Journal

Exercise 2 HW/SW Co-design

Date: **07/12-10**

Authors: **Brian Vestergaard Danielsen,**
Teddy Holm Roskvist,
Anders Hvidgaard Poder

Introduction

In this journal contains our findings and conclusions for Exercise 2. The first part of the Journal will contain the answers to the questions posed in the assignment, and the appendix contains the detailed rationales and minutes of meetings that lead us to these answers.

Assignment 2.1

During our first meeting we arranged to come up with our individual suggestions and then agree on a method.
The individual suggestions may be found in the appendix.
The conclusion was that we would use SysML as much as possible, and amend with custom timing diagrams if needed. We did however decide to not use the requirements diagram, but instead simply use a table for the non-functional requirements. We also decided to try and map our diagrams to the Y-chart, in order to maintain a good link to the HW/SW co-design methodology.
For details about the individual diagrams please refer to the appendix, or simply see them in the following parts of the journal.

The Y-chart was developed in 1983 to explain the differences between different design tools and different design methodologies in which these tools were used.
The Y-chart districts between three basic ways which different properties of a system a can be modelled from.

These three aspects can be described as:

- **Behaviour** – Specification or functionality, here the system is seen as a black box where only inputs and outputs are described.
- **Structure** – Netlist or block diagram. The block box is described as components and connections between components
- **Physical** – Layout or board design. Adds dimensions to the structure. The physical dimensions of the design are specified: Height, width and position of each component.

Futhermore, the Y-chart also introduces four levels of abstractions:

- **System** – Here the system is defined as a number of computation- and communication components. The developer identifies which processors, memory and busses that should be used in the system.
- **Processor** – Processors, memory controllers, arbiters, interface components.
- **Logic** – Gates and flip-flops, register transfer components
- **Circuit** – Transistors

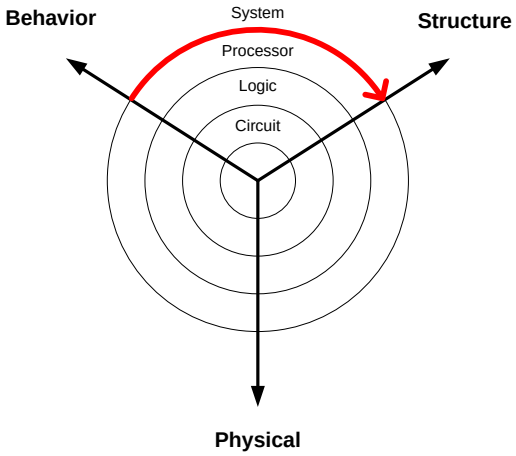

Figure 1 - Y-chart, the red line indicates where we moved from and towards.

The work we conducted during this exercise relates to the process of moving from the Behaviour-axis to the Structure-axis at the system abstraction level.
The behaviour of the system was already specified from the exercise description, so we had to figure out how to divide up the behaviour into software and hardware and define components that for filled the requirements.

Assignment 2.2

We decided to do assignment 2.2 and 2.3 in plenum during our first design session, based on prepared input from the participants with respect to use cases. The final Use-case diagram came to be as follow:

