Implementation of FPGA Based Digital Direct Synthesizer

Zepeng Chen

Contents

1 Introduction **1**

1.1 Principle of DDS 1

1.2 Encoding 2

2 Implementation by FPGA **3**

2.1 Architecture 3

2.2 Debouncing 4

2.3 Key controller 6

2.4 DDS 6

3 Experimental result **9**

1 Introduction

1.1 Principle of DDS

The DDS is generally composed of a reference clock, a phase accumulator, a phase to amplitude conversion unit, a D/A converter and a low pass filter, etc. On receiving each clock pulse f_{clk}, an addition of frequency control word X to cumulative sum of the phase data, which is output from accumulate register, is implemented on a N-bit adder, and the sum of results Y is immediately fed into accumulation register[1]. The newest phase data generated in last clock cycle are on the one hand feedback to the adder by the accumulator register, so that an addition of frequency control data X with the adder's input is implemented in the next clock cycle; the other hand, as a sample address this value is sent into the phase to amplitude conversion circuit, through which corresponding waveform data are output according to the address value. Finally, the waveform data are converted into analog waveform by the D/A converter and low-pass filter. Linear phase accumulation will continue on the phase accumulator in the role of the reference clock, and an overflow will occur when the cumulative result of the phase accumulator is equal to or greater than 2^N, then it turn back to the initial state, and waveform of a cycle is completed and output. The output frequency has the following relation with frequency control word X.

$$f_{out} = \frac{f_{clk}}{2^N} \times X$$

The architecture of DDS shown as below[2].

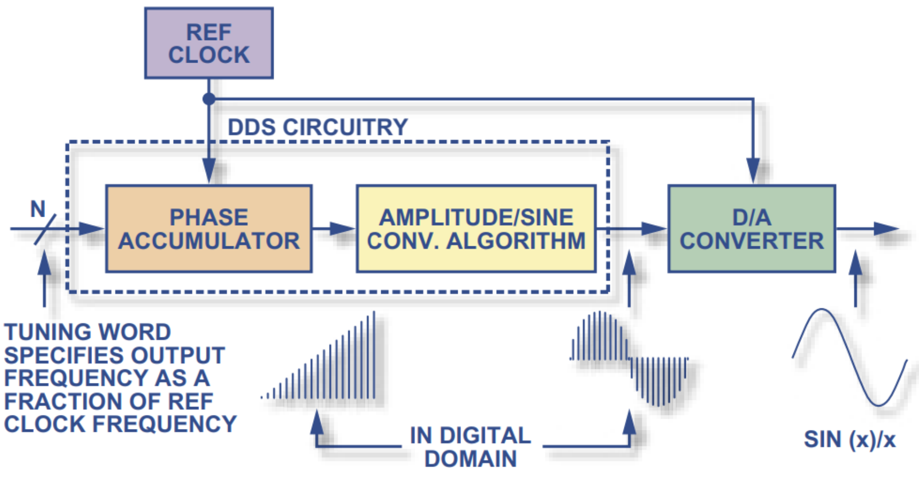

REF CLOCK
DDS CIRCUITRY
N
PHASE ACCUMULATOR
AMPLITUDE/SINE CONV. ALGORITHM
D/A CONVERTER
TUNING WORD SPECIFIES OUTPUT FREQUENCY AS A FRACTION OF REF CLOCK FREQUENCY
IN DIGITAL DOMAIN
SIN (x)/x

