**RMIT University**

**EEET2162 – Advanced Digital Design 1**

**Final Project Report**

**AUDIO SPECTRUM ANALYSER**

Student: Hieu Huynh s3634147

Phuc Nguyen s3636112

Course: EEET2162 – Advanced Digital Design 1

Lecturer: Dr. Glenn Matthews

Due Date: 5th June 2019

Contents

[I. Acknowledgement 3](#_Toc10750261)

[II. Executive summary 3](#_Toc10750262)

[III. Introduction 3](#_Toc10750263)

[IV. Signal processor (Hieu) 3](#_Toc10750264)

[1. Literature review 3](#_Toc10750265)

[2. Block diagram 5](#_Toc10750266)

[3. Verilog design 6](#_Toc10750267)

[4. RLT viewer 8](#_Toc10750268)

[5. Simulation 9](#_Toc10750269)

[6. Implementation 10](#_Toc10750270)

[V. Display driver (Phuc) 11](#_Toc10750271)

[1. Literature review 11](#_Toc10750272)

[2. Verilog design 12](#_Toc10750273)

[a. Design approach 12](#_Toc10750274)

[b. System block diagram 12](#_Toc10750275)

[c. I2C interface 13](#_Toc10750276)

[d. Timing requirements 14](#_Toc10750277)

[e. Video sync generator 15](#_Toc10750278)

[f. Frame pattern generator 16](#_Toc10750279)

[3. Simulation 17](#_Toc10750280)

[VI. Combined system 18](#_Toc10750281)

[VII. Conclusion 20](#_Toc10750282)

[VIII. Reference 20](#_Toc10750283)

[Appendix – Diagrams and commands list 21](#_Toc10750284)

# Acknowledgement

In this project, RAM two port, FFT IP core, ALT SQRT IP and PLL IP core are used. All the IP core are available in the IP catalogue of the Quartus. For the SPI communication between the FPGA and the LTC2308, the protocol is built by the student. Also, the FFT block trigger procedure and the FIFO buffer is built base on the ADC control system. The video sync is based on Analog Device’s application note for ADV7513. The I2C controller is based loosely on Digikey’s design example in VHDL.

# Executive summary

In this project, an audio spectrum is developed. Onboard ADC LTC2308 is utilized to capture the signal from 4 channels then Fast Fourier Transform is performed to analyze the signals on frequency spectrum. Then, the spectrum can be plotted onto a monitor using HDMI interface provided by ADV7513. The system can process and distinguish the 4 signals simultaneously since the ADC cannot read 4 channels at once. The system samples the input signal at 44.1 kHz, hence, it can analyze the input signals which has frequency from 0 to 20 kHz. A single 32-points FFT spectrum is computed for each channel per frame period to avoid redundant data and clashing of memory read-write. A frame pattern consisting of spectral points represented as bar graphs is drawn with display layering of channels. When implemented, a variety of input frequencies is applied at the 4 channels, all of which show accurate FFT results on the monitor, but the video driver still causes graphic glitches between the spectral bars.

# Introduction

As the need for real-time processing of multiple channels of audio data increases with more advanced musical instruments many manufacturers are beginning to utilize FPGAs to perform data processing [1]. The main reason behind this is because FPGAs can process the multiple parallel data paths independently. In this project, DE10-Nano is utilized to capture four channels of audio using onboard audio to digital converter (ADC) LTC2308, which is produced by Analog devices. Then, Fast Fourier Transform (using IP core) is applied on the incoming data to create an audio level spectrum analyzer. The spectrum is then plotted on a monitor using HDMI interface. This project involves two main parts which are the signal processor and display driver. Signal processor captures and processes the input signal to form a frequency spectrum. Then, the display driver is developed to read and display the spectrum on a monitor.

# Signal processor (Hieu)

## Literature review

DE10-Nano has LTC2308 ADC, which is low noise, provides 500ksps and has 12bit resolution [2]. The ADC has 4 wire SPI compatible serial interface, which can be illustrated as below:

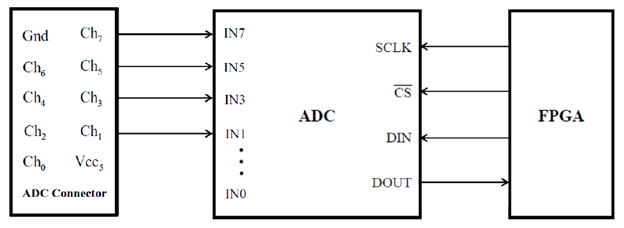


Figure 1. ADC functional block diagram [3]

The important parameters given in the datasheet, which is taken into consideration for accurate timing, are summarized as following:

* The maximum shift clock frequency is 40 MHz, that is, the SCK signal (SCLK signal) output from the FPGA is at most 40MHz
* Maximum sampling frequency is 500 kHz and be can reduced to be appropriate for this project
* Typical conversion time is 1.3 us
* Minimum acquisition time is 240 ns

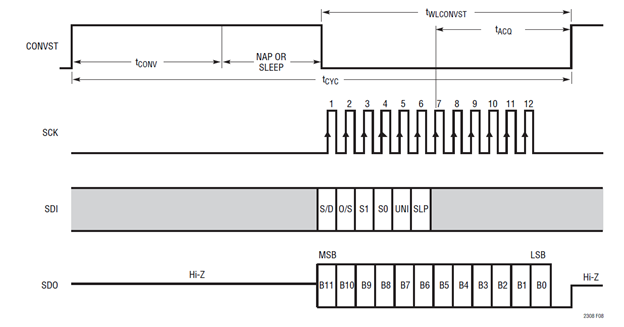


Figure 2. LTC2308 Timing with a Long CONVST Pulse [2]

When conversion signal (cs signal) is high, the ADC starts to convert the analog value, which can take 1.3 us, then waits (nap or sleep). When the conversion signal becomes low, at every rising edge of the SCK signal, the ADC captured the SDI signal (1 bit) sent from the FPGA and send out the converted data (bit by bit) from most significant bit to least significant bit. After capturing the 6 bits of SDI signal, the ADC requires at least 240 ns of acquisition time. Since the ADC is driven by 40 MHz clock, the number of clock cycles for acquisition time is:

Hence, at least 16 cycles are need for period. In term of sampling frequency, the ADC can provide a new value for every rising edge of the CONVST signal. Hence, by controlling the CONVST signal, the sampling frequency is adjustable. Since the highest frequency signal that human ear can hear is around 20kHz, the sampling frequency should be at least double that to satisfy the Nyquist theorem, which ensures that the shape of sampled signal is not significantly distorted. Thus, 44.1 kHz frequency is chosen. The conversion and wait (nap) period will be adjusted to be longer while the CONVST high time () period remains 16 clock cycles. Hence, the number of clock cycles required for the CONVST signal remains high are:

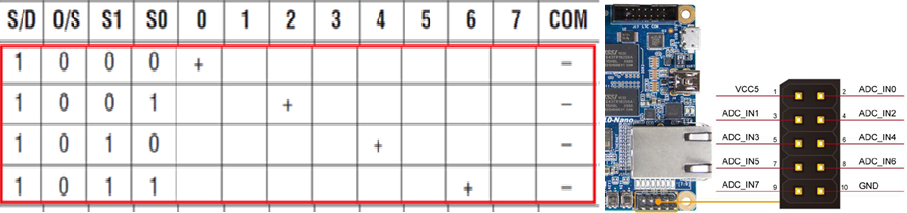


Figure 3. Channel configuration [4]

Above is the address table which helps the ADC to locate the propriate channel to read. By controlling the address, 4 channels can be simultaneously read since the ADC only read one channel at a time. In this project, only the 4 addresses in the red box are used since it is easier to connect the input to the ADC pin in that configuration. COM pin is GND and the numbers (#) in the table is corresponding to ADC\_IN# number.