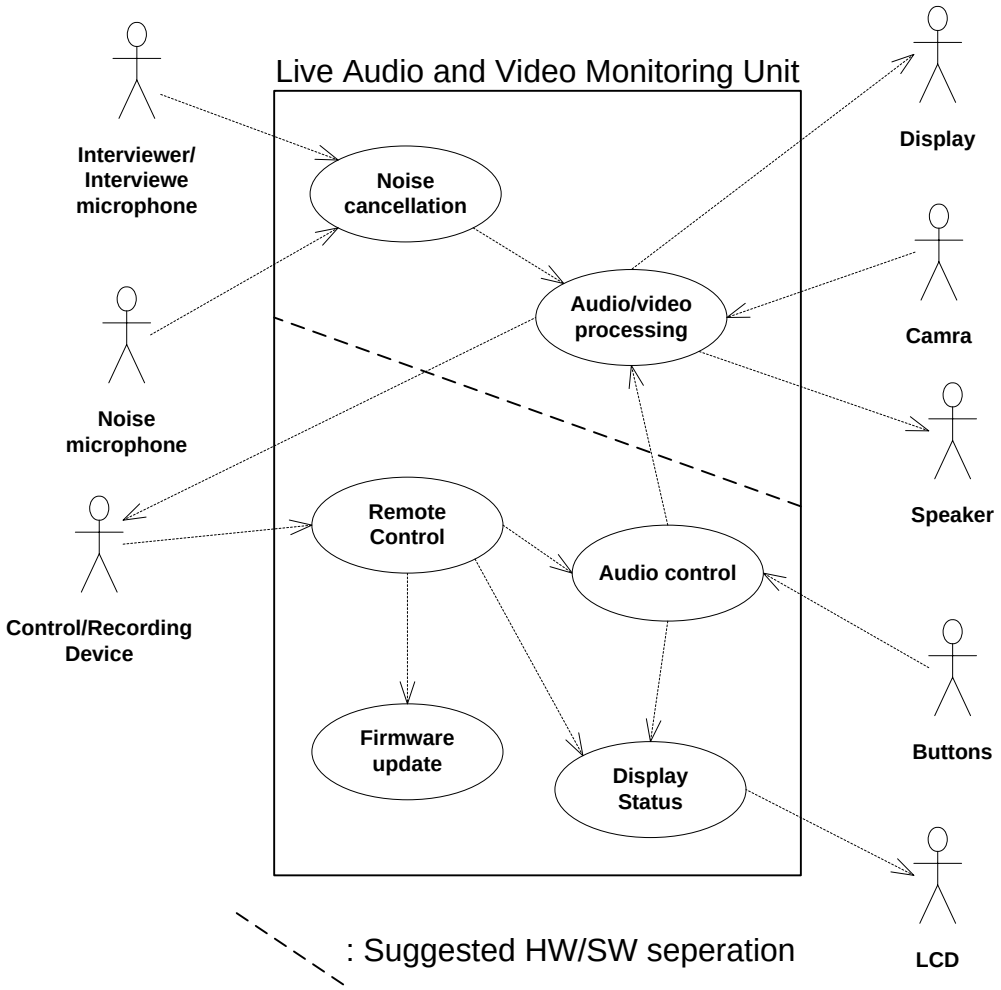

Figure 2: Use-Case diagram

It may be seen that we have included a HW/SW separation, and though this is not officially a part of the use-case diagram, we would like to illustrate that we already here started to notice an “obvious” separation into HW and SW components.

The rational for this separation may be found in detail in the appendix, but will also be discussed in assignment 2.3, but it is a matter of how likely the functionality is to change, how math-intensive it is, and what its performance requirements is. As for performance we talked about the non-functional requirements, and decided on the following:

Req. ID	Related Use Case(s)	Description
1	Noise Cancellation, Audio/Video Processing	The system shall have a 48[KHz] samplerate.
2	Noise Cancellation, Audio/Video Processing	The system samplerate shall have a 24 bit resolution.
3	Audio Control, Remote control	The system shall have a button for adjusting the volume , the adjustment shall happen within 500[ms]

4	Audio Control, Remote control	The system shall have a button for adjusting the bass (<500[Hz]), the adjustment shall happen within 500[ms]
5	Audio Control, Remote control	The system shall have a button for adjusting the treble (>4[KHz]), the adjustment shall happen within 500[ms]

Table 1 – Non-functional requirements

Req. ID	Description
6	The system shall have two microphone inputs.
7	The system shall have a two line output.
8	The system shall have a unit for displaying the current status of the unit
9	The system shall have an VGA output port.
10	The system shall have a LCD display.

Table 2 – Design constraints

This is clearly not a complete requirement specification, just like the detailed use-case descriptions have been left out. This is done on purpose to focus on the architectural design and not the requirements.

Furthermore the assignment calls for analyzing the functionality with diagrams, but this we would like to postpone to assignment 2.3, where the architecture and design will describe the functionality and design.

Assignment 2.3

There are many ways to document a system like this, and many different diagrams one may choose, but to keep the journal manageable we have decided to use the block diagram type (basic and internal) to describe the composition and communication flow of some of the important components.

Firstly we look at the static structure of the overall system with a basic block diagram