Figure 1: courtesy of analog.com

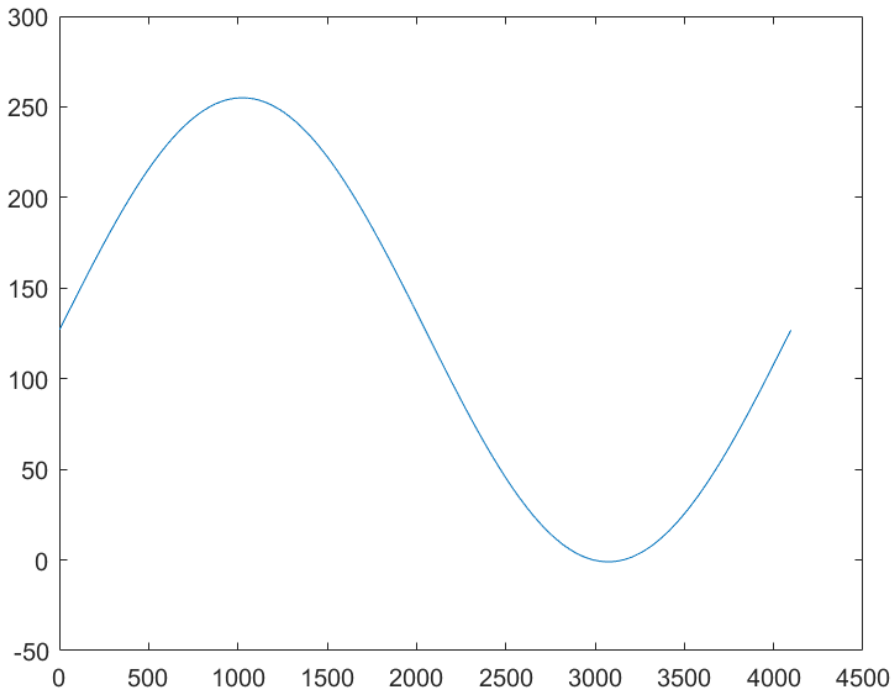
1.2 Encoding

Since we need to generate signal via digital technique, naturally, we should encode the waveform. We can use MATLAB to generate the waveform and write the sampling data into Altera standardized memory initialization file (ending with .mif). The partial content of our sinusoid shown as below.

```
WIDTH=8;
DEPTH=4096;
ADDRESS_RADIX=UNS;
DATA_RADIX=UNS;
CONTENT BEGIN
0 : 127;
1 : 127;
```

```
2 : 127;
...
...
...
4094 : 128;
4095 : 128;
END;
```

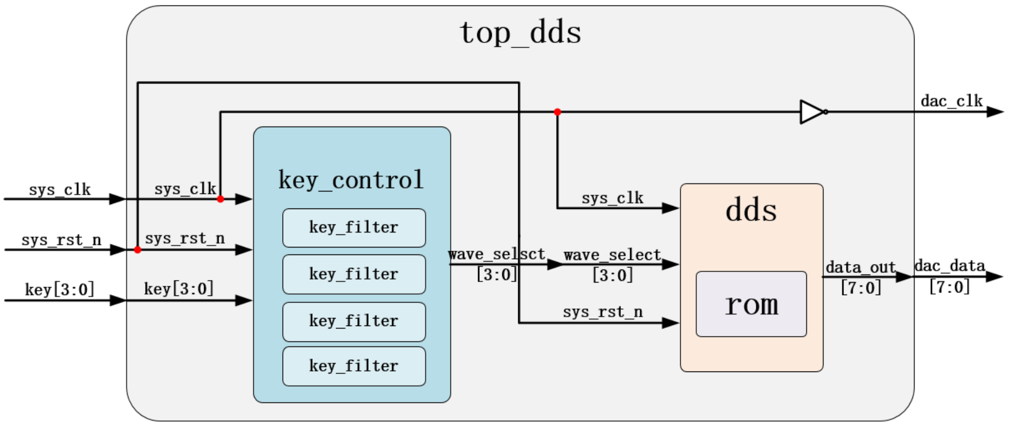
The base sinusoid as below.



