UNIVERSITÄT PADERBORN

Die Universität der Informationsgesellschaft

Faculty for Electrical Engineering, Computer Science and Mathematics
Department of Computer Science
Computer Engineering Group

PG Soundgates

Project Plan

Author:
Martin Boonk
Caius Cioran
Lukas Funke
Hendrik Hangmann
Andrey Pines
Thorbjörn Posewsky
Gunnar Wuellrich

Supervisor:
Prof. Dr. Marco Platzner
Jun.-Prof. Dr. Christian Plessl
Dipl.-Inf. Andreas Agne

July 17, 2013

Contents

1 Introduction

1.1 About this document

"'Soundgates Interactive Music Synthesis on FPGAs"' is a project initiated by the Computer Engineering Group of the University of Paderborn, aiming to synthesise music on modern FPGAs. Furthermore, users should be able to interact with the system in a playful way by manipulating the synthesised music with advanced sensors such as the acceleration sensor in modern smartphones.

This document is the project plan of our project group. Not only should this document serve as a basis for the later evaluation and grading, but also as a reference for our group during the main working phase running from ??.2013 to ??.2014.

This first chapter introduces some concepts and system we plan to use to achieve the goals outlined in Chapter **??**. Chapter **??** covers systems similar to what we are going to create, escpecially the software called MAX. Finally Chapters **??** and **??** describe how the group will organize itself and the groups' milestones.

1.2 Definitions

The following table displays some definitions, that will be used throughout this document.

Term	Definition
Composite Component	Definition
Component	A basic building block to generate music ??Haben wir uns hier auf Block als Bezeichnung geeinight? Component eher im Sinne von Softwarekomponente
Editor	The Editor is used to create a patch out of components to generate synthesizable code which can be put on a FPGA
FPGA	Field Programmable Gate Array
Patch	The entire system which consists of Components and Composite Components. A set of single Components can build a new Component
Port	The interface from one Component to another one
Simulation	The developed patch is played through the PC speakers

1.3 Generative music

1.3.1 Introduction

As pointed out in [?], there are many cultures where musicical experience is defined by people performing and people perceiving music. The only way to slightly exert influence on the performer's music is by cheering, shouting, etc. on a concert. The gap between performer and perceiver reaches its peak in the context of recorded music like CDs or MP3 files, where is no chance of manipulate the music. Actually, there is a rising trend for interacticing with music on your own or with a familiar social partner, like in the video game "'Guitar Hero"' [?, ?]. Even without any musical knowledges or talent, it is more and more possible to interact, manipulate or even create music. Generative music combines the opportunities of making music without having knowledges of how to play an instrument and explicitly exert influence on music you hear. An early an popular example for generative music was Mozart's musical dice game. Given a set of small sections of music, it was randomly chosen which part was played next.

1.3.2 Approaches

Genarative music (or algorithmic music) can be devided into the following subcategories [?].

Linguistic/Structural

Already existing songs are scanned in the first place and afterwards recomposed randomly. Therefore, a grammer is created where the production rules are made of sequent pairs of notes. The generation of the new song is achieved by executing this production rules randomly.

Creative/Procedural

There are different processes/procedures which play predefined music. This processes have an arbitrary order and starting time. A popular is Mozart's musical dice game.

Biological Emergent

Music is generated by biological phenomenons, i. e. by differing parameters within an ecology, for example wind-chimes.

Interactive/Behavioural

The interactive/behavioural generative music approach results from processes without discernable musical inputs. It is a question of synthesized music and recorded and filtered samples at the most. The music generation is fully controlled by user input and interaction.

This is the initial point on what this project is based: synthesizing music along with a highly interactive interface to exert influence on the music.

1.4 Possible User Interactions

In order to make the system interactive there need to be one or more interfaces, over which the user can take influence on the music. This could happen via midi keyboard or other electronic instruments. But with the purpose of providing an easy, playful access to music generation without knowledges of an instruments, we are focussing on advanced sensors. Sensors like Microsoft's Kinect systems can easily track e. g. the user's hand. An intuitive way of changing system's parameters is to measure and transmit the height of the hand. Other possible sensors can be found in nearly every modern smartphone, e. g. the acceleration sensors.

1.5 Introduction to sound synthesis

Sound synthesis is the artificial creation of sound. Cleverly built synthesizers can mimic many real instruments and of course are able to create sounds with no real world counterpart.