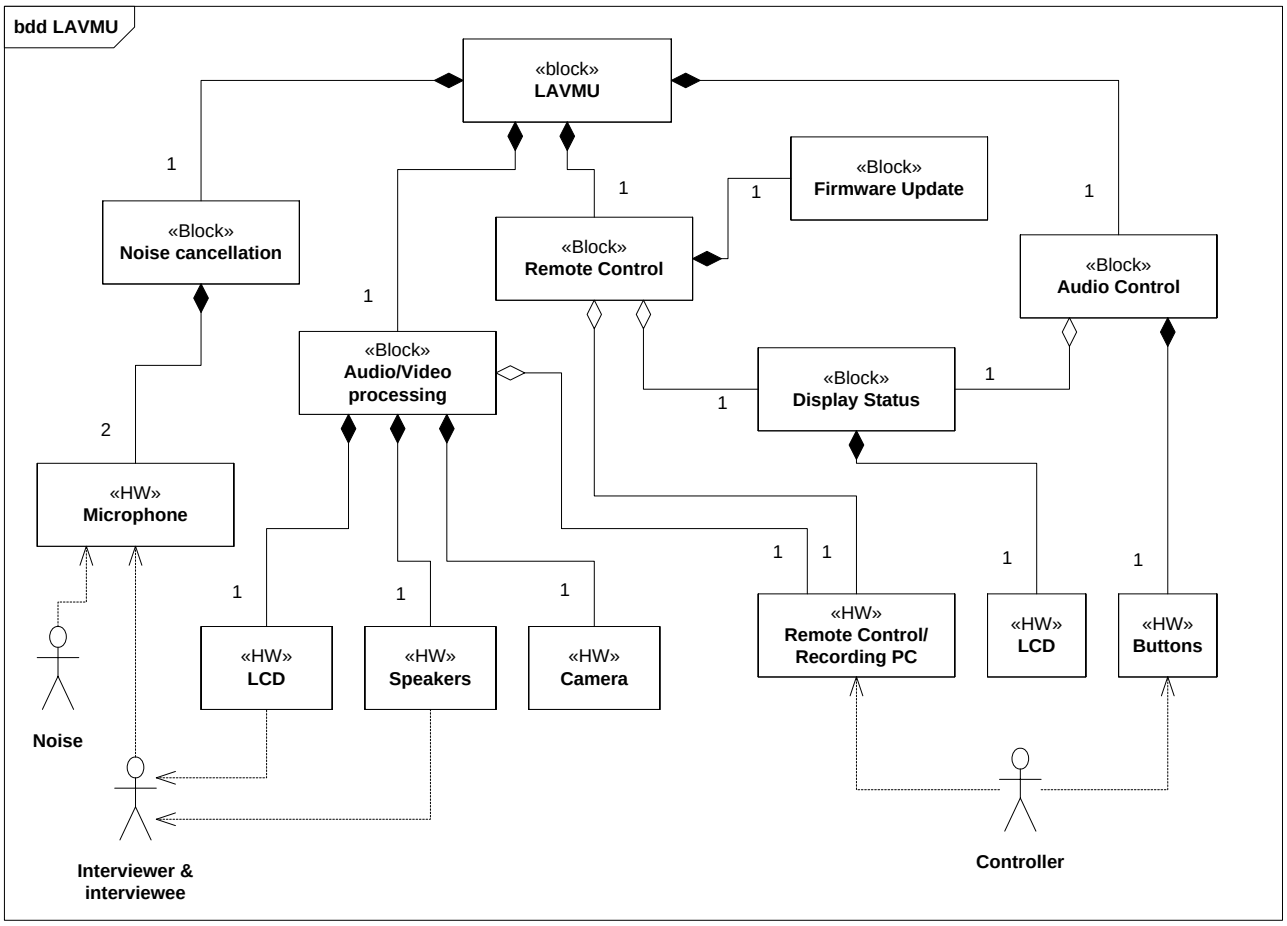

Figure 3: LAVMU Basic Block Diagram

As it may be seen no decision has been made as to what is implemented in HW or SW, except for the parts that is a physical unit, e.g. the physical microphone, which has been moved from actor to HW block.

Looking at the internal block diagram we start adding more detail, and yet we still do not have to decide on HW or SW implementation, but the added details may aid us in our decision.

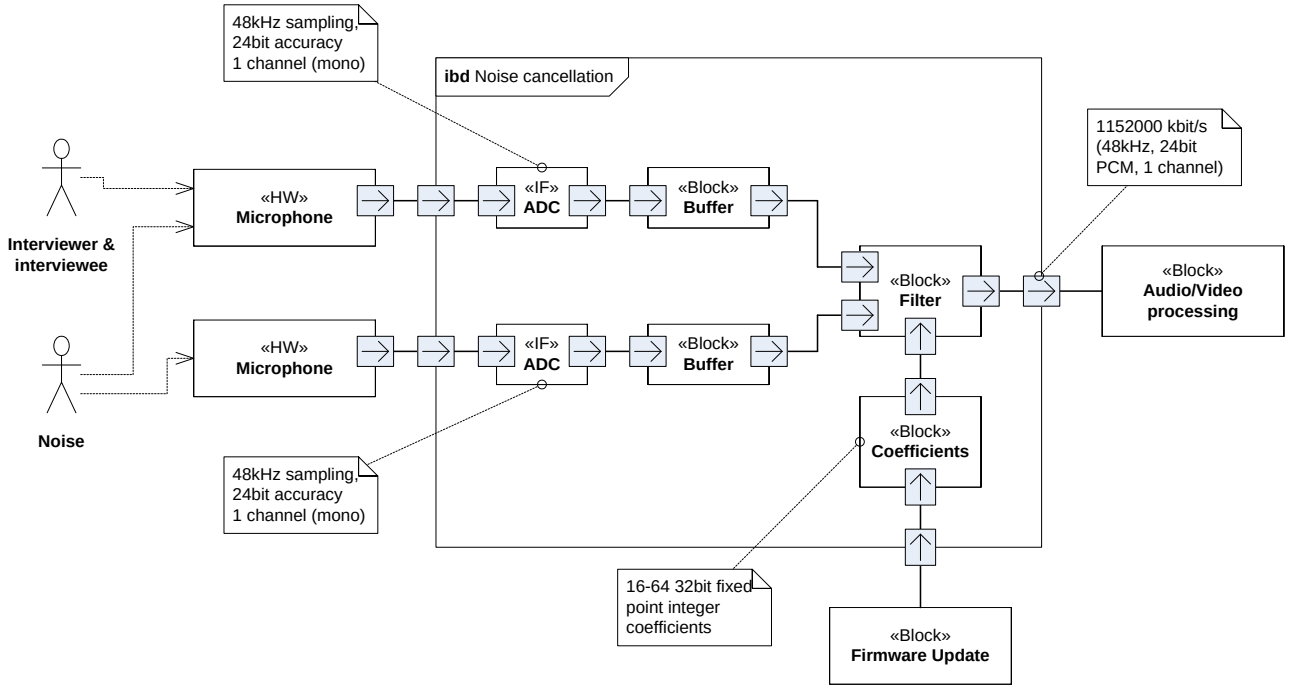

Figure 4: LAVMU Noise cancellation Internal Block Diagram

This diagram details the inner workings of the Noise cancellation on the block level (no selection of algorithm or mapping yet), but the different required parts may be seen, as may the input and output and speed of same. We can see that the flow into the noise cancellation and out of the noise cancellation is quite intense (data flow oriented), with the exception of the Firmware update which is sporadic (asynchronous means no lower limit between events) and do not have a “short” time requirement. The received audio signals must be read in and cross-correlated (or other filtering technique like LMS ☺) to remove the noise and written to the output before the next sample is ready.

