Nortsam

NumberOneRealTimeSpectrumAnalyzerMAX

Joshua Mackler(jrm267@columbia.edu)
Tecuan Flores(tf180@columbia.edu)
Philip Coakley(pwc35@columbia.edu)

Table of Contents

1. Project Overview

The purpose of this project was to create a 31-band Audio Spectrum Analyzer. A real time audio spectrum analyzer (sometimes called an RTA) is an invaluable tool for any audiophile. The ability to visualize the relative amplitude of discrete frequency ranges affords an objective measure by which to fine-tune a hi-fi audio system. Audio Spectrum Analyzers are also very common in commercial applications for the visualization of audio signals.

Nortsam uses the SpartanIIE FPGA on the XSB-300E Board with the on-board audio and video peripherals. The on-board audio input accepts an external analog stereo line-level signal through the 1/8" mini jack. The signal undergoes Analog-to-Digital conversion with the assistance of the on-board AK4565 (audio codec). A Fast Fourier Transform is applied in order to appropriately isolate independent ranges in the frequency domain. The results of the FFT are pipelined into a graphics visualization function and then sent to the video hardware to produce a proper visualization. The end result is a dynamic VGA bar graph style display.

The FPGA itself is used to implement the computations of the stereo FFT, and video frame buffer hardware. The display output consists of 31 discrete bands, each ranging 1/3 octave. Each band corresponds to a third octave segment on the range from 20 Hz to 20 KHz.

2. Project Design

2.1. The Hopeful

To achieve our goal, a suitable design plan was constructed. In our original Project Proposal, audio signal from the audio source is fed to the stereo-in port, where it is processed by the audio codec. Once processed, the sampled audio data is buffered by the FPGA and sent to the Fast Fourier Transform (FFT) for further processing. In the meantime, a second buffer is filling with new audio data to be processed by the FFT upon completion of the previous data.

The determination on how to process the audio required much background research. First, a simple Fast Fourier Transform (FFT) with 32 bins was thought to be sufficient. Unfortunately, since our desired bands are based on an exponential scale (the 1/3 octave scale), and FFT precision needs to be higher in order for the lower frequencies to be distinctive, it turned out that many more FFT bins would be needed. The implications are, even if a moderately large FFT were used, of size 512, each component of the FFT result (FFT bins), of which there are 256 due to mirroring in the frequency domain, would correspond to roughly 100 Hz, causing the first 7 frequency bands to be calculated from a single FFT result bin. Conversely, the final band would have about 40 bins associated with its' range.

To solve this problem, a staged approach was decided upon. Instead of calculating all the bands in one FFT, band segments could be calculated separately. It is known that the audio codec samples at the rate of 48 kHz. If a FFT of size 256 is used, the bottom 128 bins would be related to the positive frequency range of 0 to 24 kHz, or half the sampling rate. Knowing the center frequency for each of the bands, it was found that the accuracy of the FFT would only be suitable for approximately the top

12 bands. At the thirteenth band, less than one bin separates each band from the previous, resulting in increasing inaccuracy. This would be the cutoff point where a new FFT would begin, which contains only the nineteenth (1.25kHz band, thirteenth from top) band and below.

To reach the second FFT, two signal processing steps are needed. First, to avoid the effects of the higher frequencies, the samples would have to be filtered. The new sampling rate needs to be at least twice as high as the highest frequency sampled (Nyquist Sampling rate) to avoid aliasing. This would cause the frequency range of the new FFT to be from only 0 to 3 kHz. A second order Infinite Impulse Response Filter was designed for this purpose. Secondly, the samples would have to be downsampled, or decimated, after the filter to achieve the 3 kHz range. This translates into a 16x down-sample of the incoming samples at 48 kHz. For a 256 bin FFT, 4096 samples would be needed.

The range from 0 to 3 kHz only gives suitable accuracy down until about the 125 Hz center frequency, at about 12 Hz per bin. Although this leaves the 8 lower bands still inaccurate, any additional down-sampling would increase the number of samples needed to unacceptable limits. For example, to have more than one bin separating 20 Hz from 25 Hz, 128x downsample is needed, which mean 32768 samples are needed. At this rate it would take close to a second for there to be enough data to even begin calculating the audio spectrum.

Once all the necessary bins are calculated, a suitable average will be made for each band according to the appropriate bins, and the results will be displayed on the screen. For the display, previous lab setups will help in achieving the correct timing to display images.

2.2. The Actual

As can be found in many cases, the boundary between the theoretical and the attainable can be difficult to cross. Many design choices gave way to more practical and attainable goals. Our preliminary design was composed largely of hardware blocks which would do the majority of the signal processing, leaving the software end solely for the visualization aspects. Our design plan quickly changed when we realized the magnitude and complexity of work needed in such an elaborate hardware design. As an alternative, we chose to perform the majority of the control aspects in signal processing with the Microblaze processor, using hardware only for the most computationally expensive tasks. The software approach yielded to a much simpler interface with the SRAM, so those design aspects were avoided in hardware. Additionally, the software helped to keep track of the large sample buffers, and to group the resulting data.

Unfortunately, one of the more disappointing gaps between the optimistic and actual was the modular decomposition of the FFT. Even though all the necessary operations were decided upon for the filter and down-sampling, the task of getting a working FFT outweighed the further design option of using a filtering process. To compensate for the loss in accuracy, an FFT of size 2048 is used. Even so, lower bars have less accuracy. Even given additional time, the implementation of the filter would have been difficult, since the project was rapidly approaching its' limits: time constraint limit and the memory area limit.

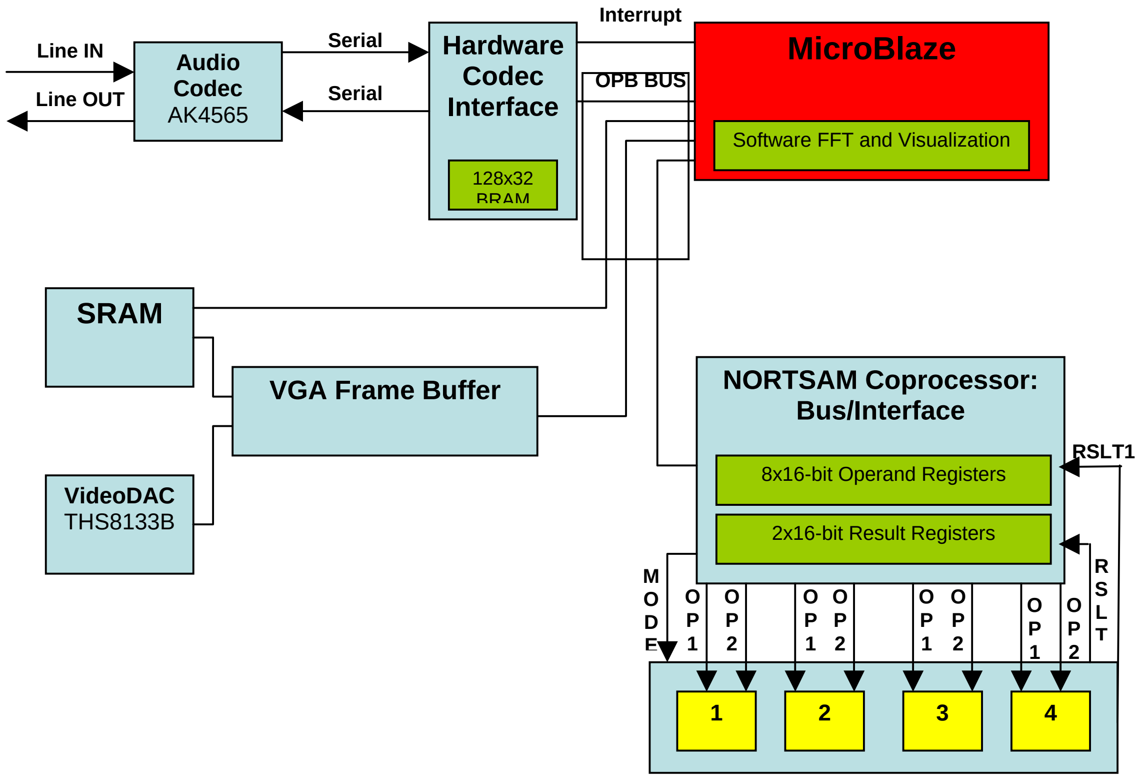
With the 2048 size FFT, there was still slight overlap between the lower frequency bands. Since the lower bars relate to such few bins, and in two cases the same bins, in the final result we changed the corresponding frequency ranges for the lower bands from the logarithmic scale in order to ensure distinctness in the visualization. The following table summarize the frequency ranges for the corresponding bin number:

Band	Corresponding FFT Bin(s)	Frequency Range (Hz)
0	0	0
1	1	0-23
2	2	23-70
3	3	70-93
4	4	93-117
5	6	117-140
6	7	140-164
7	8	164-187
8	9	187-210
9	10	210-234
10	11	234-257
11	12-13	257-304
12	14-16	304-375
13	17-20	375-468
14	21-25	468-585
15	26-31	585-726
16	32-39	726-914
17	40-49	914-1148
18	50-61	1148-1429
19	62-77	1429-1804
20	78-97	1804-2273
21	98-122	2273-2859
22	123-153	2895-3585
23	154-194	3585-4546
24	195-244	4546-5718
25	245-306	5718-7171
26	307-387	7171-9070
27	388-487	9070-11414
28	488-612	11414-14344
29	613-773	14344-18117
30	774-973	18177-22805

For every 2048 samples collected, 1216 samples are dropped while waiting for a free buffer from the FFT.

3. Design Components

3.1. Hardware Block Diagram



3.2. Hardware Description

a. Audio Codec Interface

The codec interface hardware generates three clock signals: master (12.5MHz), serial (1.5625MHz), and frame (48.828kHz). The AK4565 codec chip outputs a serial bitstream of sample data, synchronized to these clocks. The sample bitsteam is fed bit-by-bit into a BRAM module. Once 64 stereo samples have been collected (16 bits per sample-channel), the interface generates an Interrupt signal and swaps BRAM buffers to continue collecting samples.

The interface is memory-mapped to base address 0xFEFE0000 on the OPB bus. When a write is performed from software (in support of audio output), the interface prefetches the incoming sample at the corresponding address. A 32-bit value is returned upon the next read to the interface's address space, representing one stereo amplitude sample at $\Delta t=(1/48.828khz)$.

The interface was implemented in VHDL by Christian Soviani, with minor modifications made by us to scale input data and configure the output as a 'thru' jack.

b. Nortsam Coprocessor

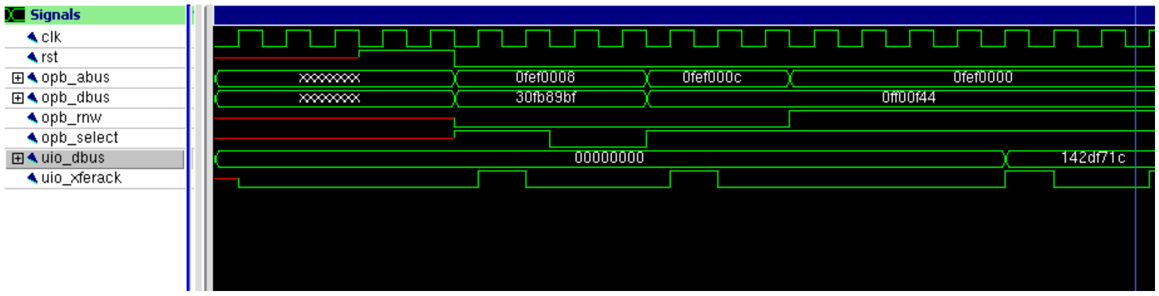
The Nortsam Coprocessor could also be known as the DSP Workhorse. The upper level module controls the interface aspects of communication through the OPB with the software level, as well as latching the appropriate values according to the function need, for further calculations in the sub-modules. The second level down controls the handshaking needed to interface with the computational units, as well as formatting the data appropriately for the upper level to receive. Once the calculations have been made, the second level sends the results, along with a done signal.

Inside the second state, there are four identical special purpose functional units, which can be assigned one of two modes. The first mode is a fixed point multiply which multiplies a signed 16 bit vector, formatted in any particular matter, and another signed 16 bit vector, with all 16 bits to the right of the binary point. The returned result is a 16 bit vector, with the same format as the first input. This function is especially important when data inside the FFT must be multiplies by a fixed coefficient which ranges from -1 to 1. The second mode takes two 16 bit signed vectors, with the same binary point, and returns a 16 bit signed value which represents their multiple. The binary point lies at whichever point that is twice as much over on the right as it is in the input format. This mode is important for the magnitude calculation necessary at the end of the butterfly output in the FFT. Although the output is signed, a 16 bit unsigned output is returned when calculating the magnitude. In both these modes, some precision is lost, since a potential 31 bit result is packed into 16 bits, but the overall loss is not significant enough to effect the values later on, since all the most significant bits are kept.

One important function the middle state handles is retrieving the result from these four sub-modules, and formatting the result appropriately. For the magnitude calculation this is fairly simple, since the module need only take two of the output and send them as output, with the first result being the real part of the magnitude calculation, and the second result being the imaginary part of the calculation. The actually summing of these results is handled in software, where saturation is additionally considered. For the data and coefficient multiplies, the job of this module is a bit more complex, effectively performing a complex multiply. The result of the real calculation is sent on result A, and the imaginary is sent on result B.

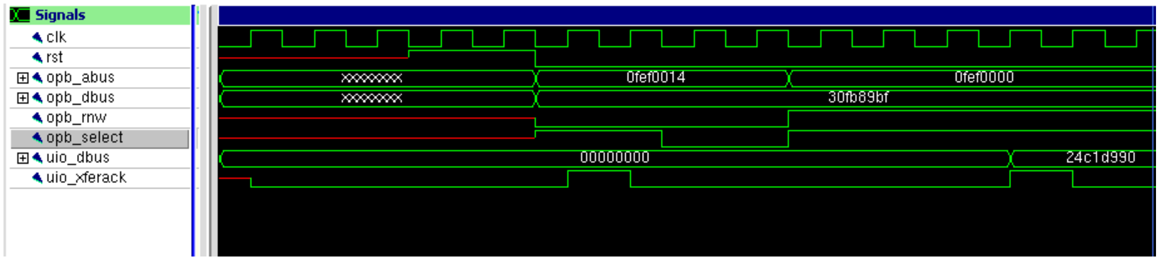
The communication scheme of the Coprocessor with the Microblaze follows the standard OPB protocol with perfecting of the results. This means that once data is written to the Coprocessor, it latches the return data even before the Microblaze requests it. This avoids some delay inherent in the reading data process, allowing the result to be returned in fewer clock cycles.

