

# Real Time Spectrogram

EDA385 Final Report

Erik Karlsson, dt08ek2@student.lth.se  
David Winér, ael09dwi@student.lu.se  
Mattias Olsson, ael09mol@student.lu.se

October 31, 2013

## **Abstract**

Our project is about visualizing sound using an FPGA board, a monitor and an A/D converter to read the source. The visualization is a spectrogram, a graph showing frequency and intensity over time. To get a signal from the time domain to the frequency domain we use a Fast Fourier Transform algorithm. In the end we learned that writing our own components was not as easy as we thought and it saves a lot of work to adjust your architecture to use premade IP cores instead.

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Architecture</b>	<b>3</b>
2.1	Hardware . . . . .	3
	AD Converter . . . . .	3
	Fast Fourier Transform . . . . .	4
	VGA Module . . . . .	4
2.2	Software . . . . .	5
<b>3</b>	<b>User Manual</b>	<b>6</b>
<b>4</b>	<b>Problems</b>	<b>7</b>
	A/D Converter . . . . .	7
	Fast Fourier Transform . . . . .	7
	VGA Module . . . . .	7
<b>5</b>	<b>Lessons Learned</b>	<b>7</b>
<b>6</b>	<b>Contributions</b>	<b>8</b>
<b>7</b>	<b>Final Result</b>	<b>8</b>
<b>8</b>	<b>References</b>	<b>9</b>

# 1 Introduction

The project is about visualizing sound on a monitor through the use of a Nexys 3 spartan 6 FPGA. There is an analog signal as input that is converted with the AD-Converter, this signal is then processed with the FFT algorithm. Our initial thoughts were to write most of the components from scratch (see [Figure 1]), since we thought this would save area on the FPGA. This was however a mistake as it took a long time to write and debug the hardware. Instead we looked at which IP cores were available to us and decided to change architecture as to use those IP cores. We wrote a custom VGA and memory controller, and we wrote a wrapper to convert the FFT core for use with FSL instead of AXI4-Stream. The AXI\_IIC was instantiated through XPS (see [Figure 2]).

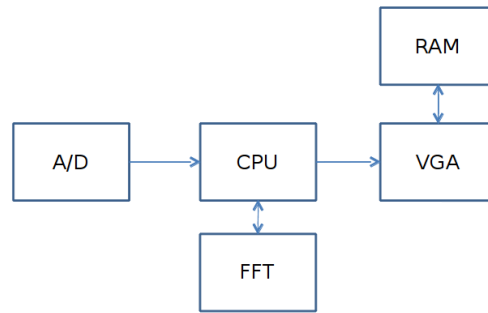


Figure 1: Proposed architecture

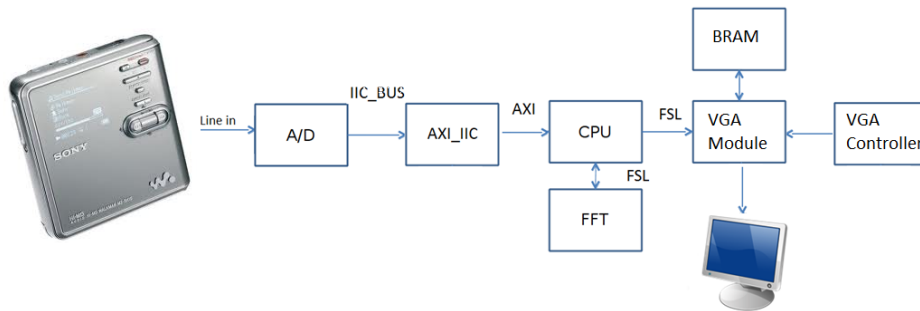


Figure 2: Final architecture

## 2 Architecture

### 2.1 Hardware

Below you will see a table outlining the utilization of our hardware. The percentages are of the entire project, not of the FPGA capability.

Component	Slices		Slice Registers		BRAMs	
	Count	Percentage	Count	Percentage	Count	Percentage
VGA	64	2.7%	65	1.6%	8	25.8%
ADC	262	10.2%	338	8.3%	0	0%
FFT	616	25.6%	2147	52.3%	7	22.6%
Microblaze	903	37.5%	1010	24.6%	16	51.6%
<i>Other components</i>	565	24%	544	13.2%	0	0%
Total	2410		4104		31	

#### AD Converter

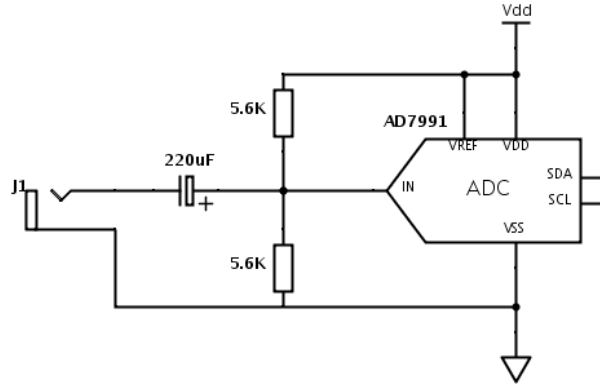


Figure 3: AD converter schematic

The signal input consists of an audio source connected to a PMOD AD2 from Digilent with some external circuitry. The PMOD contains a AD7991 12-bit A/D converter which communicates over an I2C interface. The I2C interface requires external pullups to communicate at a high enough rate since the internal pullups are too weak for the short rise time required for a sample rate of approximately 40kHz. The input signal needs to be modified to fit the converter's input range, this is done with a simple circuit that adds a DC offset to the signal without affecting the audio source ([Figure 3]).

