TAGG
The Awesome Guitar Game

CSEE 4840
Embedded System Design
Spring 2012

Academic Supervisor:
Professor Stephen A. Edwards

Imré Frotier de la Messelière (imf2108)

TAGG
The Awesome Guitar Game

CSEE 4840
Embedded System Design
Spring 2012

Academic Supervisor:
Professor Stephen A. Edwards

Imré Frotier de la Messelière (imf2108)

Overview of the project

- This work is based on the "Guitar Hero" game series.
- The user handles a game guitar.
- Once the game starts, the user tries to match the required notes with the guitar.
- His score increases each time he presses the correct key.

Objectives of the project

Objectives of the project

- Fully functional game

- Hardware conception to create the Game Guitar

- Software conception for the core of the game, using MATLAB, PYTHON, VHDL and C:
    - Detect the input of the game guitar
    - Extract the beats of a song
    - Ask for the correct key presses, based on the beats
    - Analyze the correctness of the key presses of the player
    - Keep track of the score

Project Design

Project Design

Architectual Design

Total schematics of the VHDL, C, MATLAB and PYTHON modules

Display of the beats

The beats are stored and accessed in the C program.

They mark the boundaries for the time a user has the right to try and press the correct key.

The required key to press are displayed on the Console of the NIOS2 coding environment.

Getting the user inputs from the Game Guitar

The user inputs are received by using interruptions.

Once a key has been pressed, the VHDL code sends an interruption to the NIOS2 C code.

There is one type of interruption per guitar key.

The key that had been pressed is determined by the interruption ID.

Keeping track of the score

The score is handled by the NIOS2 C program.

The score is being increased each time the correct key has been pressed during the valid time lapse.

The score is displayed continuously in the NIOS2 Console.

Timing Design

Display of the beats

- Stored inside the C code
- Displayed based on their embedded timing data

Storing and playing the music

- SDRAM
- Flash Memory
- Music in the background (raw implementation in the current state of this project)

Architectural Design

Total schematics of the VHDL, C, MATLAB and PYTHON modules

Display of the beats

The beats are stored and accessed in the C program.

They mark the boundaries for the time a user has the right to try and press the correct key.

The required key to press are displayed on the Console of the NIOS2 coding environment.

Getting the user inputs from the Game Guitar

The user inputs are received by using interruptions.

Once a key has been pressed, the VHDL code sends an interruption to the NIOS2 C code.

There is one type of interruption per guitar key.

The key that had been pressed is determined by the interruption ID.

Keeping track of the score

The score is handled by the NIOS2 C program.

The score is being increased each time the correct key has been pressed during the valid time lapse.

The score is displayed continuously in the NIOS2 Console.

Total schematics of the VHDL, C, MATLAB and PYTHON modules

- beatavg.m
- beat.m
- calclistftrs.m
- chromagram_E.m
- chromagram_IF.m
- chromagram_P.m
- chrombeatftrs.m
- chromnorm.m
- chrompwr.m
- chromrot.m
- chromxcorr.m
- coverDistMxLists.m
- coverFtrExLists.m
- coverTestLists.m

MATLAB

- distmatrixwrite.m
- fexist.m
- fft2chromamx.m
- fft2melmx.m
- history-bragg-autoco.m
- history-golddust-xcorr.m
- hz2octs.m
- ifgram.m
- ifptrack.m
- listfileread.m
- listfilewrite.m
- localmax.m
- mkblips.m
- mp3read.m
- mymkdir.m
- octs2hz.m
- tempo.m
- testlist.m
- test.m

PYTHON

- encode.py
- randomize.py
- shortest_time_dist.py
- toHexArray.py

C

- hello_world.c

- AWESOME_GUITAR.qpf
- AWESOME_GUITAR.qws
- AWESOME_GUITAR_TOP.dpf
- AWESOME_GUITAR_TOP.jdi
- AWESOME_GUITAR_TOP.qsf
- AWESOME_GUITAR_TOP.sof
- counter.vhd
- cpu.ocp
- cpu.vhd
- cpu_jtag_debug_module.vhd
- cpu_jtag_debug_module_wrapper.vhd
- cpu_ociram_default_contents.mif
- cpu_rf_ram.mif
- cpu_test_bench.vhd
- DebounceCounter.vhd
- debouncer.vhd
- de2_i2c_av_config.v
- de2_i2c_controller.v

