# 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>

1. 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:
   1. System level Behaviour -> Structure
      1. Use cases for functional requirements
      2. Simple table of requirements for non-functional requirements.
      3. Block diagram for overall architecture.
      4. Internal block diagrams without detailed description of the specific interfaces, only which blocks communicate with which.
   2. Process level Behaviour -> Structure
      1. Internal block diagrams without detailed description of the specific interfaces, only which blocks communicate with which and how the blocks are realized (HW/SW).
      2. Finite state machine to describe state behaviour.
      3. Activity diagrams to show control flow.
   3. Process level Structure -> Physical
      1. Select individual libraries for block realization in HW.
   4. Logic level Behaviour -> Structure
      1. Internal block diagrams with detailed description of the specific interfaces
      2. Sequence diagrams where needed to show control flow and possibly timing.
   5. Logic level Structure -> Physical
      1. Configure individual libraries for block realization in HW.
   6. Circuit level
      1. 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:
   1. 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.
   2. 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.
   3. 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.
   4. 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.
   5. 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 gate arrays** | **HW changeable gate arrays** | **HW/SW boundary** | **SW** |
| **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