## Block diagram

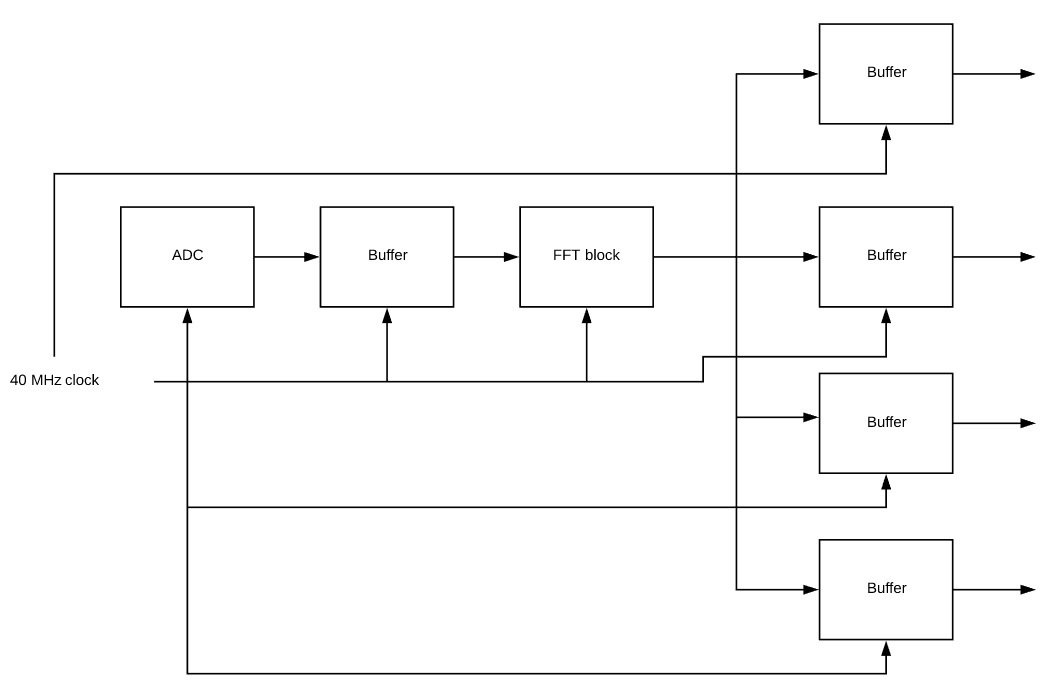


Figure 4. Signal processor system block diagram

The data flow is illustrated as above, where the whole ADC system is driven at 40 MHz to ensure that there is no mismatch in signal. The ADC’s behavior is controlled by the FPGA, which sends data back to the FPGA at 44.1 kHz. Then, the data is stored in a buffer, whose size is dependent on the FFT block configuration. The purpose of using buffer due to the mismatch of the driven clocks, where every block uses 40 MHz where the ADC has 44.1 kHz sampling frequency. The first buffer is used to store the data every time the ADC samples a new data. Once the buffer is full, it feeds its data to the FFT block and triggers it. The next four buffers are used to capture the output from the FFT block and separate the data from each channel. The first buffer starts feeding its data during the waiting period of the ADC, hence, no sample is missed. The last buffer is used for next phase, where the spectrum is plotted onto the monitor. The system is driven using a Finite State Machine. The hierarchy can be observed as following:

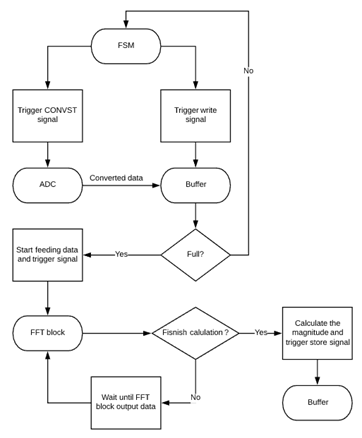


Figure 5. Workflow of the FSM controlling the ADC

The state machine controls the behavior of the ADC and write data signal of the first buffer. When the ADC acquires a new data, the FSM trigger the buffer to store the data. If the buffer is not full, the same procedure is repeated. Otherwise, it starts feeding the data to the FFT block in first-in-first-out (FIFO) order and triggers the calculation. Then, the second buffer (representing for the 4 buffers) is controlled by the FFT block, where it only starts writing data only if it is empty and the FFT block sends out the results. The FFT block sends real and imaginary results, hence, additional calculation step occurs to calculate the magnitude (absolute value) before writing it to the buffer. The writing and reading behavior of the second buffer is also in FIFO order. The buffers are empty only when all its data is read. The FFT block is configured to run in variable streaming fixed point mode with 32 data length. That is, the buffer’s size is 32 data. Hence, the FFT block is triggered and new spectrum frame is acquired at a frequency of:

The timing requirement is met, because:

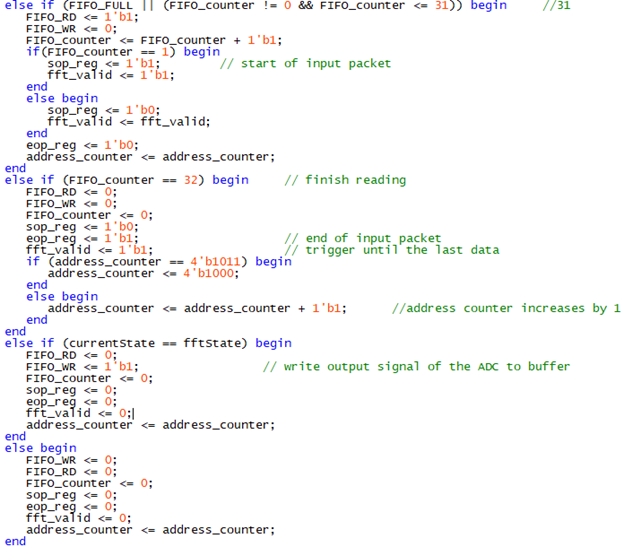
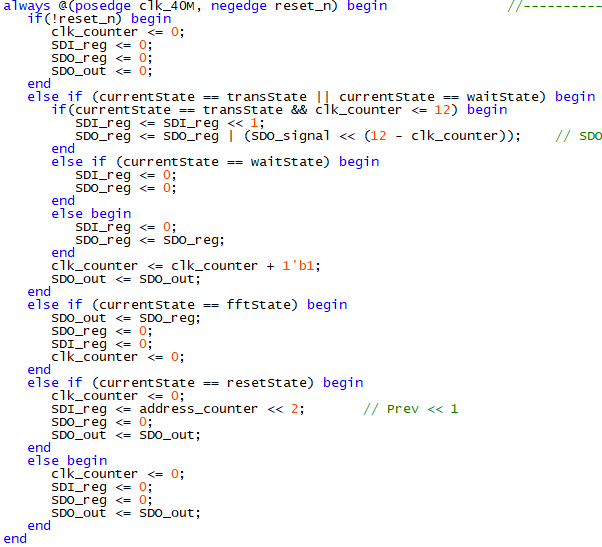
* Feeding data to FFT block process of the first buffer takes 32 cycles at 40 MHz (0.8 us) out of 891 cycles at 40 MHz (22.3 us) during waiting state
* The throughput latency is 64 cycles at 40 MHz (given by the IP parameter editor, including 32 cycles for calculation and 32 cycles for shifting out 32 results), which is 1.6 us

## Verilog design

The Moore state machine is driven by 40 MHz (same with ADC), which consists of 5 states. The role of each state is to:

* resetState: restart the routine of the ADC, CONVST signal is high for 1 clock cycles
* transState: start transferring the converted data from ADC to FPGA and send SDI signal from FPGA to ADC. CONVST signal is low for 16 clock cycles
* fftState: FPGA successfully capture the new data, trigger write signal to store the data into the first buffer. CONVST signal is high for 1 clock cycle
* waitState: include conversion time and nap time. CONVST signal is high for 891 clock cycles
* doneState: finish 1 cycle of ADC behavior, goes back to resetState to restart the routine.

Also, the state machine has asynchronous active low reset. That is, on the falling edge of the reset signal, the state machine goes back to resetState to restart the routine. Then, based on the current state of the state machine, the following block is created to capture the output data of the ADC and send the SDI signal to the ADC.



If the current state (left block) is transState, SDI\_reg shifts its pre-loaded value left to send its most significant bit to the ADC. At the same time, SDO\_reg captures the output bit of the ADC and shift it left from the most significant to the least significant. After transState is fftState, where the SDO\_out updates the new data from SDO\_reg. Otherwise, SDO\_out holds its data, which helps to create a continuous signal. If the current state is resetState, SDI\_reg update the new address from address\_counter. For other states, there is no signal sent from the ADC or the FPGA.

Also, another block (right block) is created to control the behavior of the first buffer. If the current state is fftState, a writing signal triggers the buffer to store the output from the ADC. If the buffer is full, then the buffer starts to feed its data to the FFT block and trigger the block to receive the input. A counter is increased each time a data is sent to the FFT block. The data is streamed into the FFT block until the last data, then, end of input signal is trigger and the source valid signal goes low at the next clock cycle to ensure that the last data is successfully transferred. address\_counter is a reg variable which increases by 1 every time the first buffer is full and start feeding its value to FFT block. The purpose of this variable is to create new address and to help the system to simultaneously read 4 channels one by one.