2 Implementation by FPGA

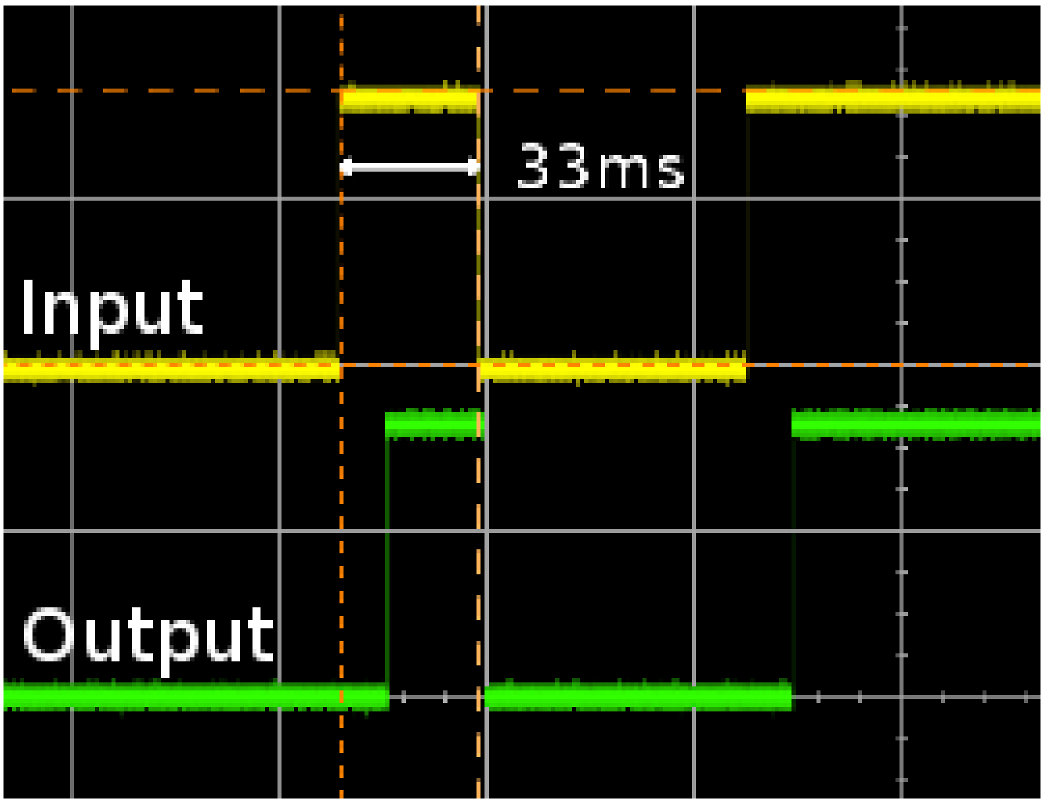
2.1 Architecture

FPGA is a good device to implement DDS. In this project, we design a multiple signals generator including triangular wave, square wave, and sinuous wave. We need to store these data of waveform in the on-chip rom which can be initialized by IP core. The data can be generated from Matlab. We can use the key to switch these waves. In order to switch smoothly, we need to implement the key debouncing module. Other modules include key control and our core module dds. The architecture diagram shown below.

