AARHUS UNIVERSITY

EXERCISE 3

Live Audio and Video Monitor Unit

Revision: 1

TIHSC1

Hardware-software Co-design of embedded systems

Deadline: 20/03-2017

Names	Initials	AU ID	Email
Frederik Andersen	FA	Au503241	frederik@fafi.dk
Morten Morberg Madsen	MMM	Au501465	morten3m@gmail.com
Thomas Holm Nielsen	THN	Au505313	thomas-h-nielsen@hotmail.com
Michael Ilkiv Misbih	MIM	Au501580	michael.misbih@gmail.com

AARHUS UNIVERSITY

Table of Contents

AARHUS UNIVERSITY

1 Document structure

1.1 Revision

Date	Revision	Author(s)	Remarks
02/03-2017	1	MMM	Initial setup of document

1.2 Document References

Reference	Document Description	Document Identifier
[DOC 1]	Hand-in – Exercise 3B – Project in HW/SW Co-design Case: Live Audio and Video Monitor Unit	TIHSC1_Exercise3B.pdf
[DOC 2]	LAVMU Usecase Specification – Chapter 2 - Requirements	Ex3B_UseCaseSpecification.pdf
[DOC 3]	"A Practical Guide to SysML" by Sanford Friedenthal, Alan Moore and Rick Steiner	A Practical Guide to SysML.pdf

2 Introduction

This exercise describes a project for using HW/SW Co-design in designing and modeling a “Live Audio and Video Monitor Unit” (LAVMU).

2.1 Description

The [Figure 1 - Live Audio and Video Monitor Unit and environment] below illustrates a new device you have to design. The “Live Audio and Video Monitor Unit” will be used for live monitoring and recording in noisy environments. The unit will be used by reporters for interviews in the field. The main purpose of the unit is audio monitoring and recording.

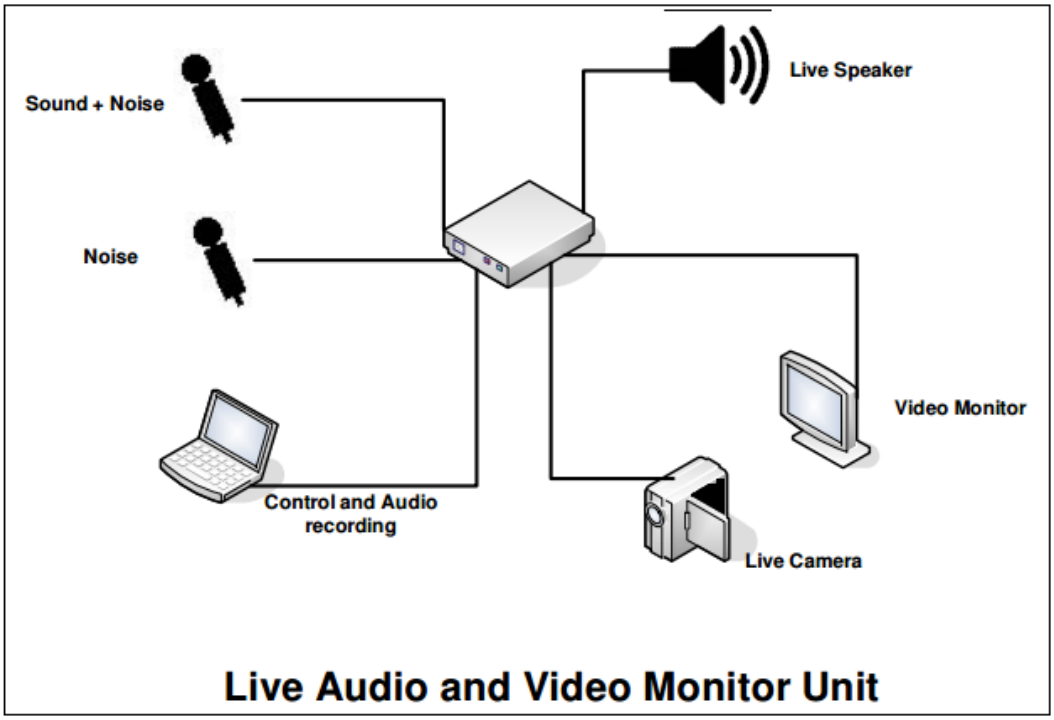

Figure 1 - Live Audio and Video Monitor Unit and environment

The unit will have 2 connected microphones using adaptive noise cancellation by subtracting the recorded noise from the surrounding. The primary microphone is placed close to the signal source to pick up the desired signal. The reference microphone is placed close to the noise source to sense the noise only. An adaptive filter must be designed on the block diagram level to remove the noise of the desired signal. The volume and tone (Bass and Treble) of the filtered sound should be able to be controlled by the user on the unit front or by using a connected computer. The LAVMU unit should have a simple LCD display and buttons to operate the unit. The output line of the filtered sound will be connected to an active speaker or connected computer.

As optional a live camera should be able to be connected to the unit and monitoring the video camera or TV input on a connected VGA Monitor. The idea is that in the future the product can be upgraded adding video processing features like edge detection or noise filtering. This upgrade of the product should be possible without changing the physical part of the hardware. From a connected computer it should be possible to update the firmware of the LAVMU unit remotely.

2.2 Physical requirements

The following points presents the provided physical requirements for the Live Audio and Video Monitor Unit:

- 2 microphone inputs
- 2 line outputs (same signal on both lines – to speaker and computer)
- Internal samplings rate of 48 kHz with a resolution of 24 bit
- Wired connection to external computer (USB, Ethernet or Serial)
- Buttons for adjustment of volume, bass (<500 Hz) and treble (>4 kHz)
- LCD-display to display status of unit
- S-Video, Composite Video Input
- HDMI output for monitor of video input

2.3 Apdaptive filter (LMS) theory

An adaptive filter consists of two functional blocks – a digital filter to perform the desired filtering and an adaptive algorithm to automatically adjust the coefficients (or weights) of that filter. In the [Figure 2 – Adaptive LMS filter], d(n) is a desired signal, y(n) is the output of a digital filter driven by an input signal x(n), and error signal e(n) is the difference between d(n) and y(n). The adaptive algorithm adjusts the filter coefficients to minimize a predetermined cost function related to e(n).

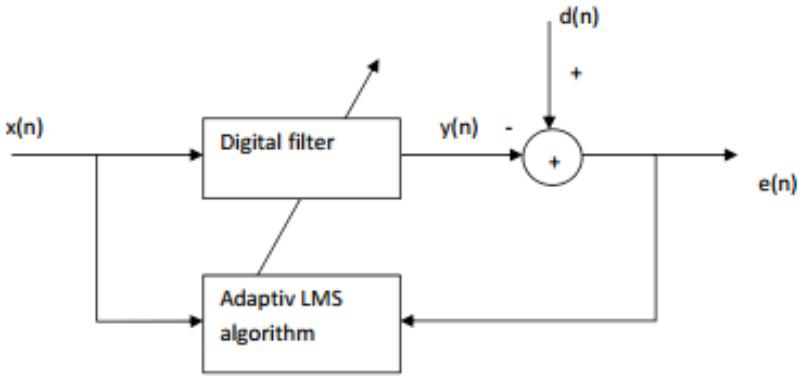

Figure 2 – Adaptive LMS filter

Adaptive noise cancellation employs an adaptive filer to cancel the noise components in the primary signal picked up by the primary sensor (Sound+Noise). The primary sensor is placed closed to the signal source to pick up the desired signal. The reference sensor (Noise) is placed close to the noise source to sense the noise only. The primary inputs d(n) consists of the signal plus noise, which is highly correlated with x(n) because they are derived from the same noise source. The reference input consists of noise x(n) alone. The adaptive filter uses the reference input x(n) to estimate the noise picked up by the primary sensor. The filter output y(n) is then subtracted from the primary signal d(n), producing e(n) as the desired signal plus reduced noise.

The digital filter is computed as

$$y(n) = \sum_{l=0}^{L-1} w(n)x(n-1)$$

The LMS algorithm updates the coefficients of the digital filter by using the steepest-decent algorithm that can be described as (LMS)

$$W(n+1) = W(n) - \mu X(n)e(n)$$

Here W is the coefficient weights vector. X is the input signal vector. The convergence of the LMS algorithm from an initial condition of W(0) to the optimum filter depends on the choice of μ. With small values of the μ factor the filter convergence speed is slow. With large values it is fast but will perhaps not find a steady performance. Typical the μ factor is in the area < 1.

The error signal is computed as

$$e(n) = d(n) - y(n)$$

A LMS filter as described above will be used suppress the noise in the LAVMU unit.

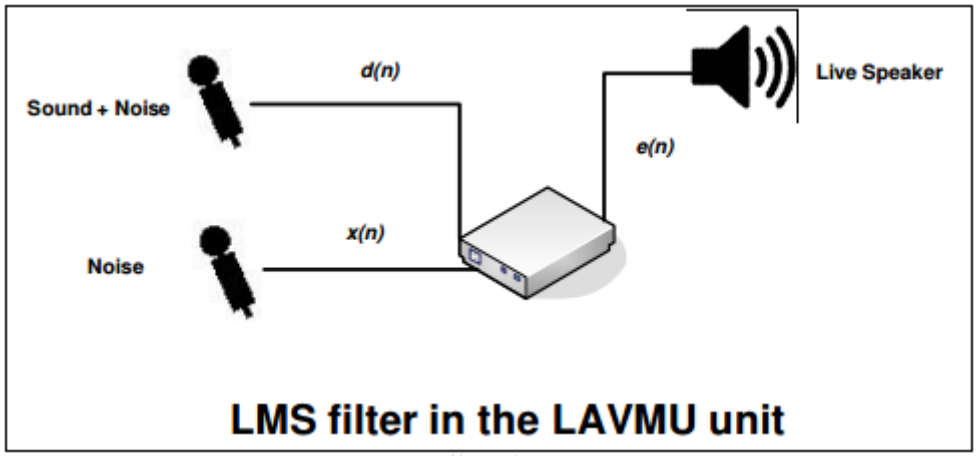

Figure 3 - LMS filter in the LAVMU unit

3 Exercise 3

3.1 Exercise 1

Description:

Decide on a method[1] using SysML/UML diagrams for the design of your system based on as minimum SysML block and activity diagrams using the principles of HW/SW co-design. Is it possible to relate these block and activity diagrams to the process state machines (PSM) as described by Gajski[2]? Make a short description of the design steps for SysML/UML diagrams and profiles you decide to use in the HW/SW co-design methodology. Remember to use references to the papers you have use as inspiration for your work. Decide on an UML[3] tool.

The diagrams and profiles that has been chosen for this exercises are pure SysML, the diagrams and standards is taken and based upon the SysML practices put forth in the book [DOC 3]. The Methodology used in this project is heavily based on the "Water Distiller Example Using Functional Analysis" Chapter 15 on page 359 of [DOC 3]. The Methodology is based on functional analysis, and uses the following approach:

- Stating the problem
- Organizing the model
- Establishing requirement
- Modeling behavior
- Modeling structure
- Model relations and allocations

The above approaches will be described in further details below:

3.1.1 Stating the problem

The step of stating the problem is the critical step of understanding the problem at hand. By analyzing and creating an initial problem definition and a problem definition diagrams one can get a better if not a concrete understanding of what the needs of the problem is. The problem definition will be captured through a various of diagrams. The problem definition diagrams of this methodology is listed below:

- Domain model

The models listed above is described in further detailed below:

3.1.1.1 Domain Model

The domain model profile of this methodology is based on the SysML Block Definition Diagram, and presents the SysML stereotype "system-of-interest" along with the items/blocks that interfaces to the "system-of-interest". The domain model helps understanding was is included in the system and what is external and interfaces to the system.

[1] See an example "A HW/SW Co-design Methodology based on UML" and literature on Blackboard
[2] Embedded System Design, Modeling, Synthesis and Verification by Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, Gunar Schirner
[3] In case you don't have a tool preference ask for a license to Enterprise Architect: http://www.sparxsystems.com.au/

3.1.2 Organizing the model

In a project it is important to organize and structure the model/mythology. To organize the model in this mythology package diagrams is used. The package diagrams follow the standard put forth in [DOC 3] page 80 – 92. The package diagram structure presents an overview of the different diagrams presented in the project in a managed and structured way. The organization of the project will consist of an overall package diagram including the following sub-packages.

- Use Case Package
- Requirement Package
- Behavior Package
- Structure Package

Each package will include one or several diagrams specifying the package category of the project. The diagrams of each package will be specified in the following sections.

3.1.3 Establishing requirements

The step of establishing requirements consists of modeling the requirement of the system so further referencing is easy and structured. The establishment of requirements consist of modeling both functional and non-functional requirements. The diagrams used for requirements in this methodology is presented below:

- Use-Case diagram
- Requirement Diagram

The above listed diagrams is described in further details below:

3.1.3.1 Use-Case diagram

The use-case diagram is used to model the functional requirements of the project. The Use-Case diagram in this methodology follows the structure and standard presented in [DOC 3] chapter 11.

3.1.3.2 Requirement diagram

The requirement diagram is used to model the non-functional requirement of the project. The requirement diagram of this methodology follows the structure and standard presented in [DOC 3] chapter 12.

3.1.4 Modeling behavior

After the establishment of the requirement, one can go to the modeling of behavior of the system. The methodology goes to the behavior modelling instead of the structure modeling, because the behavior related directly to the use cases established in the requirements. The behavior modeling in this methodology consist of two diagram types listed below:

- Activity diagram
- State machine diagram

The above listed diagram is described in further details below:

3.1.4.1 Activity diagram

The activity diagram describes the activities of the system. The activity diagrams of this methodology follows the structure and standards in [DOC 3] chapter 8.

3.1.4.2 State machine diagram

The state machine diagram describes the states of the system. The state machine diagrams of this methodology follows the structure and standards in [DOC 3] chapter 10.

3.1.5 Modeling structure

After the behavior modelling the modeling of structure can begin. The structure modeling consists of the modeling of the block or elements of the system. This methodology step consists of two types of structure diagrams which is listed below:

- Block definition diagram
- Internal block diagram

The above listed diagrams is described in further details below:

3.1.5.1 Block definition diagram

The block definition diagram describes the structure in the form of blocks of the system. The diagrams describe the composition, association and references between blocks, and can also describe allocation, flows, interfaces, constraints and parameters. The block definition diagrams of this methodology follows the structure and standard in [DOC 3] chapter 6.

3.1.5.2 Internal block diagram

The internal block diagrams model the internal flows and interfaces between block in the system. The internal block diagram of this methodology follows the standard and structure in [DOC 3] chapter 6.

3.1.6 Model relationship and allocation

The last part of this methodology is the modelling of relations and allocations between the different created diagrams. This steps involves creating relations between the different diagram, so between the requirement, behavior and structure diagram.

Lastly this step also involves the allocation or mapping for the logical elements of the system to physical element or architectures. This step can involve alto of analyzing because there can be a high number of possible physical solutions/architectures and there can be even more types of mapping/allocation.

This steps consists of the diagram type block definition diagram described in section [3.1.5.1]

3.2 Exercise 2

Description:

Read the requirement specification for the LAVMU unit found in **[DOC 2].** Are these use cases sufficient detailed for defining the software and hardware architecture for the unit? Consider non-functional requirements like performance, scale and frequency for controlling volume and tone.

During the analyzation of the requirement specification, it was found that both the use cases and the non-functional requirements was not detailed enough in some sections. The modified use cases are first presented in the following section, furthermore the adjusted non-functional requirements is listed hereafter.

3.2.1 Use Cases

For the use cases to provide sufficient information to define the software and hardware architecture for the unit, some changes were needed. The changes are indicated by brackets [] and modified into the already defined use cases from the [DOC 2]

3.2.1.1 Case 1: Control Volume

Goal	**Enable report to control the volume.**
Initialization	Interviewer.
Actors	Interviewer.
Concurrent activations	Single.
Precondition	System is started
State after succes	The volume is changed to the specific value.
State after error	The volume is unchanged.
Mainscenario	1. The actor indicates to the system, to open the volume control panel. 2. The system accepts the request [and opens the volume control panel]. 3. The actor requests the main volume to be changed [from a level range from 0-100 where 100 is highest]. 4. The system changes the main volume tone [to match the level specified by the actor].