Then, 4 instances of the first buffer are created, which is triggered to write the output of the FFT block with respect to the current channel that is being read. the fft\_out\_FIFO\_wren is triggered depending of the source\_valid signal and the address\_counter. This is to help the next phase read to data, distinguish them and plot them onto the monitor. Thereby, users can observe the input of the channel separately.

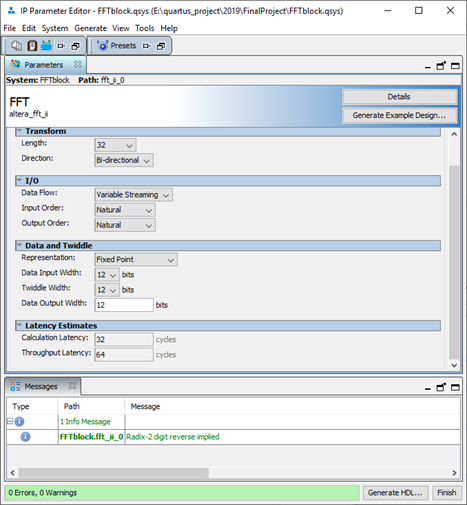


Figure 6. FFT IP configuration

Above is the configuration of the FFT block. Fixed point variable streaming is used with a length of 32 data points. The block takes the input and outputs the results in natural order, which is sequential order from 0, 1, 2… n where n is the size of the transform. In other words, the output from the FFT block is from 0 to 44.1 kHz frequency domain, no digit reversed. However, due to the nature of Fourier Transform, the one input signal has 2 frequency peaks that are symmetrical across the middle frequency. Hence, the latter half of the transform is not useful, which is then discarded in the next phase. Hence, the spectrum from 0-20 kHz is plotted onto a monitor.

## RLT viewer

Following is the RTL viewer of the ADC module. It includes a state machine, several logics and registers to control the communication between the FPGA and the ADC, buffers and FFT block to calculate and store the spectrum. Following is the detailed connection between the first buffer (bottom right) and the FFT block (top left):

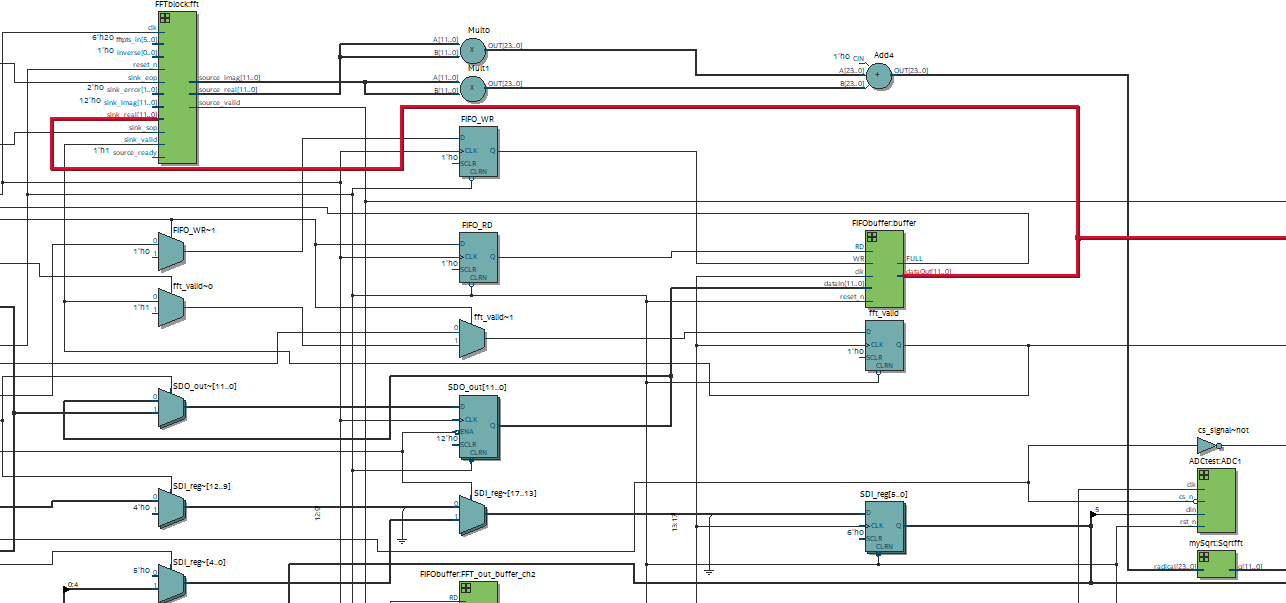


Figure 7. First buffer connection

The output from the first buffer goes directly to the real input of the FFT block. sink\_valid, sink\_eop and sink\_sop is controlled by the registers to trigger the FFT block at the waiting state of the state machine, when the buffer is full. Then, the real and imaginary output of the FFT block go to multiply and add operator to calculate , then the result goes through a square root operator (created using IP core) to calculate the magnitude then store it in the second buffer. It is important to consider that the input and output of the FFT block are signed numbers, hence, appropriate declaration is necessary to get the correct result.

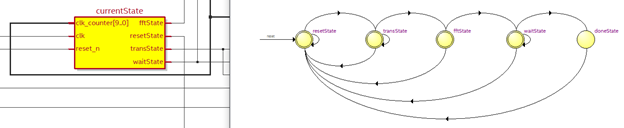


Figure 8. State machine RTL block and the transition diagram

Above is the state machine, whose input are clock, active low reset and clk\_counter reg variable. The reset is used to bring the state back the resetState. clk\_counter is used for delays in transState and waitState. This state machine is used to control the behavior of LTC2308 ADC.

## Simulation

The following simulation is conducted using ModelSim. The test bench files are created using IP parameter editor to properly setup the simulations. The following libraries are added to simulate the ADC and FFT block: fft\_ii\_0, altera\_lnsim\_ver, altera\_mf\_ver, FFTblock\_inst\_clk\_bfm, FFTblock\_inst\_rst\_bfm, FFTblock\_inst

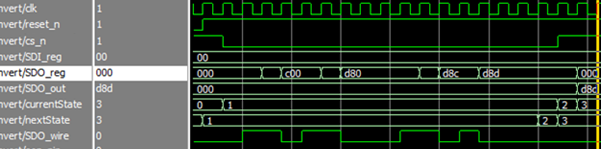


Figure 9. Cs\_n signal and data received

A module called ADCtest is created, which mimics the behavior of the ADC of receiving the SDI signal and shifting out the bits of the converted data. This module is for simulation purpose only. When the CONVST (cs\_signal) becomes low, at every rising edge of the clock, the module sends out a bit via SDO\_wire. The SDO\_reg captures that bit and shifts from the most significant bit to least significant bit. After all the 12 bits have been shifted out, at fftState, SDO\_out updates its value, which is the value of the SDO\_reg. The contents within the SDO\_reg is cleared to 0. The procedure repeats, SDO\_out updates its value at the next fftState.

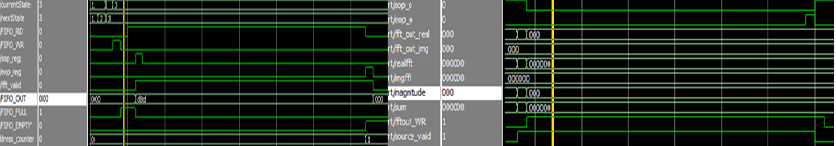


Figure 10. : Write signal and sink\_valid signal (left) and source\_valid (right)

Also, a write signal is trigger at every fftState (left figure) to write the data of the SDO\_out to the first buffer. At the next clock, the system checks whether the first buffer is full or not. If it is not full, normal operation continues. Otherwise, a buffer full signal becomes high (above figure, at the yellow cursor) to indicate the begin of feeding data to the FFT block process, where FIFO\_RD becomes high. However, the output of the buffer is delay by 1 clock process, which is considered for accurate timing. Hence, the source\_valid (fft\_valid) and sop signal only go high when the buffer begins to send output. Also, the fft\_valid signal goes low after the last data is sent out and the eop signal becomes high at the last data. This accurate timing process is to ensure that the FFT block receives full data with proper trigger signal. Then, after last data is loaded into the FFT block, address\_counter increases by 1 to change to SDI signal, which indicates the ADC to convert the signal at different channel.