When the first mode is needed, the OPB_ABus upper bits correspond to the upper bit mask of the Coprocessor (0x0FEF), and the lower eight bits have the address 0x08. Since two 32 bit writes are needed to have enough data for a complex multiply, the Coprocessor has to wait for an additional write by the Microblaze to address 0x0c before it allows the computations to take place. This order in written addresses is fixed. If 0x0c is written first, the submodule will just use the previous value written on 0x08, so the software must keep to this ordered convention. Once both addresses are written, the submodules compute the result, and send a ready signal to the upper module to let it know to latch the result. Once the Microblaze request a read to the base address, the module passes the result on the UIO_DBus, regardless of the lower bits of the address sent by the Microblaze. Below is a sample timing diagram of the communication process across the OPB.



As can be seen, the result is latched to the output only seven cycles after the second write is acknowledged. In previous versions of the multiplier, only four cycles were needed, but due to timing delay constraints, the pipeline had to be widened. Even with the additional cycles needed to communicate between the software and hardware interfaces, this is a vast improvement of the alternative, where the multiplies would be performed in software.

The second mode follows many of the same conventions as the first mode. The one main difference is only one write is needed to begin computations. As long as the first write is to an address with the bottom eight bits of 0x14, the second mode will run. The result is prefetch, and latched to the UIO_DBus once a read is requested of the appropriate base address, regardless of the lower bits of the address. The timing diagram is below.



The amount of time taken for computation is seven cycles, which is the same as mode one. This is because mode one and mode two share the same functional units and data path in order to reduce area.

3.3. *Software Description*

a. The FFT

The Fast Fourier Transform (FFT) is a Discrete Fourier Transform (DFT) algorithm which reduces the number of computations needed for N points from N^2 to N(*Log*N). The idea behind the FFT is the *divide and conquer* approach, to break up the original N point sample into two N/2 sequences. A series of smaller problems is easier to solve than one large one. The normal DFT requires $(N-1)^2$ complex multiplications and N(N-1) complex additions as opposed to the FFT's approach of breaking it down into a series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal.[1]

The format of the FFT written for the MicroBlaze comes from Numerical Recipes in C[2], with some changes to make it better suited for a join software/hardware interface. After loading the 2048 audio samples from the audio codec into an array of 4096 (to represent real and imaginary values) the array is passed to the FFT function for computation. The FFT modifies the array during computation, ultimately returning the same array propagated with the correct real and imaginary results.

One of the major changes made from the traditional C FFT was moving all the multiplications to hardware, resulting in fewer writes and reads (two writes and one read) for complex multiplies. The other major change was pre-computing the coefficients used during the FFT calculation. Calculating the coefficients and storing them in memory prevented avoided the costly repetitive computations made in the original FFT.

b. Magnitude Calculation

Once the FFT propagates the array with the real and imaginary frequency representation of the original signal, the magnitude must be computed for each resulting bin. For this, the software writes a 32 bit value, which contains the real and imaginary values, to the hardware, which then calculates the square of each value and returns them after a 32 bit read is made. The software then adds these values together to get the square of the magnitude, accounting for any saturation that

[1] http://www.spd.eee.strath.ac.uk/~interact/fourier/fft/fftideas.html#whyFFT
[2] Numerical Recipes in C: The Art Of Scientific Computing, Cambridge University Press., 1992

might occur by having the result being greater than 16 bits.

c. Visualization

Due to the nature of the octave scale, the higher frequency bands correspond to more FFT bins than the lower frequency bands. The NORTSAM product has an array of the corresponding FFT bins for each frequency band. The max value for each FFT bin is used for the band's magnitude display to the screen. For the lower frequency bands, where there is only one FFT bin corresponding to the magnitude, the one bin was used as the magnitude.

As mentioned in the magnitude section above, the result returned from the magnitude function is a 32 bit result of the summed squared values of the real and imaginary parts of the FFT results. To bypass using a computationally complex square root calculation, the square root was compensated for in the visualization.

There are 31 frequency bands according to our design specs. Each band can has 182 vertical points of precision corresponding to max value of 33124. There is a constant array storing 0 to 33124 (182 squared). A binary search was implemented to find the best approximation within this array for the square root of the number searched for. The resultant index in the squared array is the approximate magnitude of the queried value.

The magnitude, bin number, and stereo component (left or right) are all passed to the drawBar method, drawing the bar to the screen. The method uses the above listed parameters to calculate the VGA memory location to write to. The color of each component of the bar is calculated on the distance away from the 0 bar, providing the white to red color effect.

A history is kept for each of the frequency bars to smooth the visualization. This method can be commented out to demonstrate the jerky and instantaneous response of the FFT to the audio input. With a linear decrement in bars, the display is more visually enjoyable and similar to competing commercial products.

In combination with the history for each bar, a max was computed for aesthetics. The placement of this bar was calculated in a similar manner to the magnitude bar.

4. <u>Benchmarks</u>

-With hardware, with stereo visualization, 2 FFT's of size 2048 – 500 iterations

First Trial	-	38.7099838sec
Second Trial	-	38.6399570sec
Average	-	77ms per iteration = 12.92 iterations per second

-Stereo FFTs without video, in hardware, with interrupts – 500 iterations

First	-	26.99987
Second	-	27.00011
Average	-	54ms per iteration = 18.5 iterations per second

-Stereo FFTs without video, in hardware, with interrupts disabled – 500 iterations

First	-	26.009817
Second	-	25.999864
Average	-	52ms per iteration = 19.2 iterations per second

5. Work Distribution

5.1. Joshua Mackler

a. **Fixed Point Multiplier in Hardware**

b. **Magnitude Calculator in Hardware**

c. **Coefficient Generation for FFT**

5.2. Tecuan Flores

a. **Implementation of initial FFT in software**

b. **Visualization Graphics**

5.3. Philip Coakley

a. **Audio input with Audio Codec**

b. **Hardware/Software bus interfaces**

c. **Final implementation of FFT with Hardware interaction**

6. Future Advice

As with any project, it is important to start planning the project early and set up a timeline of deliverables and milestones. Once again, we learned this the hard way ☺. Project planning is extremely important as implementation time is directly dependent on the thoroughness of the planning.

It is also important to distribute the work amongst the team members. Delegation of responsibilities early in the project life reduces dead time and duplication of work. Nortsam can attribute its success to the efficient distribution of the work that was later compiled to construct the final project.

7. VHDL Code

7.1. NORTSAM Coprocessor

a. Nortsam Bus Interface - *opb_xsb300_nortsam.vhd*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

--library UNISIM;
--use UNISIM.VComponents.all;

entity opb_xsb300_nortsam is
  generic (
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector(31 downto 0) := X"0FEF_0000";
    C_HIGHADDR      : std_logic_vector(31 downto 0) := X"0FEF_FFFF"
    );

    Port (
      OPB_Clk : in std_logic;
      OPB_Rst : in std_logic;
      OPB_ABus : in std_logic_vector (31 downto 0);
      OPB_BE : in std_logic_vector (3 downto 0);
      OPB_DBus : in std_logic_vector (31 downto 0);
      OPB_RNW : in std_logic;
      OPB_select : in std_logic;
      OPB_seqAddr : in std_logic;

      UIO_DBus : out std_logic_vector (31 downto 0);
      UIO_errAck : out std_logic;
      UIO_retry : out std_logic;
      UIO_toutSup : out std_logic;
      UIO_xferAck : out std_logic;
      Interrupt : out std_logic
      );
end opb_xsb300_nortsam;

architecture Behavioral of opb_xsb300_nortsam is

component nortsam_coprocessor
  port      (
    CLK : in std_logic;
    RST : in std_logic;
    done : out std_logic;
    interrupt : out std_logic;
    rdy : in std_logic;
    mode : in std_logic_vector(1 downto 0);
    rA, rB, rC, rD, rE, rF, rG, rH : in std_logic_vector(15 downto 0);
    resultA, resultB : out std_logic_vector(15 downto 0)
    );
end component;
```

```
signal cs,cs_1, xfer, xfer_1, xfer_2 : std_logic;

signal rnw : std_logic;
signal addr : std_logic_vector (15 downto 0);
signal wdata : std_logic_vector (31 downto 0);

signal rdata : std_logic_vector (31 downto 0);
signal opb_ad_ce : std_logic;

signal we : std_logic;

signal rA, rB, rC, rD, rE, rF, rG, rH : std_logic_vector(15 downto 0);
signal resA, resB : std_logic_vector(15 downto 0);
signal res_rdy : std_logic;
signal res_rdy_1 : std_logic := '0';
signal rdy : std_logic := '0';
signal mode : std_logic_vector(1 downto 0) := "00";
begin

process(OPB_Rst, OPB_Clk)
begin

-- register adresses, data write, rnw
   if OPB_Rst = '1' then
          addr <=  X"0000";
          wdata <= X"0000_0000";
          rnw <= '0';
      elsif OPB_Clk'event and OPB_Clk ='1' then
          if opb_ad_ce = '1' then
                         wdata <= OPB_DBus;
                         addr <= OPB_ABus(15 downto 0);
                                 rnw <= OPB_RNW;
                   end if;
      end if;

-- register data read
   if OPB_Rst = '1' then
                  rdata <= X"0000_0000";
      elsif OPB_Clk'event and OPB_Clk ='1' then
                   if res_rdy  = '1' then
                           rdata(31 downto 16) <= resA;
                           rdata(15 downto 0) <= resB;
                        end if;

      end if;

end process;

-- very important
-- TO DO
-- when writing, the read data can corrupt the DBus
UIO_DBus <= rdata when (xfer or xfer_1 or xfer_2) = '1' and rnw = '1'
else X"0000_0000";

cs <= OPB_Select when OPB_ABus(31 downto 16)=X"0FEF" else '0';
```

```
-- combinational logic for BRAM
we <= (xfer or xfer_1) and not rnw;


opb_ad_ce <= not xfer;

mode <= "01" when OPB_ABus(7 downto 0) = X"14" else "00" when
OPB_ABus(7 downto 0) = X"08" or OPB_ABus(7 downto 0) = X"0C";

-- the 1st ff -- FDR
process(OPB_Clk)
begin
      if OPB_Clk'event and OPB_Clk ='1' then

          if (xfer or xfer_1) = '1' then xfer <='0';
                elsif OPB_RNW = '0' then
                  xfer <= cs;
                else
                  xfer <= res_rdy and cs;
                end if;

      end if;
end process;

process (OPB_Rst, OPB_Clk)
begin
      if OPB_Rst = '1' then
                 xfer_1 <= '0';
                 xfer_2 <= '0';
                 cs_1 <= '0';
        elsif OPB_Clk'event and OPB_Clk ='1' then
             xfer_1 <= xfer;
             xfer_2 <= xfer_1;
                 cs_1 <= cs;
        end if;
end process;

-- write to registers
process (OPB_Rst, OPB_Clk)
begin
  if OPB_Rst = '1' then
    rA <= "0000000000000000";
    rB <= "0000000000000000";
    rC <= "0000000000000000";
    rD <= "0000000000000000";
    rE <= "0000000000000000";
    rF <= "0000000000000000";
    rG <= "0000000000000000";
    rH <= "0000000000000000";
  elsif OPB_Clk'event and OPB_Clk='1' then
    if rdy = '1' then
      rdy <= '0';
    end if;
    if (xfer and we) = '1' then

      case addr(7 downto 0) is
        when X"08" =>
```

```
          -- FFT: writing Jr & Ji
          rA <= wdata(31 downto 16);
          rE <= wdata(31 downto 16);
          rC <= wdata(15 downto 0);
          rG <= wdata(15 downto 0);
         -- mode <= "00";
        when X"0C" =>
          -- FFT: writing Cr & Ci
          rB <= wdata(31 downto 16);
          rH <= wdata(31 downto 16);
          rD <= wdata(15 downto 0);
          rF <= wdata(15 downto 0);
          rdy <= '1';
        when X"14" =>
          rA <= wdata(31 downto 16);
          rB <= wdata(31 downto 16);
          rC <= wdata(15 downto 0);
          rD <= wdata(15 downto 0);
          rdy <= '1';
          -- mode <= "01";
        when others => null;
      end case;
    end if;
  end if;
end process;

--rdy <= all4 when mode = "00" else '1' when mode = "01" else '0';

-- tie unused to ground
UIO_errAck <= '0';
UIO_retry <= '0';
UIO_toutSup <= '0';
UIO_xferAck <= xfer;

nortsam : nortsam_coprocessor port map(
      CLK => OPB_Clk,
        RST => OPB_Rst,
        mode => mode,
      interrupt => interrupt,
        rdy => rdy,
        rA => rA,
        rB => rB,
        rC => rC,
        rD => rD,
        rE => rE,
        rF => rF,
        rG => rG,
        rH => rH,

        resultA => resA,
        resultB => resB,
        done => res_rdy
);

end Behavioral;
```

b. Nortsam Coprocessor – nortsam_coprocessor.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--library UNISIM;
--use UNISIM.VComponents.all;


entity nortsam_coprocessor is
  port      (
    CLK : in std_logic;
    RST : in std_logic;

    interrupt : out std_logic;
    rdy : in std_logic;
    mode : in std_logic_vector(1 downto 0);
    rA, rB, rC, rD, rE, rF, rG, rH : in std_logic_vector(15 downto 0);
    resultA, resultB : out std_logic_vector(15 downto 0);

    done : out std_logic

    );
end nortsam_coprocessor;


architecture Behavioral of nortsam_coprocessor is
component fixed_16x16_multi is
 port (
    data    : in  std_logic_vector(15 downto 0);
    coeff   : in  std_logic_vector(15 downto 0);
    rdy     : out std_logic;
    product : out std_logic_vector(15 downto 0);
    clk     : in std_logic;
    rst     : in std_logic;
    mode    : in std_logic_vector(1 downto 0));
end component;


signal resA, resB, resC, resD : std_logic_vector(15 downto 0);
signal done1 :std_logic := '0';
signal done2 :std_logic := '0';
signal done3 :std_logic := '0';
signal done4 :std_logic := '0';
signal not_mag : std_logic;
signal reel, im : std_logic_vector(15 downto 0);
signal re_rdy : std_logic := '0';
signal im_rdy : std_logic := '0';
begin

mul1 : fixed_16x16_multi port map(
  clk => CLK,
  data => rA,
  coeff => rB,
  rdy => done1,
  product => resA,
  rst => rdy,
  mode => mode
  );

mul2 : fixed_16x16_multi port map(
  clk => CLK,
  data => rC,
  coeff => rD,
  rdy => done2,
  product => resB,
  rst => rdy,
  mode => mode
  );


mul3 : fixed_16x16_multi port map(
  clk => CLK,
  data => rE,
  coeff => rF,
  rdy => done3,
  product => resC,
  rst => not_mag,
  mode => mode
  );

mul4 : fixed_16x16_multi port map(
  clk => CLK,
  data => rG,
  coeff => rH,
  rdy => done4,
  product => resD,
  rst => not_mag,
  mode => mode
  );



-- generate the 3 clocks: master, serial, frame

 process(CLK, RST)
 begin
   if clk'event and clk='1' then
     if rdy = '1' then
       im_rdy <= '0';
       re_rdy <= '0';
     else
       if (done1 and done2) = '1' and mode = "00" then
         reel <= resA - resB;
         re_rdy <= '1';
       elsif (done1 = '1' and mode ="01") then
         reel <= resA;
         re_rdy <= '1';
       end if;
       if (done3 and done4) = '1' and mode = "00" then
         im <= resC + resD;
         im_rdy <= '1';
       elsif (done2='1' and mode ="01") then
         im <= resB;
         im_rdy <= '1';
       end if;
     end if;
   end if;
 end process;

not_mag <= rdy when mode = "00" else '0';

interrupt <= '0';

resultA <= reel;
resultB <= im;

done <= im_rdy and re_rdy;

end Behavioral;
```