3.2.1.2 Case 2: Control Bass Tone

Goal	**Enable report to control the bass tone.**
Initialization	Interviewer.
Actors	Interviewer.
Concurrent activations	Single.
Precondition	System is started.
State after succes	The bass tone is changed.
State after error	The bass tone is unchanged.
Mainscenario	1. The actor indicates to the system, to open the tone control panel. 2. The system accepts the request [and opens the tone control panel]. 3. The actor requests the bass tone to be changed [from a level range from 0-100 where 100 is highest]. 4. The system changes the bass tone [to match the level specified by the actor].

3.2.1.3 Case 3: Control Treble Tone

Goal	**Enable report to control the treble tone.**
Initialization	Interviewer.
Actors	Interviewer.
Concurrent activations	Single.
Precondition	System is started.
State after succes	The treble tone is changed.
State after error	The treble tone is unchanged.
Mainscenario	1. The actor indicates to the system, to open the tone control panel. 2. The system accepts the request [and opens the tone control panel]. 3. The actor requests the treble tone to be changed [from a level range from 0-100 where 100 is highest]. 4. The system changes the treble tone [to match the level specified by the actor].

3.2.1.4 Case 4: Start Interview

Goal	**Enable the reporter to start an interview.**
Initialization	Interviewer.
Actors	Interviewer.
Concurrent activations	Single.
Precondition	System is started and firmware update not ongoing
State after success	The audio signal is streamed to output lines, without noise.
Mainscenario	1. The actor indicates to the system, that an interview should begin. 2. The system accepts the request for starting an interview [and presents the option to state the interview topic, interview name and interviewee to the user]. 3. The actor indicates the interview topic. I: The topic name is not valid. 4. The actor indicates the interviewer name to the system. I: The interview name is not valid 5. The actor indicates the interviewee name to the system. I: The interviewee name is not valid 6. The system saves the interview topic, interview name and interviewee name as a new project. I: The system couldn't save the project [and notifies the user to retry the process from [1]]. 7. The system starts the recording.
Exceptions	1. The topic name is not valid. I: The systems informs about an invalid topic name, try again. 2. The interviewer name is not valid. I: The systems informs about an invalid interviewer name, try again. 3. The interviewee name is not valid. I: The systems informs about an invalid interviewee name, try again. 4. The system couldn't save the project. I: The systems informs about a failure to save the project.

3.2.1.5 Case 5: Stop Interview

Goal	**Enable the reporter to stop an interview.**
Initialization	Interviewer.
Actors	Interviewer.
Concurrent activations	Single.
Precondition	UC4
State after succes	The system is ready to start interview.
Mainscenario	1. The actor indicates to the system, that the interview should be stopped. 2. The system accepts the request for stopping the interview [and stops the process].

3.2.1.6 Case 6: Cancel Noise

Goal	**Enable system to cancel out noise from the source microphone.**
Initialization	UC4.
Actors	System, Source Mic, Noise Mic, LineOut-1, LineOut-2.
Concurrent activations	Single.
Precondition	System is started and recording is activated.
State after succes	The environment noise is filtered from the source microphone output.
Mainscenario	1. The system listens[samples values from] to the source mic, and the noise mic and captures any noise [and captures the values from both inputs]. 2. The system adapts the filter coefficients to filter out correlated noise. 3. The system streams the filtered output stream to the two line outs.

3.2.1.7 Case 7: Firmware update

Goal	**Enable the PC to update the LAVMU firmware.**
Initialization	UC4.
Actors	System, PC.
Concurrent activations	Single.
Precondition	System is started and not interviewing.
State after succes	The firmware has been updated
State after failure	The firmware has not been updated
Mainscenario	1. The PC indicates to the LAVMU that a firmware upgrade should start. I: The systems informs that an interview is ongoing. 2. The system accepts firmware update request, and requests the actual firmware image. 3. The PC sends the requested firmware. 4. The system updates the firmware. I: The systems could not update the firmware. 5. The firmware was updated. [6. The LAVMU indicates for the actor that a update was completed]
Exceptions	[1. The systems informs that an interview is ongoing. [I: The systems informs about an ongoing interview, and requests to try again later.] [2. The systems could not update the firmware.] [I: The systems informs about failed update, and prompts to retry the update.]

3.2.2 Non-functional requirements

The modified non-functional requirements are listed below. In addition, a unique identifier is given to each requirement, so that referencing to the specific requirement is eased.

3.2.2.1 General

- [N-REQ-01-01] The internal sampling frequency for the LAVMU's Source Microphone and Noise Microphone shall be 48kHz (+/- 0Hz)
- [N-REQ-01-02] The resolution of the processed audio shall be 24bit (+/- 0bits)
- [N-REQ-01-03] Pulse-code modulation (PCM) shall be used as the method for digitally represent the sampled analog signals.

3.2.2.2 Interfaces

- [N-REQ-02-01] The interface of the control line between the "Control and Audio recording computer" & the LAVMU shall be USB 3.0
- [N-REQ-02-02] The interface of the output lines Line-out1 and Line-out2 shall be of the type female minijack connection (diameter of 3.5 mm (0.14 inches).
- [N-REQ-02-03] The interface of the input lines for the Source Microphone and Noise Microphone shall be a regular female jack connection (diameter of 6.35 mm (1/4 inches).
- [N-REQ-02-04] The LAVMU shall contain a 2x16 LCD display capable of presenting the status of the unit.
- [N-REQ-02-05] The LAVMU shall contain a S-Video input through a Mini-DIN connector.
- [N-REQ-02-06] The LAVMU shall contain a Composite Video input through a RCA connector.
- [N-REQ-02-07] The LAVMU shall contain a HDMI output with a connector of the type HDMI type A (13.9 mm X 4.45 mm).

3.2.2.3 Performance

- [N-REQ-03-01] The maximum latency for the noise cancellation process shall at no point exceed 35ms (+/- 0.5ms)
- [N-REQ-03-02] The value range of the volume shall be 0-100 where 100 is highest.
- [N-REQ-03-03] The value range of the bass tone shall be 0-100 where 100 is highest.
- [N-REQ-03-04] The value range of the treble tone shall be 0-100 where 100 is highest.
- [N-REQ-03-05] The bass tone filter shall be a low-pass filter with a cutoff frequency of 500Hz (+/- 50Hz)
- [N-REQ-03-06] The treble tone filter shall be a high-pass filter with a cutoff frequency of 4kHz (+/- 100Hz)

3.2.2.4 Development process

- [N-REQ-04-01] For the design of the LAVMU a self-developed methodology shall be used.

3.3 Exercise 3

Description:

Use your design methodology found in **3.1** to describe a SysML/UML model of your system in terms of structure and behavior. Make a suggestion for alternative HW/SW architectures. Decide on which parts of the functionality should be mapped to hardware components and software processes.

Using the methodology described in section 3.1 we start by stating the problem

3.3.1 Stating the problem

Based on the initial problem definition stated in this document a domain model has been created to define the domain in which the problem is defined. The domain model, created using the SysML block definition diagram (bdd) is shown on [Figure 4 - LAVMU Operational Domain].

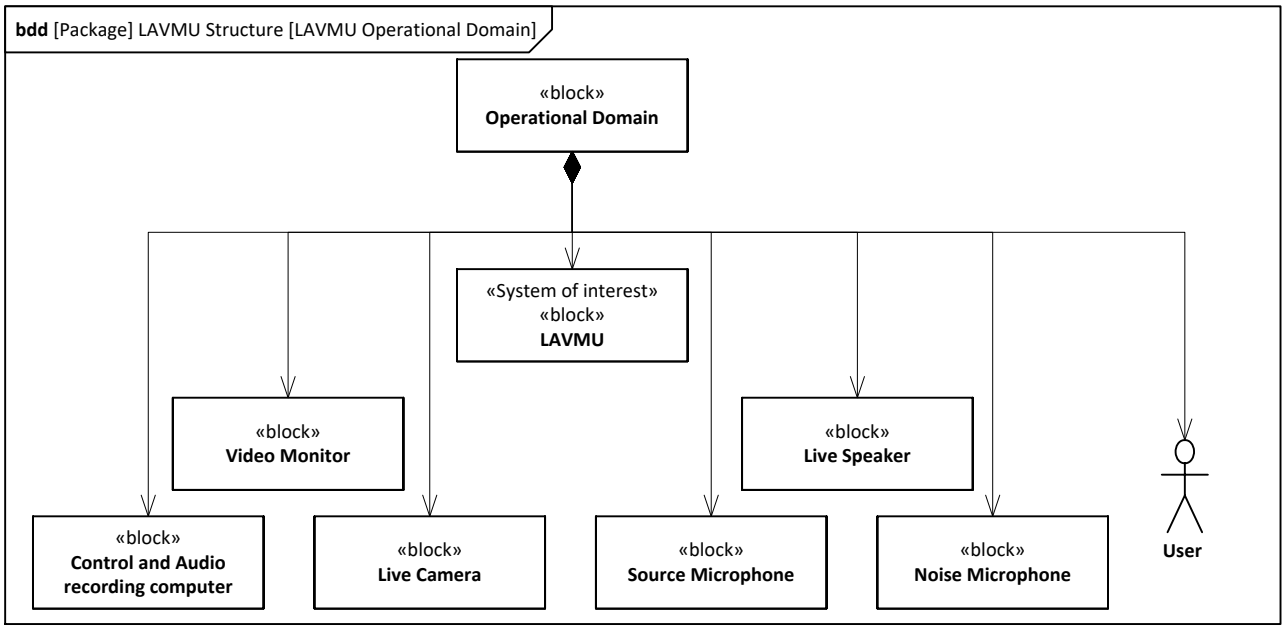

Figure 4 - LAVMU Operational Domain

The domain model represents the domain in which we work, and what in the domain that is the system of interest denoted with the SysML stereotype “system of interest”, in the domain the LAVMU is the system of interest and all the other blocks is the external elements that interact or interfaces with the system of interest. By defining the domain model early one can get a clear notation of what the external interfaces of the system are. Based on the domain model and the interfaces an internal block definition (ibd) is created showing the external interfaces of the system of interest. The ibd is shown on [Figure 5 - LAVMU Operational Domain].

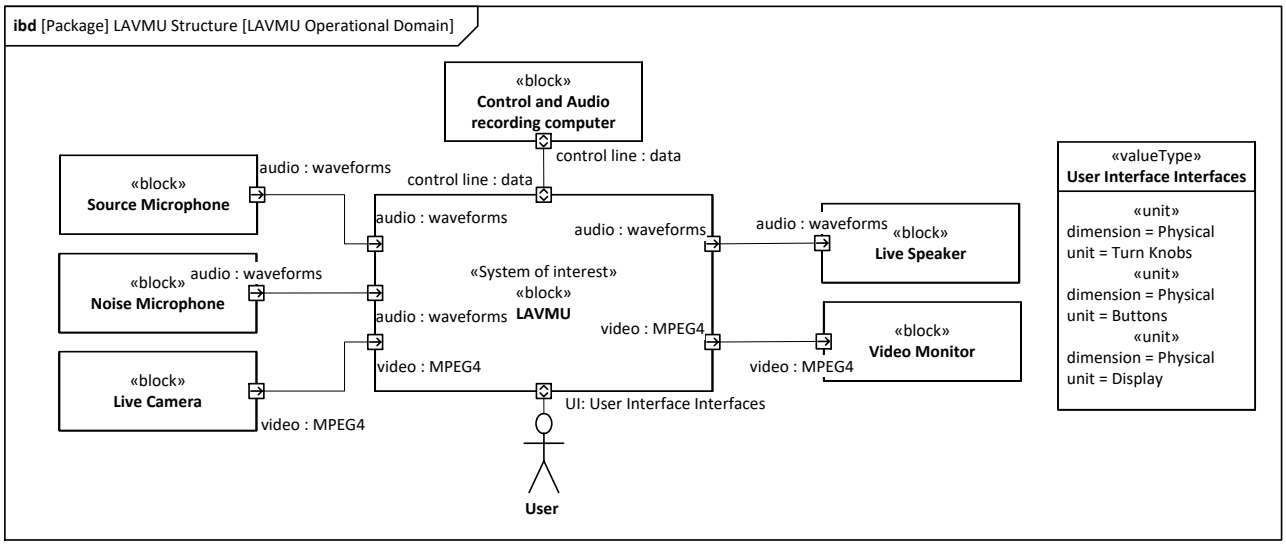

Figure 5 - LAVMU Operational Domain

On the ibd, the blocks contained within the domain is enlisted in the following way; the blocks which provide an input to the LAVMU on the left. The block and actor which control the LAVMU is located in the middle. Lastly the output blocks are presented on the right of the system of interest block. The flow gates indicate the different flows between the blocks, and indicates the direction as well. The User Interface Interfaces valueType is specified to provide a better understanding of what the interface between the LAVMU and the User is. The valueType User Interface Interfaces block should be read as a block containing three physical interfaces consisting of Knobs, Buttons and a Display.

3.3.2 Organizing the model

The next step of the methodology is to organize the model, to organize this model package diagram is used the overall package diagram of the LAVMU system is seen on [Figure 6 - LAVMU].

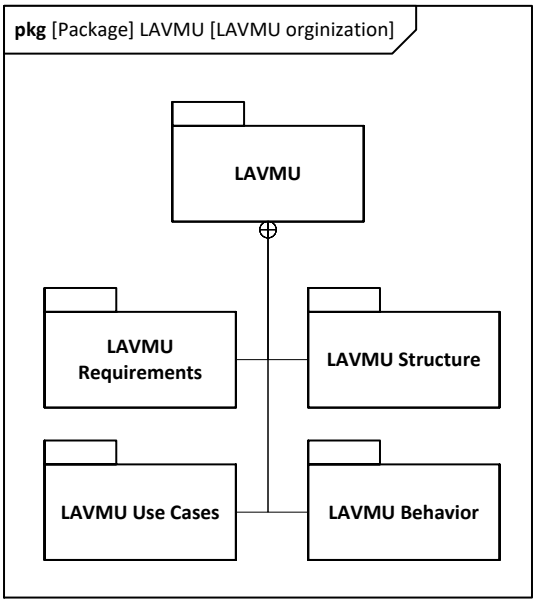

Figure 6 - LAVMU organization

3.3.3 Establishing requirements

As presented in [3.1.3 Establishing requirements], the requirements is presented through two diagram types:

- Use-Case diagram
- Requirement Diagram

First the Use-Case diagram is presented, it is based upon the already specified diagram from [DOC 2].

3.3.3.1 Use-Case diagram

The Use-Case diagram on [Figure 7 – LAVMU Use Case Diagram] contains the seven use cases presented in [3.2.1 Use Cases]. Each of the use case is linked to at least one actor which is somehow involved in the use case.