When the FFT starts to output the results (right figure), source\_valid signal becomes high. Also, sop\_o and eop\_o are the two signals indicating the start and end of the output package. The real and imaginary output go through square root block to calculate the magnitude. The source\_valid signal is used to begin the writing process of the magnitude.

## Implementation

The following implementation is only for the ADC behavior, the output result from the FFT block and the magnitude that is to be plotted on the monitor. The design is combined and loaded into the DE10-Nano. The magnitude of the FFT block, source\_valid, sink\_valid is wired to the GPIO pins. Then, the analog discovery II was used to analyze the logics of the outputs, which is captured in the following figure:

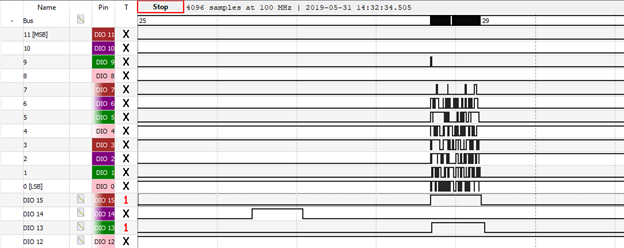
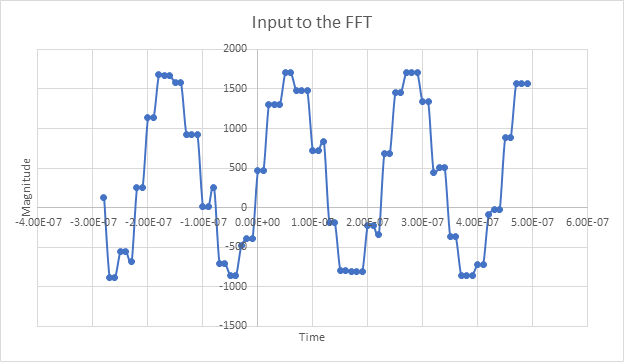


Figure 11. Input to FFT (left) and sink\_valid, source\_valid and output of FFT block (right).

The input signal was 5kHz and was given with offset voltage to ensure all the data is positive. However, the ADC has its own offset, which brings the graph down below 0. The sampling frequency was 44.1kHz, the sampled signal is quite distorted. The 12-bit bus is the magnitude, DIO 15 is source\_valid, DIO 14 is sink\_valid and DIO 13 is write enable signal of the second buffer. As first, the sink\_valid signal goes high to indicate the input data and trigger the FFT block to begin calculating. Then, output of FFT block is indicated by the source\_valid and write enable signal. The collected output of FFT block is shown below:

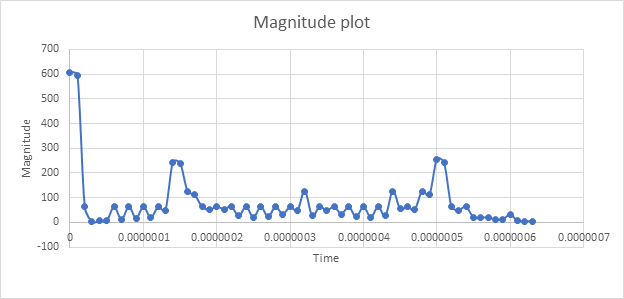


Figure 12. Magnitude of the 5 kHz signal using collected data

The input signal was 5kHz, which results in the 2 peaks at the middle. There is a peak at the beginning, which is the result of the DC offset voltage. The range of the X-axe should be from 0-44.1 kHz. In the next section, the right half of the plot is discarded, hence, the range of the X-axe is from 0 to 20kHz.

Hence, it can be confirmed that the ADC SPI communication is successfully implemented. Also, the FFT IP is properly configured to calculate the Fast Fourier Transform of a signal and store it into a buffer. Then, the data inside the buffer can be read and displayed into a monitor in the next section.

# Display driver (Phuc)

## Literature review

Digital video stream contains sequences to indicate the start and end of an active video frame, which tells the receiver when a horizontal or vertical blanking field is present and when is the video data is transmitted [video-demystified]. The video timing sequence of the transmitter is controlled by three signals vertical sync (VSYNC) and horizontal sync (HSYNC) and the display clock [5]. VSYNC synchronizes the start of a frame, HSYNC synchronizes the beginning of each horizontal scan line. DE is asserted during active video field to clock the data bus to the output. The data output contains the color space information of each pixel, either in RGB or YCbCr color-space and using 8-bit values or 10-bit values. Figure 9 dissect the video timing window of a single frame. The sync signal is a single pulse between the back and front porch, which combined make up the blanking field period.

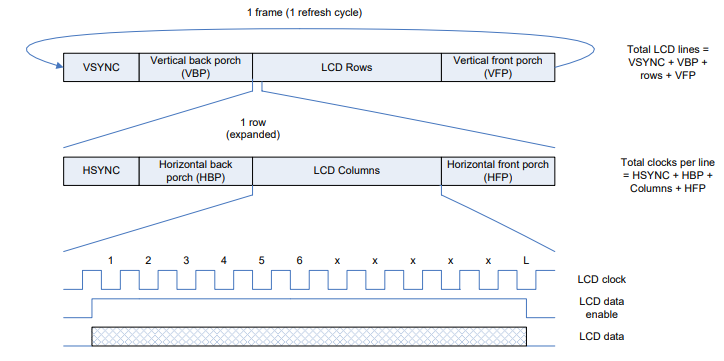


Figure 13. Video timing diagram [5]

The horizontal and vertical sync signals’ timings for each display resolution are defined by Video Electronics Standard Association (VESA). Currently, another available standard is CEA-861D [6], which follows closely the VESA timing requirements but consider also the border pixels into the blanking region.

A common communication protocol used in HDMI transmitter is I2C. The two-wire interface (SCL and SDA) allows creation of low-speed bidirectional bus between a master and a slave device. Each slave device on the bus requires a unique address, with each transmission from the master split into 1-byte packets [7].

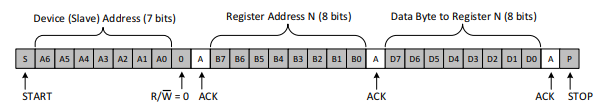


Figure 14. A typical data transmit sequence between master and slave in I2C [7]

When sending data, first the master sends a START condition (SDA transition to low while SCL is high) and transmit the slave address (7 bits) with the read/write bit. The address and subsequent bytes of data are followed by an ACK bit from the receiver. The bit indicates whether the data is successfully received so that the next byte may be sent. This requires the master to release the data line so that the slave can pull it low in the case of ACK. Once all data bytes have been transmitted a STOP condition is generated (SDA transition to high while SCL is high) [7].

## Verilog design

### Design approach

The DE10-Nano supplies the HDMI/DVI interface through the on-board ADV7513 HDMI transmitter chip from Analog Device. The module allows pixels to be transmitted to a display using the above timing standard. There are two facets to communicating with the ADV7513: setting up the internal registers to define the operational modes and parameters (DVI or HDMI, interlace or progress, aspect ratio, resolution …), then provide the video syncs and pixel data at the right intervals. For the former, an I2C bus is provided to communicate with the FPGA.

Since the video sync timing is rigid, it’s ideal to let the sync signals run continuously to refresh the screen instead of letting the signal processor’s frequency of output dictates it. For a video frame to be transmitted, the frame information needs to be written into memory and then read during the right time. With a sufficiently fast clock rate, the ADC to FFT conversion can be fast enough for 4 channels to fill up the 4 buffers during the active pixels period (when the frame in RAM is read to transmitter), then during the blanking period the magnitude values can be pulled from the buffers to generate the desired frame pattern and put into RAM again. This way, there’s guaranteed no clashing between the RAM write and read operation since they are performed during separate zones. Naturally, a sync status bus will be needed to control the start condition for conversion in the signal processor and the read/write operation of the block RAM.

The display parameters are dependent on the type of monitor used. A HP LP2275W is chosen as the base monitor for testing, which supports 4:3 resolution and DVI mode only [8]. Since the output are simple bars of color, each representing one point on the FFT spectrum, the memory space can be significantly reduced by storing each bar as a single pixel width column.

To avoid clock domain crossing problem [9], all modules are clocked with the same clock as the front-end modules, except for the sync generator which is run with the same pixel clock frequency since it operates pretty much independently from other modules.

### System block diagram

A block diagram of the display driver is presented below including the submodules and its I/O interfaces. The dashed line around the submodules indicates the whole of the display driver portion.