c. 16-bit Fixed Point Multiplier Module – *fixed_16x16_multi.vhd*

```
----------------------------------------------------------------------
--------
-- Fixed Point Multiplier
--
-- 16 bit data contains 9 bits to left of decimal and 7 to the right
-- 16 bit coeff contains all 16 bits to right of decimal
-- output has fixed point same as data
--
-- Josh Mackler
--
----------------------------------------------------------------------
--------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fixed_16x16_multi is
  port (
    data    : in  std_logic_vector(15 downto 0);
    coeff   : in  std_logic_vector(15 downto 0);
    rdy     : out std_logic;
    product : out std_logic_vector(15 downto 0);
    clk     : in std_logic;
    rst     : in std_logic;
    mode    : in std_logic_vector(1 downto 0));

end fixed_16x16_multi;

architecture archy of fixed_16x16_multi is

  signal tmp : std_logic_vector(15 downto 0);  -- shift amt of data
  signal multip : std_logic_vector(15 downto 0) := X"0000";
  signal inter1 : std_logic_vector(15 downto 0);
  signal inter2 : std_logic_vector(15 downto 0);
  signal inter3 : std_logic_vector(15 downto 0);
  signal inter4 : std_logic_vector(15 downto 0);
  signal inter5 : std_logic_vector(15 downto 0);
  signal inter6 : std_logic_vector(15 downto 0);
  signal inter7 : std_logic_vector(15 downto 0);
  signal inter8 : std_logic_vector(15 downto 0);
  signal inter9 : std_logic_vector(15 downto 0);
  signal inter10 : std_logic_vector(15 downto 0);
  signal inter11 : std_logic_vector(15 downto 0);
  signal inter12 : std_logic_vector(15 downto 0);
  signal inter13 : std_logic_vector(15 downto 0);
  signal inter14 : std_logic_vector(15 downto 0);
  signal inter15 : std_logic_vector(15 downto 0);
  signal second1, second2, second3, second4, second5 :
std_logic_vector(15 downto 0);
  signal res1 : std_logic_vector(15 downto 0);
  signal res2 : std_logic_vector(15 downto 0);
  signal res3 : std_logic_vector(15 downto 0);
  signal res4 : std_logic_vector(15 downto 0);
  signal res5 : std_logic_vector(15 downto 0);
  signal res6 : std_logic_vector(15 downto 0);
  signal res7 : std_logic_vector(15 downto 0);
  signal twos_coeff : std_logic_vector(15 downto 0);
 -- signal fm_count : std_logic_vector(0 to 3) := "0000";
  signal mul_pipe : std_logic_vector(1 downto 0);
  signal negative : std_logic := '0';
  signal neg1 : std_logic := '0';
  signal neg2 : std_logic := '0';
  signal go : std_logic := '0';
  signal tmp2 : std_logic_vector(15 downto 0);
  signal ready : std_logic := '0';
begin  -- archy


   inter1 <= '0' & tmp(15 downto 1) when twos_coeff(14)='1' and
mode="00" else
             "0000000" & tmp(15 downto 7) when twos_coeff(7)='1' and
mode="01" else X"0000";

   inter2 <= "00" & tmp(15 downto 2) when twos_coeff(13)='1' and
mode="00" else
              "000000" & tmp(15 downto 7) & '0' when twos_coeff(8)='1'
             and mode = "01" else X"0000";

   inter3 <= "000" & tmp(15 downto 3) when twos_coeff(12)='1' and
mode="00" else
              "00000" & tmp(15 downto 7) & "00" when twos_coeff(9)='1'
             and mode = "01" else X"0000";

   inter4 <= "0000" &  tmp(15 downto 4) when twos_coeff(11)='1' and
mode="00" else
              "0000" & tmp(15 downto 7) & "000" when twos_coeff(10)='1'
and mode="01"
             else X"0000";

   inter5 <=  "00000" & tmp(15 downto 5) when twos_coeff(10)='1' and
mode="00" else
               "000" & tmp(15 downto 7) & "0000" when
twos_coeff(11)='1' and mode="01"
              else X"0000";

   inter6 <=  "000000" & tmp(15 downto 6) when twos_coeff(9)='1' and
mode="00" else
              "00" & tmp(15 downto 7) & "00000" when twos_coeff(12)='1'
and mode="01"
              else X"0000";

   inter7 <=  "0000000" & tmp(15 downto 7) when twos_coeff(8)='1' and
mode="00" else
```

```
               "0" & tmp(15 downto 7) & "000000" when
twos_coeff(13)='1' and mode="01"
              else X"0000";

  inter8 <=  "00000000" & tmp(15 downto 8) when twos_coeff(7)='1' and
mode="00" else
              tmp(15 downto 7) & "0000000" when twos_coeff(14)='1' and
mode="01"
             else X"0000";
   inter9 <=  "000000000" & tmp(15 downto 9) when twos_coeff(6)='1' and
mode="00" else X"0000";
   inter10 <=  "0000000000" & tmp(15 downto 10) when twos_coeff(5)='1'
and mode="00" else X"0000";
   inter11 <=  "00000000000" & tmp(15 downto 11) when twos_coeff(4)='1'
and mode="00" else X"0000";
   inter12 <=  "000000000000" & tmp(15 downto 12) when
twos_coeff(3)='1' and mode="00" else X"0000";
   inter13 <=  "0000000000000" & tmp(15 downto 13) when
twos_coeff(2)='1' and mode="00" else X"0000";
   inter14 <=  "00000000000000" & tmp(15 downto 14) when
twos_coeff(1)='1' and mode="00" else X"0000";
   inter15 <=  "000000000000000" & tmp(15) when twos_coeff(0)='1' and
mode="00" else X"0000";

  process (clk, rst)
  begin  -- process
    if clk'event and clk='1' then
      if rst = '1' then
        if data(15) = '1' then
          tmp <= not(data) + 1;
          neg1 <= '1';
        else tmp <= data;
             neg1 <= '0';
        end if;
        if coeff(15) = '1' then
          neg2 <= '1';
          twos_coeff <= not(coeff) + 1;
        else
          twos_coeff <= coeff;
          neg2 <= '0';
        end if;
        multip <= X"0000";
        mul_pipe <= "00";
        ready <= '0';
        go <= '1';
      else
        if go = '1' then
          if mul_pipe = "00" then
            res1 <= inter1 + inter2;
            res2 <= inter3 + inter4;
            res3 <= inter5 + inter6;
            res4 <= inter7 + inter8;

           res5 <= inter9 + inter10;
            res6 <= inter11 + inter12;
            res7 <= inter13 + inter14;
            mul_pipe <= "01";
          elsif mul_pipe = "01" then       -- second clock
            second1 <= res1 + res2 + res3 + res4;
            second2 <= res5 + res6 + res7 + inter15;
            mul_pipe <= "11";
          elsif mul_pipe = "11" then       -- third clock
              if negative = '1' then product <= not(second1 + second2)
+ 1;
            else
              product <= second1 + second2;
            end if;
            go <= '0';
              ready <= '1';
          end if;
        end if;


      end if;
    end if;
  end process;

   rdy <= ready;
   negative <= neg1 xor neg2;
end archy;
```

7.2. AK4565 Audio Codec Interface

a. Bus Interface – opb_xsb300_ak4565.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;


library UNISIM;

use UNISIM.VComponents.all;


entity opb_xsb300_ak4565 is
  generic (
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector(31 downto 0) := X"FEFE_0000";
    C_HIGHADDR      : std_logic_vector(31 downto 0) := X"FEFE_FFFF"
    );

    Port (  OPB_Clk : in std_logic;
                      OPB_Rst : in std_logic;
                      OPB_ABus : in std_logic_vector (31 downto 0);
                      OPB_BE : in std_logic_vector (3 downto 0);
                      OPB_DBus : in std_logic_vector (31 downto 0);
                      OPB_RNW : in std_logic;
                      OPB_select : in std_logic;
                      OPB_seqAddr : in std_logic;

                      UIO_DBus : out std_logic_vector (31 downto 0);
                      UIO_errAck : out std_logic;
                      UIO_retry : out std_logic;
                      UIO_toutSup : out std_logic;
                      UIO_xferAck : out std_logic;


                      Interrupt : out std_logic;


            AU_CSN_N : out std_logic;
            AU_BCLK : out std_logic;
            AU_MCLK : out std_logic;
            AU_LRCK : out std_logic;
            AU_SDTI : out std_logic;
            AU_SDTO0 : in std_logic
                     );
end opb_xsb300_ak4565;

architecture Behavioral of opb_xsb300_ak4565 is


component audio_ak4565 port   (
         CLK : in std_logic;

            RST : in std_logic;


            audio_bram_clk : out std_logic;

            audio_bram_addr_dac : out std_logic_vector(11 downto 0);

            audio_bram_addr_adc : out std_logic_vector(11 downto 0);


            audio_bram_dacdata : in std_logic;

            audio_bram_adcdata : out std_logic;


            interrupt : out std_logic;

            audio_fifohalf : out std_logic;


            AU_CSN_N : out std_logic;
      AU_BCLK : out std_logic;
      AU_MCLK : out std_logic;
      AU_LRCK : out std_logic;
      AU_SDTI : out std_logic;
      AU_SDTO0 : in std_logic
        );


end component;

signal cs, xfer, xfer_1, xfer_2 : std_logic;

signal rnw : std_logic;
signal addr : std_logic_vector (15 downto 0);
signal wdata : std_logic_vector (31 downto 0);


signal rdata : std_logic_vector (31 downto 0);
signal opb_ad_ce : std_logic;



signal we, a0, ce0, ce1 : std_logic;

signal bram_rdata, bram_wdata : std_logic_vector(15 downto 0); -- as
the CPU sees them

signal bram_rdata_rev, bram_wdata_rev : std_logic_vector(15 downto 0);
-- reversed !
```

```
signal bram_addr : std_logic_vector(7 downto 0);



signal audio_bram_clk : std_logic;

signal audio_bram_addr_dac, audio_bram_addr_adc : std_logic_vector(11
downto 0);

signal audio_bram_dacdata : std_logic;

signal audio_bram_adcdata : std_logic;

signal audio_bram_dacdata_v : std_logic_vector(0 downto 0);

signal audio_bram_adcdata_v : std_logic_vector(0 downto 0);



signal audio_fifohalf : std_logic;


begin

process(OPB_Rst, OPB_Clk)
begin

-- register adresses, data write, rnw
   if OPB_Rst = '1' then
          addr <=  X"0000";
          wdata <= X"0000_0000";
          rnw <= '0';
      elsif OPB_Clk'event and OPB_Clk ='1' then
          if opb_ad_ce = '1' then

                         wdata <= OPB_DBus;

                         addr <= OPB_ABus(15 downto 0);
             rnw <= OPB_RNW;

                   end if;
      end if;


-- register data read

   if OPB_Rst = '1' then

                  rdata <= X"0000_0000";
      elsif OPB_Clk'event and OPB_Clk ='1' then

                   if(ce0='1') then rdata(15 downto 0) <= bram_rdata;
end if;

                   if(ce1='1') then rdata(31 downto 16) <= bram_rdata;
end if;
      end if;
end process;


-- very important

-- TO DO

-- when writing, the read data can corrupe the DBus

      UIO_DBus <= rdata when xfer = '1' else X"0000_0000";




cs <= OPB_Select when OPB_ABus(31 downto 16)=X"FEFE" else '0';



-- the 1st ff -- FDR

process(OPB_Clk)

begin

      if OPB_Clk'event and OPB_Clk ='1' then

            if (xfer or xfer_1) = '1' then xfer <='0'; else xfer <= cs;
end if;

      end if;

end process;



process (OPB_Rst, OPB_Clk)

begin

      if OPB_Rst = '1' then
       xfer_1 <= '0';

             xfer_2 <= '0';
   elsif OPB_Clk'event and OPB_Clk ='1' then

             xfer_1 <= xfer;

             xfer_2 <= xfer_1 and not rnw;
      end if;
end process;

-- combinational logic for BRAM
we <= (xfer or xfer_1) and not rnw;
ce0 <= xfer_1 and not rnw;
```

```
ce1 <= xfer_2;
a0 <= xfer_1;
opb_ad_ce <= not xfer;

bram_addr <= (not audio_fifohalf) & addr(7 downto 2) & a0;
bram_wdata <= wdata(31 downto 16) when a0='0' else wdata(15 downto 0);

-- tie unused to ground
UIO_errAck <= '0';
UIO_retry <= '0';
UIO_toutSup <= '0';
UIO_xferAck <= xfer;


-- instantiate the BRAMS
process (bram_rdata_rev) begin
        bram_rdata(0) <= bram_rdata_rev(8);
        bram_rdata(1) <= bram_rdata_rev(7);
        bram_rdata(2) <= bram_rdata_rev(6);
        bram_rdata(3) <= bram_rdata_rev(5);
        bram_rdata(4) <= bram_rdata_rev(4);
        bram_rdata(5) <= bram_rdata_rev(3);
        bram_rdata(6) <= bram_rdata_rev(2);
        bram_rdata(7) <= bram_rdata_rev(1);
        bram_rdata(8) <= bram_rdata_rev(0);
        -- sign extend
        bram_rdata(9) <= bram_rdata_rev(0);
        bram_rdata(10) <= bram_rdata_rev(0);
        bram_rdata(11) <= bram_rdata_rev(0);
        bram_rdata(12) <= bram_rdata_rev(0);
        bram_rdata(13) <= bram_rdata_rev(0);
        bram_rdata(14) <= bram_rdata_rev(0);
        bram_rdata(15) <= bram_rdata_rev(0);
--    for i in 6 to 15 loop
--          bram_rdata(i)<=bram_rdata_rev(21-i);
--    end loop;
end process;

process(bram_wdata) begin
      for i in 0 to 15 loop
            bram_wdata_rev(i)<=bram_wdata(15-i);
      end loop;
end process;

audio_bram_dacdata <= audio_bram_dacdata_v(0);
dac_bram : RAMB4_S1_S16 port map (
              DIA    => "0" ,
        ENA    => '1',
              WEA    => '0',
        RSTA   => '0',
              CLKA   => audio_bram_clk,
              ADDRA  => audio_bram_addr_dac,
              DOA    => audio_bram_dacdata_v,

              DIB    => bram_wdata_rev,
        ENB    => '1',
        WEB    => we,
              RSTB   => '0',
        CLKB   => OPB_Clk,
        ADDRB  => bram_addr,
        DOB    => open
);

audio_bram_adcdata_v(0) <= audio_bram_adcdata ;
adc_bram : RAMB4_S1_S16 port map (
              DIA    => audio_bram_adcdata_v,
        ENA    => '1',
              WEA    => '1',
        RSTA   => '0',
              CLKA   => audio_bram_clk,
              ADDRA  => audio_bram_addr_adc,
              DOA    => open,