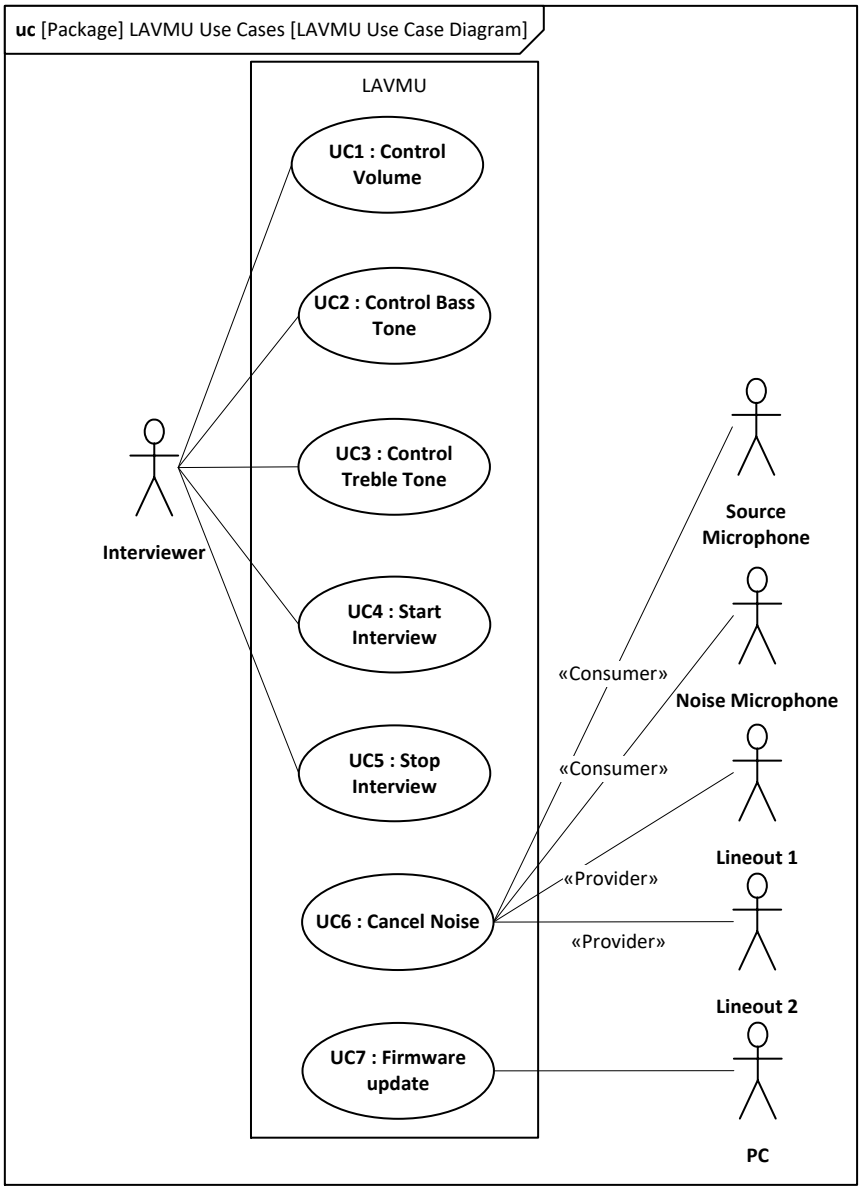

Figure 7 – LAVMU Use Case Diagram

3.3.3.2 Requirement diagram

The diagram [Figure 8 - LAVMU Requirements Overview] presented below, opens up for the LAVMU Requirements package, and identifies the four sub-packages contained within. These sub-packages are all in the category “Non-functional” and contains the requirements listed in section [3.2.2 Non-functional requirements]. To isolate the four types of non-functional requirements for the system, the four sub-packages is denoted with the following four identifiers:

- General
- Interfaces
- Performance
- Development Process

Which all match the sections created in the earlier section.

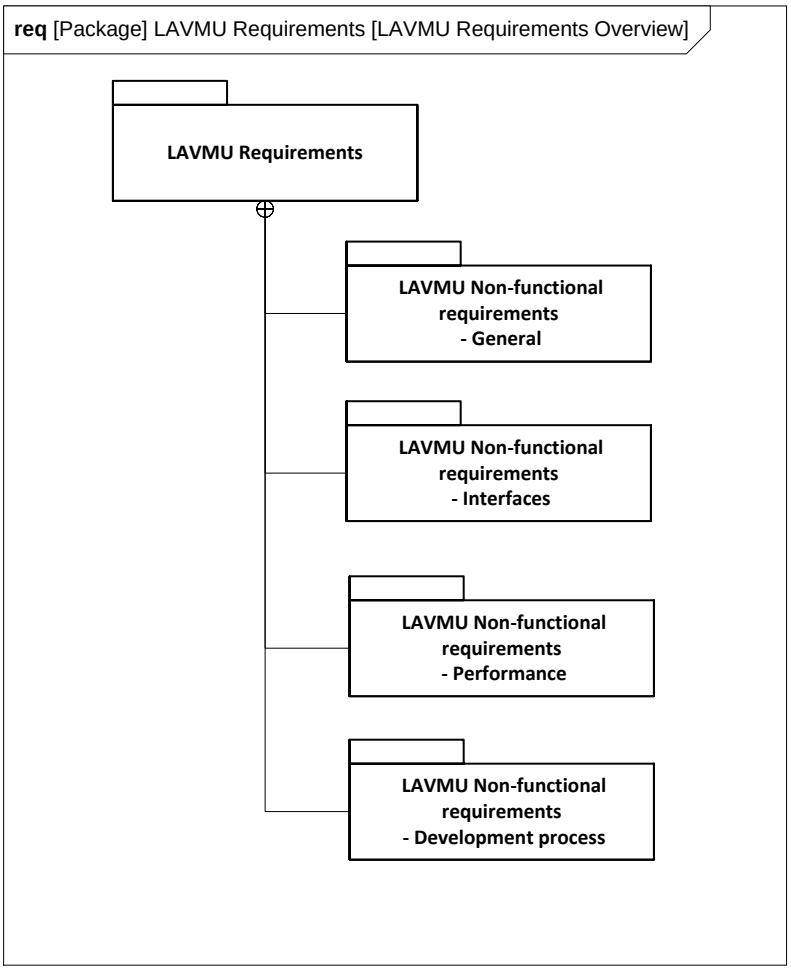

Figure 8 - LAVMU Requirements Overview

To provide an overview of the requirements, and create the possibility of further modulation of the system including the requirements, each of the sub-packages is opened up to show the contained requirements, as shown on [Figure 9 - Non-functional requirements - Interfaces]

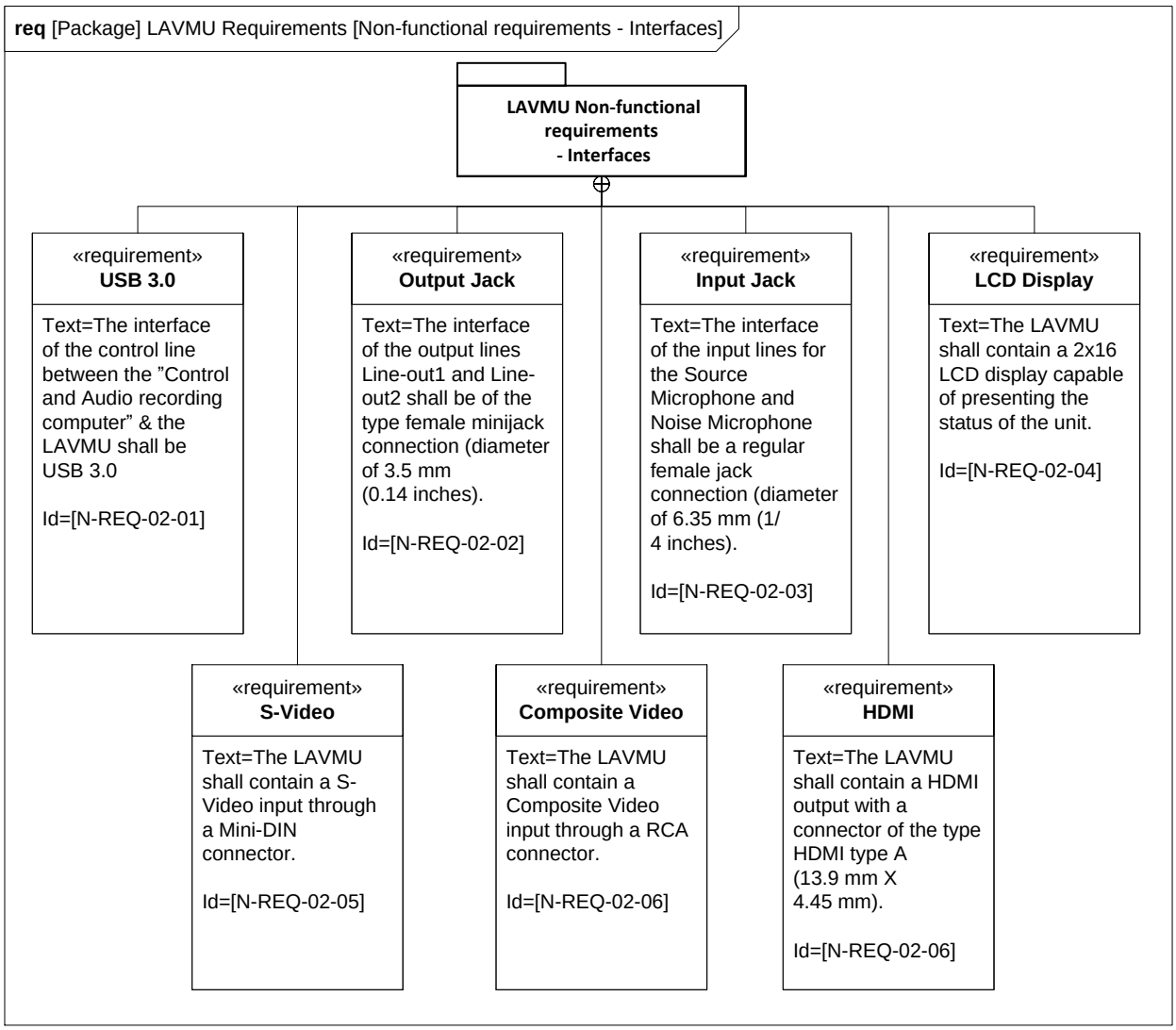

Figure 9 - Non-functional requirements - Interfaces

On the Requirement diagram above, each requirement is identified by a block with the notation «requirement». Each block follows the structure which begins with a header which names the requirement. The name should provide information regarding the context of the requirement, and as well be unique. Within the requirement block, the requirement is described through the requirement text, and the unique identifier linked to the requirement.

Each of the four sub-packages from [Figure 8 - LAVMU Requirements Overview] is opened up in the same way as [Figure 9 - Non-functional requirements - Interfaces]. To see the rest of the requirement packages, see appendix section [4.1 Requirement Diagrams].

3.3.4 Modeling behavior

Based on the established methodology the modeling of the system takes its initial steps in the behavior of the system, by doing this one can go directly form the functionality of the system in form of the Use Cases to the modeling of the behavior.

3.3.4.1 Activity

The first behavior diagram used to represent the system is an activity diagram (act). The overall activity diagram of the LAVMU system is seen on [Figure 10 - Overall LAVMU activity]

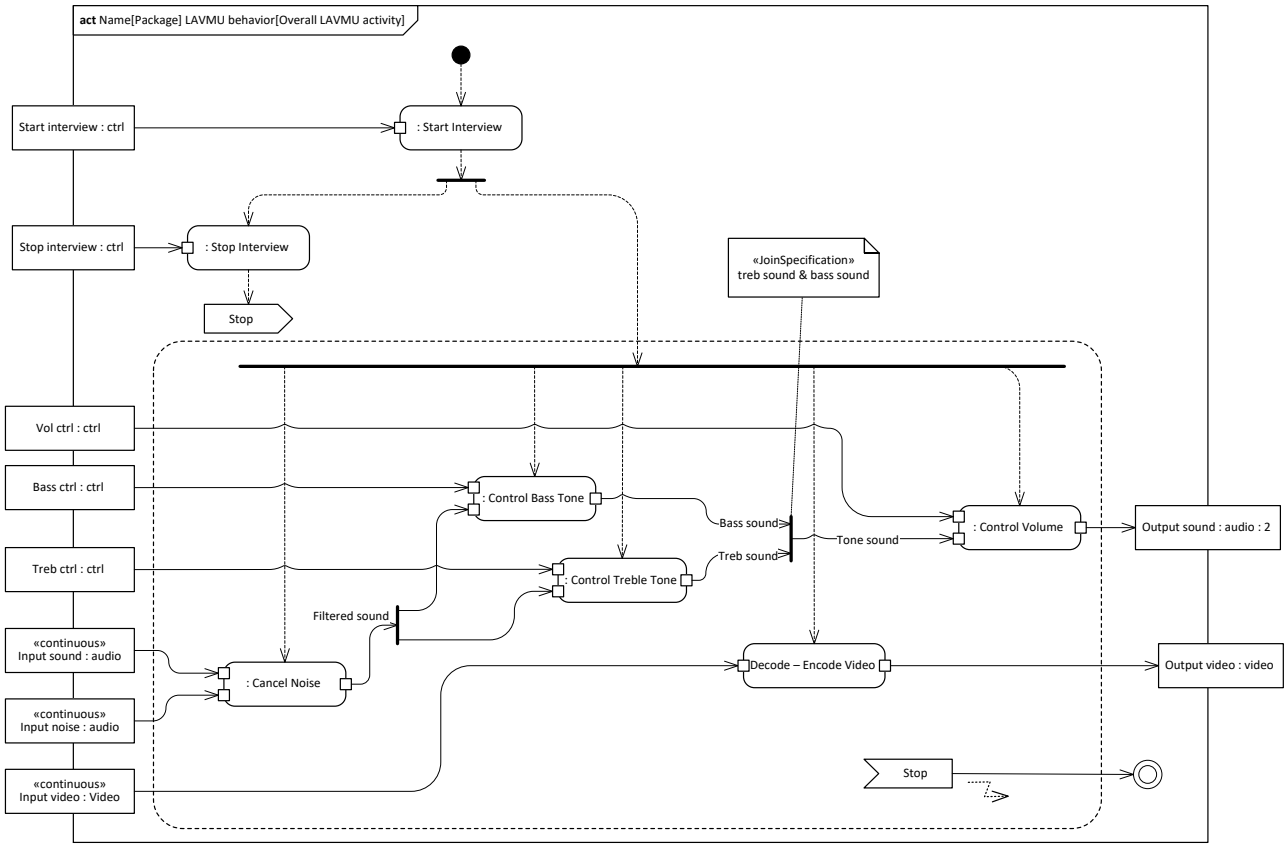

Figure 10 - Overall LAVMU activity

The diagram above shows the activities of the LAVMU, each activity in the diagram represent a use case in the Use Case diagram on [Figure 7 – LAVMU Use Case Diagram]. An exception has been made in form of the "Decode – encode video" activity with has been added for clearer understanding, also the use case "Firmware update" has not been added because it adds unnecessary complexity, and does not contribute to the understanding. The diagram both represent the flow of actions through the inputs and outputs of the activities, and the control flows of the activities. As seen the "Start Interview" is the initial activity that start all the other activities.

Based on the Overall LAVMU activity diagram on [Figure 10 - Overall LAVMU activity] each of the activities has been internally model using activity diagrams. An activity diagram of the Control Volume activity is shown on [Figure 11 - Volume Control].

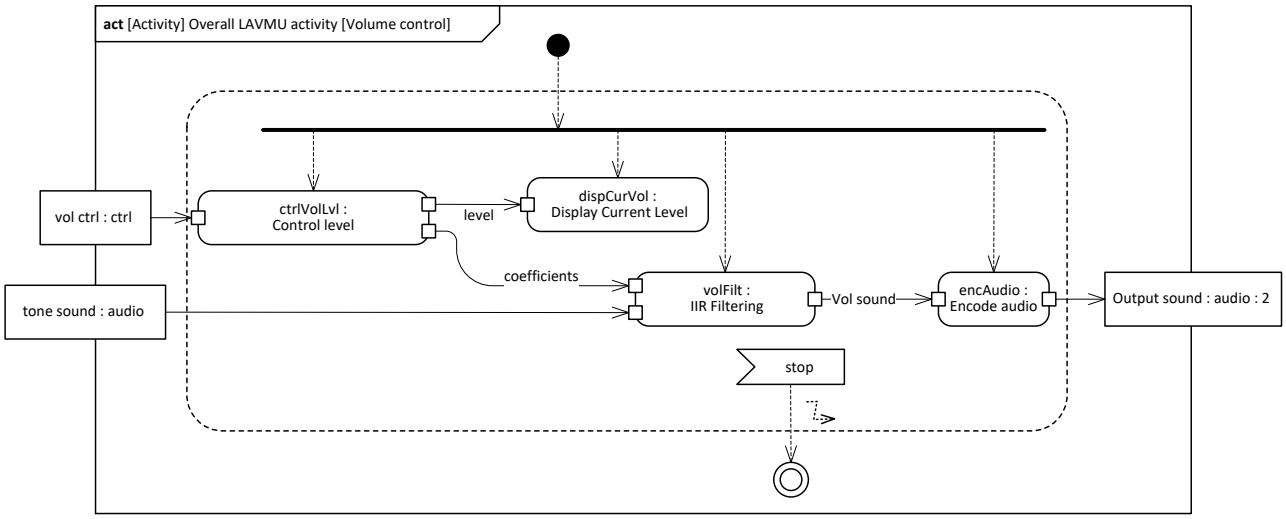

Figure 11 - Volume Control

Here the behavior of the LAVMU has been broken down inside the Volume Control activity, and shown the flow and behavior. To see the other activity breakdowns see [Figure 32 - Start Interview], [Figure 33 - Cancel Noise], [Figure 34 - Control Bass] and [Figure 35 - Control Treble] in Appendix [4.2 Behavior].

