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 15, 2013

Contents

1 Introduction

1.1 Introduction

— Von Hendrik, mit anderem Kram mergen —
"'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, one or more users should use this interactive system, by means of manipulating the synthesised music with advanced sensors such as the acceleration sensor in modern smartphones.

1.2 About this document

This document is the project plan of the project group "Soundgates" at the University of Paderborn. It introduces the topic of our project group in Chapter 1 and the goals we want to achieve in Chapter **??**. Chapter **??** covers systems similar to what we are going to create, escpecially the software called MAX. Chapters **??** and **??** describe how the group will organize itself and the groups' milestones.

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

1.3 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

1.4 Project outline

- Creating an editor for synthesizers. - Components of the synthesizer implemented in hardware (and software for simulation purpose) - Implementation of generative music concepts ——- Warum wollen wir das eigentlich tun? Was ist die Motivation dahinter (zumal es das aus Oslo ja aschon in Software gibt)

1.5 Introduction to sound synthsis

- Artifical generation of sound - Generation of basic waveforms - More complex and rich patterns by methods like additive/subtractive/... synth - Further addition of filters etc

- Originated from analog synthesizers, nowadays mostly digital. Software for general purpose PCs exist

- Building patches: job of a sound designer, rather than a musician

1.6 Generative music

— Stichpunkte Gunnar: —

- Creatioin of music depending on user interaction - Users do not need to be musicians - Playful approach to making music - tightly connected to sound synthesis

1.6.1 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.

1.6.2 Approaches

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

Linguistic/Structural

Recomposition of analyzed songs, using grammars or stochastic processes.

Creative/Procedural

Randomly reorder pre-defined musical parts and play them.

Biological Emergent

(Simulation of) biological influences.

Interactive/Behavioural

No instruments - Recorded and filtered samples at the most. Synthesized music. Music generation fully controlled by user input/interaction. (see related work)

1.7 Possible User Interactions

Microsoft Kinect, acceleration sensor of a modern smartphone, etc.

1.8 Employed systems

1.8.1 VHDL

- Hardware components implemented in VHDL

1.8.2 ReconOS

- Developed at the University of Paderborn - Operating system running on a softcore alongside our VHDL components - Especially useful for integration of external devices (generative music)

1.8.3 Eclipse/GMF

- Synthesizer Editor developed as an Eclipse Plugin. - Will use GMF

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.

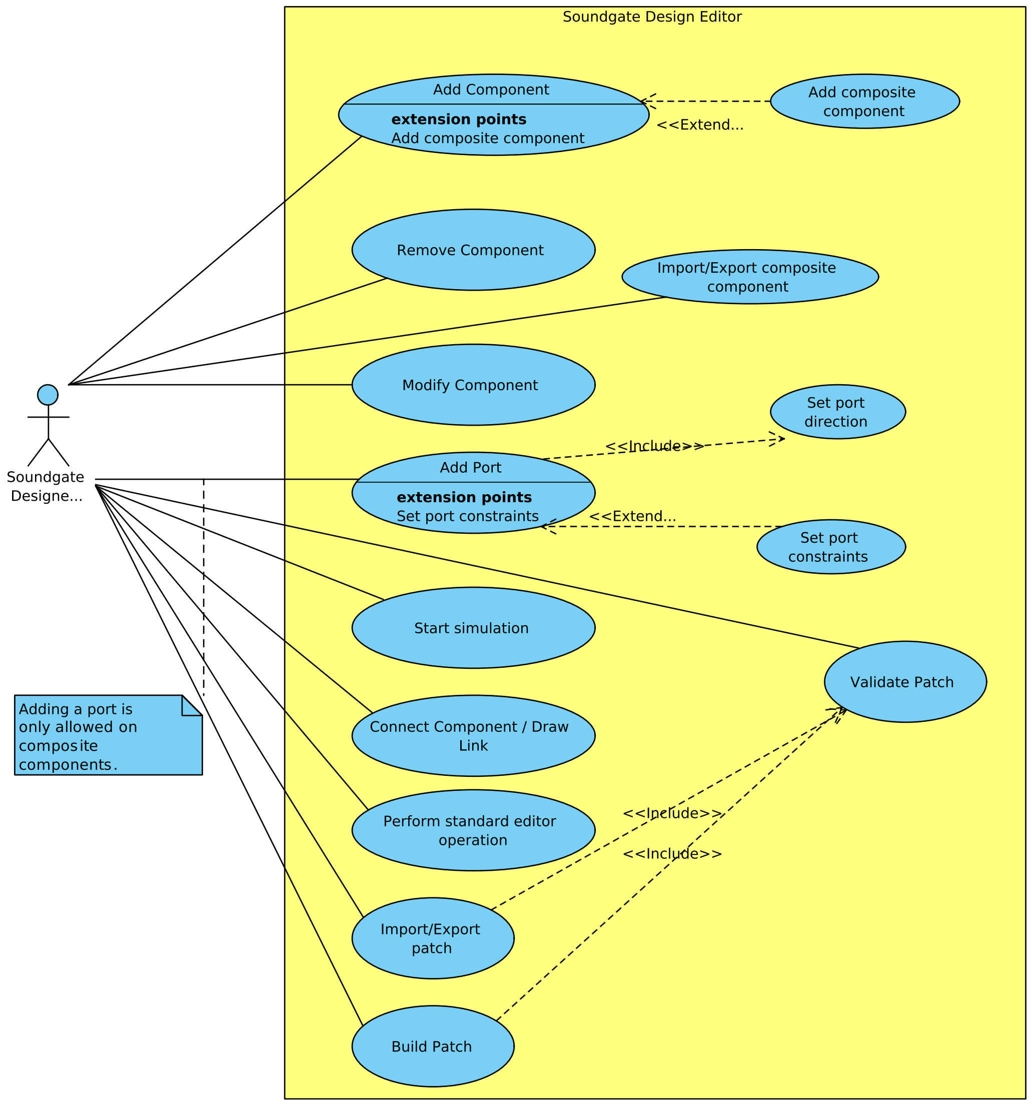

Figure 2.1: Use case diagram of the soundgate designer (editor)

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.