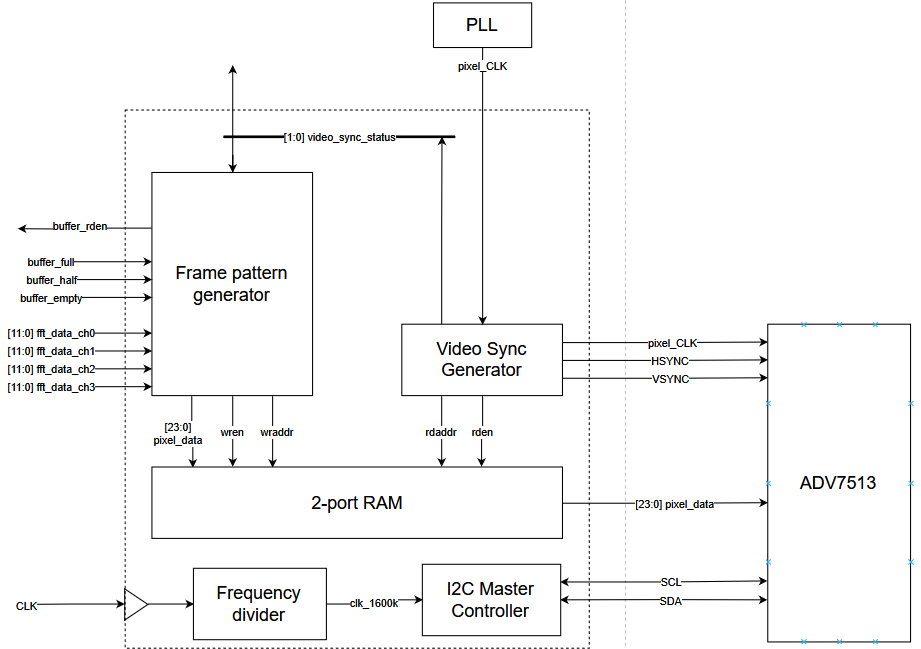


Figure 15. Display driver block diagram

### I2C interface

The I2C interface is designed based on a state machine approach. The state machine design is based loosely on the reference design from Digikey [10]. The idea is to split the transmission sequence into multiple states corresponding to different timing sections. The reference design has a very difficult to implement FSM architecture, so more states need to be padded to make the timing and logic simpler. The read functionality is also ignored since it serves no purpose.

In our implementation, each data cycle is split into 4 states which repeats until the acknowledge bit. From an **IDLE** state at startup, the I2C controller transitions to **START1** where it generates the start condition by pulling the SDA line low first and enable the tri-state buffers. The SCL line is pulled low during **ADR** state where the address is loaded to the shift register for shifting to the SDA line. Then in **OUT** state the data bit in the shift register is shifted to the SDA output. In **CLK1** and **CLK2** the SCL line is pulled high for the clock signal, then in **SHIFT** the shift register is shifted to bring the next bit to the output and the cycle repeats again to **OUT**. Once all 8 bits of data have been shifted to SDA, the last data cycle starts with **ACK1** which releases the SDA line so that the slave can acknowledge the transmission. After **CLK1** and **CLK2**, the SDA line is read for the ACK bit. If the transmission is acknowledged the register address can now be loaded for **WRITE**. If not, the **ACK2** and **STOP** generates the stop condition. Figure 12demonstrates the timing region in which each state falls into during a transmission. The state transition diagram in Quartus can be found in Appendix.

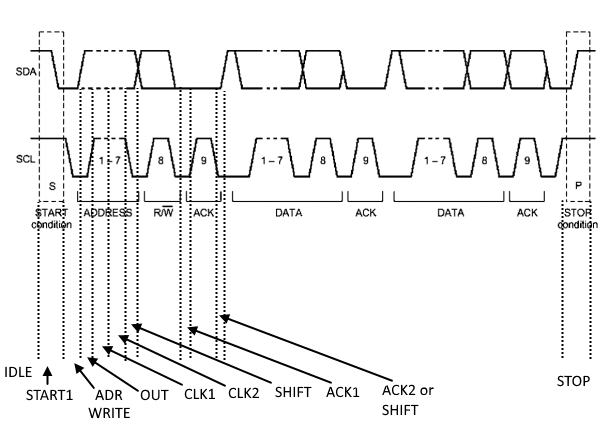


Figure 16. State transitions in time during transmission

Since each data cycle is divided into four, given a SCL rate of 400kHz (maximum of the ADV7513), the state machine needs to be clocked at four times this rate, which is 1.6MHz. This can be achieved easily with a frequency divider in the figure below instead of a PLL core to keep the clocks synchronized properly. The frequency divider divides the input clock (50MHz) by power of 2, which produces 1.56MHz when divided by 32.

Some counters are needed to time the initial startup delay, number of bytes written by the interface (3 bytes are written for each transmission) and number of commands written in total. The total number of commands is referenced from the sync generator design example for ADV7513 from Analog Devices [11]. Appendix lists the registers and the respective values required. The red highlighted line shows the key change in the command sequence. Since the chosen monitor only supports DVI mode, the register 0xAF is changed to value 0x14 to select the DVI mode. The rest of the settings are kept identical.

In the top level of the I2C master module, the controller is instantiated along with a ROM block to store the command sequence described above. A ROM can be initialized with a hex file containing the hex sequences we want to send to the ADV7513. Each word stored is 16 bits (1 byte of register address and 1 byte of register value) and the ROM contains a maximum of 32 words for the 26 commands needed.

To clock the content of the ROM to the I2C controller, another state machine is required. The FSM starts by loading the value of the ROM at initial address to the shift register, then continue to increment every time a transmission finish. Once the total number of command writes are fulfilled the state machine hangs forever until system reset.

### Timing requirements

CEA-861D timing standard defines the scan timing for progressive display with resolution 640 x 480 according to the first row in the table below [6]:

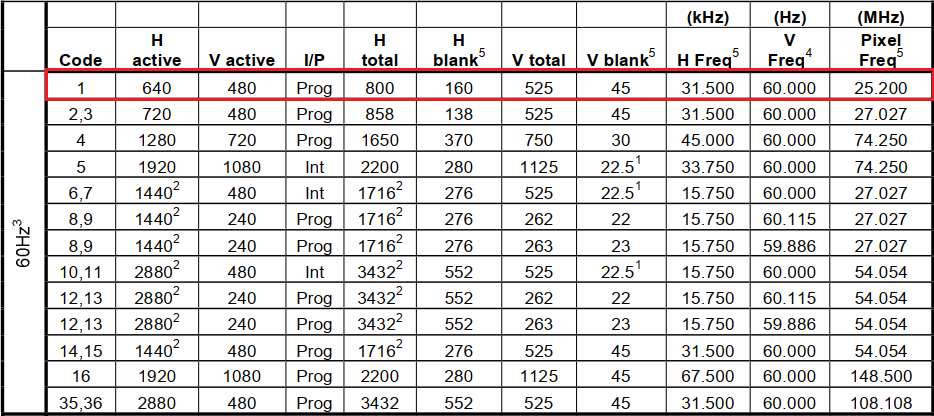


Figure 23:

Besides the general blanking, active and total pixel counts, the standard document also provides detailed timing on the blanking period, which includes the front porch width, sync pulse width, back porch width and the polarity of the sync pulse. In total we have the following values defined for the input ports:

|  |  |  |  |
| --- | --- | --- | --- |
| Vertical front porch | 10 H lines | Horizontal front porch | 16 clocks |
| Vertical sync | 2 H lines | Horizontal sync | 96 clocks |
| Vertical back porch | 33 H lines | Horizontal back porch | 48 clocks |
| Vertical active lines | 480 H lines | Horizontal active lines | 640 clocks |