After the activity modeling one can breakdown the activities for the LAVMU using a bdd. The bdd gives a better overview of the activities in the system. A bdd of the LAVMU activities is shown on [Figure 12 - LAVMU behavior breakdown]

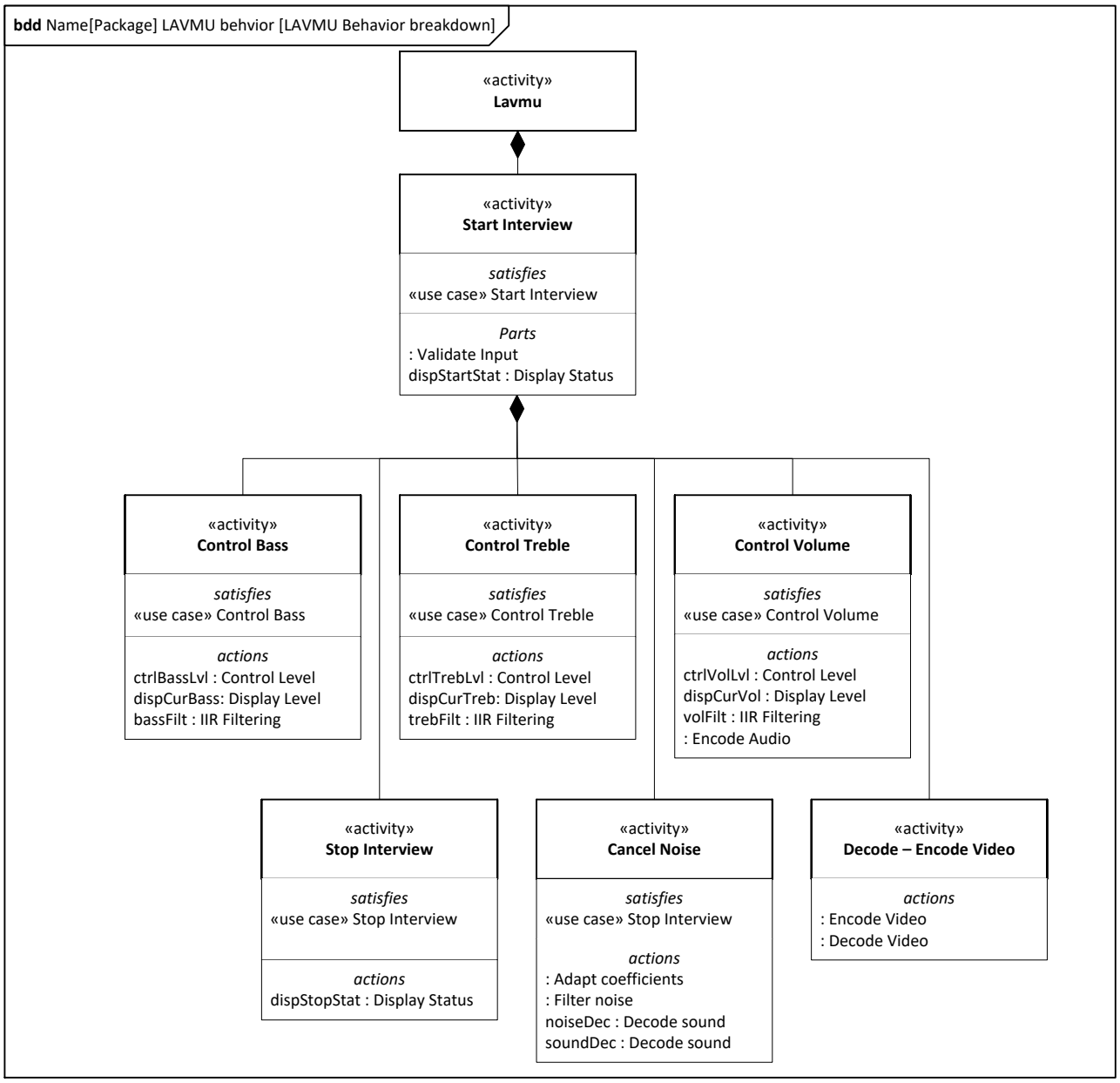

Figure 12 - LAVMU behavior breakdown

The bdd diagram shows the activities of the LAVMU. The composite association is indicating the control flow behaviors. Each block is denoted with the stereotype “activity” indicating the activity relation. Each block also has the compartment with the stereotype “actions”, which indicates all the nested activities in the activity. Last each block has the compartment stereotype “satisfies” which includes a reference to a Use Case which the activity satisfies.

3.3.4.2 State Machine

Besides the activity diagrams of the LAVMU a state machine of the interview states has been created. The state machine diagram is shown on [Figure 13 - Interview states].

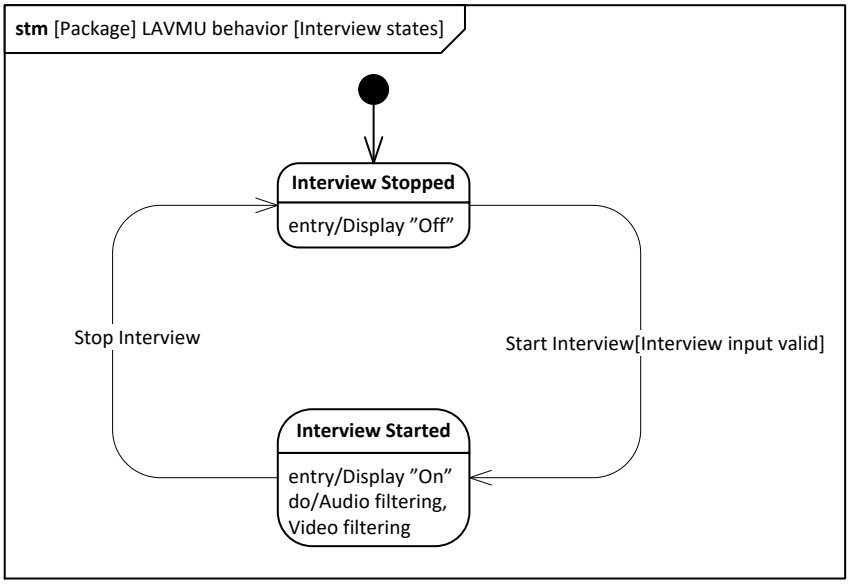

Figure 13 - Interview states

The state machine diagram shows the different states the LAVMU can be in. The LAVMU has to states "Interview stopped" and "Interview Started". The "Interview Started" state runs the audio and video filtering.

3.3.5 Modeling Structure

After the behavior modeling the modeling of the LAVMU structure can begin. The first step of the structure modeling is to breakdown the LAVMU into block, represented in a bdd.

3.3.5.1 Block definition diagram

The initial breakdown of the LAVMU is seen on [Figure 14 - Initial LAVMU breakdown].

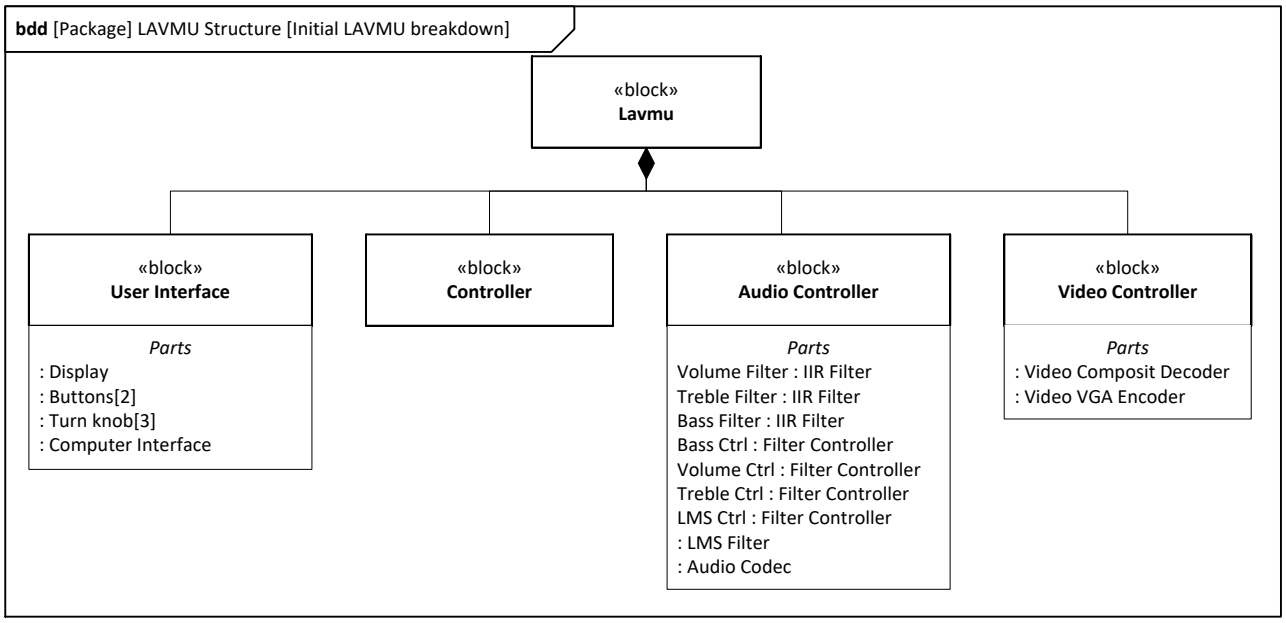

Figure 14 - Initial LAVMU breakdown

Seen on the diagram the LAVMU is divided into 4 sub-blocks, and 3 of the blocks has the compartment stereotype “parts” which indicated the components/sub-block of the block. The audio controller contains a filter block for each of the filters and a filter controller for each of the filter, which controls and creates coefficients for the filters.

From the initial block diagram one can apply the ports and flows. The flow and port apply is seen on [Figure 15 - Initial LAVMU breakdown with flows]

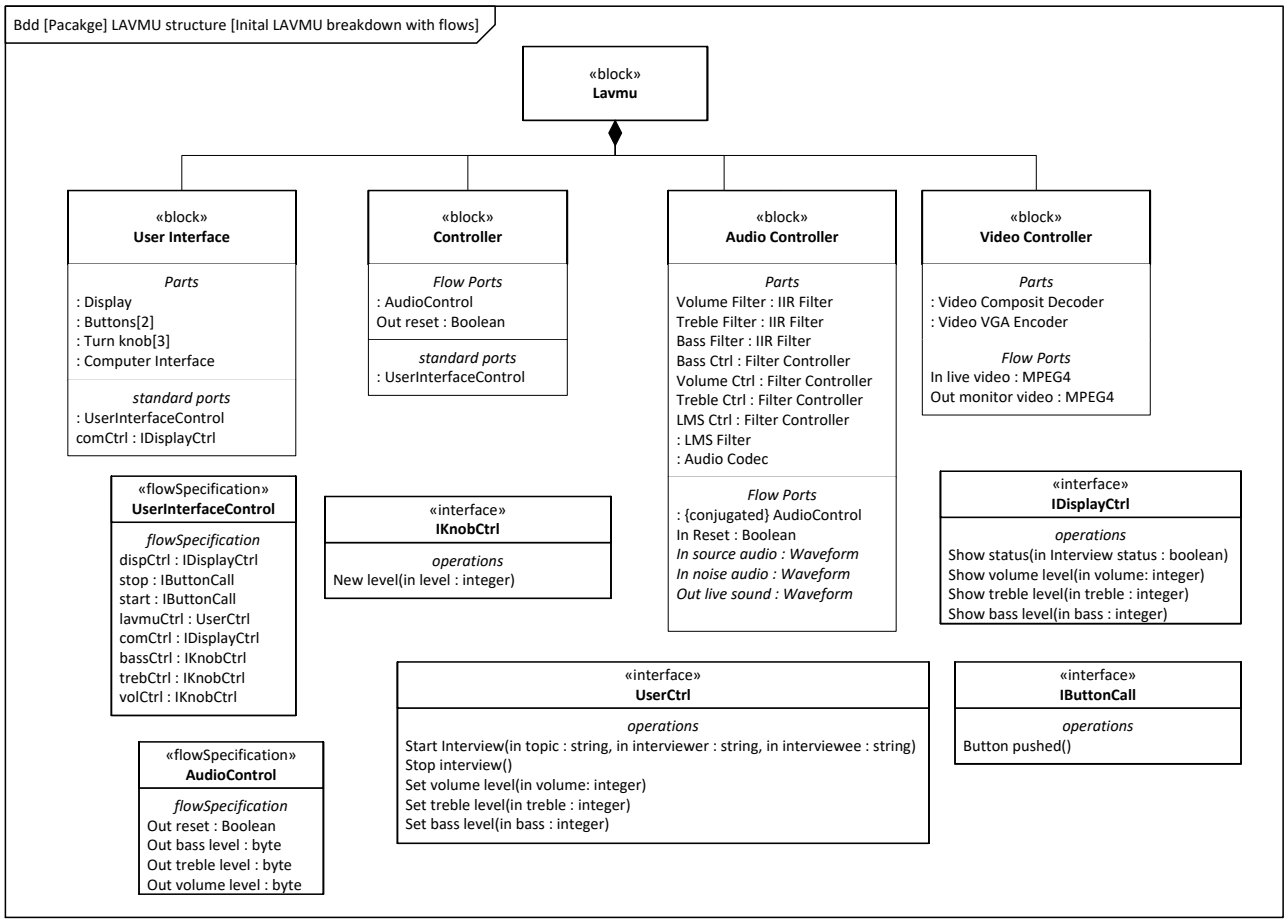

Figure 15 - Initial LAVMU breakdown with flows

The bdd shown the blocks with the two compartments “flow ports” and “standard ports” the flow port indicates the ports to the block with flow, and the standard ports indicated the interfaces to the block. Besides the blocks the specifications “interface” is added specifying the operations of the interfaces used by the blocks. There is also two “flowSpecification” the “AudioControl” indicated the flow to from block, and the “UserInterfaceControl” is uses to indicate interfaces for better overview and understanding.

Based on the bdd with the flows and interfaces an internal block diagram (ibd) can be made showing the flows between the blocks. The ibd for the LAVMU is seen on [Figure 16 - Initial LAVMU breakdown flow].