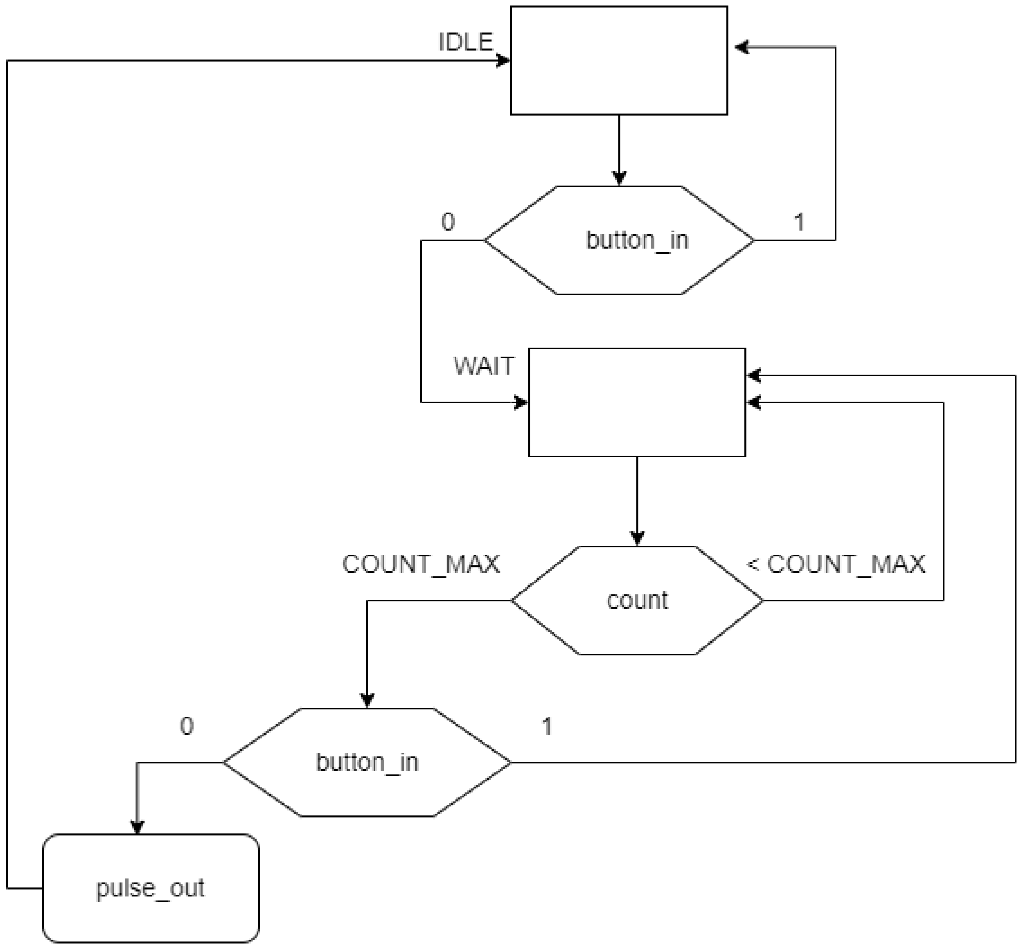


2.2 Debouncing

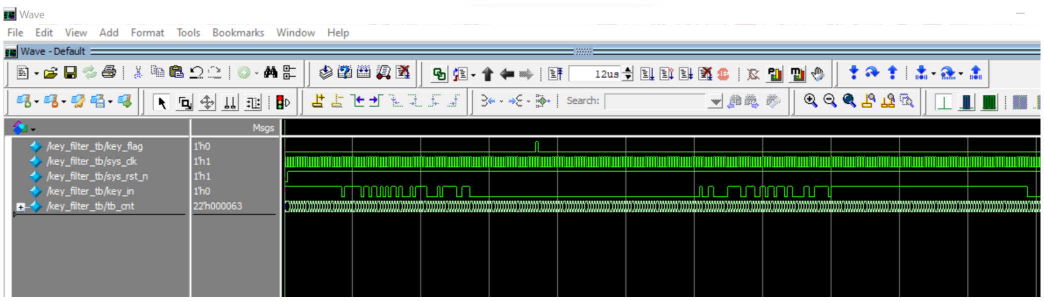
The button press and release timing captured by oscilloscope displayed as below.



From this figure, we can see that within 10 ms after the button pressed down, the electrical level of input is unstable. So we can ignore that period, and detect the input since 20 ms. We can declare a constant which is COUNT_MAX such that 20 ms will be passed when the counter reaches COUNT_MAX. For DE2-10 standard kit, the clock frequency is 50MHz, 20 ns will pass for every clock. The COUNT_MAX will be 10^6. Another two parameters "button_in" and "pulse_out" denotes the input and output respectively. When the button is pressed down, button_in is 0, after 20 ms, the pulse_out becomes 1. The process can be illustrated as the following ASM chart.



The simulation result from ModelSim shown as below.