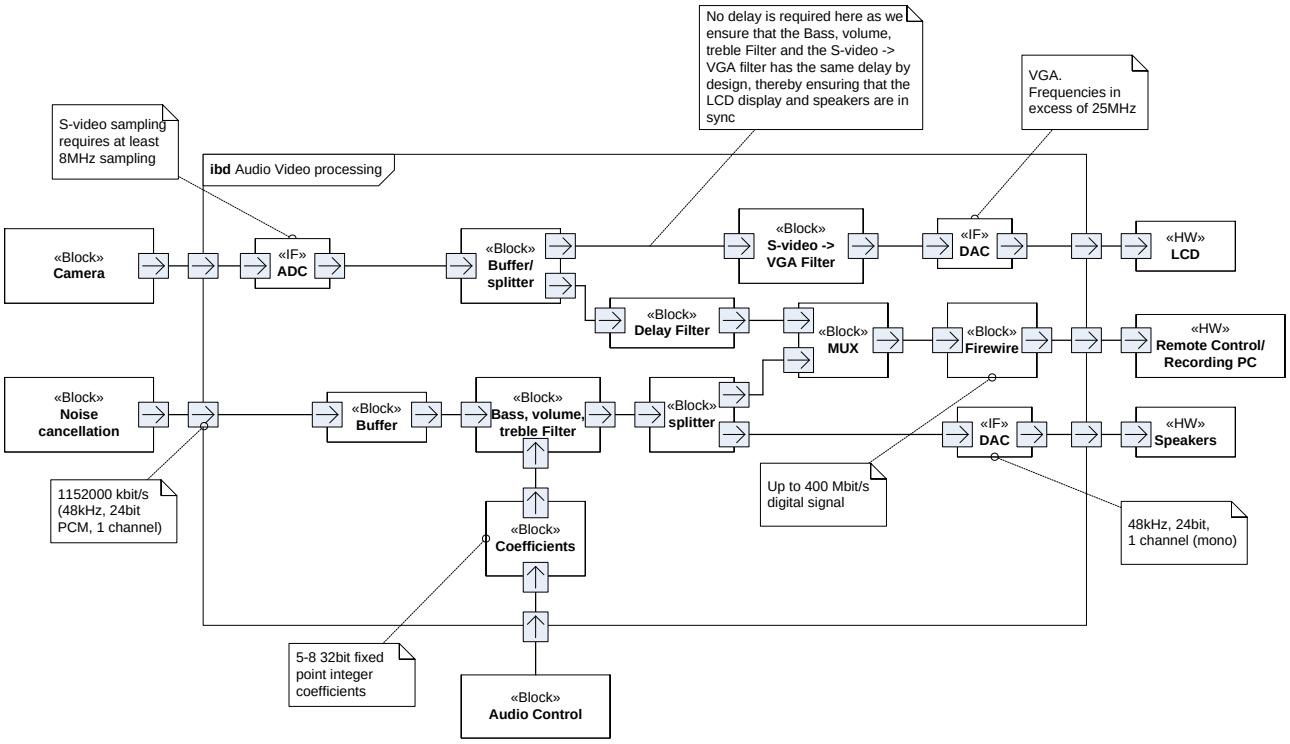

Figure 5: LAVMU Audio/Video processing Internal Block Diagram

This diagram shows the Audio/Video processing, and here the data-flow structure and high speed input and output becomes even more pronounced, with the exception of the Audio control. Here we must read and filter not only the audio signal from the noise cancellation block, but also read and process video and combine the audio and video into an Audio/Video stream, and all before the next sample is ready for processing.