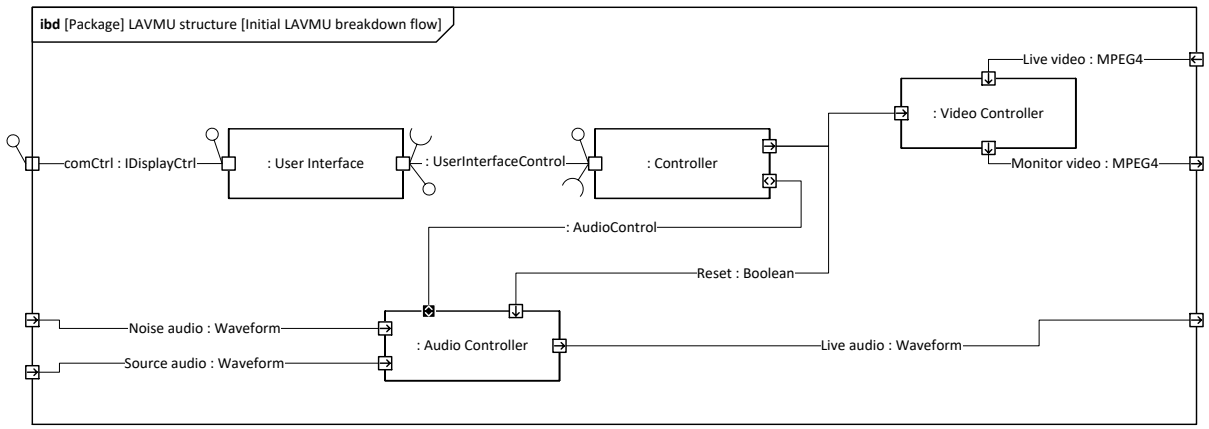

Figure 16 - Initial LAVMU breakdown flow

The ibd indicates the flows/interfaces between the block and the flow/interfaces to the external, referencing to the domain ibd seen on [Figure 5 - LAVMU Operational Domain].

An ibd for the user interface is seen on [Figure 17 - User Interface Interfaces], showing the interfaces.

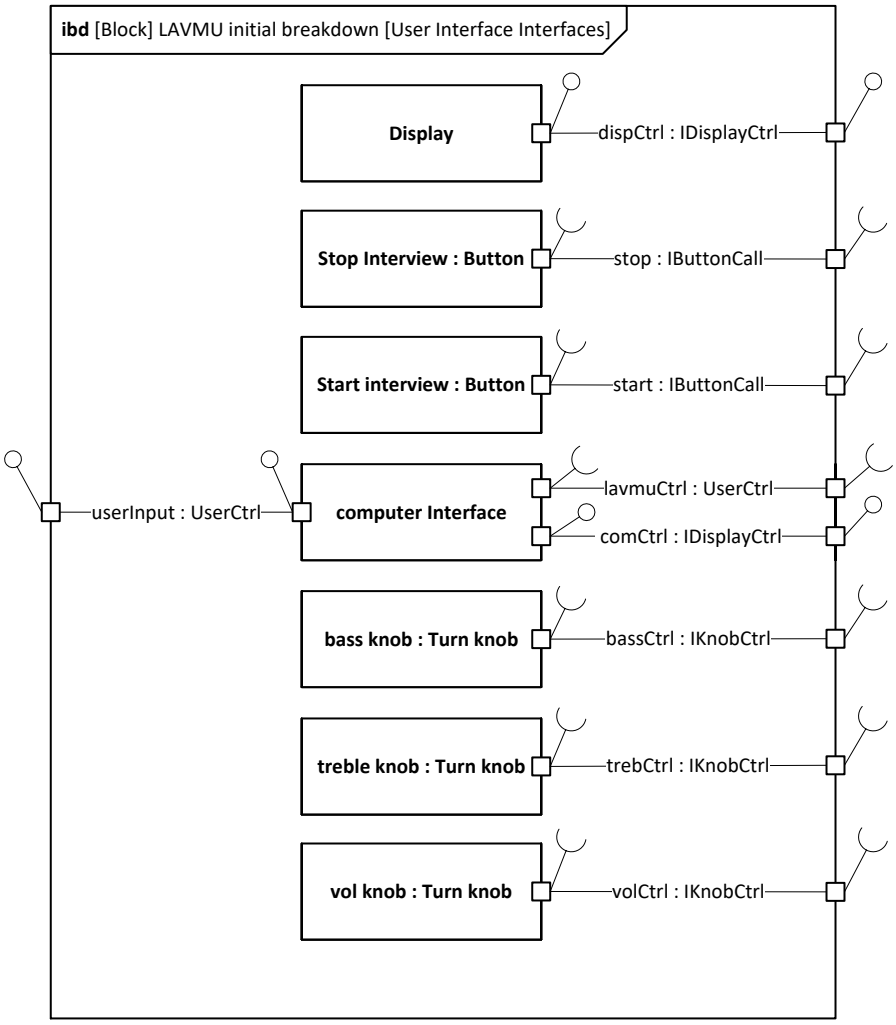

Figure 17 - User Interface Interfaces

The ibd shows the interfaces each defined in the bdd on [Figure 15 - Initial LAVMU breakdown with flows]. The diagram also shows which interfaces that is provided and which is required.

The internal block diagram for the audio controller block is seen [Figure 18 - Audio Controller flow].

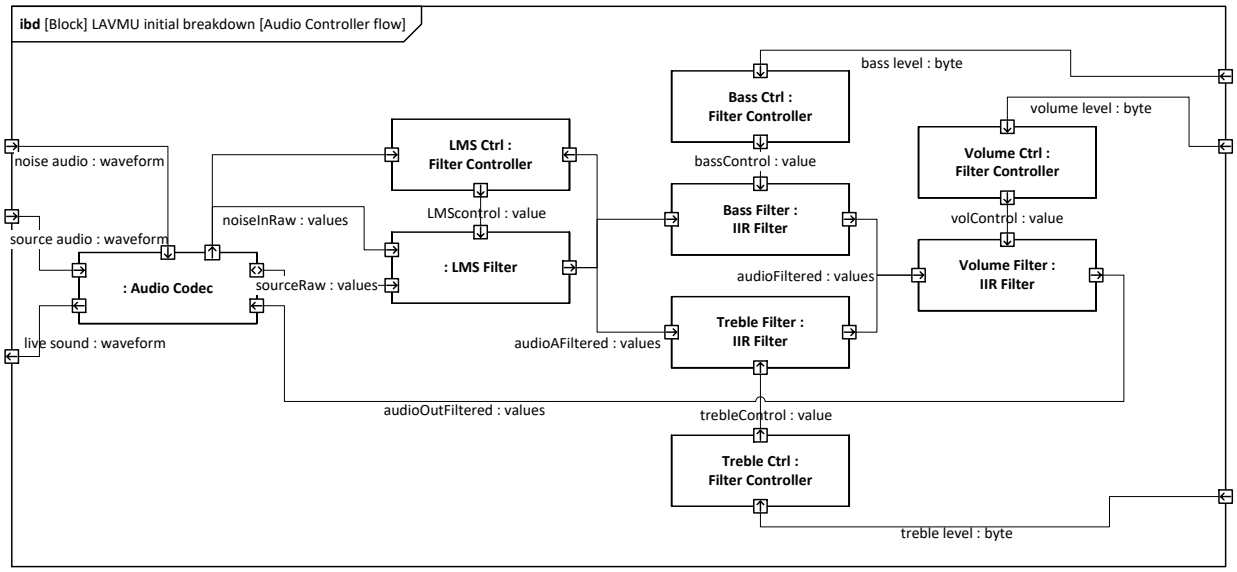

Figure 18 - Audio Controller flow

On the diagram the flows of the audio controller are shown between each of the sub-blocks.

3.3.6 Model relations and allocations

The last step of the methodology is the modeling of relations and allocations. The relation modeling includes the establishing of relations between the created diagrams, this includes the relations between the behavior, structure and requirements diagrams.

3.3.7 Relations modelling

The relations establishing has been made for the audio controller block, and is seen on [Figure 19 - Audio relationships and allocations].

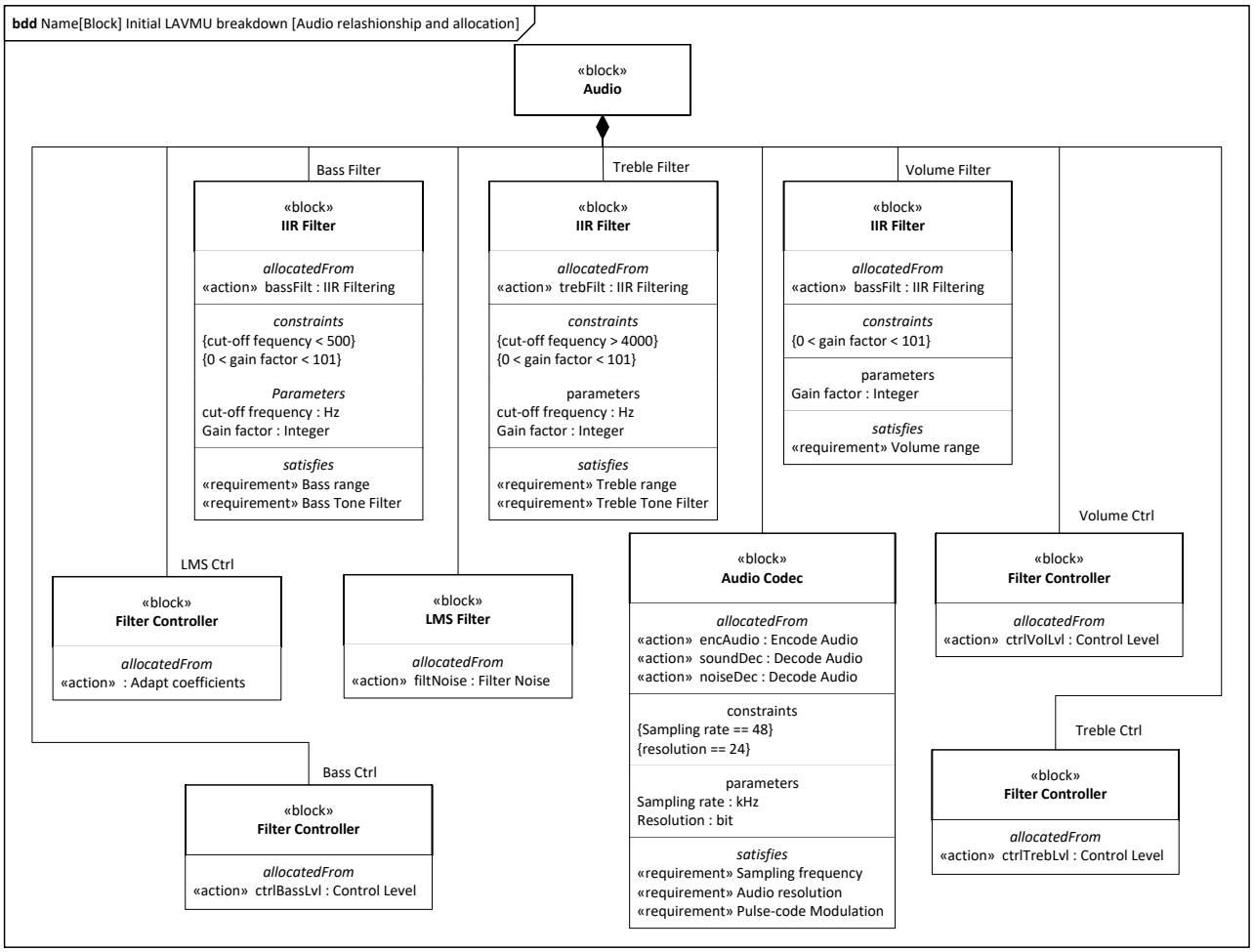

Figure 19 - Audio relationships and allocations

Seen on the diagram is the block diagram for the audio controller, each block has the compartment stereotype “allocatedFrom”, which indicates the allocation of behavior activities to the structure blocks, by doing this one can show the relation between behavior and structure of the system. The activities is allocated from the activity diagram seen on [Figure 12 - LAVMU behavior breakdown]. Besides the “allocatedFrom” compartment the compartments “constraints” and “parameters” is on some of the blocks. These compartments show some of the constraints for the block for example the cut-off frequency for the filters. Lastly the compartment stereotype “satisfies” is added to some of the blocks showing the relationship to the non-functional requirements modelled in the requirements diagram.

3.3.8 Allocation from logical to physical

Based on the modelling and the diagrams created, the last part is to allocate the logical elements to physical elements, this method appeals to the Hardware-Software Co-design principles, because one does not restrict the functionality of the system to software or hardware in the start of the project. By doing this one can also consider different design solution, and create a design space exploration.

For this project 3 types of physical solutions have been made for the understanding of the difference. The three solution is seen on [Figure 20 - Physical solution 1], [Figure 21 - Physical solution 2] and [Figure 22 - Physical solution 3].

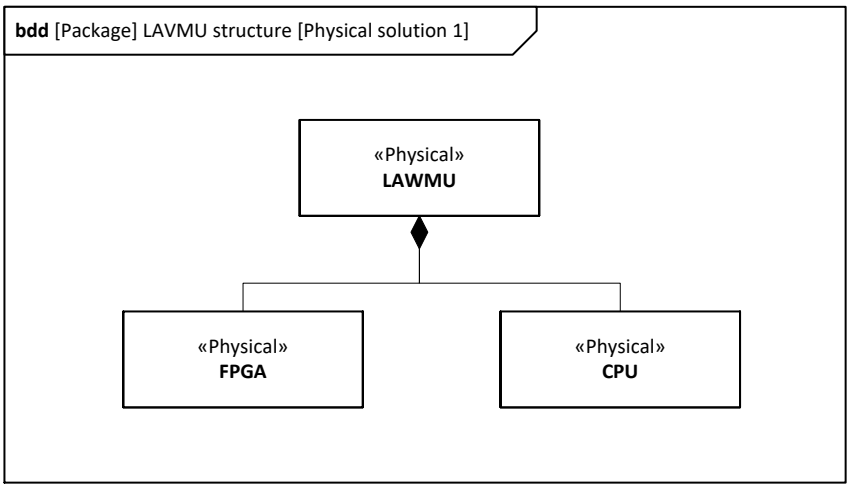

Figure 20 - Physical solution 1

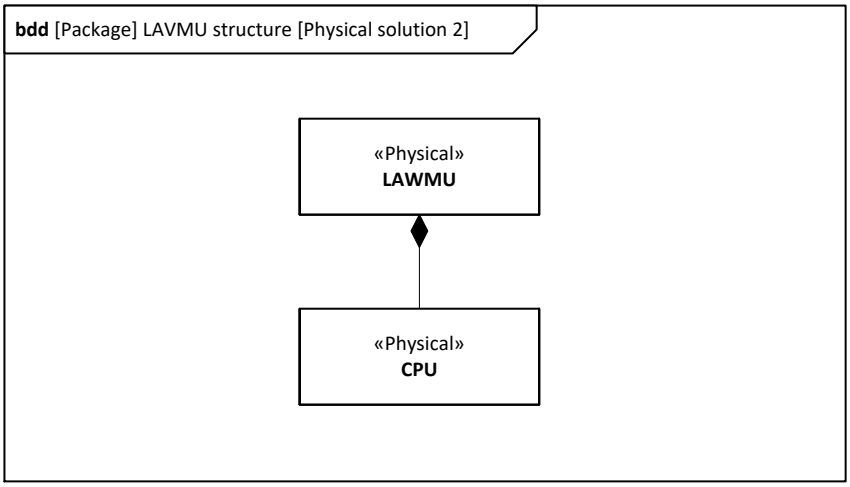

Figure 21 - Physical solution 2

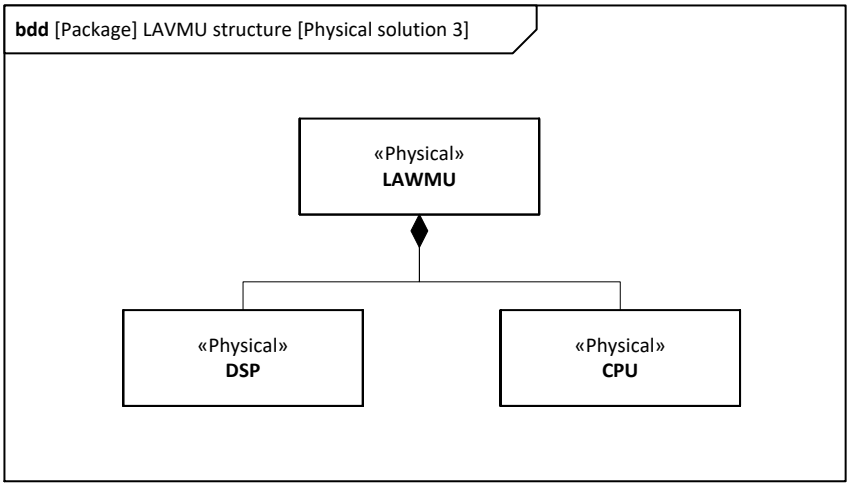

Figure 22 - Physical solution 3

The first solution is a solution consisting of a FPGA for hardware implementation and a CPU for software implementation. the second solution consist of only CPU where everything is mapped of implemented in software. The last solution consists of a DSP for the signal processing and a CPU for the software.