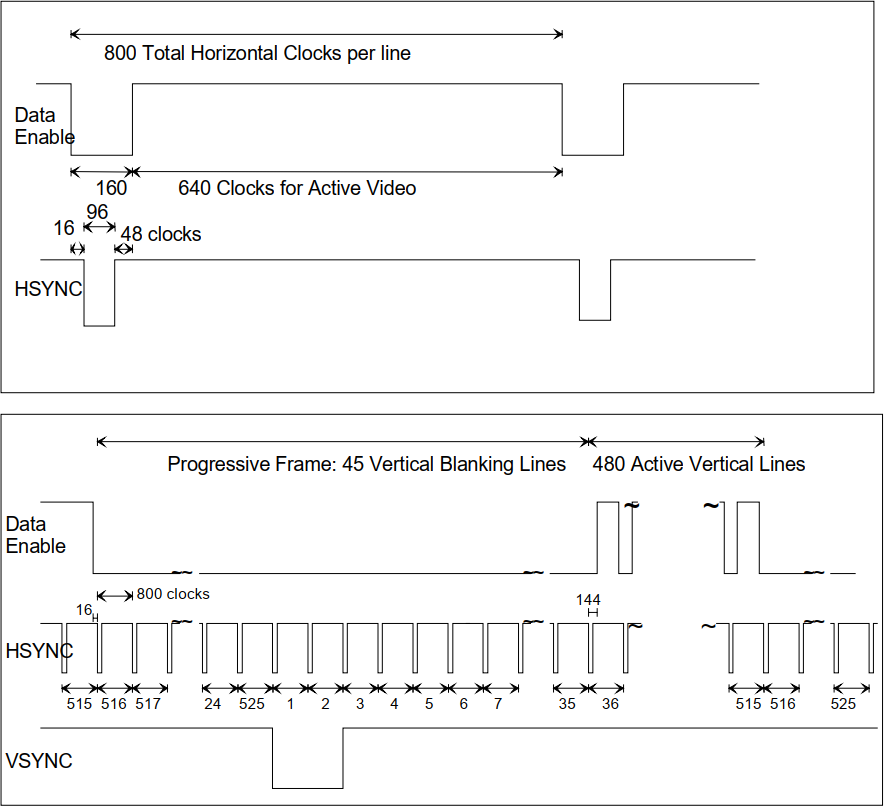


Figure 17. Timing parameters for 640 x 480p in CEA-861D document [6]

Figure **14** shows the RAM read and write window relative to the sync timing as discussed in the design approach.

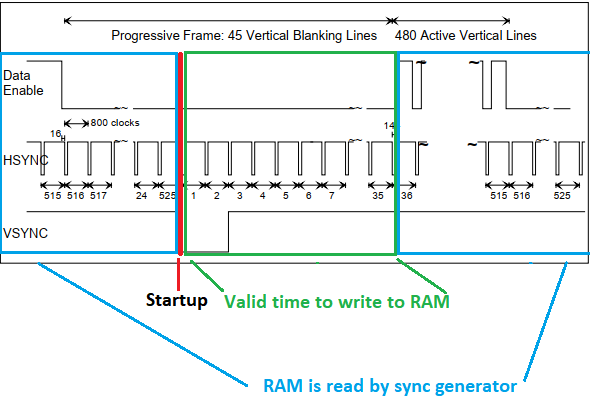


Figure . Window for RAM read and write operations highlighted with colors

### Video sync generator

To properly utilize the timing values provided, two counters are designed to count the horizontal pixels and vertical lines scanning progress. The counter’s values can be compared with the timing parameters given above to toggle the sync and data enable lines. When the counter reaches the last horizontal pixel (H\_total – 1) or vertical line (V\_total – 1), the counter is reset. For simple configurations these parameters are defined as constant input ports and with some supporting parameters.

A repeat counter is needed to count the number of pixel repetition. Considering a half spectrum of 16 points from the FFT, each point is represented as a bar on the display. Pixel width of this bar is then

Thus, the repeat counter counts to 39 then reset, at which point the RAM read address increments to the next column.

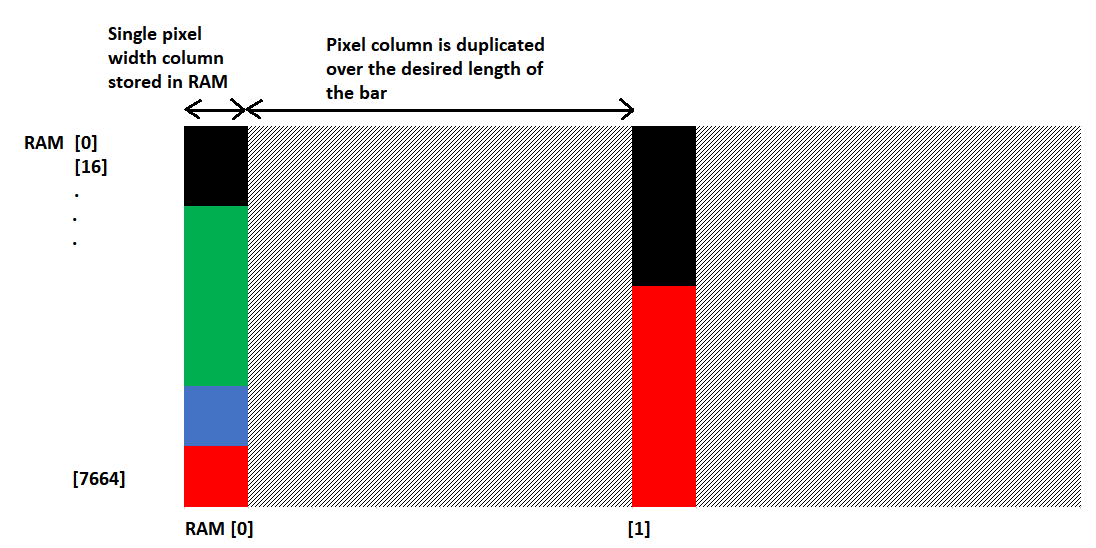


Figure 19. RAM address positioning for storing the frame

The current state of the video sync is an important information for the rest of the modules in the display driver to operate, as well as the signal processor to start converting. As a result, a vertical video status bus is extracted from module with the values corresponding to: sync (0), back porch (1), active (2) and front porch (3).

### Frame pattern generator

The pattern generator performs two tasks: reading from the FFT magnitude buffer before it and comparing the values of each channels to build a frame with the magnitude bars having the correct layering order. The generator processes the values into frames and then need to immediately write the frame to RAM, so it only operates during the vertical blanking period where the DE line is not asserted to prevent read-write clashing.

A 2-port RAM is instantiated from Altera IP Core to be used as the memory storage for each frame generated. The module allows separate read and write address which provides separate access for the sync generator and the frame generator. The input and output DFFs are clocked by the same clock and provided an asynchronous clear. The RAM stores 24-bit words with 32768 words storage size. The actual needed size is words. The extra space was initially made to allow for expanding the screen resolution afterwards, but time ran out to implement that.

The frame generator knows to read the FFT buffer by reading the buffer’s indicator signals for buffer at full, half and empty. Since all buffers are read out at the same time to compare between samples at the same spectral point from each channel, the indicators can be combined into a single indicator for simplicity. Considering the RAM read-write timing requirement, the condition for triggering the pattern generator is when all FFT buffers are full and the video sync is in the vertical blanking period. When the half full indicator is asserted, the module reads the next 16 samples without processing them since these are repeated spectrum that can be discarded. Once all the buffers are empty, the module stops until the next blanking period arrives. For each sample read, a read enable signal must be generated which will be shared between all 4 buffers. However, for the first 16 samples, each sample must be read and hold for 480 clock cycles (the vertical resolution) so that 480 pixels can be generated and put into RAM. Thus, each read enable is only asserted for a single clock cycle and then not asserted for another 479 cycles, which requires a counter to time. Another problem encountered during testing is that the reading is out of sync with the buffer output due to the flip flop delay, so the read enable signal need to be asserted before cycles where the sample is processed.

To generate each bar of FFT bins, a little more logic is required. Since we use the RAM to store the frame’s horizontal lines, when writing the color bar to RAM, the address needs to be incremented differently to ensure that the RAM addresses move in a column instead of a row. The formula is simple:

X and y counts keep track of the horizontal and vertical position of the writing process in a 16 x 480 configuration. For example: When y = 0 and x = 0, the computed RGB value is output to RAM address 0, or the top pixel of the first bar. When y = 1 and x = 0, RAM address is 16, which means the rest of the pixels of the first horizontal line has been skipped and we write to the second pixel of the first bar.

To perform the channels layering, a simple if-else priority chain will be employed. For each write cycle:

* If channel 0 (top layer) is greater than then the color of this channel (red(0) green(1) blue(2) yellow(3)) is assigned to the pixel value and written to RAM
* Due to the priority of the if-else code, the first channel to return true to the above condition will set the value of that pixel, thus forming the layering of the channels

To control all the above actions, a state machine greatly simplifies the logic. Only three states are needed: WAIT for the buffer to fill up and the blanking period to start, RUN the module to read and process the samples into frames, DISCARD the remaining half of the samples. The FSM is represented in the diagram in Appendix.

