EEC0055 - Digital Systems Design

2018/2019

Laboratory 3 – V1.0
19 November – 21 December 2018

All-digital FM stereo modulator

Version 0.3 – 26 november 2018

Revision history

date	notes	author
V0.1 Nov 18, 2018	First version	jca@fe.up.pt
V0.2 Nov 23, 2018	- Added section 3.2.1 Verification kit for the generic DDS module; - Modified section 3.1, changing the specification of the main clock frequency to 147.456000 MHz (integer multiple of 48 kHz);	jca@fe.up.pt
V0.3 Nov 26, 2018	Added section 4 – IP cores, with the description of the two intellectual property blocks provided for the FM modulator implementation: sequential multiplier and 4X interpolator	jca@fe.up.pt

1 – Introduction

In this project the students will implement a FM stereo modulator and transmitter. The input stereo audio signal is received from an audio source connected to the Line-in or Mic-in inputs of the Atlys board. These analog signals are digitized by an audio codec on the board. Inside the reference project provided for the Atlys board, where your design will be implemented, the digital audio signal is already available as two 18-bit signed words sampled at 48 kHz. The same serial interface implemented in the previous projects will be used in this design to configure some parameters of the FM modulator.

Your design will receive the stereo high-quality digital audio signal, generate the multiplexed FM-stereo signal to modulate in frequency a 5 MHz sinusoidal signal, sampled at 160 MHz. The digital output (8 bits) will be connected to an external digital to analog converter (DAC) that will produce an FM analog signal sampled at 100 Mhz. A FM radio receiver placed nearby the transmitter and tuned to 95 MHz or 105 MHz will be able to receive the FM signal.

Figure 1 shows the overall organization of the system. The FM modulator (block FMSTEREO) is detailed in the next section and a complete Matlab will be provided for generating golden simulation data for each of the blocks.

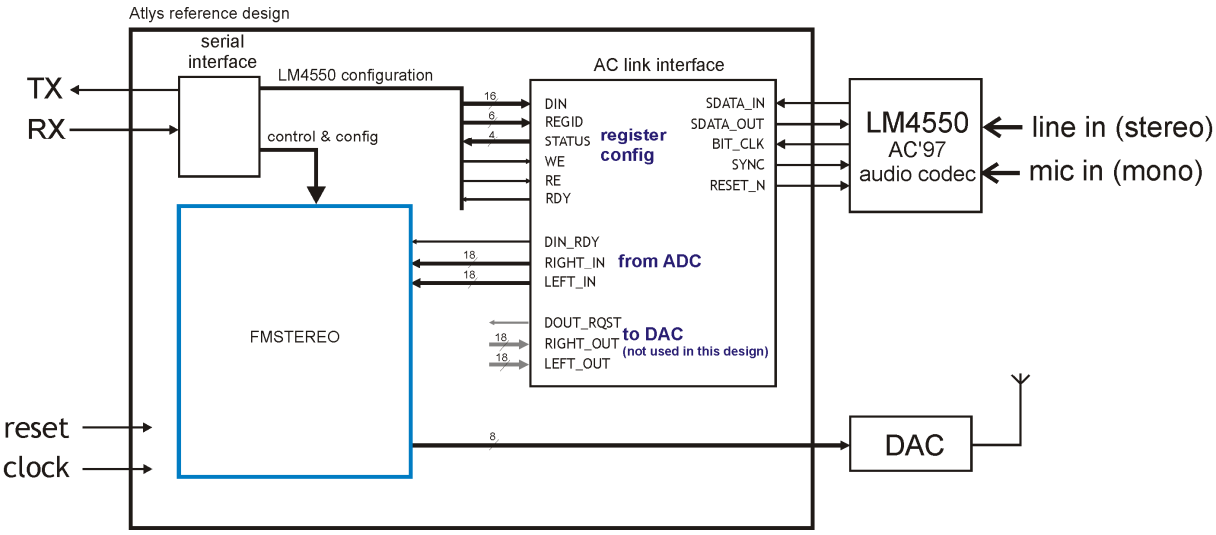


Figure 1 – Simplified block diagram of the reference project. The design to develop is represented by the block FMSTEREO.