The first synthesizers consisted of seperate analog electronic blocks. A user could connect these blocks with cables in many different ways to generate, modify and filter a signal, depending on the order and fashion of connected blocks. As with many other systems, digital components replaced the analog ones over time. The general approach, however, remained the same. Users do not necessarily plug in cables anymore but conceptually one still connects separate building blocks to create the sound that he wants.

The most basic digital synthesizer would be the one shown in Figure 1.1. In this example an oscillator generates a waveform, for example a basic sine wave. Sampled values of the waveform are fed into an Digital-Analog-Converter (DAC) which converts them to an analog signal which can then be played on a speaker. While it is not very pleasing to listen to an unmodified sine wave, it is already some artifical and therefore synthesized sound.

1.5.1 Frequency and notes

The previous example only used a single sine wave. With a small addition this setup can already be used to play notes. The only thing that needs to be added would be a way to control the frequency of the sine wave since the value of a note is determined only by the frequency of the sound wave. An A4 for example corresponds to exactly 440 Hz. This is true regardless of the actual shape of the waveform, be it a sine wave, a square wave or something completely different.

Thinking of different instruments this circumstance quickly becomes obvious. While both a violin and a piano can play the same specific note it will sound differently on

both instruments. This is due to the fact, that the sound waves those instruments create have a different shape.

1.5.2 Waveform generation

Creating a specific base shape of our sound can be done in several ways, each with it's advantages and disadvantages and a spectrum of sounds that can be created. Of course, many of these approaches are not mutually exclusive but can be combined with each other.

Additive synthesis

In additive synthesis one uses multiple waveform generators, usually sine waves, adds up their outputs and normalizes the result. For example, adding a sine wave with frequency $2F$ to a sine wave with frequency F would result in a different sound color but still with an overall frequency of F and therefore the same note.

A sound designer has very tight control over the produced sound with additive synthesis. According to the fourier theorem it is possible to approximate every periodic function by the combination of sines. Therefore, a real world sound could be analyzed and rebuilt this way. However this approach may need a lot of separate oscillators and might therefore become unfeasible to handle for very complex sounds with a broad frequency spectrum.

Subtractive synthesis

A subtractive synthesis starts with waveforms like a sawtooth, square, triangle or even periodic noise. Contrary to a bare sine wave, these patterns have a wide frequency spectrum. By feeding these waves into diferent filters (section 1.5.3) the broad spectrum can then be reduced again.

Compared to additive synthesis, it is cheaper to create rich sounds with subtractive synthesis but it sacrifices the very tight control over every single frequency.

Frequency/amplitude modulation

Another possibility is to modify the frequency or amplitude of one waveform by the value of another. This can create relatively rich results with relatively little effort. However, this approach usually creates hearable vibrato or tremolo effects.

Sampling

Any result of the aforementioned approaches, as well as real sound can of course be saved and reused. It is relatively cheap and efficient to store small sound samples. Instead of calculating the values of a sine wave, they could just be taken from a lookup table. Or instead of trying to synthesize a violin, a real one could be recorded and sampled later for further use.

Such samples are very inflexibile of course. It is possible to modify the frequency by changing the playback speed and it is possible to process them further, but they cannot serve as the only basis for a synthesizer that should be able to create arbitrary sound.

1.5.3 Filters and further modulation

- High pass / low pass - ADSR - echo /chor / delay

Figure 1.1: Easiest way to generate sound on a digital system

1.6 Employed systems

1.6.1 Xilinx Platform Studio

The overall hardware system is implemented in VHDL, using the Xilinx Platform Studio (XPS), which offers the MicroBlaze Softcore Processor.

1.6.2 ReconOS

ReconOS is an Operating System which can be run on a softcore CPU on a FPGA. Through it's support for the Linux kernel it is possible to write applications which consist of Hard- and Software threads. Therefore the software threads can be used

to communicate via sockets with sensors and hand over these values to the hardware threads, which are responsible for generating sounds.

1.6.3 Eclipse/GMF

We avoid building a new graphical editor from ground up since this is an error prone process. Instead we rely on the Model Driven Software Development process for graphical editors which is provided by the GMF framework. Therefore we can generate an editor by specifing the Meta-Model. The graphical surface with our components are connections are the result. Additionally it is neccessary to provide functions for every component, so it will be possible to simulate the generated output.