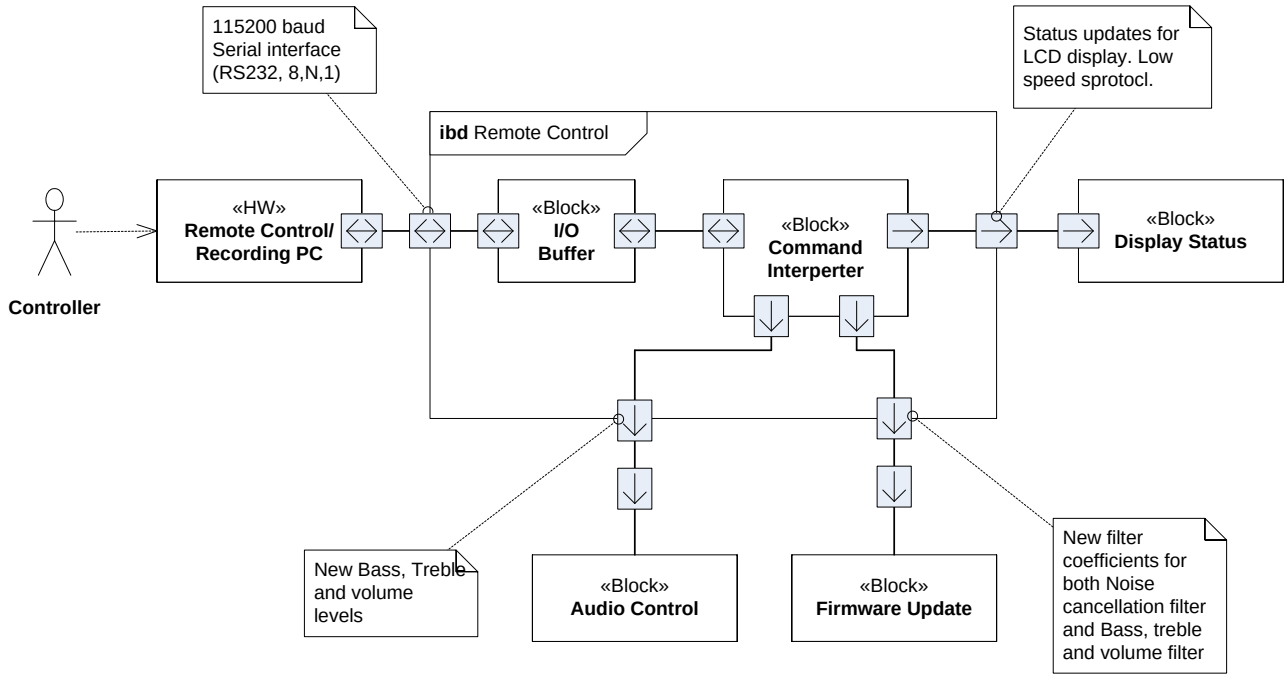

Figure 6: LAVMU Remote Control Internal Block Diagram

This diagram shows the Remote control block, and unlike the other blocks, this has no hard real-time requirements. Granted the serial bus has a data flow, but not only a rather slow one, but also one where the data comes sporadic and may therefore be buffered, and where (if we use RTS/CTS flow control) we even have the ability to request a pause in transmission if our buffer is full, and all without losing data or sacrificing any deadline. Furthermore, though it may not be seen here, the Command interpreter is pure control logic.

Instead of doing the remaining diagrams we have decided to focus on the HW/SW separation. Naturally after the HW/SW separation many more diagrams should be drawn, like state diagrams for the Remote Control would be an obvious choice, and timing diagrams for the delay/filter and mixing in Audio Control.

Separation of HW and SW

There is much information what must be considered when deciding on whether functionality should be mapped to HW or SW:

- Our current platform has no HW support, live with it.
- We have no HW development resources available.
- The given functionality cannot be implemented in SW and meet its deadline.
- The lower unit cost cannot outweigh the added cost and complexity of developing this in HW (neither ASIC or FPGA).
- We already have HW RTL components available for this functionality, so HW implementation is not a problem.

The only direct restriction that we are facing is whether a SW mapping is fast enough, which we may determine by analyzing the number of instructions required to process one sample and the speed at which the samples are arriving – that way we can determine the required size of a CPU or HW (not the same, as required instructions in SW is not the same as the required instructions in HW).

We have decided to do a simple table based approach and look at the different blocks and determine how suited they are for HW mapping, based on the following criteria:

- Performance requirements
    - If there is a high throughput performance requirement then HW is a good alternative to SW, and might also be the only possibility.
- Risk of change
    - Changing HW is much more complicated then changing SW, so if there is a high probability of change to the block then a HW implementation should be avoided.
- Data flow vs. Control logic
    - HW is not very well suited to implement control logic, but is much more suited for simple filtering and other basic block algorithms. A good way to determine this in practice is to look at the state diagram and the breakdown into Basic Blocks. If there are many states and Basic Blocks then it is not very suited for HW-mapping.
- Availability
    - Using COTS to minimize risk and development time and cost is a vital tool in a designer's tool-box. If a suitable existing tried and proven IP or SW library exists then it is almost always preferable to developing it yourself.

Based on these criteria we have come up with two suggestions for an architectural mapping. First of all we look at the tings that are "no brainers".

The high throughput, standardised protocol and well tested existing solutions makes the design of the SVideo -> VGA and RAW -> MPEG4 -> Firewire simple to map to HW and the performance requirements actually makes it difficult to map it to SW. Actually the Firewire you usually purchase as an ASIC, but technically it could also be an IP and run on the same HW as the other transformers. The blocks that are already designated as HW (e.g. the ADC and DAC) will not be mentioned, as they are already decided. These components are standardised, meaning very low risk of change. They have a high requirement for throughput, they are heavy data-flow oriented, and they are readily available to purchase as HW (either IP or ASIC).