Based on the different architectures different types of allocation or mappings can be made. For the sake of understanding an allocation of LAVMU structure has been made for the solution 1. The allocation is shown on[Figure 23 - Allocation to solution 1 with audio all on FPGA]

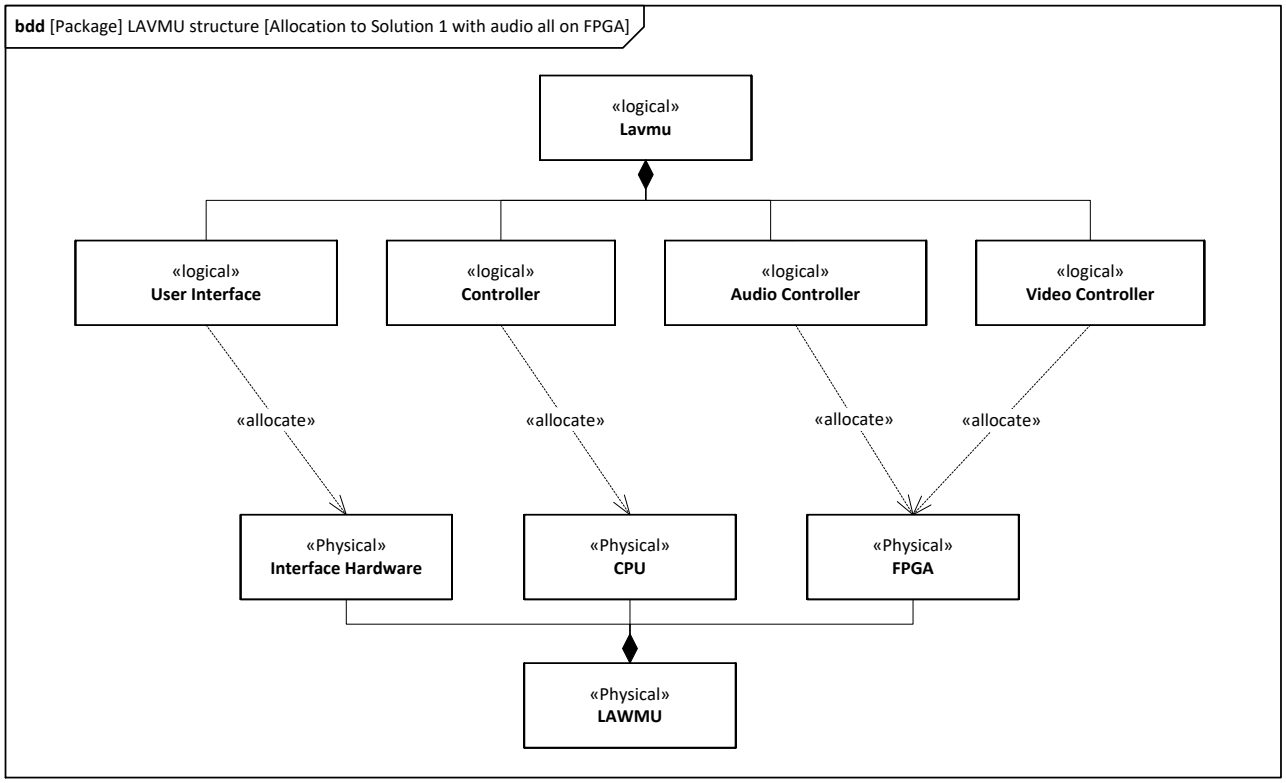

Figure 23 - Allocation to solution 1 with audio all on FPGA

Seen on the allocation diagram the logical User interface is allocated to the physical block Interface Hardware. The Interface Hardware symbolizes the hardware for the interface for example physical buttons, turn knob and LCD display. The Controller is mapped to the CPU as this has to been implemented as software. Lastly the audio controller and the video controller is mapped/allocated to the FPGA meaning implementation in hardware.

There is a lot of different allocation possibilities and even more architectures, which means that a great deal of analyzing can be done in this part of the system modelling. One could use the load balancing algorithm for this part to structure the allocation.

An example of another allocation diagram of the LAVMU logical to the physical solution 1 is seen on [Figure 24 - Allocation to solution 1 with audio split].

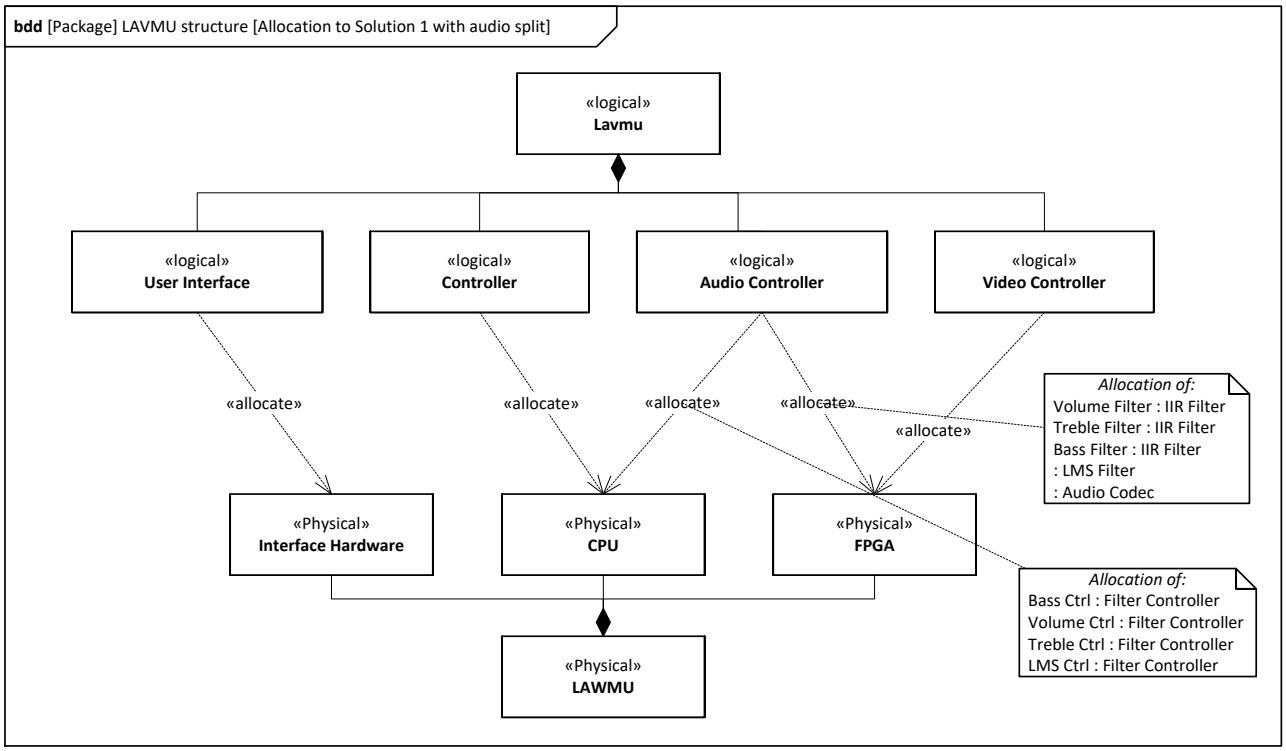

Figure 24 - Allocation to solution 1 with audio split

In this allocation diagram the audio controller sub block has been allocated to either the CPU or the FPGA. The Filters of the audio controller is allocated to the FPGA meaning implementation in hardware, the filter controllers, calculating the coefficients for the filters, of the different filters is allocated to the CPU meaning implementation in software.

3.4 Exercise 4

Description:

Examine, compile and run the LAVMU_Model (Examples) that contains a SystemC model of a digital IIR filter including the hardware interface to the simulated audio codec. Explain how SystemC has been used to model different abstraction levels (TLM or CAM) of the interface between the audio codec and IIR algorithm. What is the PView (Programmers View) module a model of? Use the [4]GTKWave viewer to analyze the interface signals behavior between the codec and IIR algorithm IP block.

SystemC has been used to model the interface between the codec and the IIR filter at a CAM (**C**ycle **A**ccurate **M**odel) abstraction level. The codecs, CodecSource and CodecSinc, are interfacing the Algo component by using ports such as *sc_in* and *sc_out*. This is in contrast to using SystemC primitive channels such as FIFOs which is a higher abstraction than using signals and ports.

The PView (Programmers View) is a component that binds the IIRFilter to the actual algorithm implementation. Through the IIRFilter component it is possible to create different kinds of IIR filters such as low-pass-, high-pass-, band-stop filter etc. This means that the IIRFilter component is used for getting the desired coefficients via the PView. From the PView it is possible to set some user parameters to the algorithm such as setting the "gain" parameter and enabling the IIR filter. In short, the PView component is a model of the application interface (software interface) to the LAVMU_Model.

The interface signals between the codec and IIR algorithm IP block is seen in [Figure 25 - Interface signals between codec and algo block]. The waveform figure has been generated using GTK wave. From here it is seen that while AudioSync is true (logical '1') sound comes out from the right channel. Likewise, once the AudioSync is false (logical '0') the sound comes out from the left channel.

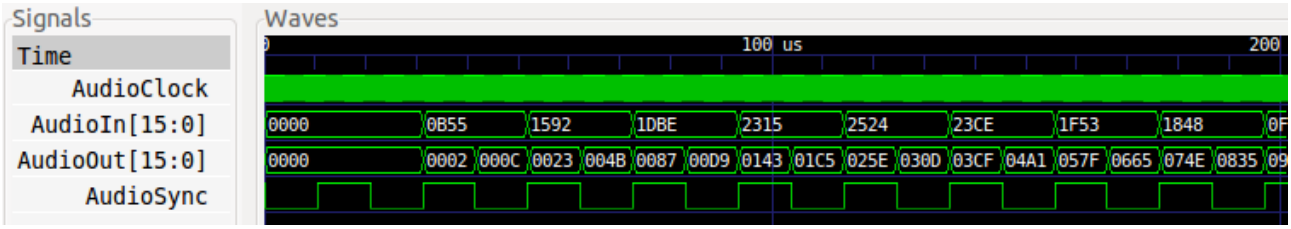

Figure 25 - Interface signals between codec and algo block

As [Figure 25 - Interface signals between codec and algo block] shows, AudioOut has different values than the AudioIn values. This is due to the fact that the output values have been modified by the IIR algorithm component.

[4] See "Guide for getting started with SystemC development"

3.5 Exercise 5

Description:

Use the **LAVMU_Model** example and modify it to model the adaptive LMS filter. The **LMSFilter** (**Examples**) contains a fixed point "C" implementation of the LMS filter. This implementation reads in signals from the "Noise.txt" and "NoiseSignal.txt" files and stores the output in a buffer. The purpose is to model the LMS filter as a hardware IP component block where the convergence factor (or step size) can be adjusted from the software running on the Soft Core processor. The **LAVMU_Model** SystemC example must be extended to read the two signal input sources from the "Noise.txt" and "NoiseSignal.txt" files and replace the IIR filter with the LMS filter.

In this exercise the LAVMU_Model example has been modified to model the adaptive LMS filter. In the modified model a noise+sound and a noise channel has been added as an input to a codec source. The codec source acts an analog to digital converter. The digital signal is then sent to the LMS filter component. The LMS filter then filters the noise from the sound and sends this further to the codec sink, which acts a digital to analog converter. The codec sink-component then outputs only sound to the speaker. In this scenario it is possible to adjust the stepsize of the LMS component from an application perspective. The flow for the modified LAVMU_Model is seen in [Figure 26 - Overview of LAVMU_Model with LMS filter].

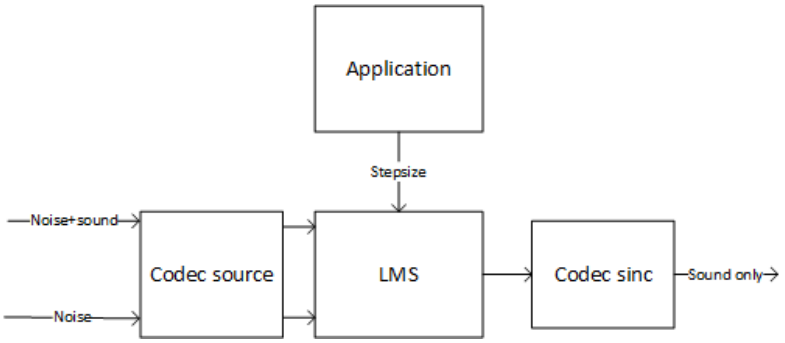

Figure 26 - Overview of LAVMU_Model with LMS filter

The CodecSource interface can be seen in [Listing 1 – CodeSource.h] and has been modified to include two Audio channels (AudioChan1 and AudioChan2), which should output the recorded noise and noise-signal. The two audio channels use the same audio synchronization signal (AudioSync). As the modified CodecSource should read from two different text files, the module now consists of two file pointers and two file paths (strings). The file paths are passed to the module at initialization by the constructor.

```
class CodecSource: public sc_module
{
public:
    sc_in_clk AudioClk;
    sc_in<bool> Reset;
    sc_out<sc_int<LMSFILTER_BITS> >  AudioInChan1;
    sc_out<sc_int<LMSFILTER_BITS> >  AudioInChan2;
    sc_out<bool> AudioSync;

    void entry();

    CodecSource(sc_module_name _n, std::string chan1, std::string chan2)
    : sc_module(_n), filename_chan1(chan1), filename_chan2(chan2)
    {
      SC_THREAD(entry);
      sensitive_pos << AudioClk;
      dont_initialize();
    }
    SC_HAS_PROCESS(CodecSource);

    ~CodecSource()
    {
      fclose(fp_chan1);
      fclose(fp_chan2);
    }
private:
    FILE *fp_chan1;
    FILE *fp_chan2;
    std::string filename_chan1;
    std::string filename_chan2;
};
```

Listing 1 – CodeSource.h

The SC_THREAD “entry()” has been modified to read from the two filepaths given at module initialization (noise and noise-signal) as seen in [Listing 2 – CodeSource.cpp – entry()]. The SourceCodec has also been modified to act as a “mono-codec”, which means that the same data is sent as the left and right channel. This is because only a single file was given for the noise and noise-signal. Another noticeable modification is that the noise and noise-signal files are comma separated which means that it is necessary to scan the file with the following string “%d,” instead of “%d”. If this is not done, the same value will be read every single time.

```
void CodecSource::entry()
{
    int tmp_val_chan1, tmp_val_chan2;
    int clk_count;
    bool rightAudio;
    fp_chan1 = fopen(filename_chan1.c_str(), "r");
    fp_chan2 = fopen(filename_chan2.c_str(), "r");

    AudioSync.write(false);
    AudioInChan1.write(0);
    AudioInChan2.write(0);
    rightAudio = true;

    cout << "CodecSource started" << endl;
    while(true)
    {
        if (Reset == true)
        {
            clk_count = 0;
        }
        else
        {
            clk_count++;
            if (clk_count == AUDIO_SYNC_CNT)
            {
                if (rightAudio)
                {
                    if (fscanf(fp_chan1, "%d,", &tmp_val_chan1) == EOF)
                    {
                        cout << "End of Input Stream: Simulation Stops"<<endl;
```

```
                        sc_stop();
                        break;
                    }
                    if (fscanf(fp_chan2, "%d,", &tmp_val_chan2) == EOF)
                    {
                        cout << "End of Input Stream: Simulation Stops"<<endl;
                        sc_stop();
                        break;
                    }
                    AudioInChan1.write(tmp_val_chan1);
                    AudioInChan2.write(tmp_val_chan2);
                    AudioSync.write(true);
                    rightAudio = false;
                }
                else
                {
                    AudioInChan1.write(tmp_val_chan1);
                    AudioInChan2.write(tmp_val_chan2);
                    AudioSync.write(false);
                    rightAudio = true;
                }
                clk_count = 0;
            }
        }
        wait();
    }
}
```