              DIB    => X"0000",
        ENB    => '1',
        WEB    => '0',
              RSTB   => '0',
        CLKB   => OPB_Clk,
        ADDRB  => bram_addr,
        DOB    => bram_rdata_rev
);


-- the sound stuff



audio : audio_ak4565 port map(



      CLK => OPB_Clk,

   RST => OPB_Rst,



      audio_bram_clk => audio_bram_clk,

      audio_bram_addr_dac => audio_bram_addr_dac,

      audio_bram_addr_adc => audio_bram_addr_adc,

      audio_bram_dacdata =>   audio_bram_dacdata,

      audio_bram_adcdata => audio_bram_adcdata,


      interrupt => interrupt,

      audio_fifohalf => audio_fifohalf,
      AU_CSN_N => AU_CSN_N,
   AU_BCLK => AU_BCLK,
   AU_MCLK => AU_MCLK,
   AU_LRCK => AU_LRCK,
   AU_SDTI => AU_SDTI,
   AU_SDTO0 => AU_SDTO0

);


end Behavioral;
```

b. Codec Interface Module – audio_ak4565.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


library UNISIM;

use UNISIM.VComponents.all;



entity audio_ak4565 is



port  (

         CLK : in std_logic;

            RST : in std_logic;


            audio_bram_clk : out std_logic;

            audio_bram_addr_dac : out std_logic_vector(11 downto 0);

            audio_bram_addr_adc : out std_logic_vector(11 downto 0);


            audio_bram_dacdata : in std_logic;

            audio_bram_adcdata : out std_logic;
            interrupt : out std_logic;

            audio_fifohalf : out std_logic;



            AU_CSN_N : out std_logic;
      AU_BCLK : out std_logic;
      AU_MCLK : out std_logic;
      AU_LRCK : out std_logic;
      AU_SDTI : out std_logic;
      AU_SDTO0 : in std_logic
        );
end audio_ak4565;





architecture Behavioral of audio_ak4565 is



component RAMB4_S1_S16
--
  generic (
       INIT_00 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_01 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_02 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000";
       INIT_03 : bit_vector :=
X"0000000000000000000000000000000000000000000000000000000000000000"

             );
--
  port (DIA    : in STD_LOGIC_VECTOR (0 downto 0);
        DIB    : in STD_LOGIC_VECTOR (15 downto 0);
        ENA    : in STD_logic;
        ENB    : in STD_logic;
        WEA    : in STD_logic;
        WEB    : in STD_logic;
        RSTA   : in STD_logic;
        RSTB   : in STD_logic;
        CLKA   : in STD_logic;
        CLKB   : in STD_logic;
        ADDRA  : in STD_LOGIC_VECTOR (11 downto 0);
        ADDRB  : in STD_LOGIC_VECTOR (7 downto 0);
        DOA    : out STD_LOGIC_VECTOR (0 downto 0);
        DOB    : out STD_LOGIC_VECTOR (15 downto 0));
end component;



signal clkcnt : std_logic_vector(4 downto 0);
signal audiocnt_dac, audiocnt_adc : std_logic_vector(11 downto 0);


signal audio_clk : std_logic;

signal lrck : std_logic;


begin


AU_CSN_N <= '1'; -- no chip select as we don't write the ctrl regs


-- generate the 3 clocks: master, serial, frame


process(CLK, RST)

begin

      if rst = '1' then

            clkcnt <= "00000";

      elsif clk'event and clk='1' then

            clkcnt <= clkcnt + 1;

      end if;

end process;

AU_MCLK <= clkcnt(1); -- master clock, 12.5 MHz

audio_clk <= clkcnt(4); -- this is the serial clock, 1.5625 Mhz

AU_BCLK <= not audio_clk; -- dont't ask but read AK4565 specs


process(audio_clk, RST)

begin

      if rst = '1' then

      audiocnt_dac <= "000000000000";

      audiocnt_adc <= "111111111111";
      lrck <= '0';

      elsif audio_clk'event and audio_clk='1' then

            audiocnt_dac <= audiocnt_dac + 1;

            audiocnt_adc <= audiocnt_adc + 1;

            lrck <= not audiocnt_dac(4);

      end if;

end process;

AU_LRCK <= lrck; -- audio clock, 48.828 kHz


interrupt <= not audiocnt_dac(10);

audio_fifohalf <= audiocnt_dac(11);


audio_bram_addr_dac <= audiocnt_dac;

audio_bram_addr_adc <= audiocnt_adc;

audio_bram_clk <= audio_clk;


--AU_SDTI <= audio_bram_dacdata;

AU_SDTI <= AU_SDTO0;

audio_bram_adcdata <= AU_SDTO0;


end architecture;
```

8. <u>C Code</u>

8.1. main.c

```
/***  FFT size is defined by the following preprocessor
defines/includes:
 *  main.c:
 *     #define FFT_SIZE
 *     #define BUFFER_SHIFT_OFFSET [=log2(FFT_SIZE)]
 *  isr.c:
```

```
 *     #define FFT_SIZE
 *  fft.c:
 *     #define FFT_SIZE
 *     #include "coeff(FFT_SIZE).c"
 *     Xuint16 nonswaps[] - must correspond to correct fft size
 *  graphics.c:
 *     Xuint16 correspond[] - must to bin correlations for correct fft
size
 ***/

#include "xbasic_types.h"
#include "xio.h"

#include "xintc_l.h"
#include "xuartlite_l.h"


#define W 640
#define H 480
#define VGA_START 0x00800000
#define AUDIO_START 0xFEFE0000
#define RED 0xE0
#define GREEN 0x1C
#define BLUE 0x03

#define FFT_SIZE 2048
#define BUFFER_SHIFT_OFFSET 11  // must =log2(FFT_SIZE), used for
double buffer

// defined in isr.c
extern void audio_intr_handler(void *callback);
extern volatile Xuint16 bufInUse[];
extern volatile Xuint16 bufReady[];
//extern Xuint32 droppedPackets, grabbedPackets, midFrameDrops;

Xuint32 audioBuffer[ FFT_SIZE*2 ];
Xuint32 audioBufferRIGHT[ FFT_SIZE ];
Xint16 *startLPtr, *startRPtr;

/*
 * setup_interrupts: Initialize the interrupt sources and handlers
 */
void setup_interrupts()
{
  /*
   * Reset the interrupt controller peripheral
   */

  /* Disable the interrupt signal */
  XIntc_mMasterDisable(XPAR_INTC_SINGLE_BASEADDR);
  /* Disable all interrupt sources */
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,0);
  /* Acknowledge all possible interrupt sources
     to make sure none are pending */
  XIntc_mAckIntr(XPAR_INTC_SINGLE_BASEADDR, 0xffffffff);

  /*
```

```
   * Install the codec interrupt handler
   */
  XIntc_InterruptVectorTable[XPAR_INTC_AK4565_INTERRUPT_INTR].Handler =
    audio_intr_handler;

  // enable interrupt sources
  /* Enable CPU interrupts */
  microblaze_enable_interrupts();
  /* Enable interrupts from the interrupt controller */
  XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);
  /* Tell the interrupt controller to accept interrupts from the codec
*/
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,
XPAR_AK4565_INTERRUPT_MASK);
}



int main()
{
  Xuint16 i, j, k, z;
  Xint32 x;

  print("[number one realtime spectrum analyzer max].\n\r");

  // Enable the instruction cache: makes the code run 6 times faster
  microblaze_enable_icache();

  setup_interrupts();

  //  clear screen
  for(x=0 ; x<H*W ; x++)
    XIo_Out8(VGA_START + x, 0);

  // this never changes, define outside loop.
  startRPtr = (Xint16*)(audioBufferRIGHT) - 1;

  i=0;
  for ( ;; ) {
    // ******************************
    // double buffer logistics
    while (bufReady[i] == 0)
      print("."); // need this delay, a busy-wait loop prevents isr
context switch?!
    bufInUse[(i+1) % 2] = 0;
    bufInUse[i] = 1;
    // ******************************


    // fudge starting pointers to account for FFT starting
    //  with index 1.
    startLPtr = (Xint16*)(&(audioBuffer[ (i<<BUFFER_SHIFT_OFFSET) ])) -
1;

    bitReverseAndLRSeparate( startLPtr, startRPtr );
    // after this point:
```

```
    //   audioBuffer[] holds bit-rearranged LEFT samples,
MSB=real/LSB=imag
    //   audioBufferRIGHT[] holds bit-rearranged RIGHT samples,
MSB=real/LSB=imag

    doFFT( startLPtr );
    doFFT( startRPtr );

    visualize( &(audioBuffer[ (i<<BUFFER_SHIFT_OFFSET) ]), 0 );
    visualize( audioBufferRIGHT, 1 );

    if (i==0) i++;
    else
      i=0;
  }

  return 0;
}
```

8.2. isr.c

```
#include "xbasic_types.h"
#include "xio.h"
#include "xparameters.h"
#include "xuartlite_l.h"

extern Xuint32 audioBuffer[];
extern Xuint32 audioBufferRIGHT[];
Xint32 bufPos = 0;
Xuint16 volatile bufInUse[] = {0, 0};
Xuint16 volatile bufReady[] = {0, 0};

// define: packet = set of 64 samples; frame= set of FFT_SIZE samples

/*
Xuint32 droppedPackets=0;
Xuint32 grabbedPackets=0;
Xuint32 midFrameDrops=0;
*/

#define FFT_SIZE 2048

/*
 * Interrupt service routine for the AK4565 CODEC
 */

void audio_intr_handler( void *callback ) {
  Xint16 i;

  // if the FFT isn't done yet, drop these samples.
  if (bufPos < FFT_SIZE) {
    if (bufInUse[0] == 1) {
      /*
      droppedPackets++;
      if (bufPos != 0)
      midFrameDrops++;
      */
      return;
    }
  }
  else if (bufInUse[1] == 1) {

    //    droppedPackets++;
    if (bufPos != FFT_SIZE)
      //      midFrameDrops++;
    return;
  }

  for (i=0; i < 64; i++) {
    XIo_Out32( XPAR_AK4565_BASEADDR + (i<<2), 0 );
    audioBuffer[bufPos++] = (XIo_In32( XPAR_AK4565_BASEADDR )); //
0xLLLLRRRR
  }

  if (bufPos == FFT_SIZE) {
    bufReady[0] = 1;
    bufReady[1] = 0;
  }
  else if (bufPos >= (FFT_SIZE * 2)) { // multiply here OK -- should
optimize to constant value
    bufReady[1] = 1;
    bufReady[0] = 0;
    bufPos = 0;
  }

  //  grabbedPackets++;
}
```

8.3. fft.c

```
#include "xbasic_types.h"
#include "xio.h"
#include "coeff2048.c"  // coefficients must be calculated for FFT size

#define FFT_SIZE 2048

extern Xuint32 coeff[];


/***********************************************************************
******/
// nonswaps[] identifies indices that do not get affected during
bitswapping.
//            index 0 holds the total # values in array


/*
const Xuint16 nonswaps256[] =
   { 17,
     1,   49,  73,  121, 133, 181, 205, 253, 259, 307, 331, 379,
     391, 439, 463, 511, 513 };
```

```
const Xuint16 nonswaps1024[] =
{ 33,
  1, 97, 145, 241, 265, 361, 409, 505, 517, 613, 661, 757, 781, 877,
925, 1021,
  1027, 1123, 1171, 1267, 1291, 1387, 1435, 1531, 1543, 1639, 1687,
1783,
  1807, 1903, 1951, 2047, 2049 };
*/

const Xuint16 nonswaps[] =
{ 65,
  1,    65,   161,  225,  273,  337,  433,  497,  521,  585,  681,
745,  793,
  857,  953,  1017, 1029, 1093, 1189, 1253, 1301, 1365, 1461, 1525,
1549, 1613,
  1709, 1773, 1821, 1885, 1981, 2045, 2051, 2115, 2211, 2275, 2323,
2387, 2483,
  2547, 2571, 2635, 2731, 2795, 2843, 2907, 3003, 3067, 3079, 3143,
3239, 3303,
  3351, 3415, 3511, 3575, 3599, 3663, 3759, 3823, 3871, 3935, 4031,
4095, 4097 };
/**********************************************************************
******/


/**
 * takes buffer data in dataL, in form 0xLLLLRRRR
 * separates L and R data and performs bit reversal of indices
 * returns two arrays, each ready to be FFT-ized
 * ---note: as with the FFT algorithm, this function will never access
idx[0]
 * fills all imaginary (even) indices with 0x0000.
 */
void bitReverseAndLRSeparate( Xint16 *dataL, Xint16 *dataR ) {
  int i, j, n, m;
  Xint16 tempL, tempR;

  n = FFT_SIZE << 1;
  j = 1;

  for ( i=1; i < n; i+=2 ) {
    if ( j > i ) {

      tempL = dataL[j];
      tempR = dataL[j+1];

      dataL[j] = dataL[i];
      dataR[j] = dataL[i+1];

      dataL[i] = tempL;
      dataR[i] = tempR;

      dataL[i+1] = 0x0000;
      dataL[j+1] = 0x0000;
      dataR[i+1] = 0x0000;
      dataR[j+1] = 0x0000;
    }

    m = FFT_SIZE;
    while ( (m >= 2) && (j > m) ) {
      j -= m;
      m >>= 1;
    }

    j += m;
  }

  // separate L and R for the indices that don't get bitreversed
  for (i=1; i < nonswaps[0]; i++) {
    dataR[ nonswaps[i] ] = dataL[ nonswaps[i]+1 ];

    dataL[ nonswaps[i]+1 ] = 0x0000;
    dataR[ nonswaps[i]+1 ] = 0x0000;
  }
}


// takes an array of size (FFT_SIZE * 2), of 16-bit values
// this function WILL NEVER access array index 0 (very important-no seg
faults)
void doFFT( Xint16 *data ) {
  //  int istep, mmax, j, i, m, n, coeff_ofs, z;
  Xint32 istep, mmax, j, i, m, n, coeff_ofs, z;
  Xint32 itr = 0;

  Xint16 *tempRI;
  Xuint32 temp;
  Xuint32* temp32;

  coeff_ofs = 0;

  n = FFT_SIZE << 1;
  mmax = 2;

  while ( n > mmax ) { // this, the outer loop, gets executed
log2(FFT_SIZE) times.
    istep = mmax << 1;

    for ( m=1; m<mmax; m+=2 ) {
      for ( i=m; i<=n; i+=istep ) {
      j = i + mmax;

      itr++;
      // And now, the guts of the whole project, the butterfly
operation:
      // (aka Danielson-Lanczos formula)

      temp32 = (Xuint32*)(&(data[j]));
      // write real and imag values of J to multiplier
      XIo_Out32( 0x0FEF0008, *temp32 );
      // write coefficients to multiplier
      XIo_Out32( 0x0FEF000C, coeff[ coeff_ofs ] );
      temp = XIo_In32( 0x0FEF0000 );

      tempRI = (Xint16*)(&temp);

      data[j] = data[i] - tempRI[0];
      data[j+1] = data[i+1] - (tempRI[1]);
      data[i] += tempRI[0];
      data[i+1] += tempRI[1];
      }
      coeff_ofs++;
    }
    mmax = istep;
  }
}
```