## Simulation

Firstly, the I2C master controller is tested. The figure below shows the simulation waveform for SCL and SDA behaving correctly from start condition, clocking for each bit and each byte, stop condition and spacing between each byte sent. This snapshot contains the first 2 commands in the list 0x01-0x00 and 0x02-0x18.

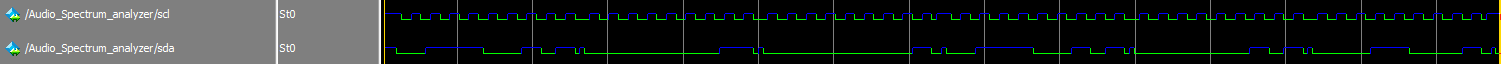


Figure 20. I2C bus in simulation

Next, the sync signals, DE and pixel clock are tested. The vertical blanking, sync pulse and porches are consistent with the parameters and the waveform illustrated in the timing requirements. The total period of a single frame is 16.67ms which is consistent with the 60Hz refresh rate of 640x480p.

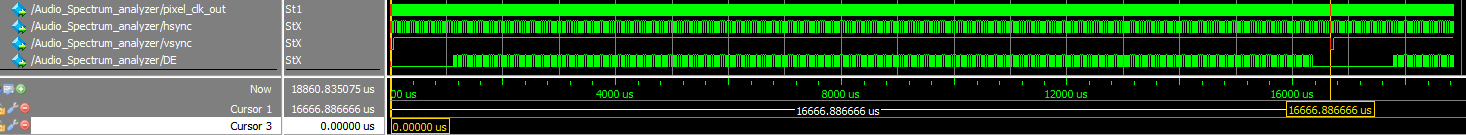


Figure 21. HSYNC, VSYNC and DE in simulation

Next, the timing between frames are tested. Here we expect the system to start the FFT conversion and fill up the 4 FFT buffers during the period (2) or active video period. The figure below shows exactly that, with 4 conversion events in “source\_valid” for each channel, the channel counter increments to convert 4 consecutive channels and the buffers are shown to be full by the end of the 4th FFT cycle. We can also see at the next VSYNC pulse, the buffers are read which lead to the buffers becoming empty. Even with four channel, it’s clear that there are plenty of space left for longer FFT window, even up to 128 points.

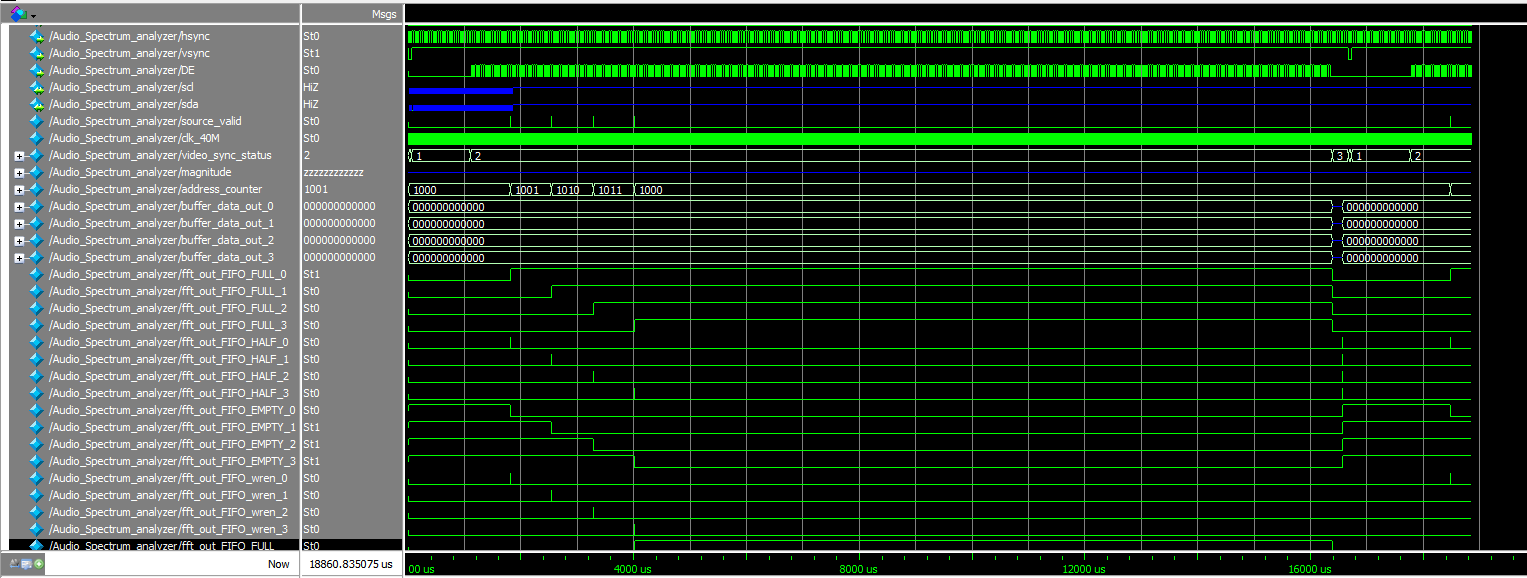


Figure 22. FFT output to buffers and buffers being read in simulation

In the same simulation, we can see that during the blanking period of the next frame, the read enable signal “fft\_out\_FIFO\_rden” is asserted multiple times to read out the buffer contents and the write enable signal “RAM\_wren” is also asserted while reading the buffers to write the computed pixel values into RAM.

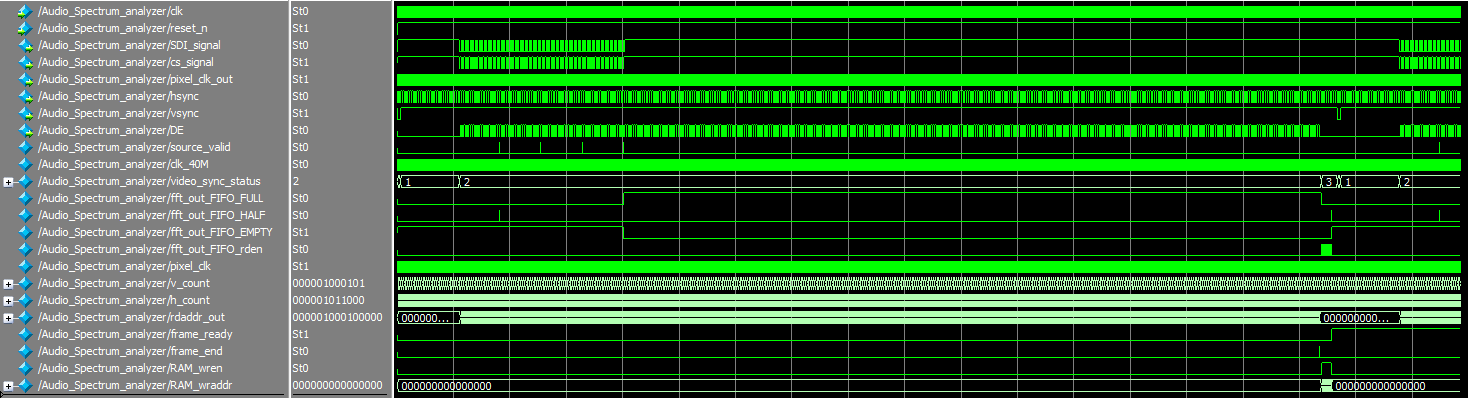


Figure 23. Writing data to RAM during blanking in simulation

# Combined system

After successfully designing and testing the two parts above, they are combined to form an audio spectrum analyzer. The system can capture the input signals from 4 channels, process the signals and display the spectrum of the signal onto a monitor.

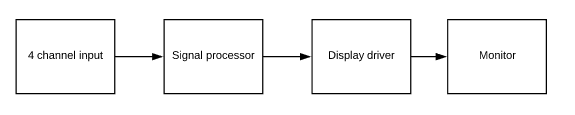


Figure 24. Full audio spectrum analyzer system block diagram

The input is generated using the Analog Discovery II. There were two input signals, which have frequencies of 10 kHz and 20 kHz. Then, the spectrum is displayed as below:



Figure 25. practical implementation of the audio spectrum analyzer

The DE10-Nano captures the input from 2 channels and processes their spectrum. Since the spectrum is discarded half, the spectrum displayed on the monitor is from 0 to 22.1 kHz from left to right. The 16 bins of the FFT are shown clearly for every channel. The red spectrum indicates to the 10 kHz signal, which has a peak as the middle. The green spectrum indicates for the 20 kHz spectrum, which has a peak at the right corner. It is noticeable that there is a high red peak on the left of the monitor. This indicates for the DC offset voltage, which is a very low frequency region.