The A/D converter is configured to convert from channel 0 and use supply voltage as reference, all other parameters are kept at their default values.

When the converter receives a read request over the i2c bus, it responds with 2 bytes of information. The first byte starts with 2 zeros, 2 channel identifier bits and the 4 most significant bits of the conversion result, the second byte contains the remaining bits of the conversion result.

Conversion value register								
Byte 1:	0	0	CH0	CH1	D11	D10	D9	D8
Byte 2:	D7	D6	D5	D4	D3	D2	D1	D0

## Fast Fourier Transform

The Fast Fourier Transform algorithm is generated with Xilinx Core Generator tool. It has a 32 bit input signal and a 32 bit output signal, 16 bits for the real value and 16 bits for the imaginary value. The phase factor (a factor that's multiplied in every step of the FFT) is 16 bits so that should need an output length of 48 bits, this is avoided by using the scaling option to scale the numbers down.

The FFT generated by Xilinx Core Generator uses the Axi4-Stream protocol to communicate. The MicroBlaze processor supports the AXI4-Stream protocol as well as the FSL protocol, but it can only use one of them, so either everything is AXI4-stream or everything is FSL. This causes an incompatibility with the other cores in our project, because they are using the FSL protocol. To solve this incompatibility we have written a wrapper for the core that converts the protocol from Axi4-Stream to FSL. In addition to the conversion we also cut the output signal in half, since one of the properties of a Fourier Transform is that it is symmetrical around the middle point. The FFT we use have 256 points, but only outputs the 128 first of the result to save a little bit of time while fetching the data.

## VGA Module

The VGA module contains a VGA controller, some BRAM for video memory and a Finite State Machine for the memory interface.

The VGA controller generates the timing signals and counters for the current screen position.

Simplified, the FSM has three states; *Draw\_Pixels*, *Read\_FSL*, and *IDLE*.

**Draw\_Pixels** is the state that is responsible for putting and holding the correct pixel color on the RGB port of the VGA. Outside of this state the RGB port is set to a constant 0, meaning black color. This state is exited

when we are outside of our visible resolution, from this state we go to the *Read\_FSL*-state.

**Read\_FSL** is where a new column of pixels is read. Because the FSL bus is 32 bits wide, but the memory is only 8 bits, we need to do this in 4 clock cycles, and increment the memory address between each of those cycles. To avoid unnecessary data movement the memory is made cyclic, and a signal is kept to keep track of where in the memory the first column is. We place the new column that we read to the left of the previously newest column, and then we decrease the signal to indicate the new first column.

From this state we jump to the *IDLE*-state as soon as we are done reading from the FSL-bus.

**IDLE** is the starting state. It is also used as a wait-state between *READ\_FSL* and *Draw\_Pixels*. From this state we jump to *Draw\_Pixels* when the VGA-controller indicates that it is the start of a new screen.

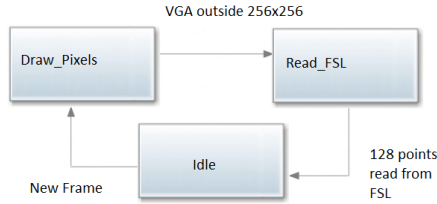


Figure 4: VGA module FSM

## 2.2 Software

Our software runs on the Microblaze CPU as a standalone process, meaning we do not have any threads or interrupts. The software starts by initializing the AD-converter to tell it which channel to use. After that it goes in to the main loop as illustrated in [Figure 5]. The loop reads values from the AD-converter, sends those values through the FFT core via FSL and finally sends the result to the VGA module. The VGA module only reads 128 pixel values every time it is done drawing on the screen, so the waiting time in the software will be waiting for the FSL queue to empty so that it can transfer the new results.