In the other end of the spectrum is the Remote Control, Firmware update, Display Status and Audio Control. Here the real-time requirements are practically non-existing. A serial protocol for commanding the system from a PC and uploading new firmware running on a serial bus (RS232) at 115200 baud does not set high performance requirements. How long it takes to upload new firmware is almost irrelevant, as it would never be done while the system is "active". As for audio control it is irrelevant if it takes 100, 200 or 500ms to update the audio – at most a very soft real-time requirement, easily met in SW. And this holds for both commands received from the Remote PC or the buttons, which can easily be polled or hooked up to an interrupt. Furthermore these blocks have quite a bit of control logic – even the audio control has to

distinguish between audio set from the remote PC and via the buttons. It cannot simply sample the buttons and set the audio coefficients accordingly, as that would override any audio settings from the remote PC. The system must have some kind of control logic to distinguish when a given input is master. Then there is the risk of change. A protocol to the remote PC and what should be written to the display status LCD is both proprietary protocols and therefore often subject to change from user responses and also not available as COTS. All of this points to an easy SW implementation.

This leaves two blocks or sub-blocks; Noise cancellation and the part of the Audio/Video processing that has to do with bass, treble and volume filtering. The noise cancellation algorithm can either be purchased as an IP, making it a no brainer as a HW implementation, but assuming the COTS noise cancellation algorithms are not sufficient then we have to implement it our selves. Naturally a SW solution is faster to implement, but filters like this generally have a high throughput requirement, making them difficult to implement in SW. This point towards making a SW solution for testing and then porting it to HW when it is good enough. Unfortunately the algorithm might have to be changed after deployment, which is not possible if we assume an ASIC or a single FPGA running everything without the ability to do partial burn. This risk of change points to a SW solution, but is it possible? Let us have a look at some figures:

Looking at “Understanding active noise cancellation” by Colin H. Hansen we see that the common noise cancellation algorithms are very clock cycle intensive in the magnitude of many million clock-cycles for processing a single sample through a 100 coefficient filter. Looking at “A FPGA-BASED ADAPTIVE NOISE CANCELLING SYSTEM” by Wolfgang Fohl and Jörn Matthies, we can see that realizing this algorithm in a 400k gates FPGA loads the FPGA about 25% when run at 48kHz. This may quickly be translated to a very serious CPU if it was to be implemented in SW. Therefore it maybe worth running the risk of change and implement it in HW, simply because the CPU requirements, if at all possible, would be so high.

The bass, treble and volume filter may most likely be implemented in SW, but if the filter does not have to be exceptionally precise then good COTS IPs exist, making it easy to implement in HW. On the other hand if it has to be more accurate than existing IPS, then it most likely requires a high order filter, meaning that the SW throughput may be a problem. This again point towards a HW implementation.

Combining this gives us the table below, which contains our suggestion for an architectural mapping.

	HW non-changeable gate arrays	HW changeable gate arrays	HW/SW boundary	SW
Noise cancellation	Noise cancellation algorithm	Coefficients registries		
Audio/Video processing	Volume, bass, treble algorithm	Volume, bass, treble coefficients registers.		
	Audio/Video MUX -> Firewire (IEEE1394)			
	S-Video -> VGA			
	Audio -> speakers			
Remote Control				Protocol for controlling audio levels.
				Protocol for controlling and transmitting firmware update.
			Firewire Audio/Video streaming – not really a boundary.	
Audio control			Update volume, bass, treble Coefficients registries	SW interface for updating bass, treble, volume coefficients.
				Poll bass, treble and volume dials.
Display Status				Update status on display according to input from Audio control and Remote control.
Firmware update			Update noise cancellation coefficients registry.	SW interface for updating coefficients.

Table 3 – Non-functional requirements

Assignment 2.4

We focussed on getting a good design and an architecture with a well thought through mapping. We believe that determining the correct mapping is more important than doing the actual realization from a learning point of view. We have had this confirmed by Kim Bjerge, who informed us that we were allowed to not complete assignment 2.4, as long as we had an understanding of how the realization could be done. We believe we have covered that in the previous sections, and will therefore not include a Quartus SoPC project with the implementation for the DE2 board.

Conclusion

It has been a great experience to model a system like this in the assignment. We have appropriated a lot of experience and knowledge about the use of UML/SysML in modelling Hardware/Software. Furthermore it has been a good exercise to do in-group because we have been forced to talk about the possibilities in developing the system and thereby increased our understanding of the concepts.

During the process of Exercise 2 we have gained a good understanding of the possibilities of mapping functionality in HW, and also about postponing the decision until later in the design, so it is possible to make an informed decision about what to map in HW and what to map in SW. It has been beneficial to have a design to serve as a red thread through the main part of the exercises, and has given us a good understanding not only about when to use HW and SW, but also about the process involved in documenting and designing the architecture which allows us to choose between the different mappings.

Though we have worked with SysML before it has been interesting to use it in a HW/SW Co design-centric manor, allowing us to truly appreciate the strength in the abstract block, which makes it possible to achieve a quite detailed design without requiring a mapping to HW or SW. And a design that both HW and SW engineers can understand.