Listing 2 – CodeSource.cpp – entry()

The LMSFilter module interface can be seen in [Listing 3 - LMSFilter.h]. This module is a modification of the "algo" module where the IIR filter has been replaced by a LMS-filter. The module now has two data inputs (in_data_noise and in_data_noise_signal) and new members used for the LMS filtering. The module consists of a "SC_THREAD" for receiving, filtering and writing data. The filtering is done in a regular method (LMSFilterProcess).

```
class LMSFilter: public sc_module
{
public:
    sc_in<bool> CLK;
    sc_in<bool> Reset;
    sc_in<sc_int<LMSFILTER_BITS> >  in_data_noise;
    sc_in<sc_int<LMSFILTER_BITS> >  in_data_noise_signal;
    sc_out<sc_int<LMSFILTER_BITS> > out_data;
    sc_in<bool> AudioSync;
    sc_in<bool> AudioClk;
    sc_in<sc_int<WBUS> >  in_step_size;
private:
    sc_int<LMSFILTER_BITS> delayLine[LMSLEN] = {0};
    sc_int<LMSFILTER_BITS> weights[LMSLEN] = {0};
    sc_int<LMSFILTER_BITS> lms_step_size = 131;
    sc_int<LMSFILTER_BITS> length = LMSLEN;
    sc_int<WBUS> step_size;
    void LMSFilterProcess(sc_int<LMSFILTER_BITS> x, sc_int<LMSFILTER_BITS> d,
sc_int<LMSFILTER_BITS> *y, sc_int<LMSFILTER_BITS> *e);
    void Entry();
public:
    SC_CTOR(LMSFilter)
    {
        SC_THREAD(Entry);
        sensitive_pos << AudioClk;
    }
};
```

Listing 3 - LMSFilter.h

The entry SC_THREAD can be seen in [Listing 4 - LMSFilter.cpp - Entry()]. This method, compared to "algo", has been modified to read from both input channels (noise and noise-signal) and then pass the read values to the LMS-filter by calling the function: "LMSFilterProcess". Afterwards the filtered value is sent on the output port of the module. The output port is connected to the "CodecSink" module which simply reads

the received values and writes them to a file. The module has also been modified to work in “mono-format” where the same sample is sent on the left and right channel.

```
void LMSFilter::Entry()
{
    sc_int<LMSFILTER_BITS> tmp_data;
    sc_int<LMSFILTER_BITS> out_tmp;
    sc_int<LMSFILTER_BITS> error_tmp;
    bool AudioSync_last = false;
    cout << "LMSFILTER started" << endl;
    while(true)
    {
        if (Reset == true)
        {
             memset(delayLine, 0, LMSLEN);
             memset(weights, 0, LMSLEN);
             lms_step_size = 131;
             length = LMSLEN;
        }
        else
        {
            if (AudioSync == true && AudioSync_last == false)
            {
                LMSFilterProcess(in_data_noise.read(),
                in_data_noise_signal.read(), &out_tmp, &error_tmp);
                out_data.write(out_tmp);
            }
            if (AudioSync == false && AudioSync_last == true)
            {
                out_data.write(out_tmp);
            }
            AudioSync_last = AudioSync;
        }
        wait();
    }
}
```

Listing 4 - LMSFilter.cpp - Entry()

The LMS-filter processing can be seen in [Listing 5 - LMSfilter.cpp - LMSFilterProcess()]. It has been modified from the “LMSFilter” project. The filter has been modified to use members of the “LMSFilter” SystemC module described in [Listing 3 - LMSFilter.h]. It should also be noted that the step size is read from the bus before it is used. The step size of the LMS-filter can therefore be changed dynamically.

```
void LMSFilter::LMSFilterProcess(sc_int<LMSFILTER_BITS> x, sc_int<LMSFILTER_BITS> d,
sc_int<LMSFILTER_BITS> *y, sc_int<LMSFILTER_BITS> *e)
{
    int yn=0, wk_i;
    short k;
    sc_int<LMSFILTER_BITS> out, err, wk_s;
    std::cout << x << std::endl;
    if(in_step_size.read())
    {
        lms_step_size = static_cast< sc_int< LMSFILTER_BITS > >( in_step_size.read() );
    }
   for(k=length-1; k > 0; k--)
   {
        delayLine[k] = delayLine[k-1];
   }
    delayLine[0] = x;
    for(k=0; k < length; k++)
    {
        yn += weights[k] * delayLine[k];
    }
    out = (yn >> 15);
    err = d - out;
    for(k=0; k < length; k++)
    {
       wk_i = err*delayLine[k];
       wk_s = (wk_i >> 15); // Truncate
       wk_i = lms_step_size*wk_s;
```

```
        weights[k] += (wk_i >> 15); // Truncate
    }
    *y = out;
    *e = err;
}
```

Listing 5 - LMSfilter.cpp - LMSFilterProcess()

The modules of the LAVMU_Model have been “linked” together in a top design as seen in [Listing 6 - top.cpp – constructor]. Here it can also be seen that the CodecSource is instantiated with two file paths passed to the constructor. The signals are traced to a file which can be analyzed after the simulation has finished.

```
Top::Top(sc_module_name nm) :
    sc_module(nm),
    clock("clock", sc_time(CLK_PERIODE, SC_NS) ),
    AudioClock("AudioClock", sc_time(AUDIOCLK_PER, SC_NS) ),
    AudioInNoise("AudioInNoise"),
    AudioInNoiseSignal("AudioInNoiseSignal"),
    AudioSync("AudioSync"),
    AudioOutLMS("AudioOutLMS")
{
    pSource = new CodecSource("CodecSource", LMSFILTER_NOISE, LMSFILTER_NOISE_SIGNAL);
    pLMSFilter = new LMSFilter("LMSFilter");
    pSink = new CodecSink("CodecSink");
    pPView = new PView("PView");
    reset.write(true);
    pLMSFilter->AudioSync(AudioSync);
    pLMSFilter->in_data_noise(AudioInNoise);
    pLMSFilter->in_data_noise_signal(AudioInNoiseSignal);
    pLMSFilter->out_data(AudioOutLMS);
    pLMSFilter->in_step_size(in_step_size);
    pLMSFilter->CLK(clock);
    pLMSFilter->AudioClk(AudioClock);
    pLMSFilter->Reset(reset);
    pSource->Reset(reset);
    pSource->AudioInChan1(AudioInNoise);
    pSource->AudioInChan2(AudioInNoiseSignal);
    pSource->AudioSync(AudioSync);
    pSource->AudioClk(AudioClock);
    pSink->AudioOut(AudioOutLMS);
    pSink->AudioSync(AudioSync);
    pSink->AudioClk(AudioClock);
    pPView->reset(reset);
    pPView->set_step_size(in_step_size);
    pPView->CLK(clock);

    tracefile = sc_create_vcd_trace_file("LMSFilter_Wave");
    ……
}
Top::~Top()
{
    sc_close_vcd_trace_file(tracefile);

    cout << "Created LMSFilter_Wave.vcd" << endl;
}
```

Listing 6 - top.cpp – constructor

3.6 Exercise 6

Description:

Extend the SystemC model of the adaptive filter with control of volume and tone (Treble and Bass) by using the IIR filter provided in the **LAVMU_Model** together with the adaptive filter. Insert the volume and tone control as a new SystemC module after the adaptive filter.

In this exercise the LAVMU_Model has been extended from exercise 3.5. The principle is the same but with addition of an IIR component. In this case it is also possible to adjust the treble and bass from an application perspective. The flow for the extended LAVMU_Model is seen in [Figure 27 - Overview of LAVMU_Model with LMS filter and IIR component].

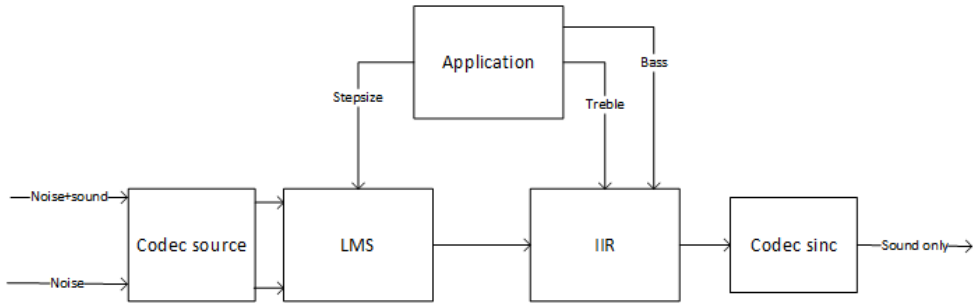

Figure 27 - Overview of LAVMU_Model with LMS filter and IIR component

The IIR filter is almost identical to the “algo” module in the LAVMU_Model example, except for the modification to a “mono-model” where only the right channel is read and then filtered. The same filtered value is also sent on the left channel as seen in [Listing 7 - algo.cpp - Entry()].

```
void algo::Entry()
{
    sc_int<ALGO_BITS> tmp_data;
    bool AudioSync_last = false;
    cout << "Algo started" << endl;
    while(true)
    {
        if (Reset == true)
        {
            m_x1 = 0;
            m_x2 = 0;
            m_y1 = 0;
            m_y2 = 0;
        }
        else
        {
            if (AudioSync == true && AudioSync_last == false)
            {
                // Right channel
                m_sample = in_data.read();
                AlgoProcess();
                out_data.write(m_sample);
            }
            if (AudioSync == false && AudioSync_last == true)
            {
                out_data.write(m_sample);
            }
            AudioSync_last = AudioSync;
        }
        wait();
    }
}
```

Listing 7 - algo.cpp - Entry()

The IIR filter has been connected to the LMS-filter and the CodecSink by modified the “top.cpp”. The additions to the file can be seen in [Listing 8 - top.ccp - IIR filter additions].

```
    pIIR = new algo("IIR");
    pPView->set_gain(in_reg1);
    pPView->set_user(in_reg2),
    pPView->get_peak(out_reg2);
    pPView->set_coeff0(in_coef0);
    pPView->set_coeff1(in_coef1);
    pPView->set_coeff2(in_coef2);
    pPView->set_coeff3(in_coef3);
    pPView->set_coeff4(in_coef4);
    pIIR->AudioClk(AudioClock);
    pIIR->AudioSync(AudioSync);
    pIIR->CLK(clock);
    pIIR->Reset(reset);
    pIIR->in_data(AudioOutLMS);
    pIIR->out_data(AudioOutIIR);
    pIIR->in_coef0(in_coef0);
    pIIR->in_coef1(in_coef1);
    pIIR->in_coef2(in_coef2);
    pIIR->in_coef3(in_coef3);
    pIIR->in_coef4(in_coef4);
    pIIR->in_reg1(in_reg1);
    pIIR->in_reg2(in_reg2);
    pIIR->out_reg2(out_reg2);
```

Listing 8 - top.ccp - IIR filter additions

It is now possible to change the volume by setting the gain of the filter by the function: set_gain(). The bass and treble can be modified by changed the IIR coefficients by the functions: set_coeff0/1/2/3/4(). To be able to change both treble and bass, two IIR filter with respectively a low and high pass filter can be combined sequentially. This has not been included in this model for simplicity.

A snippet of the recorded wave file can be seen in [Figure 28 - GTKWave form]. Here it can be seen that the output changes for every sample as the noise is subtracted from the noise-signal. The change of stepsize and filter coefficients of the LMS-filter and IIR-filter are stimulated by the “PView” module which simulates the programmers view.

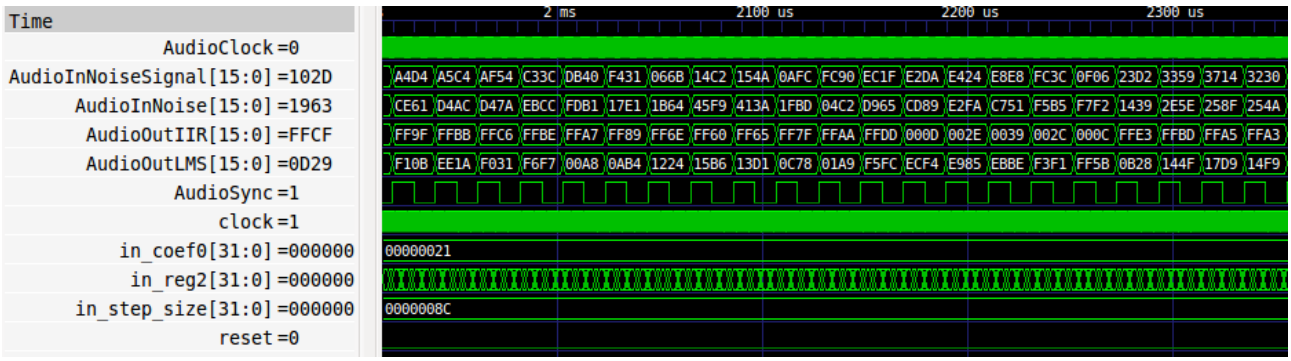

Figure 28 - GTKWave form

4 Appendix

The following section consists of an appendix of diagrams for the system. Through the earlier sections of the document these diagrams are referenced.