VHDL

- de2_sram_controller.vhd
- de2_sram_controller_hw.tcl
- de2_wm8731_audio.vhd
- guitar_top.vhd
- InputController.vhd
- InputController_hw.tcl
- InputController_inst.vhd
- InputController2_inst.vhd
- InputController3_inst.vhd
- InputController4_inst.vhd
- InputController5_inst.vhd
- InputControllers_inst.vhd
- jtag_uart.vhd
- nios_system.bsf
- nios_system.ptf
- nios_system.qip
- nios_system.sopc
- nios_system.vhd
- nios_system_generation_script
- nios_system_log.txt
- nios_system.ptf.pre_generation_ptf
- nios_system_setup_quartus.tcl
- pulser.vhd
- sopc_builder_log.txt
- sram.vhd
- timer.vhd
- timer.vhdl
- timer_hw.tcl
- timer_inst.vhd

- beatavg.m
- beat.m
- calclistftrs.m
- chromagram_E.m
- chromagram_IF.m
- chromagram_P.m
- chrombeatftrs.m

- chromnorm.m
- chrompwr.m
- chromrot.m
- chromxcorr.m
- coverDistMxLists.m
- coverFtrExLists.m
- coverTestLists.m

MATLAB

- distmatrixwrite.m
- fexist.m
- fft2chromamx.m
- fft2melmx.m
- history-bragg-autoco.m
- history-golddust-xcorr.m
- hz2octs.m
- ifgram.m
- ifptrack.m
- listfileread.m

- listfilewrite.m
- localmax.m
- mkblips.m
- mp3read.m
- mymkdir.m
- octs2hz.m
- tempo.m
- testlist.m
- test.m

PYTHON

- **encode.py**
- **randomize.py**
- **shortest_time_dist.py**
- **toHexArray.py**

C

- **hello_world.c**

- AWESOME_GUITAR.qpf
- AWESOME_GUITAR.qws
- AWESOME_GUITAR_TOP.dpf
- AWESOME_GUITAR_TOP.jdi
- AWESOME_GUITAR_TOP.qsf
- AWESOME_GUITAR_TOP.sof
- counter.vhd
- cpu.ocp
- cpu.vhd
- cpu_jtag_debug_module.vhd
- cpu_jtag_debug_module_wrapper.vhd
- cpu_ociram_default_contents.mif
- cpu_rf_ram.mif
- cpu_test_bench.vhd
- DebounceCounter.vhd
- debouncer.vhd
- de2_i2c_av_config.v
- de2_i2c_controller.v

VHDL

- de2_sram_controller.vhd
- de2_sram_controller_hw.tcl
- de2_wm8731_audio.vhd
- guitar_top.vhd
- InputController.vhd
- InputController_hw.tcl
- InputController_inst.vhd
- InputController2_inst.vhd
- InputController3_inst.vhd
- InputController4_inst.vhd
- InputController5_inst.vhd
- InputController6_inst.vhd
- jtag_uart.vhd
- nios_system.bsf
- nios_system.ptf
- nios_system.qip
- nios_system.sopc
- nios_system.vhd
- nios_system_generation_script
- nios_system_log.txt
- nios_system.ptf.pre_generation_ptf
- nios_system_setup_quartus.tcl
- pulser.vhd
- sopc_builder_log.txt
- sram.vhd
- timer.vhd
- timer.vhdl
- timer_hw.tcl
- timer_inst.vhd

Display of the beats

The beats are stored and accessed in the C program.

They mark the boundaries for the time a user has the right to try and press the correct key.

The required key to press are displayed on the Console of the NIOS2 coding environment.