We have focussed on the understanding of the HW/SW architecture and the possible mappings, and have therefore, on your suggestion, chosen to disregard the implementation of a prototype, yet it is our belief that such an implementation is fully within our capabilities after this Exercise (and Exercise 1), as is also making informed decisions with respect to HW/SW mapping and architectural design in an industry project.

Annex

29/10-2010: Project start-up

We arrange first deadline (8/11-2010) and exchange email addresses. At first deadline we should have read up on the exercise and have finished a suggestion for the model to use for the exercise (SysML/UML).

8/11-2010: We exchanged emails with suggestions

http://code.google.com/p/masterofit2009/downloads/list
http://code.google.com/p/masterofit2009/wiki/Exercise2Suggestions

17/11-2010: First architecture/design meeting

1. We talked about the Use Cases, and created a solution by combining input from the different proposals and adding new ideas that we got on the spot. We managed to close the Use case specification. We spent some time talking about whether the use case (i.e. requirement specification) could be thought of as part of the Y-model, or if it is a pre-condition to the Y-model. I

was decided that it did not really matter as we needed to finish the system architecture before the first curved line (Behaviour -> Structure for System level) was completed. http://masterofit2009.googlecode.com/svn/trunk/syseng_hwco/hwco/Exercise2/ahp_UseCases.docx

2. We talked about the Y-model and how it mapped to our process and choice of method (SysML with a twist). We came up with the following mapping:
    a. System level Behaviour -> Structure
        i. Use cases for functional requirements
        ii. Simple table of requirements for non-functional requirements.
        iii. Block diagram for overall architecture.
        iv. Internal block diagrams without detailed description of the specific interfaces, only which blocks communicate with which.
    b. Process level Behaviour -> Structure
        i. Internal block diagrams without detailed description of the specific interfaces, only which blocks communicate with which and how the blocks are realized (HW/SW).
        ii. Finite state machine to describe state behaviour.
        iii. Activity diagrams to show control flow.
    c. Process level Structure -> Physical
        i. Select individual libraries for block realization in HW.
    d. Logic level Behaviour -> Structure
        i. Internal block diagrams with detailed description of the specific interfaces
        ii. Sequence diagrams where needed to show control flow and possibly timing.
    e. Logic level Structure -> Physical
        i. Configure individual libraries for block realization in HW.
    f. Circuit level
        i. Actual realization on selected FPGA

We furthermore talked about how, by employing risk minimization, it is not necessary to defined the SW <-> SW interfaces until after the HW is fully designed, as there really is no risk in that part (and also not much learning from our point of view). The Y-model will therefore focus on separating the blocks in HW and SW, and then on defining and realizing the HW blocks.

The above separation is a preliminary separation, and we may realise that this is not an optimal separation.

We discussed realization of the use cases in an architecture and agreed on a number of matters:

1. The connection with the Recording and control PC will be a serial connection. This is beneficial is e.g. it is a more static setup, where there is a mixer connected serially to the unit. It is also a simple communication form which is widely used. An alternative could be Ethernet, where the LAVMU e.g. has a built in web server via which the control access can be gained, and an address to which it "streams" the recorded data, e.g. UDP or TCP. This is beneficial if the LAVMU is designed to be moved around and to be used at external locations. For the stationary location where the recording and control PC is close to the LAVMU the audio and video feed for recording should be transmitted via Firewire.

2. The Use case diagram can be separated into two, with the Audio/video processing and Noise cancellation being realized in HW and the Remote Control, Firmware Update and Audio Control being realized in SW. The reasons for this is:
    a. Audio/Video processing has a very predefined behaviour, separated into tree; S-video -> VGA, Audio + Video MUX -> Recording stream format, Volume and Bass, Treble control. The first two is a predefined non-changing standard and can be done solely in HW no problem. The latter is a combination of amplification, high-pass and low pass filtering. Given a fixed number of coefficients (N-order filter with N constant) it is possible to change the value of the coefficients, but not the filter itself (the number of coefficients and calculation). As the bass, treble and volume may be done with a standard filter it is not a big problem having a fixed number of coefficients and a fixed algorithm. For these reasons all Audio/Video processing may be placed in HW. The Audio/Video processing also has a specific hard real-time requirement with respect to the S-video, VGA, speakers and recording stream, and a deterministic HW solution is therefore always preferable.
    b. Noise cancellation is just like the Volume and Bass, Treble control a series of filters. If a filter with a fixed maximum number of coefficients and a fixed algorithm may be defined, then it can easily be done in HW, and it can (though perhaps not easily). The Noise cancellation also has a specific hard real-time requirement with respect to the input audio sampling of the two microphones and the audio output, and a deterministic HW solution is therefore always preferable.
    c. Remote Control has the purpose of allowing a remote user to set the volume, treble and bass. As a user per definition is not real-time (and non-deterministic), there is no requirement that the setting of the volume, treble and bass should be. If the remote user changes the volume twice and the first time it takes 15ms to change it and the next time it takes 25ms, it is not important – had the communication channel been Ethernet the communication channel itself would have introduced a “big” non-deterministic delay. Also this form of control logic is simpler to implement in SW, and with no reason to move it into HW (it does not matter if it takes 25ms to change a value or 18ms) it should be done in SW.
    d. Audio control has two interfaces, but only one purpose. It should sample interface 1 (the physical volume, treble and bass buttons on the LAVMU) and present a SW interface (functions) to the Remote Control block where the volume, bass and treble can be set from SW. Naturally if the volume, bass and treble is set from SW we should not reset them next time we poll the buttons. This may be accomplished by only setting changed values, but here we had to define “change”. We decided that the buttons are simple “potentiometers” which are then samples by an ADC, but as some imprecision exist here (LSB may be change due to noise) a certain minimum change before update must be define. This specific value will be defined later – the SW interface needs no such functionality.
    e. Firmware Update is also control logic, though here we had a much longer discussion about it. The main question is this: “If we have only one chip is it then possible to re-program the entire FPGA (or part of the FPGA) while also executing from it?” We believe it is not possible. There may be several solutions; Have a secondary update chip which takes over the firmware update from the main chip and simply has the capability of re-programming the main chip. Alternatively we simply define the firmware as the part we can update, i.e.