4.1 Requirement Diagrams

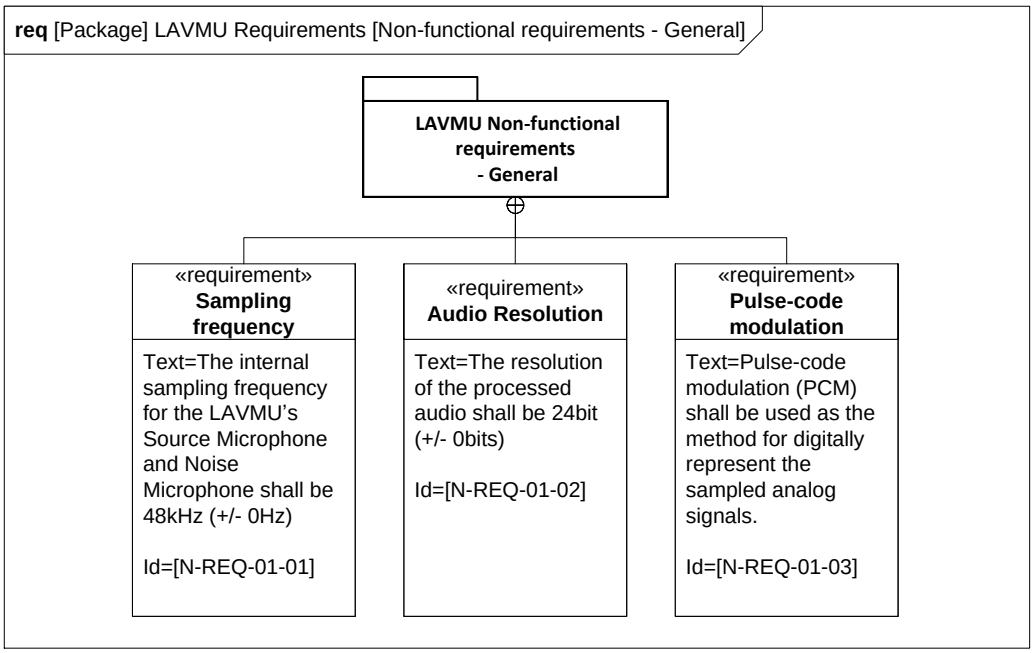

Figure 29 - Non-functional requirements - General

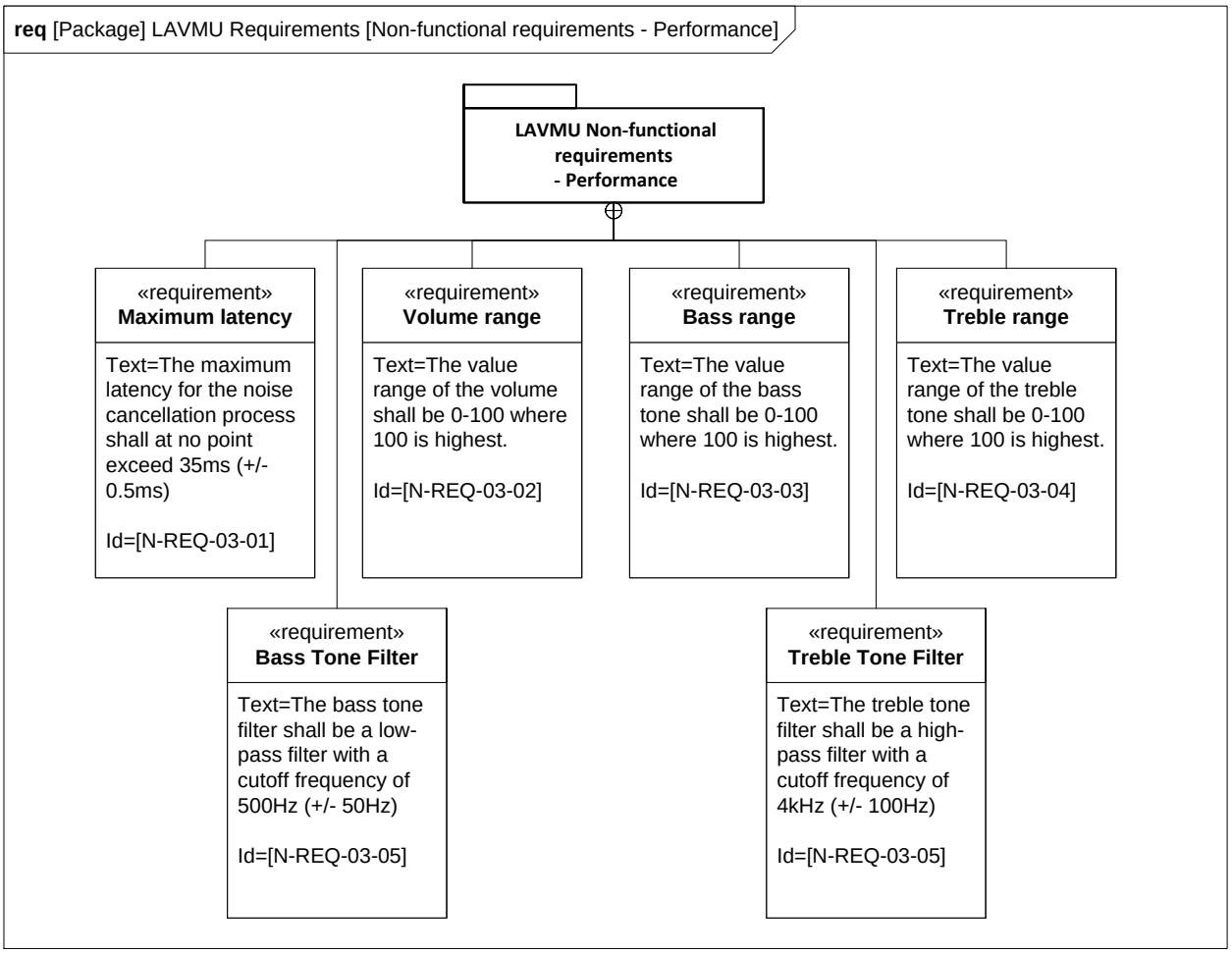

Figure 30 - Non-functional requirements - Performance

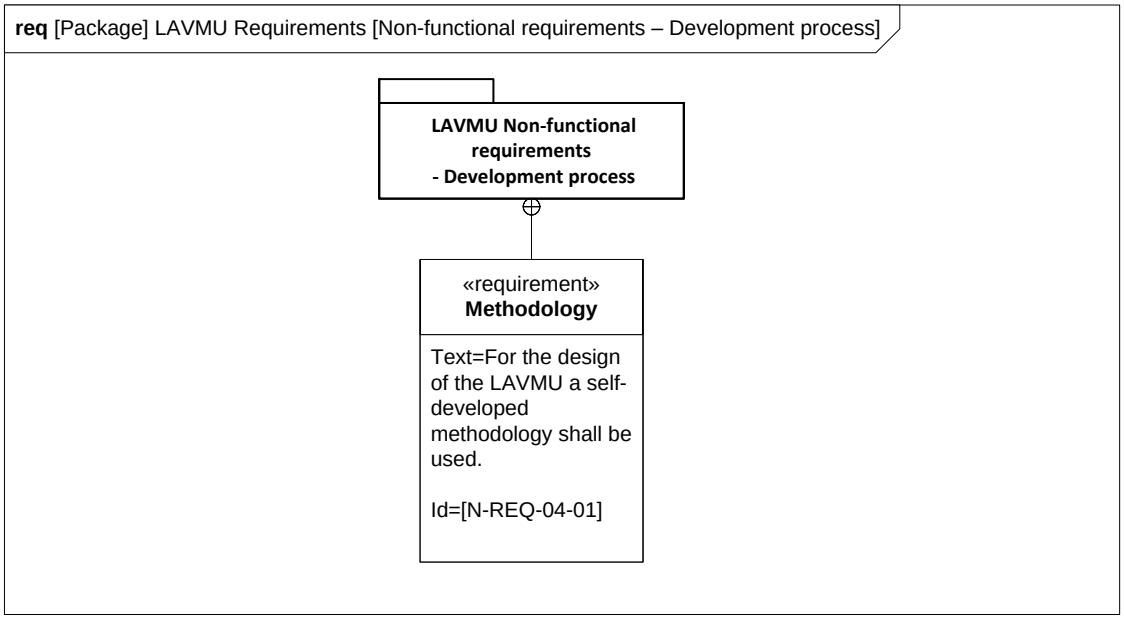

Figure 31 - Non-functional requirements - Development process

4.2 Behavior

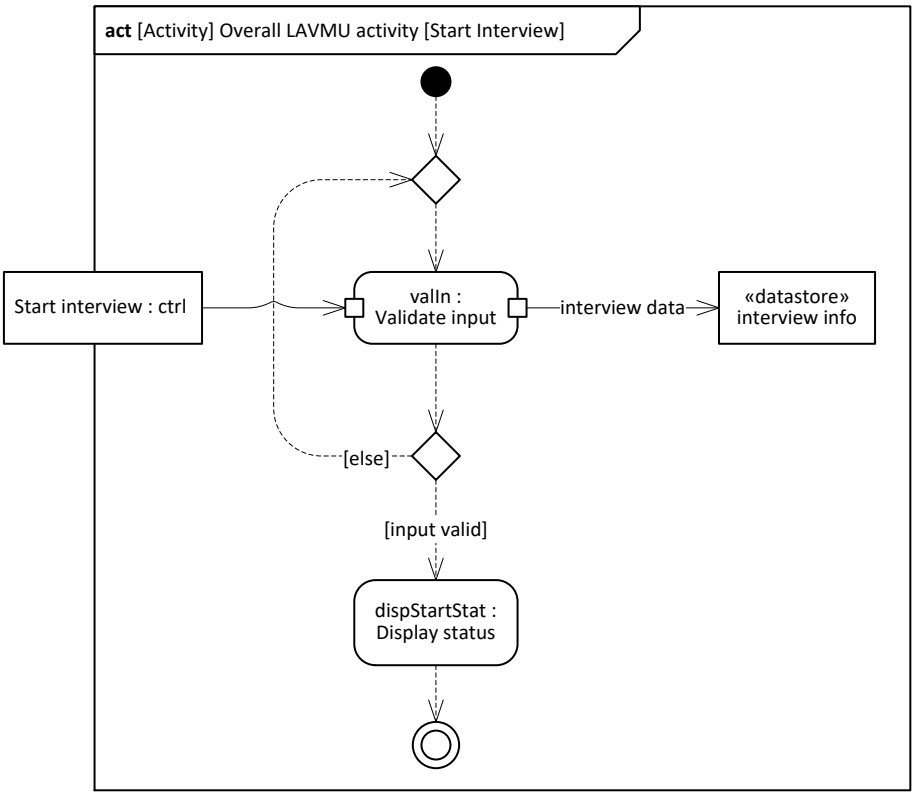

Figure 32 - Start Interview

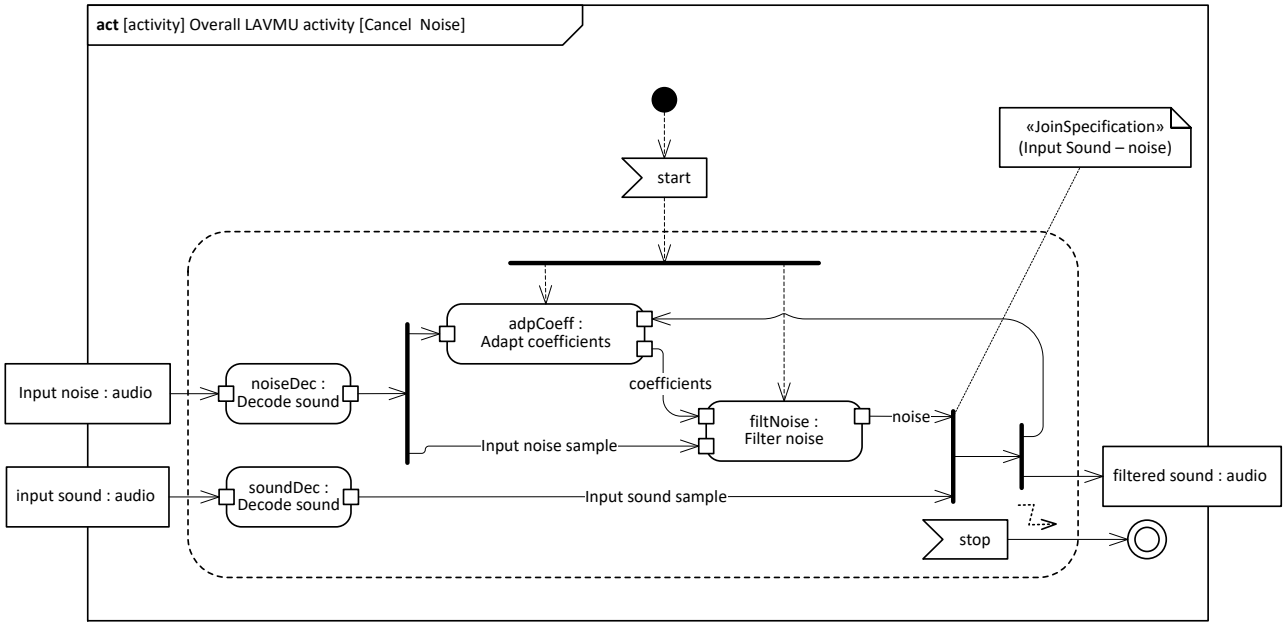

Figure 33 - Cancel Noise

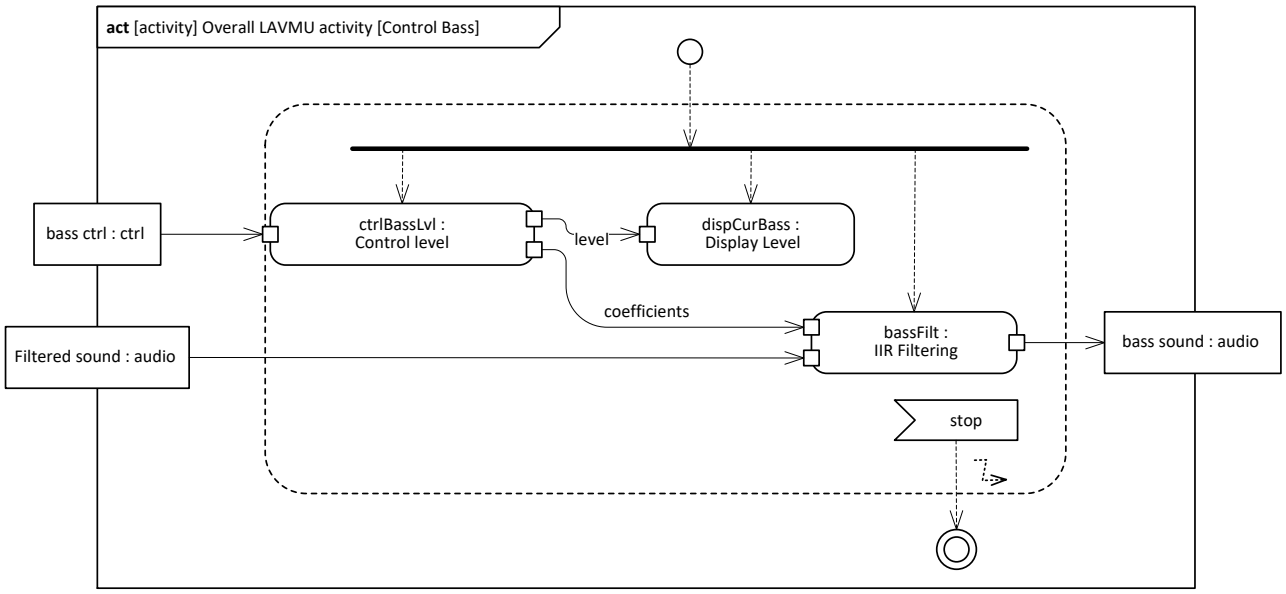

Figure 34 - Control Bass

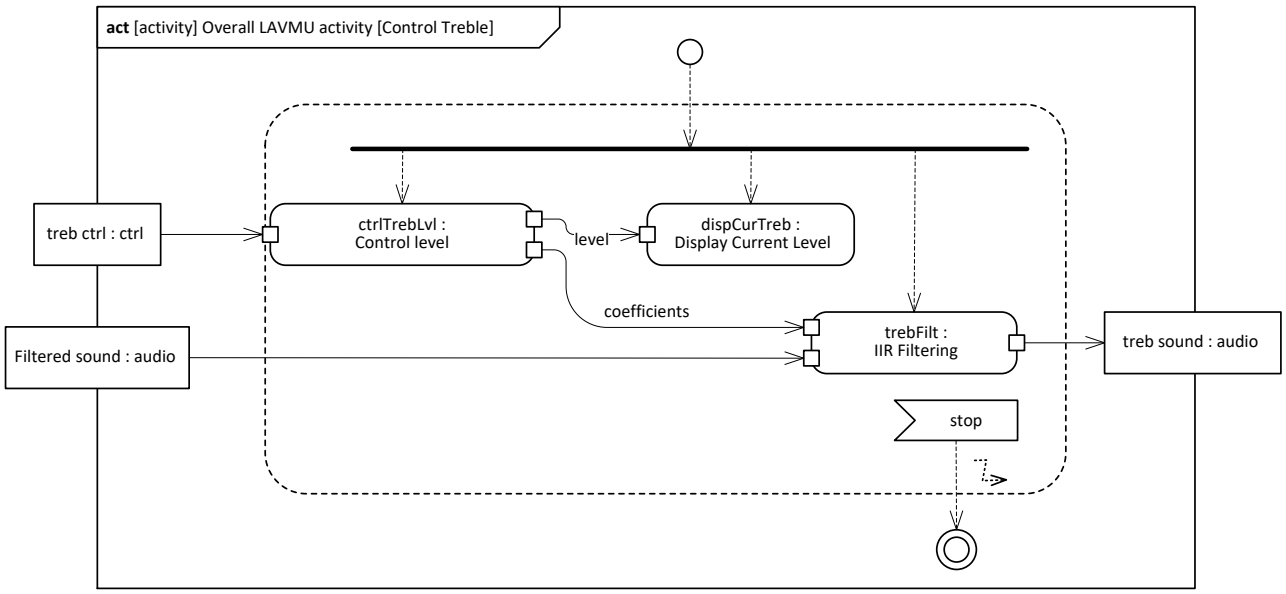

Figure 35 - Control Treble