2 – FM and FM stereo basics

This section introduces the basics concepts of frequency modulation and the process of encoding the two channels of a stereo signal. If you are familiar with these concepts you can skip this section.

2.1 – Frequency modulation

FM or frequency modulation is a process to encode analog information (a low frequency audio signal (t)) on a high frequency carrier signal (for example the FM broadcast radio signal in the band 88 – 108 MHz). In a FM signal, the instantaneous frequency is a fixed value ω_c (the frequency tuned to listen to a FM radio station) summed to a deviation in frequency that is proportional to the amplitude of the encoded signal $m(t)$. The FM modulation used in the commercial FM broadcast in Europe uses a maximum frequency deviation of ±50 kHz.

The instantaneous angular frequency of a FM signal $\omega_i(t)$ is thus defined as:

$$\omega_i(t) = \omega_c + K_f m(t) \quad (1)$$

where K_f is a frequency deviation constant that depends on the maximum amplitude of the signal $m(t)$.

A sinusoidal signal with constant angular frequency ω_c is represented by:

$$x(t) = A.\cos(\omega_c t) = A.\cos(\theta(t)) \quad (2)$$

where $\omega_c t = \theta(t)$ is the instantaneous angle argument of the cosine function. If the frequency varies with time, as $\omega_i(t)$ in equation 1, the angle argument of the cosine function is the integral with time of that frequency:

$$\theta(t) = \int_{-\infty}^{t} \omega_i(\alpha)\, d\alpha = \int_{-\infty}^{t} (\omega_c + K_f m(\alpha))\, d\alpha \quad (3)$$

The process to generate a FM signal uses directly the relationship in equations 2 and 3: the output signal is the cosine of an angle $\theta(t)$ obtained by accumulating along time a constant ω_c added to the signal to modulate.

2.2 - FM Stereo

A FM stereo broadcast encodes a two channel audio signal by calculating the sum of the two channels $L + R = (left(t) + right(t))$ and the difference between them $L - R = (left(t) - right(t))$. The signal $m(t)$ is obtained by summing $L + R$, $L - R$ multiplied by a 38 kHz sine wave and a 19 kHz pilot sine wave (the constants K_s, K_d and K_p will be defined later).

$$m_{stereo}(t) = K_s(left(t) + right(t)) + K_d(left(t) - right(t)) \times \cos(2\pi \times 38k\, t) + K_p \cos(2\pi \times 19k\, t)$$

Figure 2 shows the spectrum of a FM stereo signal.

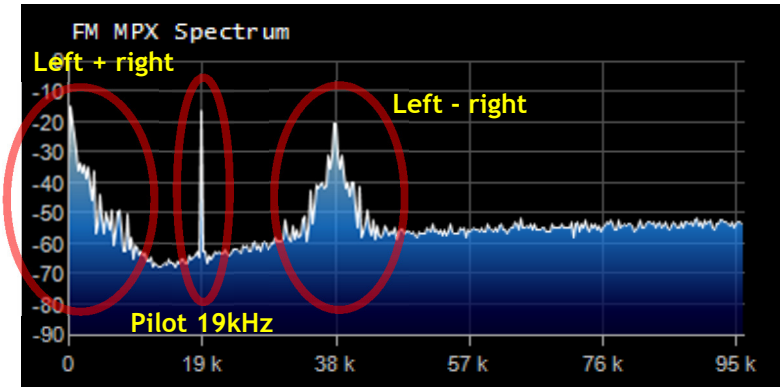
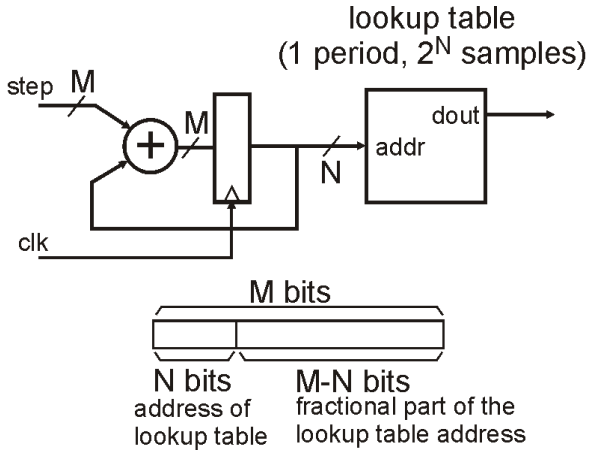