8.4. graphics.c

```
#include "xbasic_types.h"
#include "xio.h"

#include "xintc_l.h"
#include "xuartlite_l.h"

#define W 640
#define H 480
#define VGA_START 0x00800000
#define X_START 40
#define Y_START 240

#define RED 0xE0
#define GREEN 0x1C
#define BLUE 0x03
#define WHITE RED | GREEN | BLUE
#define BLACK 0x00

#define MAX_PIX 200

#define TRUE 1
#define FALSE 0

/*******************************************************************/
//correspond2048:

/*  [0]   [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]*/

Xuint16 correspond[] = {
     0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
    10,   11,   13,   16,   20,   25,   31,   39,   49,   61,
    77,   97,  122,  153,  194,  244,  306,  387,  487,  612,
   773,  973};
/*******************************************************************/


static Xuint16 logScale[] = {
 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256,
289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900,
961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764,
1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916,
3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356,
4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084,
6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100,
8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000, 10201,
10404, 10609, 10816, 11025, 11236, 11449, 11664, 11881, 12100, 12321,
12544, 12769, 12996, 13225, 13456, 13689, 13924, 14161, 14400, 14641,
14884, 15129, 15376, 15625, 15876, 16129, 16384, 16641, 16900, 17161,
17424, 17689, 17956, 18225, 18496, 18769, 19044, 19321, 19600, 19881,
20164, 20449, 20736, 21025, 21316, 21609, 21904, 22201, 22500, 22801,
23104, 23409, 23716, 24025, 24336, 24649, 24964, 25281, 25600, 25921,
26244, 26569, 26896, 27225, 27556, 27889, 28224, 28561, 28900, 29241,
29584, 29929, 30276, 30625, 30976, 31329, 31684, 32041, 32400, 32761
};

Xuint16 history[] = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

Xuint16 barHistory[] = {
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

/* Method for drawing the Magnitude bar for each frequency band */
void drawBar(Xuint16 bin, int mag,  unsigned char direction) {
  Xuint32 vgaMem;
  Xint32 i;
  Xuint16 color= 0xffff, middle=0x49;
  Xint32 offset =0;
  unsigned char higher=0;
  unsigned int fourPixel = 0; //0xFF; //f0ffff;

  if (direction ==1)
    offset =30+bin;
  else
    offset = bin;

  /* Zero Out Noise */
  if (mag < 15) mag = 0;

  /* Update History if mag is higher */
  if(((history[offset]-6) < mag)) {
    //history[direction][bin] =
    barHistory[((offset))] = mag;
    history[(offset)] = mag;
    higher =1;
  } else {
    if((history[(offset)]) >6)
      history[(offset)]-= 6;  // comment this line and uncomment next
line
```

```
      //history[offset] = mag;  //    to disable slowed decay of bars
      else
        history[(offset)] =0;
      mag = history[(offset)];
    }

    /* draw black above color so screen refreshes */
    for (i=mag; i < MAX_PIX; i++) {
      if (direction)
        vgaMem = VGA_START + bin*16 + X_START + (Y_START - i-2)*640;
      else
        vgaMem = VGA_START + bin*16 + X_START + (Y_START + i+2)*640;

      XIo_Out32(vgaMem, 0x0);
      XIo_Out32(vgaMem+4, 0x0);
    }

    /* Print Middle Spacer Bar */
    for(i=0; i < 32; i++){

      fourPixel = (middle<<24);
      fourPixel = fourPixel | (middle<<16);
      fourPixel = fourPixel | (middle<<8);
      fourPixel = fourPixel | middle;

      /* Draw 3 pixel thick bar to separate Left and Right Results */
      vgaMem = VGA_START + (bin<<4) + X_START + (Y_START -1)*640 + i;
      XIo_Out32(vgaMem, fourPixel);
      XIo_Out32(vgaMem+4, fourPixel);

      vgaMem = VGA_START + (bin<<4) + X_START + (Y_START)*640 + i;
      XIo_Out32(vgaMem, fourPixel);
      XIo_Out32(vgaMem+4, fourPixel);

      vgaMem = VGA_START + (bin<<4) + X_START + (Y_START+1)*640 + i;
      XIo_Out32(vgaMem, fourPixel);
      XIo_Out32(vgaMem+4, fourPixel);
    }

    // If the Bar is Magnitude is smaller than History //
      if ( (higher==0) && (mag-6) > 0 ) {

        for (i=0 ; i < (mag-6); i++) {

        if (direction)
          vgaMem = VGA_START + (bin<<4) + X_START + (Y_START - i -
  2)*640;
        else
          vgaMem = VGA_START + (bin<<4) + X_START + (Y_START + i +
  2)*640;

        // If higher than 160, print RED //
        if(i > 160) {
          color = 224;
        }
        else { // ELSE, get color based on Magnitude //
          if((i&0x000000FF) == 4)
            color = 252;
          else if ((i&0x001f) == 16)
            color-=4;
        }

        // Set Color and Print Bar //
        fourPixel = (color<<24);
        fourPixel = fourPixel | (color<<16);
        fourPixel = fourPixel | (color<<8);
        fourPixel = fourPixel | color;
        XIo_Out32(vgaMem, fourPixel);
        XIo_Out32(vgaMem+4, fourPixel);
        }

      } else  if (higher == 1) {

        for (i=0; i < mag; i++) {

        if (direction)
          vgaMem = VGA_START + (bin<<4) + X_START + (Y_START - i -
  2)*640;
        else
          vgaMem = VGA_START + (bin<<4) + X_START + (Y_START + i +
  2)*640;
        /* If higher than 160, print RED */
        if(i > 160) {
          color = 224;
        }
        else { /* ELSE, get color based on Magnitude */
          if((i&0x000000FF) == 4)
            color = 252;
          else if ((i&0x001f) == 16)
            color-=4;
        }

        /* Set Color and Print Bar */
        fourPixel = (color<<24);
        fourPixel = fourPixel | (color<<16);
        fourPixel = fourPixel | (color<<8);
        fourPixel = fourPixel | color;
        XIo_Out32(vgaMem, fourPixel);
        XIo_Out32(vgaMem+4, fourPixel);
        }
      }

      mag = barHistory[(offset)];

      if ( (mag > 3 ) && (higher ==0) ){
        mag -= 3;
        barHistory[(offset)] = mag;

        if ( direction ==0 )
        vgaMem = VGA_START + (bin<<4) + X_START + (Y_START + mag )*640;
        else
        vgaMem = VGA_START + (bin<<4) + X_START + (Y_START - mag )*640;
      fourPixel = (middle<<24);
      fourPixel = fourPixel | (middle<<16);
      fourPixel = fourPixel | (middle<<8);
      fourPixel = fourPixel | middle;

      XIo_Out32(vgaMem, fourPixel);
      XIo_Out32(vgaMem+4, fourPixel);
      }
}

Xuint16 calcMagnitude( Xuint32 data ) {
  Xint16 *tempRI;
  Xuint32 result;
  short high = 182, low = 0, middle;
  unsigned char less = 0;
  short temp;

  // calculate magnitudes of buckets corresponding to positive
frequencies
  // hw address 0x0fef_0014 squares real and imag.

  XIo_Out32( 0x0FEF0014, data );
  result = XIo_In32( 0x0FEF0000 );

  tempRI = (Xuint16*)(&result);
  result = (Xuint32)(tempRI[0]) + (Xuint32)(tempRI[1]);

  // saturate, don't overflow
  if (result > 32767)
    result = 32750;

  while( low <= high )
    {
      middle = ( low  + high ) / 2;
      if( result == logScale[  middle ] ) //match
      return middle;
      else if( result < logScale[ middle ] ) {
      high = middle - 1;     //search low end of array
      less =1;
      }
      else {
      low = middle + 1;      //search high end of array
      less = 0;
      }
    }
  if(less == 0) {
    if (middle > 180)
      return 180;
    else
      return middle;
  }
  else
    return middle-1;
}
```

```
void visualize( const Xuint32* fftData, const unsigned char direction )
{
  Xint32 i;
  Xint32 currBin, endBin;
  Xuint16 secondHighest;
  Xuint16 tempo;
  Xuint16 binMagnitude;

  // figure out which bin magnitudes pair up with which bands

  /*[0]   [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]
     0,  0/1,    1,  1/2,    2,  2/3,    3,    4,    5,    7,
     8,   10,   13,   16,   20,   25,   31,   39,   49,   61,
    77,   97,  122,  153,  194,  244,  306,  387,  487,  612,
   773, [973] */

  // fill upper bands (bin resolution >= 1) with peak from range
  for (i=0; i<31; i++) {
    binMagnitude = 0;
    secondHighest = 0;
    endBin = correspond[i+1];

    for (currBin = correspond[i]; currBin < endBin; currBin++) {
      tempo = calcMagnitude( fftData[currBin] );
      if (tempo > secondHighest) {
      if (tempo >= binMagnitude) {
        secondHighest = binMagnitude;
        binMagnitude = tempo;
      }
      else {
        secondHighest = tempo;
      }
      }
    }

    if (i < 15)
      drawBar(i, binMagnitude, direction);
    else
      drawBar(i, secondHighest, direction);
  }

}
```

