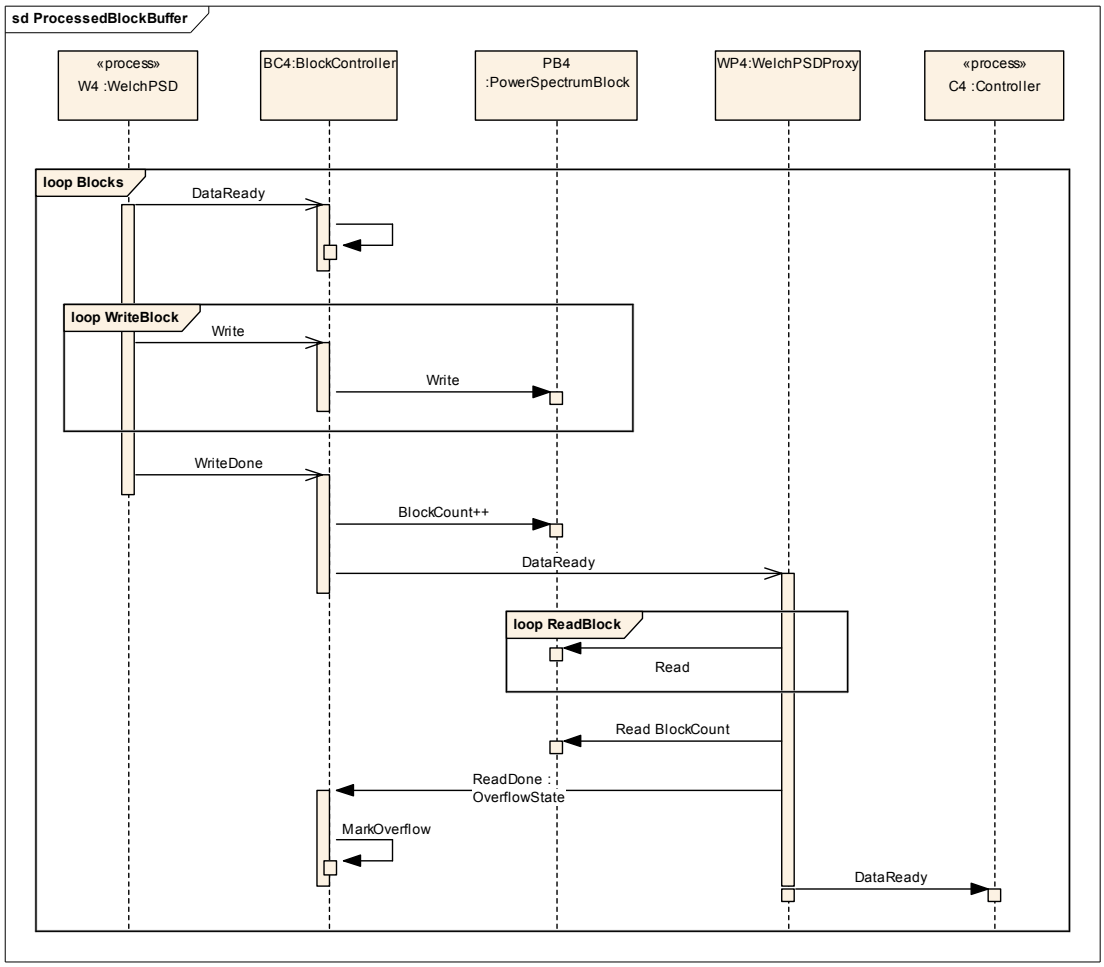

Figure 11 WechPSD interface sequence diagram

The sequence diagram identifies a number of signals and variables like **DataReady**, **BlockCount** and **OverflowState** which will be described in details in the memory map between the hardware and software layer.

The memory map is a structural description of data/register variables to be exchanged between the hardware and software layer. Every variable is named with the same identification used in the UML diagrams. A variable has a number of attributes like: **description, words, direction, reset value, range and address**. These attributes describe a variable in detail and should be seen as the software application programming interface (**API**) to the hardware.

The **direction** attribute indicates to the software whether the variable can be written to or read from (W/R). The default **reset value** is the value which the hardware layer has updated after a reset condition. The **range** attribute indicates the range in which the software is allowed to modify a variable. All addresses are relative to a base address of the memory mapped hardware block.