Getting the user inputs from the Game Guitar

The user inputs are received by using interruptions.

Once a key has been pressed, the VHDL code sends an interruption to the NIOS2 C code.

There is one type of interruption per guitar key.

The key that had been pressed is determined by the interruption ID.

Keeping track of the score

The score is handled by the NIOS2 C program.

The score is being increased each time the correct key has been pressed during the valid time lapse.

The score is displayed continuously in the NIOS2 Console.

Timing Design

Display of the beats

- Stored inside the C code
- Displayed based on their embedded timing data

Storing and playing the music

- SDRAM
- Flash Memory
- Music in the background (raw implementation in the current state of this project)

Display of the beats

- Stored inside the C code
- Displayed based on their embedded timing data

Storing and playing the music

- SDRAM
- Flash Memory
- Music in the background (raw implementation in the current state of this project)

key has been pressed during the valid time lapse.

The score is displayed continuously in the NIOS2 Console.

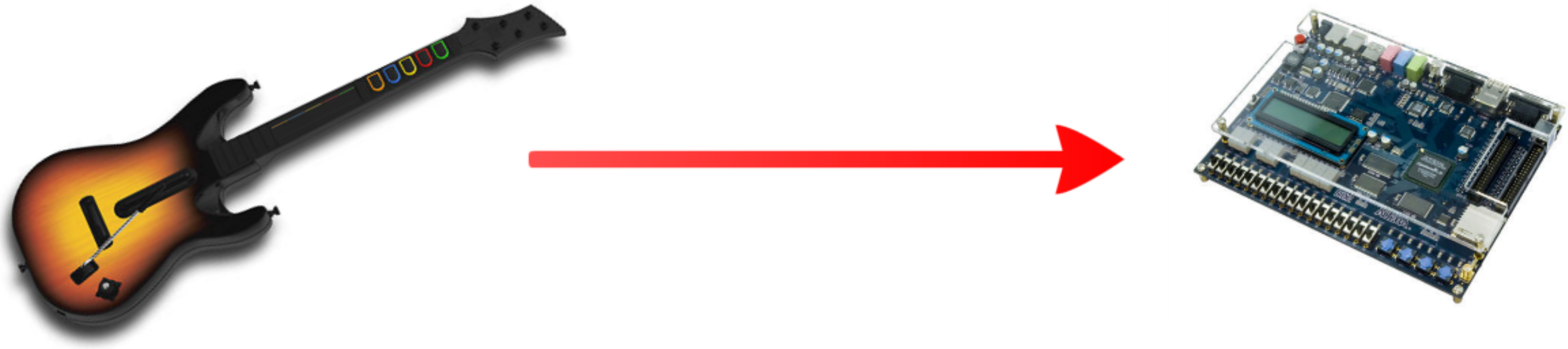
implementation in the current state of this project)

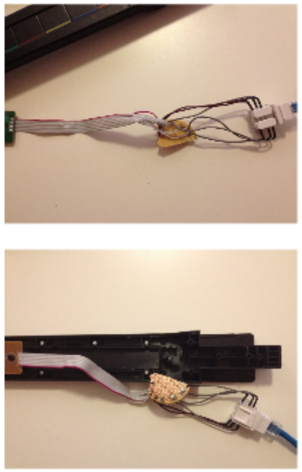
Software

- MATLAB LabRosa:
  - getting the beats
- PYTHON:
  - formatting the song
- NIOS 2 platform:
  - launch of the beats
  - keeping track of the score
  - The NIOS 2 console is the visual display for the user.