the SW, or perhaps only a part of the SW – the coefficients for the filters. This has the disadvantage that we cannot change (increase) the number of coefficients in the HW filters nor update the algorithm. Alternatively we could place everything (that has changeable coefficients) in SW, which allows for a complete update of everything. This is actually the two architectural considerations we will use. The S-Video -> VGA and Audio/Video MUX so not have to be placed in SW, as they are standards, i.e. no changeable parts. The Noise cancellation and volume, treble, bass adjusting is specifically designed to this system, and may therefore be improved after the fact. Unfortunately placing everything in SW means loosing the speed and guaranteed determinism of the HW implementation, and we believe that changeable coefficients are sufficiently flexible and we will recommend the HW solution. Also a SW solution has the problem that the speakers and Audio/Video MUX has a hard real-time requirement, which means that the data must be available early enough. This can be handled with buffers, a fast enough CPU and control of the scheduling, but by implementing it in HW the problem goes away entirely.

	HW non-changeable gate arrays	**HW changeable gate arrays**	**HW/SW boundary**	**SW**
Noise cancellation	Noise cancellation algorithm	Coefficients registries		
Audio/Video processing	Volume, bass, treble algorithm	Volume, bass, treble coefficients registers.		
	Audio/Video MUX -> Firewire (IEEE1394)			
	S-Video -> VGA			
	Audio -> speakers			
Remote Control				Protocol for controlling audio levels.
				Protocol for controlling and transmitting firmware update.
			Firewire Audio/Video streaming – not really a boundary.	
Audio control			Update volume, bass, treble Coefficients registries	SW interface for updating bass, treble, volume coefficients.
				Poll bass, treble and volume dials.
Firmware update			Update noise cancellation coefficients registry.	SW interface for updating coefficients.

Tabel 1: Suggestion 1 (Mixed realization)

	HW non-changeable	**HW changeable gate**	**HW/SW boundary**	**SW**

	gate arrays	arrays		
Noise cancellation				Noise cancellation algorithm and sampling.
			Audio buffer for Audio/Video processing	
Audio/Video processing				Volume, bass, treble algorithm
	Audio/Video MUX -> Firewire (IEEE1394)			
	S-Video -> VGA			
	Audio -> speakers			
Remote Control				Protocol for controlling audio levels.
				Protocol for controlling and transmitting firmware update.
			Firewire Audio/Video streaming– not really a boundary.	
Audio control				SW interface for updating bass, treble and volume coefficients.
				Poll bass, treble and volume dials.
Firmware update				SW interface for updating Noise cancellation algorithm and coefficients.
				SW interface for updating treble, bass, volume algorithm.

Tabel 2: Suggestion 2 (SW realization)

We made a tentative plan to meet again on the 24/11-2010. Until then we will formalize the decisions we made at the meeting and begin the architectural design.

08/12-2010: Finishing touches meeting

We talked about the proposed construction of the Journal and created a list of missing parts and assigned responsibilities for them:

- Kommentarer til figurer (generelt, specielt interne blokdiagrammer) - Teddy
- Fix løsningstabeller samt kommentarer og opsætning (Findes i referatet fra forrige møde, indsættes hvor "Insert table") - Anders
- Konklusion - Alle forbereder noget tekst til en konklusion, vi mødes onsdag 20.00 på Skype!
- Non-functional requirements og Design Constraints (tabeller) - Brian
- 2.4: Vi har valgt at nedprioritere 2.4 - Anders
- Figurer- og tabelnumre - Teddy
- Y-chart mapping - Brian

We talked about what Kim said at the Friday lesson and we decided to follow his proposal and not worry about assignment 2.4, and focus on the other parts.

We agreed on the following schedule:

1. Monday afternoon the above assignments must be completed and committed to subversion.
2. We all read peer review on the material and by Tuesday evening the changes must be committed, and at the same time the suggested conclusions must be mailed out.
3. Wednesday we meet on Skype at 20:00 and finalize a conclusion and then hand in the journal.

15/12-2010: Skype meeting - Conclusion

We had a Skype meeting where we discussed the Conclusion. We all presented a suggestion for a conclusion and we combined them in a way that suited us all. Then we read final corrections in plenum and handed in.