Figure 2 – Spectrum of a FM stereo signal (image created by the software-radio application SDRSharp – free download from airspy.com)

3 - Digital implementation

The construction of the FM stereo modulator requires the implementation of 3 different cosine functions, two with constant frequencies (19 kHz and 38 kHz) and one with the angular argument defined by equation 3. This is thus a fundamental building block for our system and we will start by implementing a parameterizable digital system to generate a sinusoidal wave.

One process to implement a digital calculator of the function cos(x) is called Direct Digital Synthesizer (DDS[1], see figure 3) and is based on a look-up table (a ROM-like memory) holding N equally spaced samples of one period of the function cosine (or any other periodic function). That memory is addressed by an accumulator register whose instantaneous value represents the angle argument of the cosine. The step of increment of the accumulator (the value accumulated at each clock cycle) dictates the frequency of the output cosine function. To generate the FM signal, that step must be proportional to $\omega_c + K_f m(t)$.



The frequency of the output wave is determined by equation 4, where F_{sine} represents the frequency of the output signal and F_{clk} Is the clock frequency.

$$F_{sine} = \frac{F_{clk}}{2^N} \times step \qquad (4)$$

Input **step** is a fixed point value with N integer bits and (M-N) fractional bits, defining the phase increment for each clock cycle.

Figure 3 – Direct Digital Synthesizer

Figure 4 presents a simplified block diagram of the complete FM stereo modulator. The target clock frequency will be 160 MHz and the whole circuit should be synchronous with the main clock, using a global synchronous reset.

The input signals (the left and right audio channels) are available as 18 bit two's complement words sampled at 48 kHz. Signals **LpR** and **LmR** (addition and subtraction of the two input channels) are multiplied by the constants **Ks** and **Kd** (4 bit unsigned) representing positive numbers between 0 (0.000b) and 1.875 (1.111b). The results of these multiplications are scaled to 18 bits and up-sampled to 4 x 48kHz = 192 kHz using a first-order interpolator (this block will be available as a RTL Verilog module). The **LmR** signal is then multiplied by a 38 kHz cosine function generated by a DDS, using the same 192 kHz sampling frequency. The result of this multiplication is scaled down again to fit into 18 bits. Finally, the **mstereo** signal is constructed by adding the scaled **LpR** signal, the **LmR** multiplied by the 38 kHz cosine and the output of a second DDS generating a 19 kHz cosine function. The output of this addition is scaled down to fit the 20 bit dynamic range.

The FM modulation is performed by a third DDS, where the accumulator increment **Phase** is a linear function of the **mstereo** signal: **Phase = stepWc + mstereo * Kf**. This DDS will have a lookup-table with 32 entries (for the whole cosine period), holding samples with 8 bits in two's complement signed format. With **stepWc**=1 and the clock frequency equal to 160 MHz, the central frequency of the output sine wave will be 5 MHz. The output of this DDS is applied to an external digital to analog converter to generate the output analog signal.

The parameters **Ks**, **Kd**, **Kp**, **Kf** and **stepWc** will be provided by output ports of the serial interface. Additional ports can be used to configure other parameters of your design.

[1] A brief introduction to DDS can be found in ftp://ftp.ni.com/evaluation/pxi/Direct_Digital_Synthesis.pdf