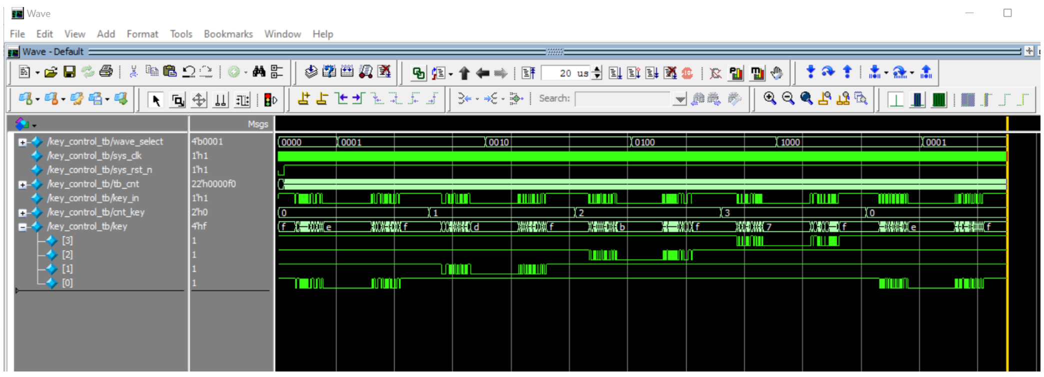
```
/key_filter_tb/key_flag     1'h0
/key_filter_tb/sys_clk      1'h1
/key_filter_tb/sys_rst_n    1'h1
/key_filter_tb/key_in       1'h0
/key_filter_tb/tb_cnt       22'h000063
```

2.3 Key controller

In this project, the output data depends on which key pressed down such that different waveform can be switched over.This module is to generate a signal named ”wave_select”. It is encoded as below.

- 0001 when key0 is 0.
- 0010 when key1 is 0.
- 0100 when key2 is 0.
- 1000 when key3 is 0.

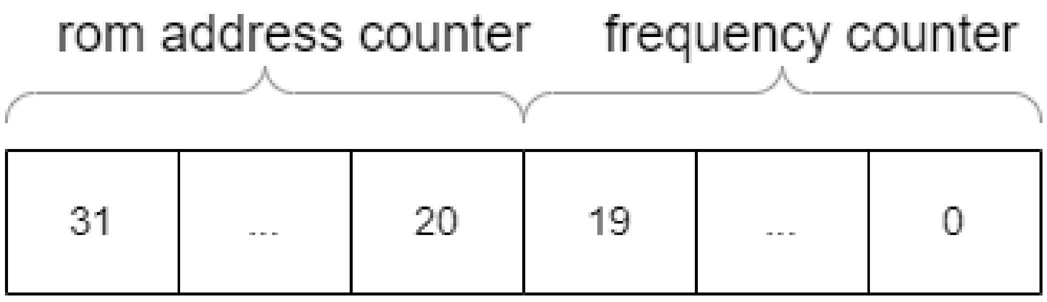
The simulation result shown as below.



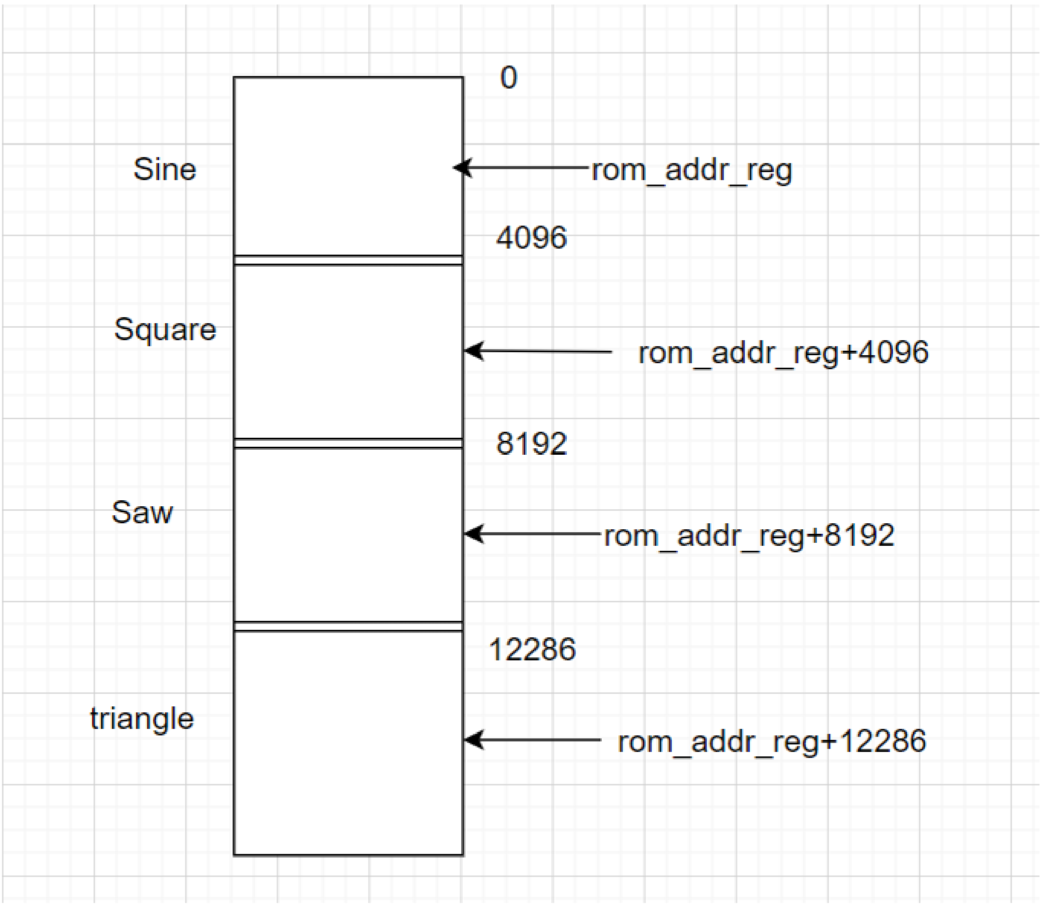
```
/key_control_tb/wave_select   4'b0001
/key_control_tb/sys_clk       1'h1
/key_control_tb/sys_rst_n     1'h1
/key_control_tb/tb_cnt        22'h0000f0
/key_control_tb/key_in        1'h1
/key_control_tb/cnt_key       2'h0
/key_control_tb/key           4'hf
    [3]                       1
    [2]                       1
    [1]                       1
    [0]                       1
```