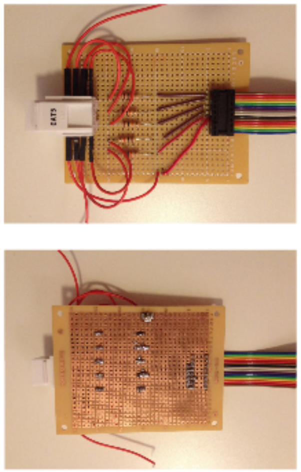
Hardware

Hardware





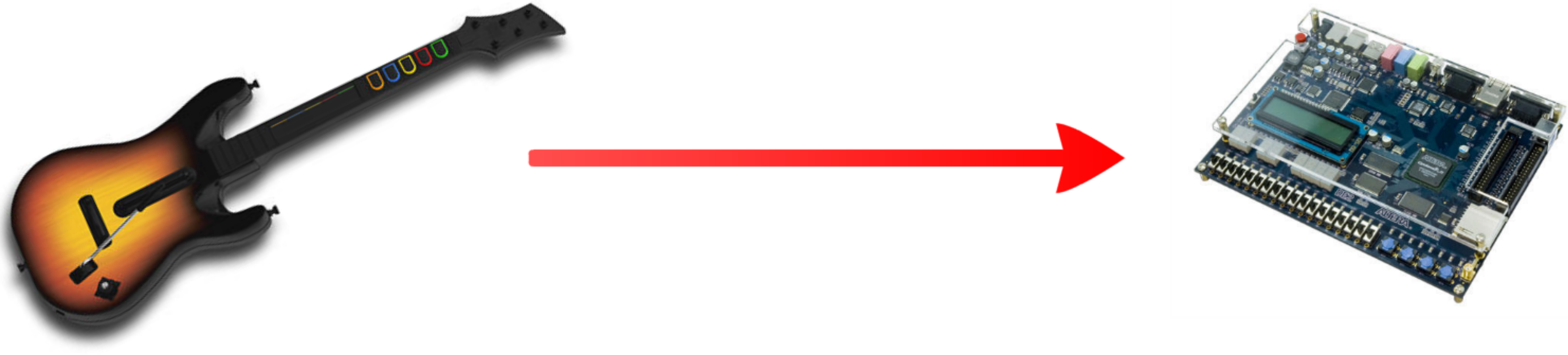
Link between the guitar and the cardboard