Variable Name	Description	Words (32bit)	Direction	Reset value	Range	Addr.
NumSampleBlocks	Number of sample blocks to be used in calculation of a PowerSpectrumBlock.	1	W/R	41	32-128	0
SampleBlockSize	Size of sample block. (PowerSpectrumBlock)	1	W/R	512	256-1024	1
WindowDWidth	Window width of the Welch curve.	1	W/R	1024	512-2048	2
Control	Control register for the Welch PSD.	1	W/R	0		3
OverflowState	Overflow counter. Incremented every time a block overflow is detected. Cleared automatically by reading. This register will also be used to signal that a new PowerSpectrumBlock has been read.	1	R	0		4
Status	Status register for the Welch PSD.	1	R	0		5
BlockCount	Incremented each time a new block is updated. Wraps around when upper limit reached.	1	R	0		6
PowerSpectrumBlock	Processed PowerSpectrumBlock. Valid words in block depends on the SampleBlockSize. There will be new data in block when Status register DataReady bit is 1. Buffer is emptied by reading NumSampleBlocks.	1	R	0		7

Figure 12 WechPSD interface memory map

The table below describes in detail the bits of the control and status registers.

Control register				
Bit Name	**Description**	**Bit Pos**	**Direction**	**Reset value**
Restart	Control Reset bit - as long this bit is set to 1 the Welch PSD block will stay in reset mode. The Welch PSD block will be enabled setting this bit to 0.	0	W/R	1
InterruptEnable	This bit will enable the PowerSpectrumBlock interrupt from the WelchPSD.	1	W/R	0
WindowStart	Set to 1 for start updating a new Welch curve.	2	W/R	0
EnableSerialInterface	Enable the serial interface.	3	W/R	0

Status register				
Bit Name	**Description**	**Bit Pos**	**Direction**	**Reset value**
DataReady	This bit will be set by the Welch PSD when a new PowerSpectrumBlock is processed and ready for reading. Cleared by reading the overflow register.	0	R	0
SampleReady	This bit is set to 1 every time a new sample is avaible. Cleared when RawSample is read.	1	R	0
ChannelID	Id of the channel for the last spectrum	31 - 24	R	0

Figure 13 WechPSD interface memory map control and status registers

System Design Views

To give a brief overview of the architecture modelling steps, the four different views are listed below. For every view a number of artifacts (documents and diagrams) will be produced depending on the complexity of the embedded system. The use-case view is the driving factor which is why the system design views are called the "4+1 views".

- Process View
  - Adding concurrency and resources
  - Software resources, processes and threads
  - Hardware resources and processing blocks
  - Relations and synchronisation

- Logical View
  - Refinement of class diagram
  - Dividing classes into logical packages

- Deployment View
  - Components
  - Partitioning
  - Deployment

- Implementation View
  - Signals and Timing
  - Sequence, activity and state diagrams
  - HW/SW Interfaces (IP memory maps)
  - External Interfaces

Summary

This article has described a sample case using a methodology based on UML for modeling real-time and embedded systems, covering both hardware and software elements. The methodology is a guide for the steps to do - coming from a specification to a design based on UML.

Our experience is that the use of a common visual UML language improves communication and understanding for the system to be implemented for teams of both hardware and software developers. The methodology ensures that you will be closer to a fine solution as an alternative to starting implementation without a structured coordination of the software and hardware team. The methodology does not guarantee a better result but it helps you to make the right decisions early in the development process. It visualizes the decisions made during the analysis and design phases covering functionality, partitioning, platform and interface specification to ensure a common understanding of how the system should be implemented.

We recommend that a development process using this methodology includes design and review meetings where the UML diagrams are reviewed by the customer and hardware and software developers, depending on where you are in the phase of the project.

References

Below are listed some books, training material and tools which are used as background material for this article.

Books

The following books will serve as a good starting point for learning about UML, Methodologies and Model-Driven Development for real-time systems.

"UML Distilled, Applying the standard object modeling language" by Martin Fowler

"UML and the Unified Process, Practical Object-Oriented Analysis & Design" by Jin Arlow, Ila Neustadt (Describes the Unified Process)

"Real Time UML Third Edition" by Bruce Powel Douglass
(Describes the ROPES process)

"Real-Time UML Workshop for Embedded Systems" by Bruce Powel Douglass
(Describes the Harmony process)

Training

The following training material has been used as input for this article:

"OO analyse, design og realisering med UML", course material, Danish Technological Institute (Describes the "Teknologisk Unified Process" TUP)

Links and UML tools

OMG Group web site: http://www.uml.org/

Papyrus (Open Source eclipse based UML tool) web site: http://www.papyrusuml.org/

Sparx System – Enterprise Architect (UML tool) web site: http://www.sparxsystems.com.au/

Visual Paradigm – Visual UML modeling tool http://www.visual-paradigm.com/

Telelogic - Rhapsody (MDD tool) web site: http://www.telelogic.com/

IBM – Rational Systems Developer (MDD tool - Formerly called Rational Rose) web side: http://www.developers.net/ibmshowcase/view/1328
Xilinx – FPGA tools and boards web site: http://www.xilinx.com/

Tool versions

Enterprice Architect ver. 6.5
Xilinx FPGA tools: EDK and ISE ver. 9.2