8.5. coeff2048.c

```
[font size reduced]
static                  0x30fb7640,             0x00007ffb,             0x70e13c53,             0xe70b7d7f,
Xuint32                 0x00007ffe,             0xe7097d85,             0x6a6c4719,             0xdadd7a72,
coeff[] = {             0xcf05763f,             0xcf07763c,             0x62f0512f,             0xcf0b7637,
0x7FFF0000,             0xa5805a80,             0xb8e76a68,             0x5a805a7d,             0xc3b270d8,
0x7FFF0000,             0x89c230fa,             0xa5835a7d,             0x513162ec,             0xb8ed6a63,
0x00007FFF,             0x7FFF0000,             0x95994718,             0x471a6a67,             0xaed862e8,
0x7FFF0000,             0x7d8a18f8,             0x89c630f8,             0x3c5470db,             0xa58b5a79,
0x5a825a82,             0x764130fa,             0x827e18f6,             0x30f97639,             0x9d1d512b,
0x00007fff,             0x6a6d471b,             0x7FFF0000,             0x25267a74,             0x95a34715,
0xa57f5a81,             0x5a825a80,             0x7f620c8b,             0x18f77d81,             0x8f2f3c50,
0x7FFF0000,             0x471c6a6b,             0x7d8a18f7,             0x0c8b7f58,             0x89d130f6,
0x764130fb,             0x30fb763e,             0x7a7c2526,             0x00007ff5,             0x85972524,
0x5a825a81,             0x18f87d86,             0x764030f9,             0xf3767f57,             0x828a18f6,
0x80b30c8b,        0x763c30f2,        0xa381581c,        0x68974993,        0xfb517f98,
0x7FFF0000,        0x74ff33d5,        0xa15f55d0,        0x67ad4ada,        0xf9c17f86,
0x7fd80647,        0x73b036b0,        0x9f4c5376,        0x66bf4c1e,        0xf8317f6f,
0x7f610c8a,        0x724f3982,        0x9d485110,        0x65cd4d5f,        0xf6a17f54,
0x7e9c12c6,        0x70dc3c4b,        0x9b534e9d,        0x64d74e9d,        0xf5127f34,
0x7d8918f6,        0x6f583f0b,        0x996d4c1e,        0x63de4fd8,        0xf3837f0f,
0x7c281f17,        0x6dc341c1,        0x97974993,        0x62e15110,        0xf1f57ee5,
0x7a7b2524,        0x6c1d446d,        0x95d146fd,        0x61e05245,        0xf0677eb6,
0x78822b1b,        0x6a66470e,        0x941c445c,        0x60db5376,        0xeeda7e82,
0x763f30f7,        0x689f49a4,        0x927741b0,        0x5fd254a4,        0xed4d7e49,
0x73b336b5,        0x66c84c2f,        0x90e33efa,        0x5ec655cf,        0xebc17e0b,
0x70e03c51,        0x64e14eae,        0x8f603c3b,        0x5db656f7,        0xea367dc8,
0x6dc741c8,        0x62ea5121,        0x8def3972,        0x5ca2581b,        0xe8ac7d81,
0x6a6a4716,        0x60e45388,        0x8c8f36a0,        0x5b8b593c,        0xe7237d35,
0x66cc4c38,        0x5ecf55e2,        0x8b4133c6,        0x5a705a59,        0xe59b7ce4,
0x62ee512b,        0x5cac582f,        0x8a0530e4,        0x59525b73,        0xe4147c8e,
0x5ed355ec,        0x5a7a5a6e,        0x88db2dfb,        0x58305c89,        0xe28e7c33,
0x5a7e5a78,        0x583a5c9f,        0x87c42b0a,        0x570b5d9c,        0xe1097bd4,
0x55f15ecc,        0x55ed5ec2,        0x86bf2813,        0x55e35eab,        0xdf857b70,
0x512f62e6,        0x539360d6,        0x85cd2516,        0x54b75fb6,        0xde027b07,
0x4c3b66c3,        0x512c62db,        0x84ee2213,        0x538860be,        0xdc817a99,
0x47186a60,        0x4eb864d1,        0x84221f0b,        0x525661c2,        0xdb017a27,
0x41c96dbc,        0x4c3866b7,        0x83691bfe,        0x512162c2,        0xd98279b0,
0x3c5270d4,        0x49ac688d,        0x82c318ed,        0x4fe963be,        0xd8057934,
0x36b673a6,        0x47156a53,        0x823115d8,        0x4eae64b7,        0xd68a78b4,
0x30f87631,        0x44736c09,        0x81b212bf,        0x4d7065ac,        0xd510782f,
0x2b1c7873,        0x41c66dae,        0x81460fa4,        0x4c2f669d,        0xd39877a5,
0x25257a6b,        0x3f0f6f42,        0x80ee0c86,        0x4aeb678a,        0xd2217717,
0x1f177c17,        0x3c4f70c5,        0x80aa0966,        0x49a46873,        0xd0ac7684,
0x18f67d77,        0x39857237,        0x80790645,        0x485a6958,        0xcf3975ed,
0x12c67e89,        0x36b37397,        0x805c0323,        0x470d6a39,        0xcdc87551,
0x0c8a7f4d,        0x33d874e5,        0x7FFF0000,        0x45bd6b16,        0xcc5974b0,
0x06477fc3,        0x30f57621,        0x7ffd0192,        0x446b6bef,        0xcaec740b,
0x00007fea,        0x2e0b774b,        0x7ff50324,        0x43166cc3,        0xc9817362,
0xf9ba7fc2,        0x2b197863,        0x7fe804b5,        0x41bf6d93,        0xc81872b4,
0xf3787f4b,        0x28217968,        0x7fd60646,        0x40656e5f,        0xc6b17202,
0xed3d7e86,        0x25237a5a,        0x7fbf07d7,        0x3f096f27,        0xc54c714b,
0xe70e7d73,        0x221f7b39,        0x7fa30968,        0x3daa6fea,        0xc3ea7090,
0xe0ee7c12,        0x1f167c05,        0x7f820af8,        0x3c4970a9,        0xc28a6fd1,
0xdae27a65,        0x1c087cbe,        0x7f5d0c88,        0x3ae57164,        0xc12c6f0d,
0xd4ed786d,        0x18f67d64,        0x7f330e17,        0x397f721a,        0xbfd16e45,
0xcf12762a,        0x15e07df7,        0x7f040fa6,        0x381772cc,        0xbe786d79,
0xc955739e,        0x12c67e76,        0x7ed01134,        0x36ad7379,        0xbd226ca9,
0xc3ba70cb,        0x0faa7ee2,        0x7e9712c2,        0x35417422,        0xbbce6bd4,
0xbe446db3,        0x0c8b7f3a,        0x7e59144f,        0x33d374c7,        0xba7d6afb,
0xb8f76a57,        0x096a7f7e,        0x7e1615db,        0x32637567,        0xb92f6a1e,
0xb3d666b9,        0x06487faf,        0x7dce1766,        0x30f17603,        0xb7e3693d,
0xaee462dc,        0x03257fcc,        0x7d8218f0,        0x2f7d769a,        0xb69a6858,
0xaa245ec2,        0x00017fd5,        0x7d311a79,        0x2e07772c,        0xb554676f,
0xa5995a6e,        0xfcde7fcb,        0x7cdb1c01,        0x2c8f77ba,        0xb4116682,
0xa14555e2,        0xf9bc7fad,        0x7c801d88,        0x2b167843,        0xb2d16591,
0x9d2c5121,        0xf69b7f7b,        0x7c201f0e,        0x299b78c8,        0xb194649c,
0x99504c2e,        0xf37b7f36,        0x7bbc2093,        0x281e7948,        0xb05a63a3,
0x95b3470c,        0xf05d7edd,        0x7b532217,        0x26a079c3,        0xaf2362a6,
0x925841be,        0xed427e70,        0x7ae52399,        0x25207a39,        0xadef61a6,
0x8f413c48,        0xea2a7df0,        0x7a72251a,        0x239f7aab,        0xacbe60a2,
0x8c6f36ad,        0xe7157d5d,        0x79fb2699,        0x221c7b18,        0xab915f9a,
0x89e430f0,        0xe4047cb6,        0x797f2817,        0x20987b80,        0xaa675e8e,
0x87a22b15,        0xe0f77bfc,        0x78fe2993,        0x1f137be4,        0xa9405d7f,
0x85aa251f,        0xddef7b2f,        0x78792b0e,        0x1d8d7c43,        0xa81c5c6c,
0x83fe1f12,        0xdaec7a4f,        0x77ef2c87,        0x1c067c9d,        0xa6fc5b56,
0x829e18f2,        0xd7ef795c,        0x77602dfe,        0x1a7d7cf2,        0xa5df5a3c,
0x818c12c3,        0xd4f87856,        0x76cd2f74,        0x18f37d42,        0xa4c6591f,
0x80c80c88,        0xd208773e,        0x763530e8,        0x17697d8d,        0xa3b057fe,
0x80520646,        0xcf1f7614,        0x7599325a,        0x15de7dd4,        0xa29e56da,
0x7FFF0000,        0xcc3d74d7,        0x74f833ca,        0x14527e16,        0xa18f55b2,
0x7ff60324,        0xc9637388,        0x74533538,        0x12c57e53,        0xa0845487,
0x7fd80647,        0xc6927227,        0x73a936a4,        0x11377e8b,        0x9f7d5359,
0x7fa60969,        0xc3ca70b5,        0x72fb380e,        0x0fa97ebe,        0x9e7a5228,
0x7f610c8a,        0xc10b6f32,        0x72483976,        0x0e1a7eec,        0x9d7a50f4,
0x7f080fa9,        0xbe566d9d,        0x71913adb,        0x0c8b7f15,        0x9c7e4fbc,
0x7e9b12c5,        0xbbab6bf8,        0x70d53c3e,        0x0afb7f39,        0x9b864e81,
0x7e1b15de,        0xb90a6a42,        0x70153d9f,        0x096b7f59,        0x9a924d43,
0x7d8718f4,        0xb674687b,        0x6f513efd,        0x07da7f74,        0x99a24c02,
0x7ce01c06,        0xb3ea66a4,        0x6e884059,        0x06497f8a,        0x98b64abe,
0x7c261f14,        0xb16c64be,        0x6dbb41b2,        0x04b87f9b,        0x97ce4978,
0x7b59221d,        0xaefa62c8,        0x6cea4309,        0x03277fa7,        0x96ea482f,
0x7a792521,        0xac9460c3,        0x6c15445d,        0x01957fae,        0x960a46e3,
0x7986281f,        0xaa3b5eaf,        0x6b3c45af,        0x00037fb0,        0x952e4594,
0x78802b17,        0xa7ef5c8c,        0x6a5f46fe,        0xfe727fad,        0x94564443,
0x77672e08,        0xa5b15a5b,        0x697d484a,        0xfce17fa5,        0x938242ef,
0x92b24198,         0x7ca61cba,         0x5fc7547e,         0x2bc977c1,         0xed5e7df5,
0x91e7403f,         0x7c781d7d,         0x5f415514,         0x2b0c7805,         0xec997dd7,
0x91203ee3,         0x7c491e40,         0x5eba55a9,         0x2a4f7848,         0xebd47db7,
0x905d3d85,         0x7c181f03,         0x5e32563d,         0x29917889,         0xeb0f7d96,
0x8f9e3c25,         0x7be61fc5,         0x5daa56d0,         0x28d378c9,         0xea4a7d74,
0x8ee43ac2,         0x7bb32087,         0x5d215762,         0x28157908,         0xe9867d51,
0x8e2e395d,         0x7b7f2149,         0x5c9757f3,         0x27567946,         0xe8c27d2d,
0x8d7c37f6,         0x7b4a220a,         0x5c0c5884,         0x26977983,         0xe7fe7d07,
0x8ccf368d,         0x7b1322cb,         0x5b805914,         0x25d779bf,         0xe73a7ce0,
0x8c263522,         0x7adb238c,         0x5af359a3,         0x251779f9,         0xe6767cb8,
0x8b8233b5,         0x7aa2244c,         0x5a655a31,         0x24577a32,         0xe5b37c8f,
0x8ae23246,         0x7a68250c,         0x59d65abe,         0x23967a6a,         0xe4f07c65,
0x8a4730d5,         0x7a2d25cc,         0x59475b4a,         0x22d57aa1,         0xe42d7c39,
0x89b02f62,         0x79f1268b,         0x58b75bd5,         0x22147ad7,         0xe36b7c0c,
0x891e2ded,         0x79b3274a,         0x58265c5f,         0x21527b0b,         0xe2a97bde,
0x88912c76,         0x79742808,         0x57945ce9,         0x20907b3e,         0xe1e77baf,
0x88082afd,         0x793428c6,         0x57015d72,         0x1fce7b70,         0xe1257b7f,
0x87842983,         0x78f32984,         0x566d5dfa,         0x1f0b7ba1,         0xe0647b4d,
0x87042807,         0x78b12a41,         0x55d85e81,         0x1e487bd1,         0xdfa37b1a,
0x8689268a,         0x786e2afe,         0x55435f07,         0x1d857bff,         0xdee27ae6,
0x8613250b,         0x78292bba,         0x54ad5f8c,         0x1cc27c2c,         0xde227ab1,
0x85a1238b,         0x77e32c76,         0x54166010,         0x1bfe7c58,         0xdd627a7b,
0x85342209,         0x779c2d32,         0x537e6093,         0x1b3a7c83,         0xdca27a44,
0x84cc2086,         0x77542ded,         0x52e56115,         0x1a767cad,         0xdbe37a0b,
0x84691f02,         0x770b2ea8,         0x524c6196,         0x19b27cd5,         0xdb2479d1,
0x840a1d7d,         0x76c12f62,         0x51b26216,         0x18ed7cfc,         0xda657996,
0x83b01bf7,         0x7675301c,         0x51176295,         0x18287d22,         0xd9a7795a,
0x835b1a6f,         0x762830d5,         0x507b6313,         0x17637d47,         0xd8e9791d,
0x830b18e6,         0x75da318e,         0x4fde6390,         0x169e7d6b,         0xd82b78df,
0x82c0175c,         0x758b3246,         0x4f41640c,         0x15d87d8d,         0xd76e789f,
0x827a15d2,         0x753b32fe,         0x4ea36488,         0x15127dae,         0xd6b1785e,
0x82381447,         0x74ea33b5,         0x4e046503,         0x144c7dce,         0xd5f5781c,
0x81fb12bb,         0x7498346c,         0x4d64657d,         0x13867ded,         0xd53977d9,
0x81c3112e,         0x74453522,         0x4cc465f6,         0x12c07e0b,         0xd47d7795,
0x81900fa1,         0x73f035d8,         0x4c23666e,         0x11f97e27,         0xd3c27750,
0x81620e13,         0x739a368d,         0x4b8166e5,         0x11327e42,         0xd3077709,
0x81390c84,         0x73433742,         0x4adf675b,         0x106b7e5c,         0xd24d76c1,
0x81150af5,         0x72eb37f6,         0x4a3c67d0,         0x0fa47e75,         0xd1937678,
0x80f60966,         0x729238aa,         0x49986844,         0x0edd7e8c,         0xd0da762e,
0x80db07d6,         0x7238395d,         0x48f368b7,         0x0e167ea2,         0xd02175e3,
0x80c50646,         0x71dd3a10,         0x484e6929,         0x0d4f7eb7,         0xcf697597,
0x80b404b6,         0x71813ac2,         0x47a8699a,         0x0c877ecb,         0xceb1754a,
0x80a80326,         0x71243b74,         0x47016a0a,         0x0bbf7ede,         0xcdfa74fb,
0x80a10195,         0x70c63c25,         0x465a6a79,         0x0af77eef,         0xcd4374ab,
0x7FFF0000,         0x70663cd5,         0x45b26ae6,         0x0a2f7eff,         0xcc8c745a,
0x7fff00c9,         0x70053d85,         0x45096b52,         0x09677f0e,         0xcbd67408,
0x7ffd0192,         0x6fa33e34,         0x44606bbd,         0x089f7f1c,         0xcb2073b5,
0x7ff9025b,         0x6f403ee3,         0x43b66c27,         0x07d77f28,         0xca6b7361,
0x7ff40324,         0x6edc3f91,         0x430b6c90,         0x070f7f33,         0xc9b7730c,
0x7fee03ec,         0x6e77403e,         0x42606cf8,         0x06477f3d,         0xc90372b6,
0x7fe704b4,         0x6e1140eb,         0x41b46d5f,         0x057f7f46,         0xc850725f,
0x7fde057c,         0x6daa4197,         0x41076dc5,         0x04b77f4e,         0xc79d7206,
0x7fd40644,         0x6d424242,         0x405a6e2a,         0x03ef7f54,         0xc6eb71ac,
0x7fc9070c,         0x6cd942ed,         0x3fac6e8e,         0x03267f59,         0xc6397151,
0x7fbd07d4,         0x6c6f4397,         0x3efe6ef1,         0x025d7f5d,         0xc58870f5,
0x7fb0089c,         0x6c044441,         0x3e4f6f53,         0x01947f60,         0xc4d77098,
0x7fa10964,         0x6b9844ea,         0x3d9f6fb4,         0x00cb7f61,         0xc427703a,
0x7f910a2c,         0x6b2b4592,         0x3cef7014,         0x00027f61,         0xc3786fdb,
0x7f800af4,         0x6abd463a,         0x3c3e7073,         0xff3a7f60,         0xc2c96f7b,
0x7f6e0bbc,         0x6a4e46e1,         0x3b8d70d1,         0xfe727f5e,         0xc21b6f1a,
0x7f5a0c84,         0x69de4787,         0x3adb712d,         0xfdaa7f5a,         0xc16d6eb8,
0x7f450d4b,         0x696d482c,         0x3a287188,         0xfce27f55,         0xc0c06e55,
0x7f2f0e12,         0x68fb48d1,         0x397571e2,         0xfc1b7f4f,         0xc0136df1,
0x7f180ed9,         0x68884975,         0x38c1723b,         0xfb547f48,         0xbf676d8c,
0x7f000fa0,         0x68144a18,         0x380d7293,         0xfa8d7f40,         0xbebc6d26,
0x7ee61067,         0x679f4abb,         0x375872ea,         0xf9c67f36,         0xbe116cbe,
0x7ecb112e,         0x67294b5d,         0x36a37340,         0xf8ff7f2b,         0xbd676c55,
0x7eaf11f5,         0x66b24bfe,         0x35ed7395,         0xf8387f1f,         0xbcbe6beb,
0x7e9212bb,         0x663a4c9e,         0x353773e9,         0xf7717f12,         0xbc156b80,
0x7e731381,         0x65c14d3e,         0x3480743c,         0xf6aa7f03,         0xbb6d6b14,
0x7e531447,         0x65474ddd,         0x33c9748d,         0xf5e37ef3,         0xbac66aa7,
0x7e32150d,         0x64cc4e7b,         0x331174dd,         0xf51c7ee2,         0xba1f6a39,
0x7e1015d3,         0x64504f18,         0x3259752c,         0xf4557ed0,         0xb97969ca,
0x7ded1698,         0x63d34fb5,         0x31a0757a,         0xf38e7ebd,         0xb8d4695a,
0x7dc8175d,         0x63555051,         0x30e775c7,         0xf2c77ea8,         0xb82f68e9,
0x7da21822,         0x62d650ec,         0x302d7613,         0xf2017e92,         0xb78b6877,
0x7d7b18e7,         0x62565186,         0x2f73765e,         0xf13b7e7b,         0xb6e86804,
0x7d5319ab,         0x61d55220,         0x2eb876a7,         0xf0757e63,         0xb6456790,
0x7d2a1a6f,         0x615352b9,         0x2dfd76ef,         0xefaf7e49,         0xb5a3671b,
0x7cff1b33,         0x60d05351,         0x2d417736,         0xeee97e2e,         0xb50266a5,
0x7cd31bf7,         0x604c53e8,         0x2c85777c,         0xee237e12,         0xb462662e,
0xb3c265b6,          0x8cab34ee,          0x7fe8044c,          0x7ac82374,          0x6e284063,
0xb323653d,          0x8c593438,          0x7fe404b0,          0x7aac23d4,          0x6df540b9,
0xb28564c3,          0x8c083382,          0x7fe00514,          0x7a8f2434,          0x6dc2410f,
0xb1e86448,          0x8bb832cb,          0x7fdb0578,          0x7a722494,          0x6d8e4165,
0xb14b63cc,          0x8b693214,          0x7fd605dc,          0x7a5524f4,          0x6d5a41ba,
0xb0af634f,          0x8b1b315c,          0x7fd10640,          0x7a372554,          0x6d26420f,
0xb01462d1,          0x8acf30a4,          0x7fcb06a4,          0x7a1925b3,          0x6cf14264,
0xaf7a6252,          0x8a842feb,          0x7fc50708,          0x79fb2612,          0x6cbc42b9,
0xaee061d3,          0x8a3a2f32,          0x7fbf076c,          0x79dc2671,          0x6c87430e,
0xae476153,          0x89f12e78,          0x7fb907d0,          0x79bd26d0,          0x6c524363,
0xadaf60d2,          0x89a92dbe,          0x7fb20834,          0x799e272f,          0x6c1c43b7,
0xad186050,          0x89622d03,          0x7fab0898,          0x797f278e,          0x6be6440b,
0xac825fcd,          0x891c2c48,          0x7fa408fc,          0x795f27ed,          0x6bb0445f,
0xabec5f49,          0x88d82b8d,          0x7f9c0960,          0x793f284c,          0x6b7a44b3,
0xab575ec4,          0x88952ad1,          0x7f9409c4,          0x791f28ab,          0x6b434507,
0xaac35e3e,          0x88532a15,          0x7f8c0a28,          0x78fe290a,          0x6b0c455b,
0xaa305db7,          0x88122958,          0x7f830a8c,          0x78dd2968,          0x6ad545ae,
0xa99e5d2f,          0x87d2289b,          0x7f7a0af0,          0x78bc29c6,          0x6a9e4601,
0xa90d5ca6,          0x879327de,          0x7f710b54,          0x789b2a24,          0x6a664654,
0xa87c5c1c,          0x87552720,          0x7f670bb8,          0x78792a82,          0x6a2e46a7,
0xa7ec5b92,          0x87192662,          0x7f5d0c1c,          0x78572ae0,          0x69f646fa,
0xa75d5b07,          0x86de25a3,          0x7f530c80,          0x78352b3e,          0x69be474d,
0xa6cf5a7b,          0x86a424e4,          0x7f490ce3,          0x78122b9c,          0x6985479f,
0xa64259ee,          0x866b2425,          0x7f3e0d46,          0x77ef2bfa,          0x694c47f1,
0xa5b65960,          0x86332365,          0x7f330da9,          0x77cc2c58,          0x69134843,
0xa52b58d1,          0x85fc22a5,          0x7f280e0c,          0x77a92cb6,          0x68da4895,
0xa4a05841,          0x85c721e5,          0x7f1c0e6f,          0x77852d13,          0x68a048e7,
0xa41657b1,          0x85932124,          0x7f100ed2,          0x77612d70,          0x68664939,
0xa38d5720,          0x85602063,          0x7f040f35,          0x773d2dcd,          0x682c498a,
