# EDA385: FPGA-pod

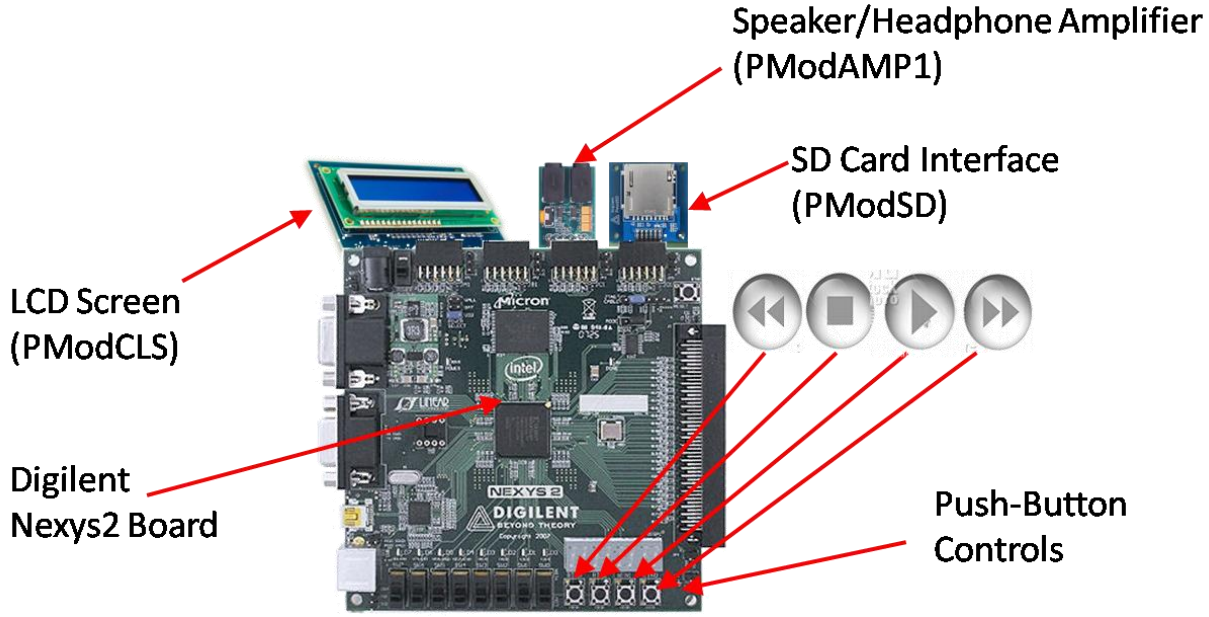
Pierre-Adrien Lefebvre: pieradrien.lefebvre@gmail.com

Kristian Samppa: et06ks7@student.lth.se

Matt Moles: mkmoles@ncsu.edu

## Description

Our planned FPGA-pod is a rudimentary music player based on the Digilent Nexys2 FPGA development board. Included in the design are three add-on modules to allow for data storage and user interaction with the FPGA-pod, these add-on modules include: one SD card interface (PModSD), one LCD screen (PModCLS), and one speaker/headphone amplifier (PModAMP1). Also acting as a point of user interaction are the push buttons already on the Nexys2 board. The push buttons on the board will act as user controls for the FPGA-pod. There are four push buttons on the board, their planned implementation in this user control scheme are as follows (from left to right): skip to previous track, stop, play/pause, and skip to next track. The LCD screen (PModCLS) will provide the user with a graphical user interface and will be controlled by the afforementioned push buttons already on the board. While a song is playing, that song's artist and title will be displayed on the LCD screen. If skip to next track is selected, the next artist and song title will be displayed on the LCD screen. Conversely, if skip to previosu track is slected, the previous artist and song title will be displayed on the LCD screen. Once displayed, the user has the option to play the track with the play/pause button. While the song is playing, the user has to option to stop or pause the track via the stop or play/pause button. The SD card interface (PModSD) will be the primary data storage unit of the FPGA-pod. Music will be stored on an SD card where it can later be opened, read, and eventually output by the Nexys2 board. Mono music output will occur at the speaker/headphone amplifier (PModAMP1). While a song is playing, headphones or speakers be connected allowing the user to listen to the track they chose. A visual description of this planned implenentaion is shown below.