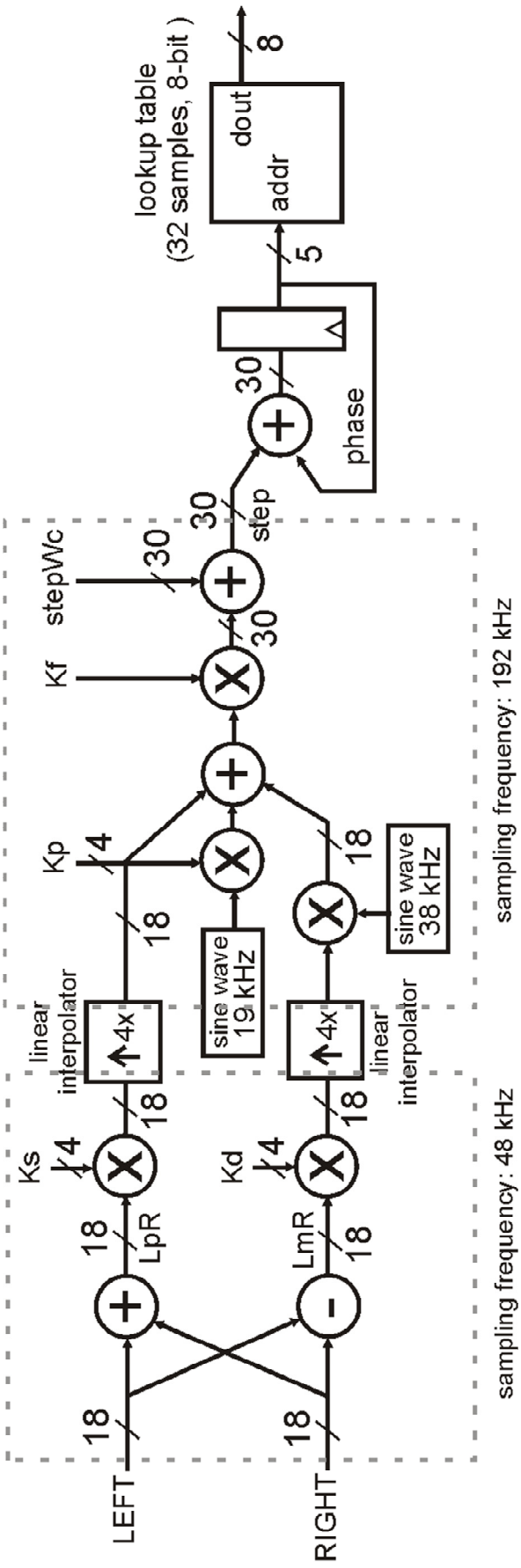


Figure 4 – FM stereo digital modulator

3.1 Design goals and constraints

The global design goal should be to minimize the area, measured as the number of lookup tables and flip-flops. The arithmetic blocks available in the FPGA (DSP48) should not be used in this design. A small sequential multiplier will be available to implement the multiplications shown in the datapath.

The clock frequency of the final DDS should be 147.456000 MHz. The rest of the circuit can use that same clock frequency (the easiest solution) or slower clocks with frequencies adjusted to the requirements of each section. This later option is more challenging as it requires to solve the clock domain crossing (CDC) synchronization issues. Note that a 48 kHz clock would be sufficient for the initial section before the interpolators, but in that case combinational (and large) multipliers need to be used.

3.2 Development stages – generic DDS

The first block to develop is a parameterizable digital sine wave generator, using the DDS architecture referred above. Build a Verilog module implementing the circuit represented in figure 2. The same Verilog module should be flexible enough to implement the 3 DDS blocks needed in the FM modulator. The contents of the lookup table can be loaded at compile time from a text file (either for simulation and synthesis) using the Verilog system task **`$readmemh( )`**. This module must have a clock enable input and a synchronous reset.

3.2.1 Verification kit for the generic DDS module

This verification kit is prepared to perform the logic simulation the Verilog module that implements the DDS sine generator. This is provided as a set of files organized with the same directory structure used for the previous projects. The kit contains:

./matlab/dds.m	Matlab/Octave script to create the simulation data required for the testbench and to analyse the sine wave for different configurations of the DDS parameters;
./simdata	The script dds.m will create in this directory two data files that will be used by the Verilog testbench;
./src/verilog-tb/dds_tb.v	The testbench to perform the functional and the post-synthesis simulation;

To use this kit **you must follow exactly the instructions below** (this same text is included in the beginning of the testbench dds_tb.v:

1. Edit the script ./matlab/dds.m and adjust the following parameters to match the DDS configuration you want to verify:

```
duration = 0.1;       % Duration of the output test signal (seconds):
Fs = 192000;          % Sampling frequency (Hz):
Fout = 19000;         % required output frequency (Hz):
Nbits_sine_LUT = 9;   % Number of bits per sample in lookup-table
Nsamples_LUT   = 128; % Number of samples in the lookup-table (int power of 2)
Nfrac = 6;            % Number of bits of the fractional phase:
                      %   note that the number of bits in the integer part of the
                      %   phase is given by log2( NsamplesLUT )
```

2. Run the script dds.m in Matlab/Octave, in directory ./matlab. This will generate the data for the DDS lookup table (file ./simdata/DDSLUT.hex) and a vector with the output sine samples generated by the DDS (file ./simdata/DDSout.hex). These files are ASCII with one signed data sample per line in hexadecimal format and the high-order bits to

the left of the Nbits_sine_LUT bit padded with zeros (only the lower Nbits_sine_LUT are meaningful). To use the DDSLUT.hex file to load you lookup-table (register array sineLUT), you can include the following Verilog code into your module:

```
reg [31:0] sineLUT[ 0 : Nsamples_LUT-1 ];
initial
  $readmemh("../simdata/DDSLUT.hex", sineLUT );
```

The lookup-table should be defined as a 32-bit register array, even if the data samples use less bits. The output port of the DDS module should only output the meaningful bits. This script also outputs the required phase increment for the output frequency set.

Register this number as it will be necessary to configure the parameter PHASE_INCREMENT in the testbench

3. Adjust the following simulation parameters in the testbench:

```
parameter FS             = 192000; // Sampling frequency
parameter MAX_SIM_SAMPLES = 19200;  // Maximum simulation time is 0.1 second
parameter N_OUTPUT_BITS   = 9;      // Number of valid bits in the output word
parameter PHASE_INCREMENT = 32'b001100_101010; // 12.6562500 in binary: 001100.101010
                                    // for generate a 19 kHz sine wave
```

4. Setup and run the simulation in QuestaSim:

4.1 Create a QuestaSim project in ./sim

4.2 Import to the project your dds.v and this testbench (./src/verilog-tb/dds_tb.v). You may need to adjust the module and signal names and define the parameters needed to configure your module: number of samples in the DDS LUT, number of bits per sample and number of bits of the fractional part of the phase. Note that the example of instantiation included in this testbench does not define any parameter.

4.3 The testbench compares automatically the results generated by the DDS module with the results generated by the Matlab script. If errors are found and you need to analyse the signals in more detail, the signal 'outsineNbits' contains the output with only the number of bits defined by parameter 'Nbits_sine_LUT'. This signal can be plotted in the waveform window using radix decimal and format analog.

4.4 If no errors are reported for the various configurations needed, congratulations! You have created a fundamental building block of the FM modulator.

5. If the simulation succeed, you can proceed with the RTL synthesis and post-synthesis verification, using this same testbench (refer to the guide of laboratory project 2).

4 - IP blocks

Two IP (*intellectual property*) blocks are made available for free: a signed multiplier and a 4X linear interpolator. These are provided as Verilog RTL synthesizable modules, including very basic testbenches to illustrate the module instantiation and the respective eco systems. These modules are available in directory **./IP-cores/seqmultNM** and **./IP-cores/interpol4x**.

4.1 – Sequential signed multiplier – module seqmultNM (seqmultNM.v)

The sequential signed multiplier implements the shift-add sequential multiplication algorithm in **N+2** clock cycles (**N** is the number of bits of the multiplier). The module is instantiated as:

```
seqmultNM     #(          // definition of parameters
                .N( N ),  // parameter N = number of bits of the multiplier
                .M( M )   // parameter M = numbero of bits of the multiplicand
               )
seqmult_1                 // instance name
        (
          .clock( clock ), // Master clock
          .reset( reset ), // Master reset, synchronous and active high
          .start( start ), // Set to 1 during one clock cycle to start the multiplication
          .ready( ready ), // Set to 1 when the multiplier is ready to accept a new start
          .A( A ),         // Multiplicand, signed  M bits
          .B( B ),         // Multiplier, signed    N bits
          .R( R )          // Result, signed        M+N bits
        );
```

The module receives two parameters to specify the number of bits of the multiplicand (parameter **M**) and the number of bits of the multiplier (parameter **N**). This module requires **N+2** clock cycles to complete one multiplication and the signed result has **N+M** bits. Setting input **start** to 1 during one clock cycle will load the operands **A** and **B** (multiplicand and multiplier, respectively) and start the shift-add iterative process. When the iterative process is running, the output **ready** is set to 0 and returns to 1 when the multiplication ends. The result is ready at output **R** when output **ready** is 1 after an activation of **start.** While the process is running, the output **R** has meaningless data.