2.4 DDS

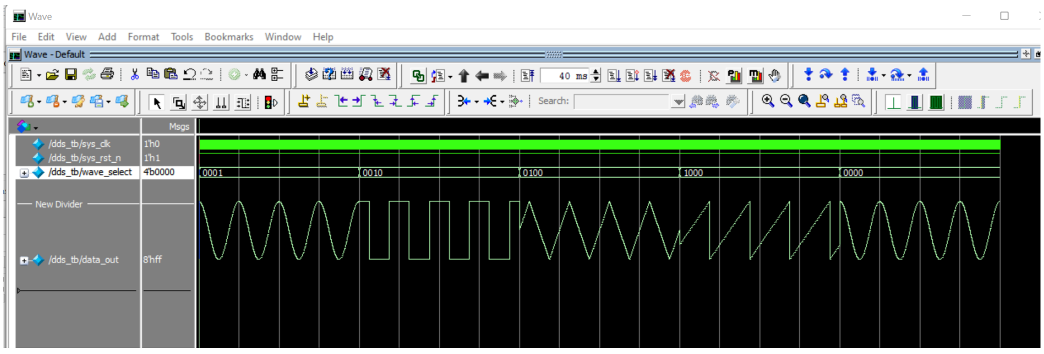
There are two functionalities of DDS module, the first one is to control output frequency, the second one is two control initial phase. For frequency controller, we can declare a 32 bit register as the following figure shown.



The lower 20 bit is used for frequency counter and the upper 12 bit is used for rom address. If the frequency counter overflows, the rom address counter will be added by 1. The frequency control word is used to control the step of counter such that the incremental time of rom address counter will be changed. For each waveform, the number of sampling points is 4096 which is 2^{12} corresponding with rom address counter. While the four waves have been put in one mif file. So the rom address ranges from 0 to 4096×4. We need another register which should be 14 bit to reach all the possible address. This technique is illustrated as below.