## I/O Processes

**SD Card Interface (PModSD):**
This module will permit us to import data from music files stored in a SD memory card. The SD driver is composed of a software and a hardware part. The software portion is responsible for sending SD-compliant commands in order to initialize, identify, and send block-read orders to the SD card. The hardware will drive the clock signal and the data signals (DI/DO/CS) to communicate with the card in SPI mode. The command data is serialized before being sent on the DI line. Incoming data flow will be stored in a FIFO memory, with a flag that indicates to the Microblaze microprocessor that one entire block (512 bytes) is ready. The software has to poll this flag and retrieve music data through the *Processor Local Bus* (PLB) interface.

This block will be entirely designed by our group.

**LCD Screen (PModCLS):**
The LCD screen will be driven in UART mode. This implies implementing an UART IP core. Menus and button actions will be designed in software.

**Speaker/Headphone Amplifier (PModAMP1):**
Microblaze microprocessor will bufferize sound samples in a FIFO memory. Since sound output is analog, we need a digital to analog converter. This will be done thanks to an integrated Sigma-Delta DAC : *xps_deltasigma_adc* IP core. Audio will first be output in mono mode. Stereo mode - which implies two FIFO and two DAC – could be an interesting extension of our project.

**On-Board Push Buttons:**
The state of each push button will be polled in software thanks to the implementation of the Xilinx IP core *xps_gpio*.

**Debug serial output:**
An RS232 UART will be used as a debug output to make debugging easier.

## Memory Requirements

We will need to implement two FIFOs (incoming and outgoing data flow) of 1024 bytes each in the mono music player. The size of the FIFOs should correspond to the size of two memory blocks since the old block is being read while the new one is written to the FIFO. This will also prevent data overflow from occurring.

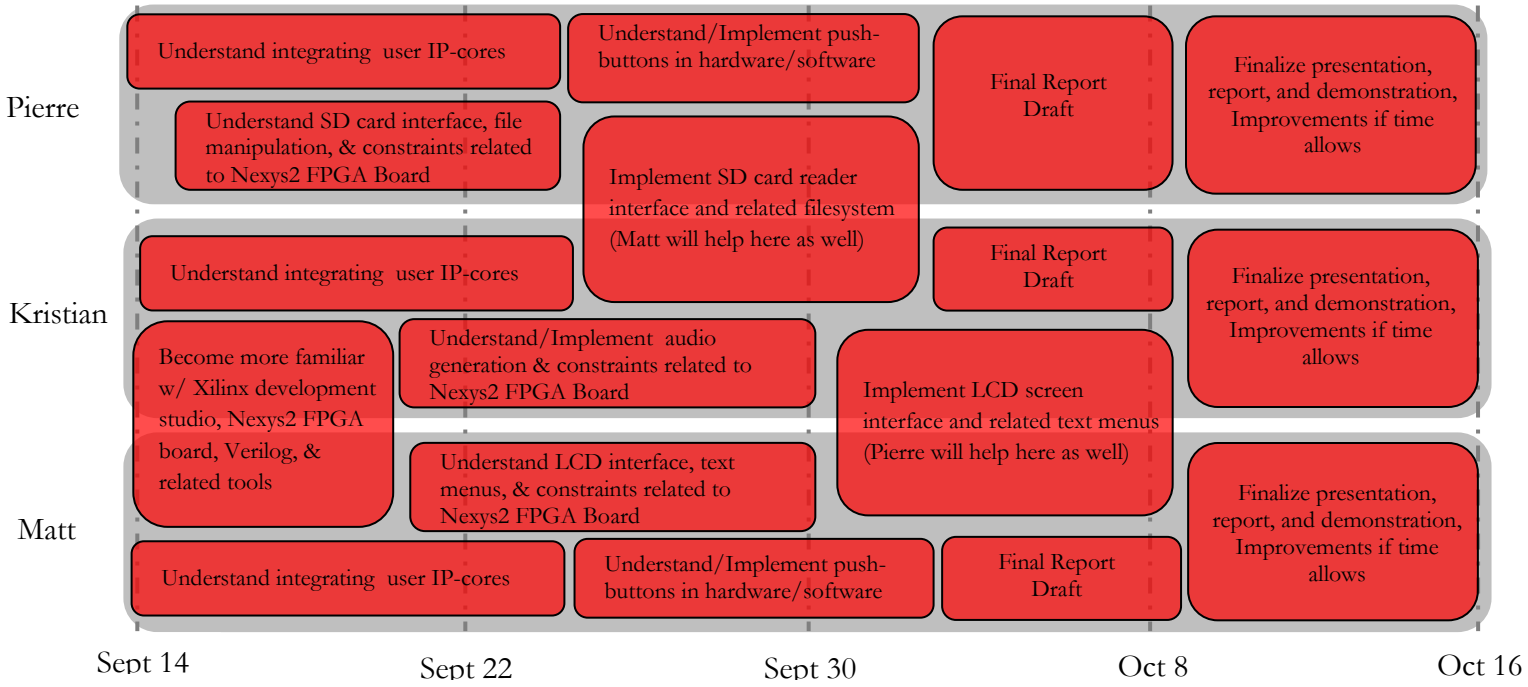
Memory needs: 2048 bytes of BRAMs for a mono player, 3092 bytes of BRAMs for a stereo player.