Figure 5 shows the simulation waveforms for a multiplication with a 14 bit multiplicand (input **A**) and a 12 bit multiplier (input **B**). The result in output **R** is ready **N+2** clock cycles after **start** is set.

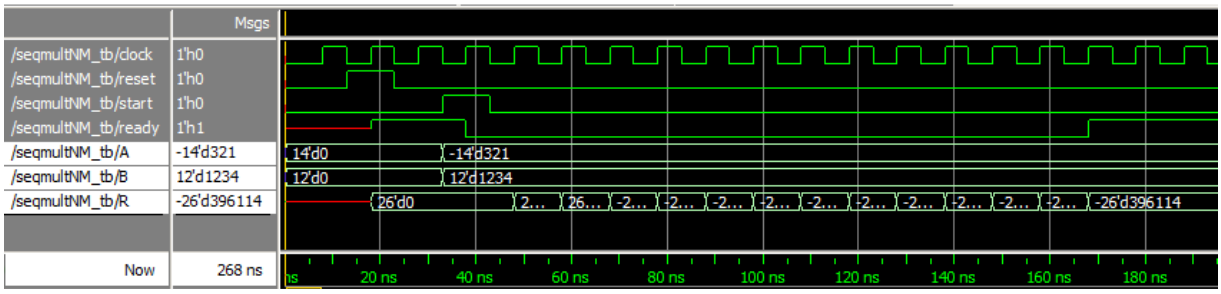


Figure 5 – Example of simulation waveforms for the sequential multiplier.

4.2 - Linear interpolator x4 - module interpol4x (interpol4x.v)

The linear interpolator receives a stream of 18-bit samples (input **xkin**) sampled at a rate defined by the clock enable signal **clkenin** and generates a stream of 18-bit samples (output **ykout**) with a sampling frequency determined by the clock enable signal **clken4x**. The output sampling frequency must be exactly 4 times higher than the input sampling frequency and the additional samples at the output are obtained by the linear interpolation between each two samples of the input stream.

The module can be instantiated as:

```
interpol4x
interpol4x_1(
     .clock( clock ),         // Master clock
     .reset( reset ),         // Master reset, synchronous active high
     .clkenin( en48000Hz ),   // Input clock enable
     .clken4x( en192000Hz ), // Output clock enable (4X the input enable)
     .xkin( xkin ),           // Input signal, 18 bit signed
     .ykout( ykout )          // Output signal, 18 bit signed
     );
```

Signals **clkenin** and **clken4x** are clock enable signals (active only during one clock cycle) whose frequencies dictate the input sampling rate and the output sampling rate. In your design these frequencies will be 48 kHz and 192 kHz.

Figure 6 shows in detail the relationship between the two clock enable signals and figure 7 presents a simulation waveform created with the simple testbench included in the kit. To observe the analog view, configure the signal with radix decimal (signed, two's complement) and format analog (set the maximum and minimum values in the signal and the height of the display window reserved for that signal).