The layering of the channels can be observed as the red bars for channel 0 lays on top the green bars which is the second channel while channel 3 and 4 are obscured since their FFT is near 0. It can be seen, however, that the bars have vertical stripes with different colors along it which was not intended. This can possibly be attributed to the data write or read timing to RAM. The colors on the stripes also contain some colors not defined in local parameter in the HDL code, so clearly some timing error has caused undefined output in RAM which creates this random color for the stripe. Otherwise, the system responds correctly for different tests: all channels connected to function generators, shifting frequency when the FGENs change their frequencies, reduced magnitude when the FGENs change their amplitude level. When the monitor is reset, the DVI signal is still received correctly and the spectrum continues to be displayed, proving the video timing signals working to design specifications.

# Conclusion

In conclusion, the project is successfully conducted. The system can capture, process the incoming signals from 4 channel, perform spectrum analysis and plot the spectrum on a monitor. The SPI communication between the DE10-Nano and its onboard LTC2308 ADC is successfully implemented. The Fast Fourier Transform is successfully implemented, which performs spectrum analysis to create the data for the display driver to plot the spectrum onto a monitor using HDMI interface. While glitches can still be observed on the display, but the various frequency and amplitude tests performed on each channel show the system performing to expectations. A lot more improvements can still be made to the FFT such as increasing the window size for more resolution, increasing the display resolution to prevent smearing from the display scaling, adding log spectrum for better peak distinguishing. Overall, given the time constraint and the complexity of the design, the outcome is satisfactory.

# Reference

[1] G. Matthews, "EEET2162/2035 – ADVANCED DIGITAL DESIGN 1 / DESIGN WITH HARDWARE DESCRIPTION LANGUAGES MAJOR PROJECT TOPIC LIST AND ASSESSMENT GUIDELINES – 2019," RMIT University, 2019.

[2] "LTC2308 - Low Noise, 500ksps, 8-Channel, 12-Bit ADC," 2007, [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/2308fc.pdf>.

[3] "Using the DE0-Nano ADC Controller," 2014.

[4] "DE10-Nano User Manual," 2017.

[5] "Introduction to graphics and LCD technologies," [Online]. Available: <https://www.nxp.com/wcm_documents/techzones/microcontrollers-techzone/Presentations/graphics.lcd.technologies.pdf>.

[6] *CEA Standard - A DTV Profile for Uncompressed High Speed Digital Interfaces: CEA-861-D*, 2006.

[7] J. Valdez and J. Becker, "Understanding the I2C Bus," 2015.

[8] "HP LP2275w and LP2475w LCD Monitors User Guide," 2008, [Online]. Available: <https://images10.newegg.com/User-Manual/User_Manual_24-176-104.pdf>.

[9] A. Tan. (2018). *Clock Domain Crossing in FPGA* [Online]. Available: <https://semiwiki.com/eda/aldec/7344-clock-domain-crossing-in-fpga/>.

[10] S. Larson. (2015). *I2C Master (VHDL)* [Online]. Available: <https://www.digikey.com/eewiki/pages/viewpage.action?pageId=10125324#space-menu-link-content>.

[11] W. Kaczurba, "ADV7511-/ADV7511W-/ADV7513-Based Video Generators ", MA, U.S.A, 2013.

# Appendix – Diagrams and commands list

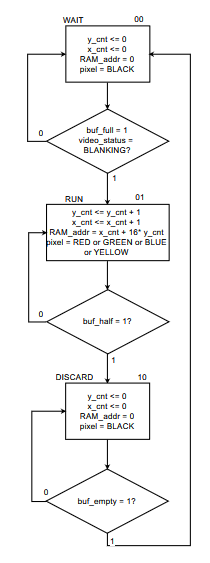


Figure 26. ASM diagram for the frame generator

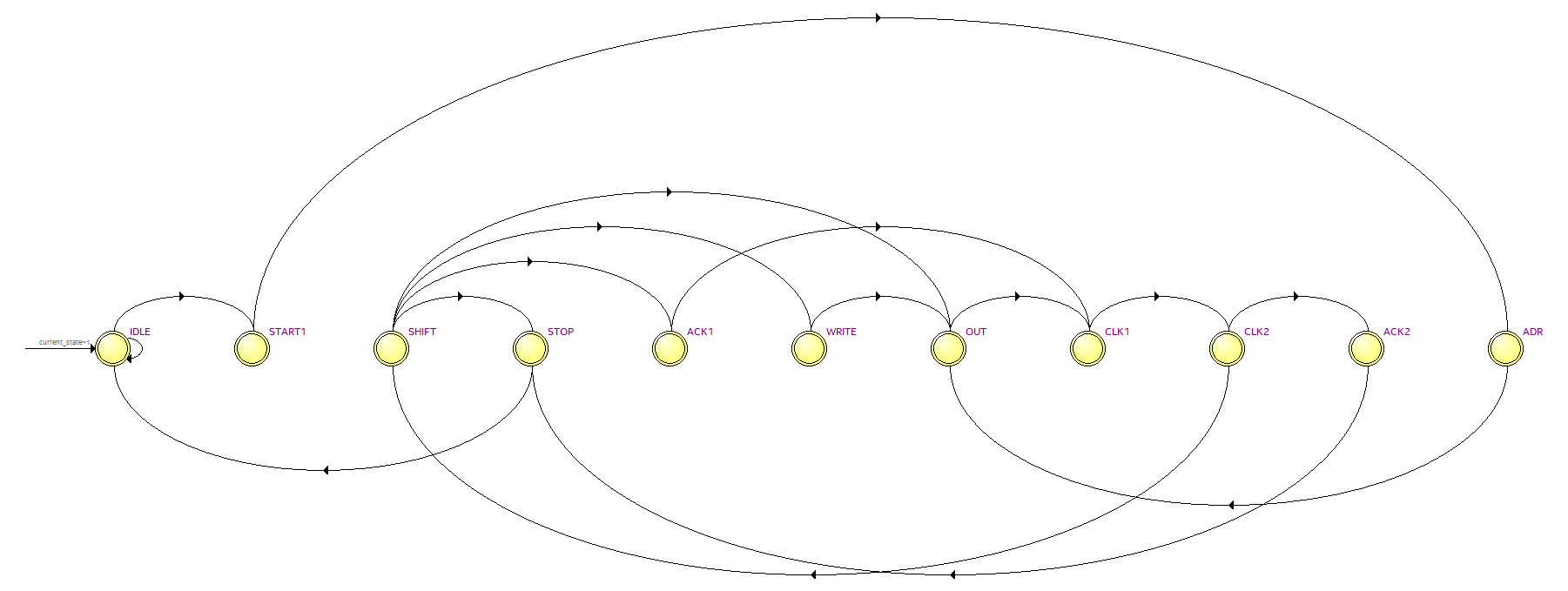


Figure 27. State transition diagram in Quartus for I2C controller

*List of commands:*

72 01 00 ; Set N Value(6144)  
72 02 18 ; Set N Value(6144)  
72 03 00 ; Set N Value(6144)  
72 15 00 ; Input 444 (RGB or YCrCb) with Separate Syncs  
72 16 61 ; 44.1kHz fs, YPrPb 444  
72 18 46 ; CSC disabled  
72 40 80 ; General Control Packet Enable  
72 41 10 ; Power Down control  
72 48 48 ; Reverse bus, Data right justified  
72 48 A8 ; Set Dither\_mode - 12-to-10 bit  
72 4C 06 ; 12 bit Output  
72 55 00 ; Set RGB444 in AVinfo Frame  
72 55 08 ; Set active format Aspect  
72 96 20 ; HPD Interrupt clear  
72 98 03 ; ADI required Write  
72 98 02 ; ADI required Write  
72 9C 30 ; ADI required Write  
72 9D 61 ; Set clock divide  
72 A2 A4 ; ADI required Write  
72 43 A4 ; ADI required Write  
**72 AF 14 ; Set DVI Mode**72 BA 60 ; No clock delay  
72 DE 9C ; ADI required write  
72 E4 60 ; ADI required Write  
72 FA 7D ; Nbr of times to search for good phase