2 Goals

Our goal is to develop a graphical editor for music generation. With it's help it will be possible to create patches, test their output in a simulation and produce a netlist to generate that sound on a FPGA. A patch can be build with different components like wave generators and filters. In order to create specific sounds, the components have to be connected. The exported netlist can be put on a FPGA so the user of the synthesizer (e.g. a musician) can connect a MIDI device or specific sensors to the FPGA to modify input values at runtime.

To describe the functions, which we want to develop, in detail, we provide use case diagrams.

2.1 Editor for the designer

The designer can add components to the patch and remove them from the patch. He can connect components by drawing links between their ports. A component can be an atomic component like a sound generator, filter etc. The atomic components are stored within XML-libraries. Some of these components have static properties, which the designer can modify (e.g. the value of a constant-block). The other kind of components are composite components. The designer can add a composite component to the patch and fill it on his own with other components and links between them. He can add ports to a composite component and define their direction and constraints. We want to make it possible to export finished composite components as XML-files and to import existing ones to the editor such that different designers can exchange their components. Furthermore, the designer must define the interfaces of his patch. The interfaces are the points where the end-user interacts with the system. These can be MIDI-inpus, sensor values etc. It a patch is finished, it can be validated. The program tests if all constraints are fulfilled (e.g. if all ports are connected and so on). The designer can export his patch as XML-file and build it, which means that he can generate VHDL-code out of the XML-file.

2.2 Simulator

We want to develop a software simulator for a patch such the designer and the user can test the sound of the patch before putting it on a FPGA. A designer must be able to import a patch to the simulator and to start the simulation. Afterwars he is able to pause or even stop the simulation. An optional feature we would like to implement is the possibility to save the output of a simulation on the HDD.