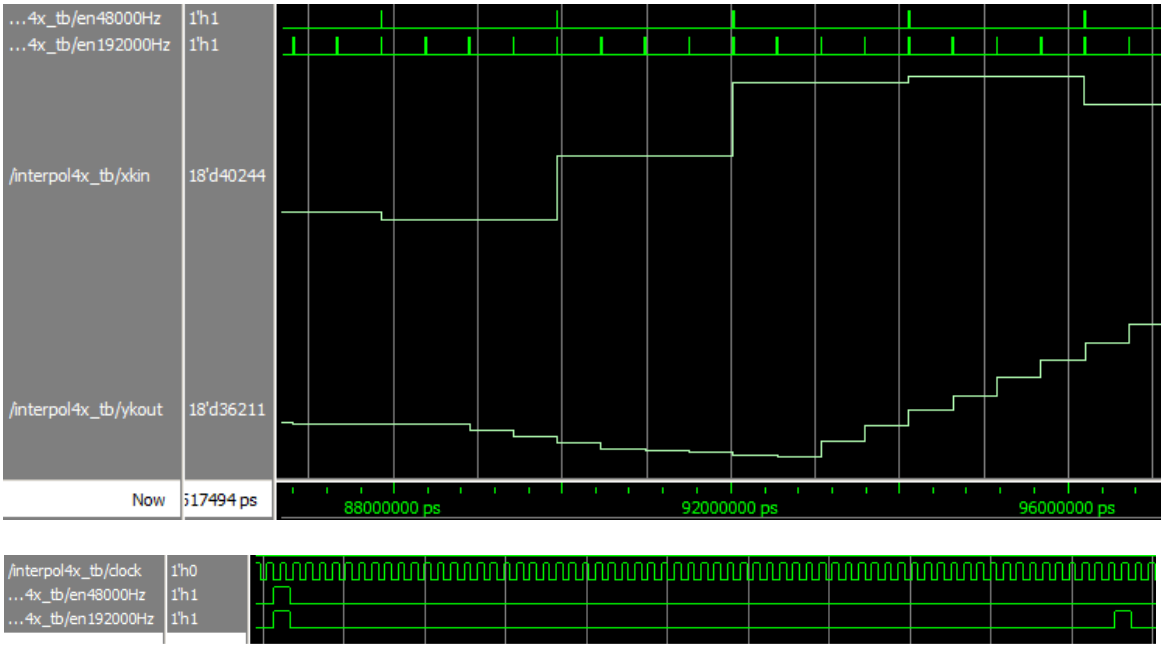


Figure 6 – Example of simulation waveforms for the 4X interpolator – relationship between the two clock enable signals.

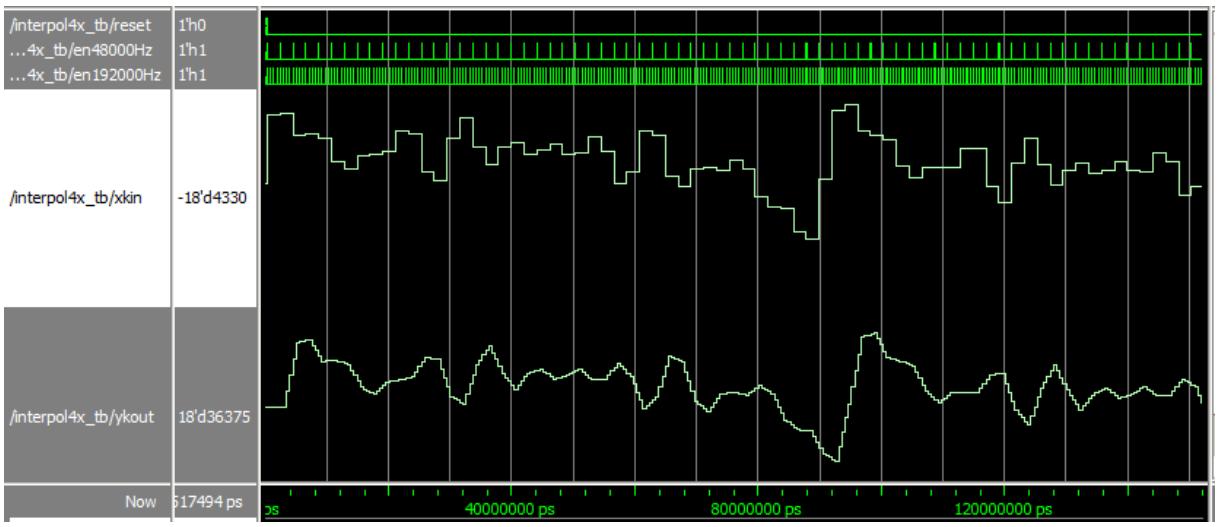


Figure 7 – Example of simulation waveforms for the 4X interpolator – input signal samples at Fs and output signal sampled at 4xFs with linear interpolation.