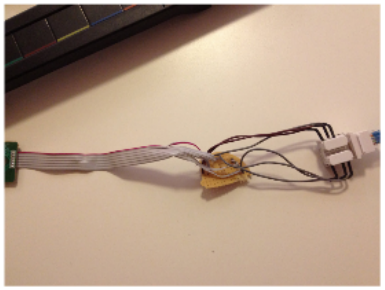


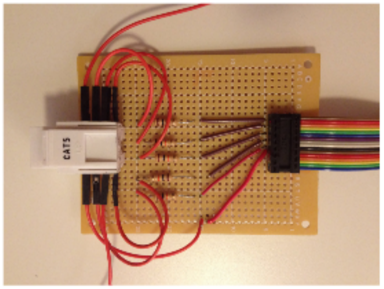
Cardboard

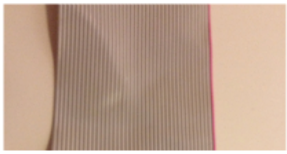
Link between the cardboard and the FPGA

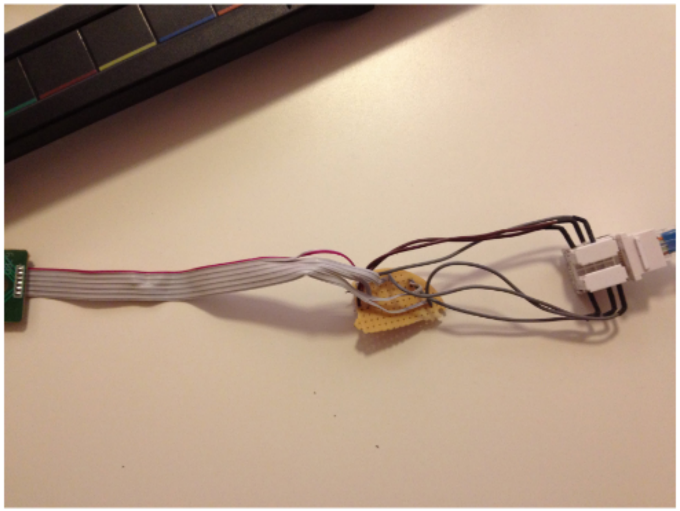
Hardware

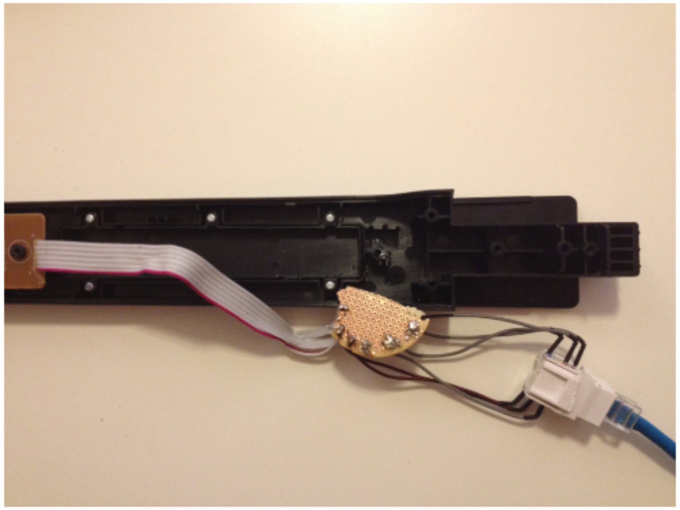




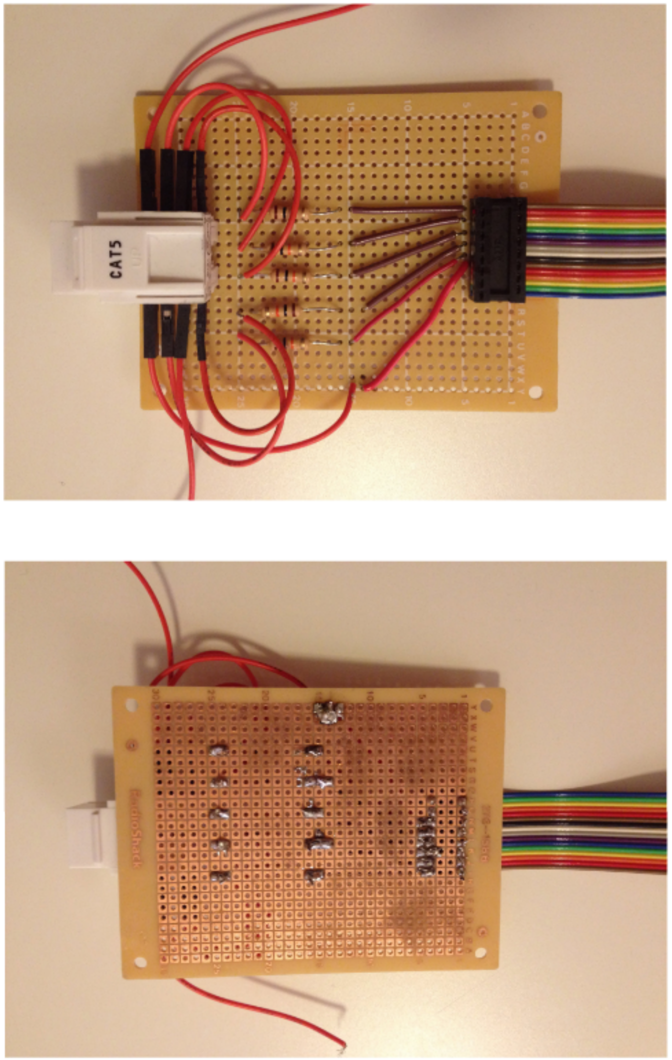








Link between the guitar and the cardboard



Cardboard

Link between the cardboard and the FPGA

Project Timeline

Milestone 1

March 27th

- I will buy and construct the game guitar.
- I shall detect key inputs with the game guitar.
- I will play a song (raw sound format) from a SD card in the FPGA.
- I will develop a program to build a script of a given song. This means, to produce a file that contains the notes and their corresponding positions for this particular song.
- I shall finally make a prototype of the base game engine in Java.