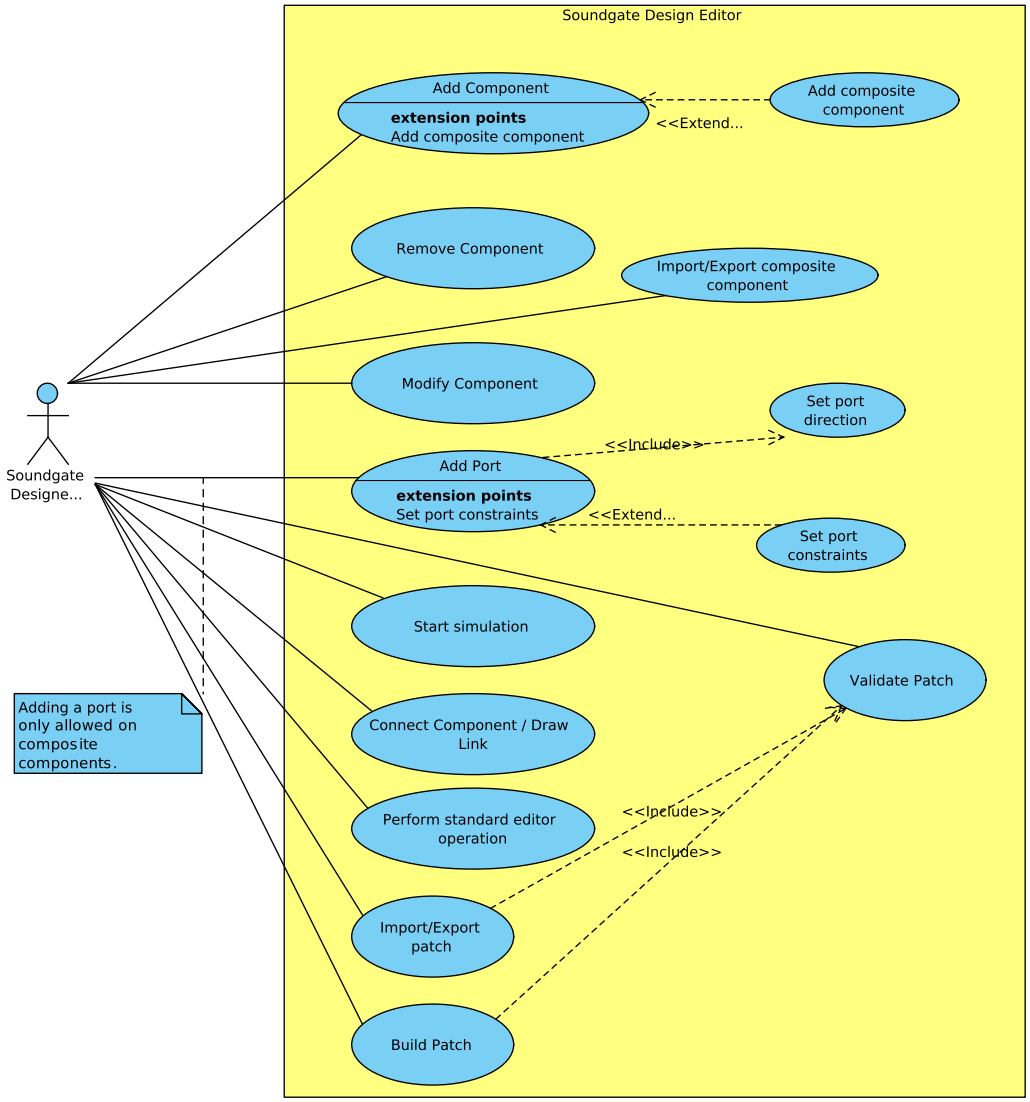

Figure 2.1: Use case diagram of the soundgate designer (editor)

2.3 End-user product

As mentioned before, a patch created in the editor can be exported as a XML-file which is used to generate VHDL-code. This VHDL-code is synthesized and put on a FPGA. The end-user (e.g. a musician or a performer) maps sensor values and other inputs to the interfaces defined by the designer (see ..). He starts the session by pushing a button. During the session he performs some sensor actions or actions with other devices to create input values for the system.