0xa305568e,          0x852e1fa2,          0x7ef70f98,          0x77182e2a,          0x67f249db,
0xa27e55fb,          0x84fd1ee0,          0x7eea0ffb,          0x76f32e87,          0x67b74a2c,
0xa1f85567,          0x84ce1e1e,          0x7edd105e,          0x76ce2ee4,          0x677c4a7d,
0xa17354d2,          0x84a01d5c,          0x7ecf10c1,          0x76a92f41,          0x67414ace,
0xa0ef543d,          0x84731c9a,          0x7ec11124,          0x76832f9e,          0x67064b1f,
0xa06c53a7,          0x84471bd7,          0x7eb31187,          0x765d2ffb,          0x66ca4b6f,
0x9fea5310,          0x841c1b14,          0x7ea511ea,          0x76373057,          0x668e4bbf,
0x9f685278,          0x83f31a51,          0x7e96124d,          0x761030b3,          0x66524c0f,
0x9ee751df,          0x83cb198e,          0x7e8712b0,          0x75e9310f,          0x66164c5f,
0x9e675146,          0x83a418ca,          0x7e781313,          0x75c2316b,          0x65d94caf,
0x9de850ac,          0x837e1806,          0x7e681376,          0x759b31c7,          0x659c4cfe,
0x9d6a5011,          0x83591742,          0x7e5813d9,          0x75733223,          0x655f4d4d,
0x9ced4f75,          0x8336167e,          0x7e48143c,          0x754b327f,          0x65224d9c,
0x9c714ed8,          0x831415b9,          0x7e37149f,          0x752332db,          0x64e44deb,
0x9bf64e3b,          0x82f314f4,          0x7e261502,          0x74fa3336,          0x64a64e3a,
0x9b7c4d9d,          0x82d3142f,          0x7e151565,          0x74d13391,          0x64684e88,
0x9b034cfe,          0x82b4136a,          0x7e0415c7,          0x74a833ec,          0x642a4ed6,
0x9a8b4c5e,          0x829712a5,          0x7df21629,          0x747f3447,          0x63eb4f24,
0x9a144bbe,          0x827b11df,          0x7de0168b,          0x745534a2,          0x63ac4f72,
0x999e4b1d,          0x82601119,          0x7dce16ed,          0x742b34fd,          0x636d4fc0,
0x99294a7b,          0x82461053,          0x7dbb174f,          0x74013558,          0x632e500d,
0x98b549d9,          0x822d0f8d,          0x7da817b1,          0x73d635b3,          0x62ef505a,
0x98424936,          0x82160ec7,          0x7d951813,          0x73ab360d,          0x62af50a7,
0x97d04892,          0x82000e01,          0x7d811875,          0x73803667,          0x626f50f4,
0x975f47ed,          0x81eb0d3b,          0x7d6d18d7,          0x735536c1,          0x622f5141,
0x96ef4748,          0x81d70c74,          0x7d591939,          0x7329371b,          0x61ef518e,
0x968046a2,          0x81c50bad,          0x7d45199b,          0x72fd3775,          0x61ae51da,
0x961245fb,          0x81b40ae6,          0x7d3019fd,          0x72d137cf,          0x616d5226,
0x95a54554,          0x81a40a1f,          0x7d1b1a5f,          0x72a53829,          0x612c5272,
0x953944ac,          0x81950958,          0x7d061ac1,          0x72783882,          0x60eb52be,
0x94ce4403,          0x81870891,          0x7cf01b23,          0x724b38db,          0x60a9530a,
0x9464435a,          0x817b07ca,          0x7cda1b85,          0x721e3934,          0x60675355,
0x93fb42b0,          0x81700703,          0x7cc41be7,          0x71f0398d,          0x602553a0,
0x93934206,          0x8166063c,          0x7cad1c48,          0x71c239e6,          0x5fe353eb,
0x932c415b,          0x815d0575,          0x7c961ca9,          0x71943a3f,          0x5fa05436,
0x92c640af,          0x815604ae,          0x7c7f1d0a,          0x71663a98,          0x5f5d5481,
0x92614003,          0x815003e7,          0x7c681d6b,          0x71373af0,          0x5f1a54cb,
0x91fd3f56,          0x814b031f,          0x7c501dcc,          0x71083b48,          0x5ed75515,
0x919b3ea8,          0x81470257,          0x7c381e2d,          0x70d93ba0,          0x5e94555f,
0x913a3dfa,          0x8144018f,          0x7c201e8e,          0x70aa3bf8,          0x5e5055a9,
0x90da3d4b,          0x814300c7,          0x7c071eef,          0x707a3c50,          0x5e0c55f2,
0x907b3c9c,          0x7FFF0000,          0x7bee1f50,          0x704a3ca8,          0x5dc8563b,
0x901d3bec,          0x7fff0064,          0x7bd51fb1,          0x701a3d00,          0x5d845684,
0x8fc03b3b,          0x7ffe00c8,          0x7bbb2012,          0x6fe93d57,          0x5d3f56cd,
0x8f643a8a,          0x7ffd012c,          0x7ba12073,          0x6fb83dae,          0x5cfa5716,
0x8f0939d8,          0x7ffb0190,          0x7b8720d4,          0x6f873e05,          0x5cb5575e,
0x8eaf3926,          0x7ff901f4,          0x7b6d2134,          0x6f563e5c,          0x5c7057a6,
0x8e563873,          0x7ff70258,          0x7b522194,          0x6f243eb3,          0x5c2b57ee,
0x8dfe37c0,          0x7ff502bc,          0x7b3721f4,          0x6ef23f0a,          0x5be55836,
0x8da7370c,          0x7ff20320,          0x7b1c2254,          0x6ec03f61,          0x5b9f587e,
0x8d523658,          0x7fef0384,          0x7b0022b4,          0x6e8e3fb7,          0x5b5958c5,
0x8cfe35a3,          0x7fec03e8,          0x7ae42314,          0x6e5b400d,          0x5b13590c,
0x5acc5953,          0x41e46cbe,          0x24fb796f,          0x05d57e9f,          0xe6a07c06,
0x5a85599a,          0x418e6cf1,          0x249b798b,          0x05717ea3,          0xe63f7bf1,
0x5a3e59e0,          0x41386d24,          0x243b79a7,          0x050d7ea7,          0xe5de7bdc,
0x59f75a26,          0x40e26d57,          0x23db79c3,          0x04a97eaa,          0xe57d7bc7,
0x59b05a6c,          0x408c6d89,          0x237b79df,          0x04457ead,          0xe51c7bb2,
0x59685ab2,          0x40356dbb,          0x231b79fa,          0x03e17eb0,          0xe4bb7b9c,
0x59205af8,          0x3fde6ded,          0x22bb7a15,          0x037d7eb2,          0xe45a7b86,
0x58d85b3d,          0x3f876e1f,          0x225b7a30,          0x03197eb4,          0xe3fa7b70,
0x58905b82,          0x3f306e50,          0x21fa7a4a,          0x02b57eb6,          0xe39a7b59,
0x58485bc7,          0x3ed96e81,          0x21997a64,          0x02517eb7,          0xe33a7b42,
0x57ff5c0c,          0x3e826eb2,          0x21387a7e,          0x01ed7eb8,          0xe2da7b2b,
0x57b65c51,          0x3e2a6ee2,          0x20d77a97,          0x01897eb9,          0xe27a7b13,
0x576d5c95,          0x3dd26f12,          0x20767ab0,          0x01257eba,          0xe21a7afb,
0x57245cd9,          0x3d7a6f42,          0x20157ac9,          0x00c17eba,          0xe1ba7ae3,
0x56da5d1d,          0x3d226f72,          0x1fb47ae2,          0x005d7eba,          0xe15a7acb,
0x56905d61,          0x3cca6fa1,          0x1f537afa,          0xfffa7eba,          0xe0fa7ab2,
0x56465da4,          0x3c726fd0,          0x1ef27b12,          0xff977eb9,          0xe09a7a99,
0x55fc5de7,          0x3c1a6fff,          0x1e917b2a,          0xff347eb8,          0xe03a7a80,
0x55b25e2a,          0x3bc1702e,          0x1e307b41,          0xfed17eb7,          0xdfda7a66,
0x55675e6d,          0x3b68705c,          0x1dcf7b58,          0xfe6e7eb5,          0xdf7a7a4c,
0x551c5eaf,          0x3b0f708a,          0x1d6e7b6f,          0xfe0b7eb3,          0xdf1a7a32,
0x54d15ef1,          0x3ab670b8,          0x1d0d7b85,          0xfda87eb1,          0xdebb7a18,
0x54865f33,          0x3a5d70e5,          0x1cab7b9b,          0xfd457eaf,          0xde5c79fd,
0x543b5f75,          0x3a047112,          0x1c497bb1,          0xfce27eac,          0xddfd79e2,
0x53ef5fb7,          0x39ab713f,          0x1be77bc7,          0xfc7f7ea9,          0xdd9e79c7,
0x53a35ff8,          0x3951716c,          0x1b857bdc,          0xfc1c7ea6,          0xdd3f79ab,
0x53576039,          0x38f77198,          0x1b237bf1,          0xfbb97ea2,          0xdce0798f,
0x530b607a,          0x389d71c4,          0x1ac17c06,          0xfb567e9e,          0xdc817973,
0x52bf60bb,          0x384371f0,          0x1a5f7c1a,          0xfaf37e9a,          0xdc227956,
0x527260fb,          0x37e9721c,          0x19fd7c2e,          0xfa907e95,          0xdbc37939,
0x5225613b,          0x378f7247,          0x199b7c42,          0xfa2d7e90,          0xdb64791c,
0x51d8617b,          0x37357272,          0x19397c55,          0xf9ca7e8b,          0xdb0578ff,
0x518b61bb,          0x36db729d,          0x18d77c68,          0xf9677e85,          0xdaa778e1,
0x513e61fa,          0x368072c7,          0x18757c7b,          0xf9047e7f,          0xda4978c3,
0x50f06239,          0x362572f1,          0x18137c8e,          0xf8a17e79,          0xd9eb78a5,
0x50a26278,          0x35ca731b,          0x17b17ca0,          0xf83e7e73,          0xd98d7886,
0x505462b7,          0x356f7345,          0x174f7cb2,          0xf7db7e6c,          0xd92f7867,
0x500662f5,          0x3514736e,          0x16ed7cc4,          0xf7787e65,          0xd8d17848,
0x4fb86333,          0x34b97397,          0x168a7cd5,          0xf7157e5e,          0xd8737829,
0x4f696371,          0x345e73c0,          0x16277ce6,          0xf6b27e56,          0xd8157809,
0x4f1a63af,          0x340373e8,          0x15c47cf7,          0xf64f7e4e,          0xd7b777e9,
0x4ecb63ed,          0x33a77410,          0x15617d07,          0xf5ec7e46,          0xd75977c9,
0x4e7c642a,          0x334b7438,          0x14fe7d17,          0xf5897e3d,          0xd6fb77a8,
0x4e2d6467,          0x32ef7460,          0x149b7d27,          0xf5267e34,          0xd69e7787,
0x4dde64a4,          0x32937487,          0x14387d37,          0xf4c37e2b,          0xd6417766,
0x4d8e64e1,          0x323774ae,          0x13d57d46,          0xf4607e22,          0xd5e47745,
0x4d3e651d,          0x31db74d5,          0x13727d55,          0xf3fd7e18,          0xd5877723,
0x4cee6559,          0x317f74fc,          0x130f7d64,          0xf39a7e0e,          0xd52a7701,
0x4c9e6595,          0x31237522,          0x12ac7d72,          0xf3387e04,          0xd4cd76df,
0x4c4e65d1,          0x30c67548,          0x12497d80,          0xf2d67df9,          0xd47076bc,
0x4bfd660c,          0x3069756e,          0x11e67d8e,          0xf2747dee,          0xd4137699,
0x4bac6647,          0x300c7593,          0x11837d9b,          0xf2127de3,          0xd3b67676,
0x4b5b6682,          0x2faf75b8,          0x11207da8,          0xf1b07dd7,          0xd35a7653,
0x4b0a66bd,          0x2f5275dd,          0x10bd7db5,          0xf14e7dcb,          0xd2fe762f,
0x4ab966f7,          0x2ef57602,          0x105a7dc1,          0xf0ec7dbf,          0xd2a2760b,
0x4a686731,          0x2e987626,          0x0ff77dcd,          0xf08a7db3,          0xd24675e7,
0x4a16676b,          0x2e3b764a,          0x0f947dd9,          0xf0287da6,          0xd1ea75c2,
0x49c467a5,          0x2dde766e,          0x0f317de5,          0xefc67d99,          0xd18e759d,
0x497267de,          0x2d807691,          0x0ece7df0,          0xef647d8c,          0xd1327578,
0x49206817,          0x2d2276b4,          0x0e6b7dfb,          0xef027d7e,          0xd0d67553,
0x48ce6850,          0x2cc476d7,          0x0e087e06,          0xeea07d70,          0xd07a752d,
0x487b6889,          0x2c6676fa,          0x0da57e10,          0xee3e7d62,          0xd01f7507,
0x482868c1,          0x2c08771c,          0x0d417e1a,          0xeddc7d53,          0xcfc474e1,
0x47d568f9,          0x2baa773e,          0x0cdd7e24,          0xed7a7d44,          0xcf6974ba,
0x47826931,          0x2b4c7760,          0x0c797e2d,          0xed187d35,          0xcf0e7493,
0x472f6969,          0x2aee7781,          0x0c157e36,          0xecb67d26,          0xceb3746c,
0x46dc69a0,          0x2a9077a2,          0x0bb17e3f,          0xec547d16,          0xce587445,
0x468869d7,          0x2a3177c3,          0x0b4d7e48,          0xebf27d06,          0xcdfd741d,
0x46346a0e,          0x29d277e3,          0x0ae97e50,          0xeb907cf6,          0xcda273f5,
0x45e06a45,          0x29737803,          0x0a857e58,          0xeb2e7ce5,          0xcd4773cd,
0x458c6a7b,          0x29147823,          0x0a217e60,          0xeacc7cd4,          0xcced73a5,
0x45386ab1,          0x28b57843,          0x09bd7e67,          0xea6a7cc3,          0xcc93737c,
0x44e46ae7,          0x28567862,          0x09597e6e,          0xea097cb1,          0xcc397353,
0x448f6b1c,          0x27f77881,          0x08f57e75,          0xe9a87c9f,          0xcbdf732a,
0x443a6b51,          0x279878a0,          0x08917e7b,          0xe9477c8d,          0xcb857300,
0x43e56b86,          0x273978be,          0x082d7e81,          0xe8e67c7b,          0xcb2b72d6,
0x43906bbb,          0x26da78dc,          0x07c97e87,          0xe8857c68,          0xcad172ac,
0x433b6bef,          0x267b78fa,          0x07657e8c,          0xe8247c55,          0xca787282,
0x42e66c23,          0x261b7918,          0x07017e91,          0xe7c37c42,          0xca1f7257,
0x42906c57,          0x25bb7935,          0x069d7e96,          0xe7627c2e,          0xc9c6722c,
0x423a6c8b,          0x255b7952,          0x06397e9b,          0xe7017c1a,          0xc96d7201,
0xc91471d6,       0xb15462aa,       0x9d884e99,       0x8eb336a5,       0x858c1bfc,
0xc8bb71aa,       0xb107626c,       0x9d4b4e4b,       0x8e89364b,       0x85771b9b,
0xc862717e,       0xb0ba622d,       0x9d0e4dfd,       0x8e5f35f1,       0x85621b3a,
0xc8097152,       0xb06d61ee,       0x9cd14daf,       0x8e353597,       0x854d1ad9,
0xc7b17125,       0xb02161af,       0x9c954d61,       0x8e0c353d,       0x85391a78,
0xc75970f8,       0xafd56170,       0x9c594d12,       0x8de334e3,       0x85251a17,
0xc70170cb,       0xaf896130,       0x9c1d4cc3,       0x8dba3489,       0x851119b6,
0xc6a9709e,       0xaf3d60f0,       0x9be14c74,       0x8d91342f,       0x84fd1955,
0xc6517070,       0xaef160b0,       0x9ba64c25,       0x8d6933d5,       0x84ea18f4,
0xc5f97042,       0xaea66070,       0x9b6b4bd6,       0x8d41337a,       0x84d71893,
0xc5a17014,       0xae5b602f,       0x9b304b86,       0x8d19331f,       0x84c41832,
0xc54a6fe6,       0xae105fee,       0x9af54b36,       0x8cf132c4,       0x84b217d1,
0xc4f36fb7,       0xadc55fad,       0x9abb4ae6,       0x8cca3269,       0x84a01770,
0xc49c6f88,       0xad7a5f6c,       0x9a814a96,       0x8ca3320e,       0x848e170f,
0xc4456f59,       0xad305f2b,       0x9a474a46,       0x8c7c31b3,       0x847d16ae,
0xc3ee6f29,       0xace65ee9,       0x9a0d49f6,       0x8c563158,       0x846c164c,
0xc3976ef9,       0xac9c5ea7,       0x99d449a5,       0x8c3030fd,       0x845b15ea,
0xc3406ec9,       0xac525e65,       0x999b4954,       0x8c0a30a1,       0x844a1588,
0xc2ea6e99,       0xac085e23,       0x99624903,       0x8be43045,       0x843a1526,
0xc2946e68,       0xabbf5de0,       0x992948b2,       0x8bbf2fe9,       0x842a14c4,
0xc23e6e37,       0xab765d9d,       0x98f14861,       0x8b9a2f8d,       0x841a1462,
0xc1e86e06,       0xab2d5d5a,       0x98b9480f,       0x8b752f31,       0x840b1400,
0xc1926dd5,       0xaae45d17,       0x988147bd,       0x8b512ed5,       0x83fc139e,
0xc13c6da3,       0xaa9b5cd4,       0x9849476b,       0x8b2d2e79,       0x83ed133c,
0xc0e66d71,       0xaa535c90,       0x98124719,       0x8b092e1d,       0x83df12da,
0xc0916d3f,       0xaa0b5c4c,       0x97db46c7,       0x8ae52dc1,       0x83d11278,
0xc03c6d0d,       0xa9c35c08,       0x97a44675,       0x8ac22d64,       0x83c31216,
0xbfe76cda,       0xa97b5bc4,       0x976d4622,       0x8a9f2d07,       0x83b511b4,
0xbf926ca7,       0xa9345b7f,       0x973745cf,       0x8a7c2caa,       0x83a81152,
0xbf3d6c74,       0xa8ed5b3a,       0x9701457c,       0x8a5a2c4d,       0x839b10f0,
0xbee86c41,       0xa8a65af5,       0x96cb4529,       0x8a382bf0,       0x838e108e,
0xbe946c0d,       0xa85f5ab0,       0x969544d6,       0x8a162b93,       0x8382102c,
0xbe406bd9,       0xa8185a6b,       0x96604483,       0x89f42b36,       0x83760fca,
0xbdec6ba5,       0xa7d25a25,       0x962b442f,       0x89d32ad9,       0x836a0f68,
0xbd986b70,       0xa78c59df,       0x95f643db,       0x89b22a7c,       0x835f0f06,
0xbd446b3b,       0xa7465999,       0x95c14387,       0x89912a1f,       0x83540ea4,
0xbcf06b06,       0xa7005953,       0x958d4333,       0x897129c1,       0x83490e42,
0xbc9d6ad1,       0xa6ba590c,       0x955942df,       0x89512963,       0x833e0de0,
0xbc4a6a9b,       0xa67558c5,       0x9525428b,       0x89312905,       0x83340d7d,
0xbbf76a65,       0xa630587e,       0x94f14236,       0x891128a7,       0x832a0d1a,
0xbba46a2f,       0xa5eb5837,       0x94be41e1,       0x88f22849,       0x83200cb7,
0xbb5169f9,       0xa5a657f0,       0x948b418c,       0x88d327eb,       0x83170c54,
0xbafe69c2,       0xa56257a8,       0x94584137,       0x88b4278d,       0x830e0bf1,
0xbaac698b,       0xa51e5760,       0x942540e2,       0x8896272f,       0x83050b8e,
0xba5a6954,       0xa4da5718,       0x93f3408d,       0x887826d1,       0x82fd0b2b,
0xba08691d,       0xa49656d0,       0x93c14038,       0x885a2673,       0x82f50ac8,
0xb9b668e5,       0xa4525688,       0x938f3fe2,       0x883c2614,       0x82ed0a65,
0xb96468ad,       0xa40f563f,       0x935d3f8c,       0x881f25b5,       0x82e50a02,
0xb9126875,       0xa3cc55f6,       0x932c3f36,       0x88022556,       0x82de099f,
0xb8c1683d,       0xa38955ad,       0x92fb3ee0,       0x87e524f7,       0x82d7093c,
0xb8706804,       0xa3465564,       0x92ca3e8a,       0x87c92498,       0x82d008d9,
0xb81f67cb,       0xa304551b,       0x929a3e34,       0x87ad2439,       0x82ca0876,
0xb7ce6792,       0xa2c254d1,       0x926a3dde,       0x879123da,       0x82c40813,
0xb77d6759,       0xa2805487,       0x923a3d87,       0x8775237b,       0x82be07b0,
0xb72c671f,       0xa23e543d,       0x920a3d30,       0x875a231c,       0x82b9074d,
0xb6dc66e5,       0xa1fc53f3,       0x91db3cd9,       0x873f22bd,       0x82b406ea,
0xb68c66ab,       0xa1bb53a9,       0x91ac3c82,       0x8724225e,       0x82af0687,
0xb63c6671,       0xa17a535e,       0x917d3c2b,       0x870a21ff,       0x82ab0624,
0xb5ec6636,       0xa1395313,       0x914e3bd4,       0x86f0219f,       0x82a705c1,
0xb59c65fb,       0xa0f852c8,       0x91203b7c,       0x86d6213f,       0x82a3055e,
0xb54c65c0,       0xa0b8527d,       0x90f23b24,       0x86bd20df,       0x829f04fb,
0xb4fd6585,       0xa0785232,       0x90c43acc,       0x86a4207f,       0x829c0498,
0xb4ae6549,       0xa03851e6,       0x90963a74,       0x868b201f,       0x82990435,
0xb45f650d,       0x9ff8519a,       0x90693a1c,       0x86721fbf,       0x829603d2,
0xb41064d1,       0x9fb9514e,       0x903c39c4,       0x865a1f5f,       0x8294036f,
0xb3c16495,       0x9f7a5102,       0x900f396c,       0x86421eff,       0x8292030c,
0xb3736458,       0x9f3b50b6,       0x8fe33914,       0x862a1e9f,       0x829002a9,
0xb325641b,       0x9efc5069,       0x8fb738bb,       0x86131e3f,       0x828f0246,
0xb2d763de,       0x9ebd501c,       0x8f8b3862,       0x85fc1ddf,       0x828e01e3,
0xb28963a1,       0x9e7f4fcf,       0x8f5f3809,       0x85e51d7f,       0x828d0180,
0xb23b6364,       0x9e414f82,       0x8f3437b0,       0x85ce1d1f,       0x828c011d,
0xb1ee6326,       0x9e034f35,       0x8f093757,       0x85b81cbe,       0x828c00ba,
0xb1a162e8,       0x9dc54ee7,       0x8ede36fe,       0x85a21c5d,       0x828c0057};
```