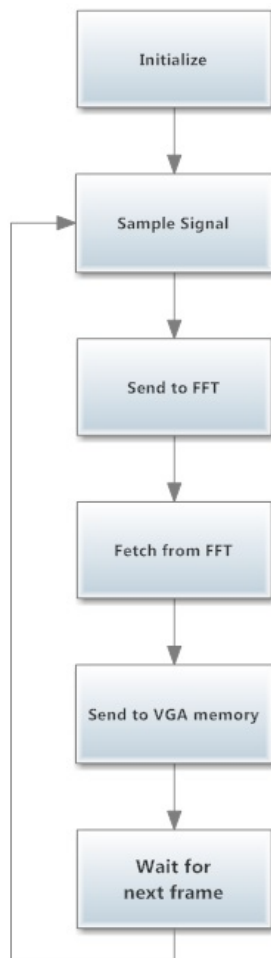


Figure 5: Software flowchart

### 3 User Manual

1. Connect A/D Converter with pull-ups and DC-centering: This component is described earlier in this report.
2. Connect monitor: connect a VGA cable from the FPGA VGA port to the monitor.
3. Connect the programming and power cable to the Nexys 3 board
4. Turn on the power to the board
5. Open Digilent Adept and find the file download.bit included in our project



6. Press program in the Adept interface

## 4 Problems

### A/D Converter

It took us some time to get the hardware for the A/D converter to work. As mentioned previously some pullups needed to be soldered on to the IIC bus to get it to work at a high enough rate. In addition to this the PMOD AD2 converter only works on positive voltages, but the line-in source delivers voltages centered around the zero point. This was solved by soldering a DC offset on to the circuit before the A/D converter.

### Fast Fourier Transform

At first we intended to implement our own FFT algorithm since we could not find any other way. This was very tedious work and in the end it took up almost all of the space of the FPGA board, the best we could get working was a 32 point FFT. Later on though we found that Xilinx Core Generator could generate an FFT core for us, and this worked very well.

### VGA Module

The initial plan was to write the memory controller from scratch and use the RAM memory, we soon found that this approach was too slow for us. To solve this we first tried to enable burst mode of the memory, but the problem still remained. After that we tried enabling page mode [2], this made the data stream not constant so a buffer was introduced. This solved the timing problem but instead our VHDL hardware became too big to fit in the FPGA.

The solution was to use the BRAM instead of the RAM. The BRAM is smaller and faster, which means our timing problem was solved, unfortunately it also meant that we could not print on the full resolution of the VGA screen as we first intended.

## 5 Lessons Learned

This project has taught us a lot of things, for example the Xilinx toolchain and efficient workflow with it. We have learned how to build an entire project with both hardware and software and how to connect them together. Developing hardware is a much "slower" process than developing software, it is hard to debug and having a good test bench is *very* important. If the whole

project is compiled and a bug shows up it's hard to know where it came from, so proper testing should be done on small pieces first.

## 6 Contributions

Most things AD-related is done by David, this includes constructing the hardware and writing the software to communicate with it.

The early, self implemented, as well as the final wrapped Fast Fourier Transformis written by Erik.

The early version of the VGA Module with built in DMA (which was working in simulation but not tested or used) was written by Mattias.

And this final report is written by the group members equally.

## 7 Final Result

The sample rate is determined by the communication frequency of the I2C bus, which is set to 800kHz, this should give a sample rate of 44.44 kHz but this is not verified, the system detects 20kHz input signals indicating that it is atleast 40kHz. If the external pullups are too weak for the specified communication frequency it could lead to a lower sampling rate.

The target sample rate of 44.1 kHz would contain 735 sample points for every frame. The A/D converter only reads 256 samples, of the 735 available, which means it might miss some information. This could be improved by using a larger FFT or repeating the process multiple times per frame, but it did not seem to be a problem.

The sampling process starts when the fifo buffer of the last iteration is emptied, which happens when the VGA module has finished drawing a frame. The result is then put in the fifo buffer and the VGA module puts it into the bram when it is done with the current frame. The time required for this process is the time to position the VGA to begin drawing a new frame and draw a complete frame, which in our case is 25.3 ms.

See a picture of the final result in [Figure 6], unfortunately it is rather blurry because our camera is not fast enough to capture the moving image.

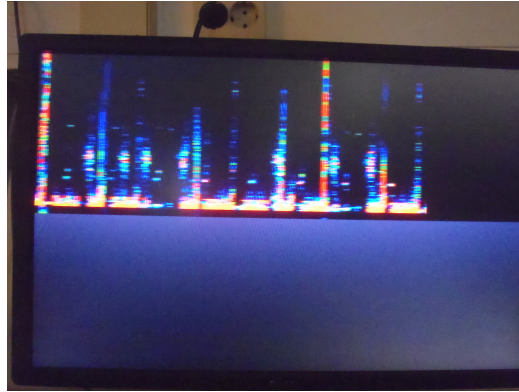


Figure 6: Final result

## 8 References

1. Digilent, Nexys3 Board Reference Manual, [http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3\\_rm.pdf](http://www.digilentinc.com/Data/Products/NEXYS3/Nexys3_rm.pdf)
2. CellularRam Memory, [https://www.micron.com/~media/Documents/Products/Data%20Sheet/DRAM/Mobile%20DRAM/PSRAM/16mb\\_burst\\_cr1\\_0\\_p23z.pdf](https://www.micron.com/~media/Documents/Products/Data%20Sheet/DRAM/Mobile%20DRAM/PSRAM/16mb_burst_cr1_0_p23z.pdf)