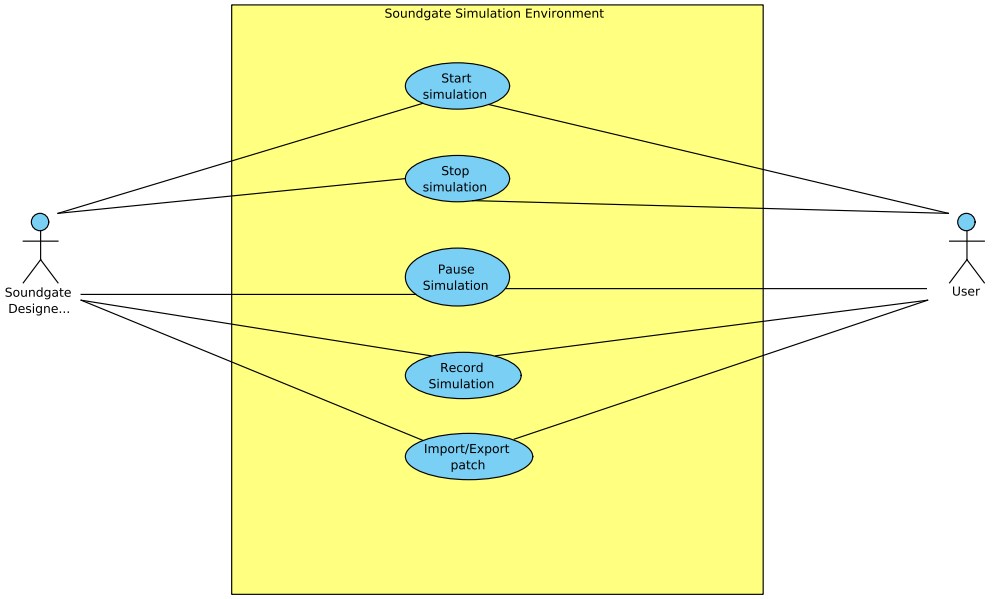

Figure 2.2: Use case diagram of the soundgate simulation environment

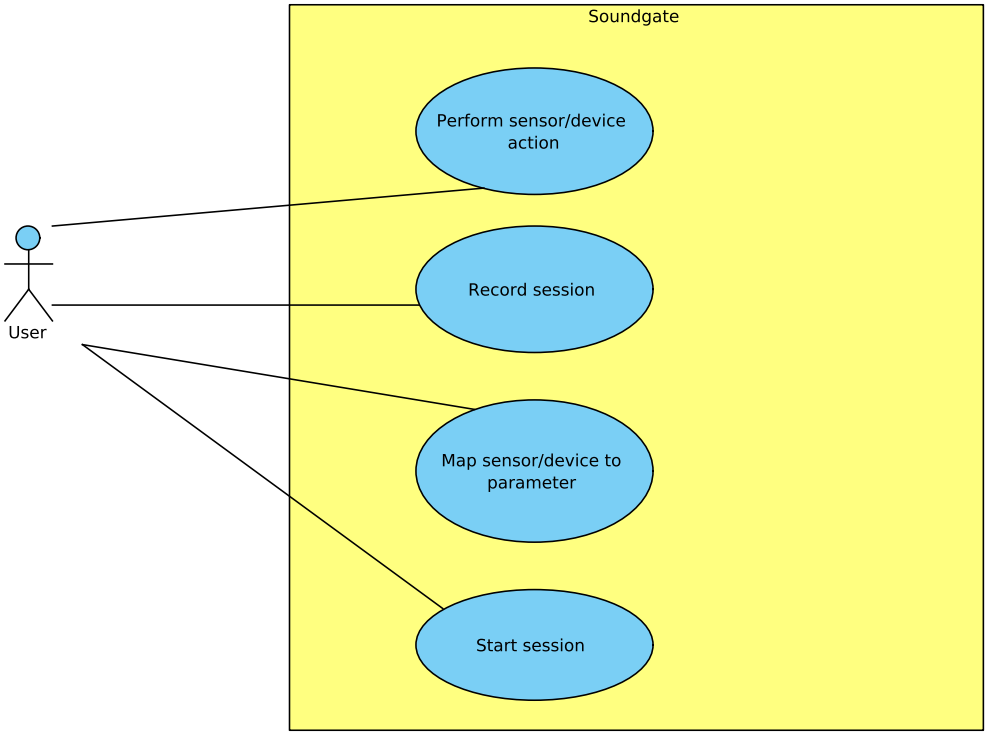

Figure 2.3: Use case diagram of the soundgate user interface

2.3.1 Input

We will provide different ways of modfiying the patch at runtime through sensors. The first approach uses a device as a sensors whereas the other one will use the body of the musician.

Mobile Device

Today's mobile phones are full of different sensors which can easily be accessed with a small application. The Android SDK offers three different groups of sensors, namely:

- **Motion Sensors**
  This group contains accelerometers, gravity sensors, gyroscope, and rotational vector sensors (TODO see ref to Android page).
- **Environmental Sensors**
  With their help it is possible to check the air temperature illumination and humidity.
- **Position Sensors**
  Orientation sensors and magnometers.

However environmental sensors are not suitable since modfiying their values can not be done quickly enough. The other sensors are perfectly suited for changing values since they are modified simply by moving the phone and changing it's orientation. We will develop an Android Application because the Android OS is widespred. On the one hand the sensors have to be read by the application and on the other hand these values have to be forwared to the simulation environment. This can be done with Sockets to send these values over a network connection.

3D depth camera

A 3D camera like the Microsoft Kinect system is able to calculate the depth of an image. Therefore it is possible to track the movement of a person by tracking different joints, which have to be calibrated at the beginning. So it is possible to connect parametes of a sound component i.e. to the altitude of the left arm. This method abstracts the sensors so it feels like if the person himself is possible to change the sound.

Furthermore the end-user should be able to use MIDI devices.

2.4 Open Sound Control

- message based communication protocol
- signals are created by hardware (midi keyboard) or software and send to device
- not transportprotocol dependent. We will use ethernet/wifi and sockets to communicate. They are provided by ReconOS/linux on the FPGA

- OSC message
  - 1 Byte contains length of packet and the rest data. length = 2^x
  - datatypes: int32, float32, OSCString, OSCTimetag, OSCBlob

2.5 Component library

We plan to provide following components:

- Generators:
  - Sinus
  - Sawtooth
- Filters:
  - Ramp
  - Low pass
  - Delay
- Arithmetic:
  - Addition
  - Multiplication
  - Equals
- Mixer
- Sources:
  - Constant
- Sinks:
  - Sound output
- MIDI:
  - MIDI note to frequency converter
  - MIDI scheduler

3 Related Work

3.1 Cycling '74 Max

Our editor is in some way similar to other music composition applications especially cycling74's MAX. Also it offers a drawing canvas, which can be filled with connected components. At any time the patch, which of the combination of every used component, can be simulated to hear the generated sound. However, this covers only the simulation part of our entire system. These components can be exported as a combination of VHDL code and prepared netlists in order to create a bitfile to generate that sound live on a FPGA. Furthermore it is possible to change the behavior of some components at runtime my manipulating their input values through different sensors.

3.2 Reactable

Another interactive music generator is Reactable, which is an innovative electronic musical instrument table where the user can deploy cubes on (also available as an smartphone app). Each cube represents a either waveforms, loops, filters or other synthesizer components. These components can also be combined and linked. Similar to that, our editor can arrange different synthesizer components in order to generate music. Furthermore, we adapt the interactive input possibilities of the Reactable app. It is possible to connect nearly each component to an acceleration sensor or gyroscope of the phone the Reactable app is running on. This sensor measures the phone's tilt angle and thereby for instance can effect the music's pitch.