## Implementation Alternatives and Improvements

We believe that successfully manipulating music data with the SD card interface will be one of the harder and most time consuming portions of this project. We also believe that this portion of the project will provide for the greatest amount of flexibility implementation wise. Since only audio data will be read from the SD card, it may be possible to implement most of the SD card controller with software, if not exclusively with software. The SD card interface in conjunction with the Nexys2's on-board file system and already included file management library functions may provide for the easiest way to handle this sound processing. However, reading and manipulating this sound data might be better implemented in hardware. This is because the SD card reader needs to serialize data sent from the processor in parallel form; implementing the SD card interface in hardware may allow for faster data read times, a potentially key factor in trying to realize an output of CD-quality audio. A third, and perhaps best alternative is to implement the SD controller into both hardware and software. The software portion could handle complex commands and file management while hardware portion interpreted and carried out those commands. This would allow for both simpler hardware implementation and faster data processing while the complexities of this task were hidden in the software portion of the SD card controller.

Time permitting, there can be more features added to the FPGA-pod. One of the first features we would probably upgrade is the mono sound output. Once we learn more about the restraints of sound processing on the Nexys2 board and become more familiar with working with the board itself, implementing stereo sound will hopefully be fairly easy to achieve. Also, the user control buttons on the board could be used for different actions depending on what the FPGA-pod was doing. If music is not being played, the skip to previous and next track control buttons would do just that, skip to the previous or next track. Conversely, if music is being played, the skip to previous and next track buttons could be assigned new functionality, such as volume control. This would mean the user would have to stop the song they are currently listening to before they can browse for another track. Another added feature could be the implementation of a sort of track timer. Whenever the FPGA-pod is playing music, the LCD display could be used to show how much time has elapsed or is left in a song. The LCD display could alternate on a given time interval (~3 seconds) between the track's artist and title and the timer. Possibly the greatest, and most complex potential for improvement is the implementation of a MP3 decoder. We plan to first implement the FPGA-pod with Wave files because while they are larger, they are easier to read and manipulate. Adding an MP3 decoder would allow for many more songs on the SD card but also require much more sound processing.

## Timeline


Pierre
Understand integrating user IP-cores
Understand/Implement push-buttons in hardware/software
Final Report Draft
Finalize presentation, report, and demonstration, Improvements if time allows
Understand SD card interface, file manipulation, & constraints related to Nexys2 FPGA Board
Implement SD card reader interface and related filesystem (Matt will help here as well)
Kristian
Understand integrating user IP-cores
Final Report Draft
Finalize presentation, report, and demonstration, Improvements if time allows
Become more familiar w/ Xilinx development studio, Nexys2 FPGA board, Verilog, & related tools
Understand/Implement audio generation & constraints related to Nexys2 FPGA Board
Implement LCD screen interface and related text menus (Pierre will help here as well)
Matt
Understand LCD interface, text menus, & constraints related to Nexys2 FPGA Board
Finalize presentation, report, and demonstration, Improvements if time allows
Understand integrating user IP-cores
Understand/Implement push-buttons in hardware/software
Final Report Draft
Sept 14
Sept 22
Sept 30
Oct 8
Oct 16