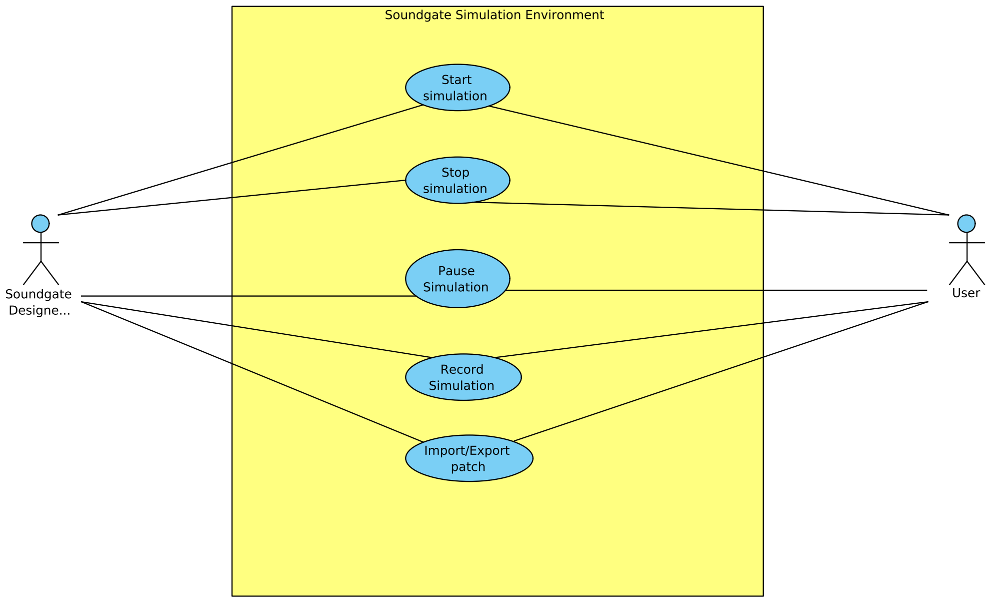

Figure 2.2: Use case diagram of the soundgate simulation environment

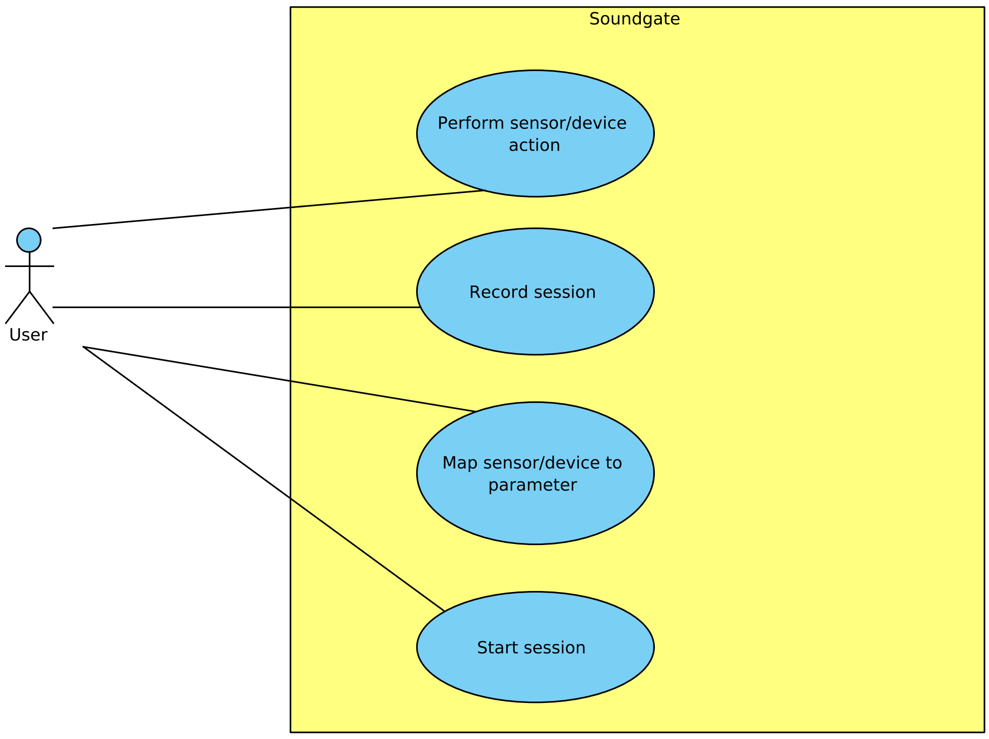

Figure 2.3: Use case diagram of the soundgate user interface

2.3.1 Input

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.

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

2.4 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.
Aufbau, Elements, User Interaction

3.2 Reactable

Aufbau, Elements, User Interaction

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
    - ...