4 Organization

Basically we split our group in two teams. The one team is the Software Team and the other the Hardware team. The affilation to one of these teams is based on the previous experiences of the specific person. – 1 or 2 manager for special tasks, timing/deadlines and team coordination?

4.1 Agile inspired development

The following reasons inspired the agile-like development process that we discussed in the last section:

- Testing is an integrated part of the whole development process, which improves the quality of the project.
- Incremental development of the features by the iterative nature of agile software development
- Regular release of a working project code at the end of each iteration.
- Easy adaptation for any change in circumstances.
- Self organizing development process provides more team member involvement and friendly environment.
- Less end to end project overhead is required.

4.2 Meetings

Since the beginning of our project group we have a mandatory team meeting every week. Right now it is scheduled each Monday at six o'clock.

The mandatory meeting is attended by all members of the group. In this meeting the current status of all tasks and the whole project is discussed. Further tasks for the next week are developed by discussion among the team members. The members also track the progress towards the approaching milestone on each of this mandatory meetings. Tasks are created and the members are assigned to the tasks in this meeting. Tasks assignment is either done due to volunteering or by asking a member. A member with knowledge about a given task by previous experience or a seminar topic is more likely assigned to that task.

– Additionally time slot, when?

Furthermore a single person or a subgroup consisting of two or maximal three members that is working on a certain task is given a time slot where he/they can work on the provided computers and FPGAs. The time slot must be kept by the person or the subgroup in order to ensure project progress and a controlled work flow. Each member or subgroup has at least one time slot per week such that every group member is working each work on the project.

4.3 Tools

4.4 Member role and specialization

4.5 Milestone presentation

5 Workplan

5.1 Overview

The whole project is divided into multiple milestones. Each milestones consists of set of tasks. One milestone must be completed before antoher milestone can start.

5.2 Milestones

Five milestones are planned, where each milestones should be completed in approximately five to six weeks. Depending on the set of tasks in each milestone, some may consume more time, some less.

5.2.1 Milestone 1 - Prototyping infrastructure/environment

The fundamental infrastructure to achive the project objectives is prototyped in the first milestone. This involves three basic tasks:

1. prototype a graphical design environment (editor) for audio-generation as well as audio-processing blocks
2. prototype a simulation environment to simulate a patch created with the editor
3. prototype a hardware infrastructure such as digital-to-analog (dac) controller, (rotary) switches and an interface to a external sensors (Reconos)

The goal of this milestone is not to have a complete working toolflow, but to have a quick start of the development.

In detail the basic tasks can be further subdevided:

1. Editor prototype
   - Implementation of a basic meta-model
   - Prototye editor:
     - Create/delete atomic blocks
     - Connect atomic blocks
     - Save and load a model (patch)
     - Serialize the patch to a human readable format (e.g XML)

- Create at least two atomic blocks (a sound generator block and an audio output)

2. Simulator prototype
   - Evaluate JAVA-Sound API
   - Design a framework
   - Proof-of-concept implementation
     - Import a patch-file
     - Simulate system input
3. Hardware prototype
   - Setup Reconos environment
   - Prototype I/O expension interface
     - DAC-Controler should work
     - Rotary switch should work
   - Setup communication between a host pc and reconos
   - Encapsulate access to the XILINX FPGA-Toolchain (e.g. Eclipse plugin)

5.2.2 Milestone 2 - Topic of milestone

In the second milestone more featues will be added to the whole system. The goal of this milestone is to move the system to a working toolflow.

1. Editor continued
   - Add the ability to create and import user-defined atomic blocks
   - Add the ability to create composite blocks
2. Basic sensor interaction
   - empty
3. Codegeneration
   - Transform a patch to a syntesizable HDL description
   - Import of existing HW-Framework components (e.g. DAC, switches)
4. Implement a set of basic atomic blocks
   - wave generator (saw, square, sine)
   - multiplication
   - addition
   - ramp generator

5.2.3 Milestone 3 - Topic of milestone

1. Implement additional audio processing blocks
    - Arithmetic/Logic-Blocks (like "'Equals"')
    - Routing objects
    - midi processing blocks
    - Low/high-pass filters
    - delay
    - ...