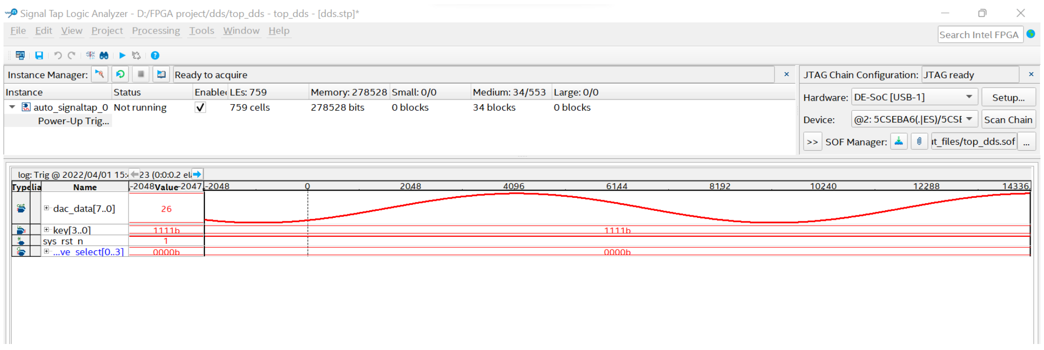


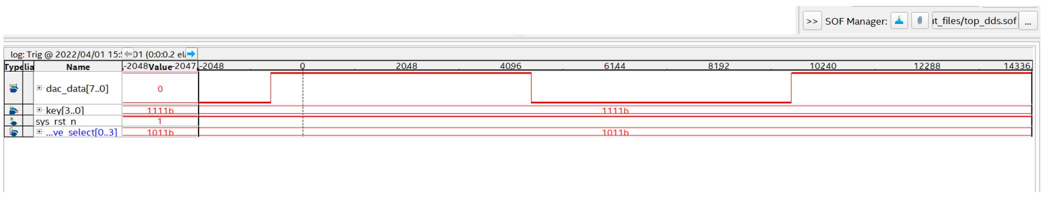
The rom_addr_reg ranges from 0 to 4095, if we want to switch over from sine to square, we just need to add an offset 4096. The offset for saw and triangle wave is 8192 and 12286. The simulation result displayed as below.

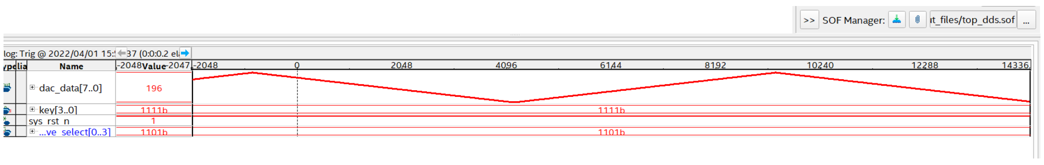


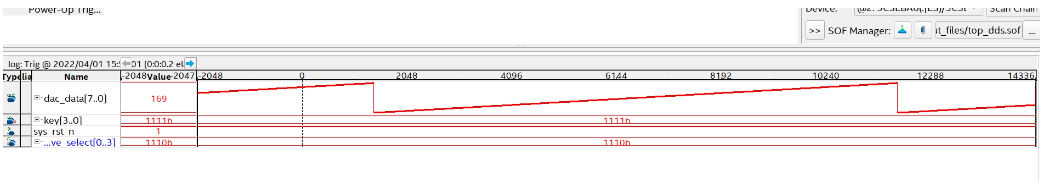
3 Experimental result

The SignalTap II Logic Analyzer Editor allows us to debug our design in real-time and at high-speed while performing an analysis in the Quartus II software. With the SignalTap II Logic Analyzer Editor we create one SignalTap II File (.stp) that contains all SignalTap II Logic Analyzer configuration data. When we run a SignalTap II analysis we capture data and save it to the SignalTap II File, which is then included in our design[3]. SignalTap II just likes a mini oscilloscope which can display the variable we want to peep. For this project, I use this tool to demonstrate the result. The results figures are listed as below.









References

[1] Jiang, Wei; Yuan, Fang; Yang, Liu Qing. Applied Mechanics and Materials; Zurich Vol. 713-715, (Jan 2015):1031-1033. DOI:10.4028/www.scientific.net/AMM.713-715.1031

[2] https://www.analog.com/en/analog-dialogue/articles/all-about-direct-digital-synthesis.html

[3] https://www.intel.com/content/www/us/en/programmable/quartushelp/13.0/mergedProje
view_using.htm