Milestone 2

April 10th

- I will work on the sprites and study how to do graphics and how to encode the sound efficiently (how many bits, how much information I can store…).
- The Java game prototype will integrate the work on scripts from the first milestone.
- I shall have designed the game internal functioning to ensure a constant frame rate (on paper).
- I will have started implementing the game.

Milestone 3

April 24th

- I shall finalize the game.
- I will develop an algorithm to compute the score.
- I shall improve the performance of the game and work on the graphics.

Final Run

May 23rd

- End of the VHDL programming
- End of the C programming
- Test of the global game

September 30th

- End of the project report
- End of the project presentation

Milestone 1

March 27th

- I will buy and construct the game guitar.

- I shall detect key inputs with the game guitar.

- I will play a song (raw sound format) from a SD card in the FPGA.

- I will develop a program to build a script of a given song. This means, to produce a file that contains the notes and their corresponding positions for this particular song.

- I shall finally make a prototype of the base game engine in Java.

Milestone

April 10th

- I will work on the sprites and st how to do graphics and how to encode the sound efficiently (ho many bits, how much informatic can store...).

- The Java game prototype will integrate the work on scripts fro the first milestone.

- I shall have designed the game internal functioning to ensure a constant frame rate (on paper).

- I will have started implementin game.

ilestone 1

27th

buy and construct the game

l detect key inputs with the guitar.

play a song (raw sound format) SD card in the FPGA.

develop a program to build a of a given song. This means, to ce a file that contains the notes eir corresponding positions for rticular song.

l finally make a prototype of se game engine in Java.

Milestone 2

April 10th

- I will work on the sprites and study how to do graphics and how to encode the sound efficiently (how many bits, how much information I can store…).

- The Java game prototype will integrate the work on scripts from the first milestone.

- I shall have designed the game internal functioning to ensure a constant frame rate (on paper).

- I will have started implementing the game.

Milestone

April 24th

- I shall finalize the game.

- I will develop an algorithm to compute the score.

- I shall improve the performan the game and work on the grap

ilestone 2

0th

work on the sprites and study do graphics and how to e the sound efficiently (how bits, how much information I ore...).

ava game prototype will te the work on scripts from t milestone.

I have designed the game l functioning to ensure a nt frame rate (on paper).

have started implementing the

Milestone 3

April 24th

- I shall finalize the game.
- I will develop an algorithm to compute the score.
- I shall improve the performance of the game and work on the graphics.

Final Run

May 23rd

- End of the VHDL programmi
- End of the C programming
- Test of the global game

September 30th

- End of the project report
- End of the project presentat

Ailestone 3

4th

l finalize the game.

develop an algorithm to ıte the score.

l improve the performance of me and work on the graphics.

Final Run

May 23rd

- End of the VHDL programming
- End of the C programming
- Test of the global game

September 30th

- End of the project report
- End of the project presentation

Experiences and issues in implementation

- Storage using SRAM/SDRAM
- Playing the song
- Bugs solving, difficulty to trace the source of a crash
- Merging several parts of the project

Lessons learned

- Start the project as early as possible.
- Code and debug small step per small step.
- Synchronize all the parts of the project as soon as possible.
- Do not venture into too many directions at once.
- Share CLIC laboratory resources with the other students.
- Make regular copies of the global projects.
- Be very careful with the SOPC builder.
- The most important lesson of all: enjoy your project!

Conclusion

- A very enjoyable game to code and debug
- A great hands on experience: using, modifying and building hardware by myself
- My first coding experience with VHDL and good training in C, as well as in MATLAB and PYTHON
- Additional tracks of study:
  - more developed visual interface for the game, using sprites;
  - have several different songs available instead of just one;
  - introduce a multiplayer mode.

Thank you for your attention!

Now is the time for a demo!