9. Other Files

9.1. system.mhs

```
# Parameters
PARAMETER VERSION = 2.0.0

# Global Ports

PORT PB_A = PB_A, DIR = OUT, VEC = [19:0]
PORT PB_D = PB_D, DIR = INOUT, VEC = [15:0]
PORT PB_LB_N = PB_LB_N, DIR = OUT
PORT PB_UB_N = PB_UB_N, DIR = OUT
PORT PB_WE_N = PB_WE_N, DIR = OUT
PORT PB_OE_N = PB_OE_N, DIR = OUT
PORT RAM_CE_N = RAM_CE_N, DIR = OUT
PORT VIDOUT_CLK = VIDOUT_CLK, DIR = OUT
PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N, DIR = OUT
PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N, DIR = OUT
PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N, DIR = OUT
PORT VIDOUT_RCR = VIDOUT_RCR, DIR = OUT, VEC = [9:0]
PORT VIDOUT_GY = VIDOUT_GY, DIR = OUT, VEC = [9:0]
PORT VIDOUT_BCB = VIDOUT_BCB, DIR = OUT, VEC = [9:0]
PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN
PORT AU_CSN_N =  AU_CSN_N, DIR=OUT
PORT AU_BCLK =  AU_BCLK, DIR=OUT
PORT AU_MCLK = AU_MCLK, DIR=OUT
PORT AU_LRCK = AU_LRCK, DIR=OUT
PORT AU_SDTI = AU_SDTI, DIR=OUT
PORT AU_SDTO0 = AU_SDTO0, DIR=IN

# Sub Components


BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_ADDR_TAG_BITS = 6
 PARAMETER C_CACHE_BYTE_SIZE = 2048
 PARAMETER C_ICACHE_BASEADDR = 0x00860000
 PARAMETER C_ICACHE_HIGHADDR = 0x0087FFFF
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 PORT Interrupt = intr
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

BEGIN opb_intc
```

```
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFEFF0000
 PARAMETER C_HIGHADDR = 0xFEFF00FF
 PORT OPB_Clk = sys_clk
 PORT Intr =  uart_intr & audio_intr
 PORT Irq =   intr
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_xsb300
 PARAMETER INSTANCE = xsb300
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF
 PORT PB_A = PB_A
 PORT PB_D = PB_D
 PORT PB_LB_N = PB_LB_N
 PORT PB_UB_N = PB_UB_N
 PORT PB_WE_N = PB_WE_N
 PORT PB_OE_N = PB_OE_N
 PORT RAM_CE_N = RAM_CE_N
 PORT OPB_Clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT VIDOUT_CLK = VIDOUT_CLK
 PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N
 PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N
 PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N
 PORT VIDOUT_RCR = VIDOUT_RCR
 PORT VIDOUT_GY = VIDOUT_GY
 PORT VIDOUT_BCB = VIDOUT_BCB
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN opb_xsb300_ak4565
 PARAMETER INSTANCE = ak4565
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFE0000
 PARAMETER C_HIGHADDR = 0xFEFEFFFF

 PORT OPB_Clk = sys_clk
 PORT Interrupt = audio_intr
 PORT AU_CSN_N =  AU_CSN_N
 PORT AU_BCLK =  AU_BCLK
 PORT AU_MCLK = AU_MCLK
 PORT AU_LRCK = AU_LRCK
 PORT AU_SDTI = AU_SDTI
 PORT AU_SDTO0 = AU_SDTO0

 BUS_INTERFACE SOPB = myopb_bus
```

```
END

BEGIN opb_xsb300_nortsam
 PARAMETER INSTANCE = nortsam
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x0FEF0000
 PARAMETER C_HIGHADDR = 0x0FEFFFFF

 PORT OPB_Clk = sys_clk
 PORT Interrupt = nortsam_intr

 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

BEGIN lmb_lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 PORT Interrupt = uart_intr
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END
```