**ROM logic**

The Digital Pattern Generator (Patterns) lets you define three types of output objects: Signal, Bus or ROM logic. Multiple objects of same or different types can be built.

The figure below shows the block diagram associated with a single DIO pin. A Digital Input/Output (DIO) line can be assigned as Output in at most one object. When a line is used by the Static I/O as an output element (Slider, Button, or Switch), this has priority over the signal configuration in Patterns (not shown in the block diagram).

Signals and Buses generate time sequenced output values: the sampling rate derives from the system clock, and the sample values are either algorithmically generated (clocks, pulses, counters, etc.) or read from a Lookup Table (custom). In the latter case, a Counter points the current sample in the Lookup Table Memory.

The ROM logic is different as it generates output values as Boolean function of DIO arguments. The function Truth Table is stored in the Lookup Table Memory. In the figure below, MUX2 selects Lookup Table Memory, and MUX1 selects the n-bit input bus. The input buffer IB is only present for the first n DIO pins (0 to n-1). In this configuration, ANY DIO pin can implement a Boolean function of DIO pins 0 to n-1. Since the Lookup Table Memory is implemented in an FPGA Block RM, the Boolean function is synchronized on the prescaled clock.

Lookup Table Memory

MUX1

n-bit Counter

MUX2

Algorithmic generator

OB

n-bit input bus

IB

DIOx

Prescaler

system clock

Figure 1 DIO pin block diagram

## Using ROM logic to implement combinatorial Boolean functions

As long as none of the DIO pins defined as outputs in a ROM logic object is also used as input in the same object, there is no feedback path and the Boolean function is combinatorial. However, the output is synchronized to the prescaled clock.

## Using ROM logic to implement combinatorial Boolean functions

## A DIO pin used as input (argument) in its’ own Boolean function results in a feedback loop, and the DIO pin is a state bit. One or more state bits build a state variable in a Finite State Machine (FSM).

The figure below shows the generic block diagram of a Mealy FSM mapped on a ROM logic object. A Moore FSM does not use the Inputs in Output logic block.

Notice that pins DIOx used both as inputs and outputs form the State Variable, pins used as inputs only (DIOy) are FSM inputs and pins used as outputs only (DIOz) are FSM (synchronized) outputs.

The FSM structure can be implemented in a single or in multiple ROM logic objects. In the latter case, a clean design would use a ROM logic object for the Next State Logic and the State Register and another one for the Output Logic and the Output Register.

Figure 2 DIO pin block diagram

Lookup Table Memory

OB

Current state

IB

DIOx

Prescaler

system clock

Next State Logic (comb)

State Register (synchronous)

Output Logic (comb)

Output Register (synchronous)

Inputs

DIOy

IB

OB

DIOz

## ROM logic User Interface in WaveForms 2015

## In the Pattern Generator, click the green “plus” sign and select ROM logic, as shown below.

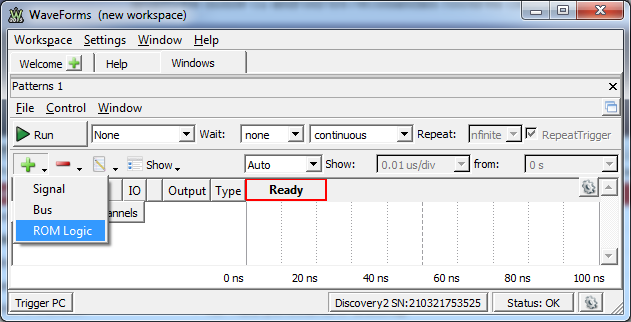


Figure 3 Opening a new ROM logic object.

In the Pop-up window, select the Properties tab. Select or type in the synchronizing frequency. Use the green “plus” signs to add input and output signals in the left and right lists. Use the green arrows to move signals up or down in the lists. The list order will be preserved in the Truth Table tab.

Select the Truth Table tab. Edit or import the truth table. Use the Truth Table rules:

1. Change the signal names, if convenient (optional). Changed names will be used in the Truth Table.
2. If a signal is used in both input and output panes, it is a state variable; the input pane shows its’ value in the Current State and the output pane indicates its’ value in the Next State.
3. Use 0, 1 or X (don’t care) in the input pane.
4. An expanded Truth Table (no X input values) has 2^n lines for n input signals. A line including k X values is the collapsed version of 2^k lines of the expanded Truth Table.
5. Make sure that all possible input cases are covered, i.e. the logical reunion of all input lines is 2^n.
6. Use 1, 0, “signal name” or “not signal name” in the output pane.
7. The table lines are “sequential” as a line can override lines above it in the table. For example, the first line in the table might be used to describe the “default” output values, while subsequent lines override particular cases.
8. The same functional block might be split in multiple ROM logic objects, to optimize the Truth Table(s).

The rules and ROM logic GUI operation is demonstrated in the following examples.

## Example: Three input, synchronous AND gate.

In the figure below, the ROM logic object is named AND3 (optional). DIO0…DIO2 are selected as inputs, DIO3 as output. The signal names is changed to i0…i2, respectively o3 (optional). The synchronizing frequency is set to 5MHz, meaning the output signal will be synchronized to a 5MHz clock with 0 degrees of phase, relative to the moment of running the Pattern Generator. This way, the output signal is synchronized to other signals generated by the Pattern generator.

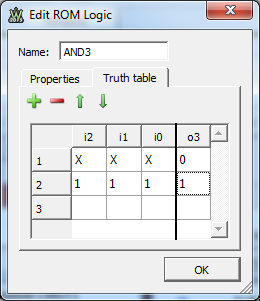
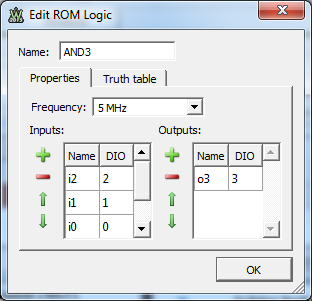


Figure 4. 3 input synchronous AND gate ROM logic definition

In the truth Table tab, the first line describes the “default” case, with the output signal value of 0. The three x-es in the input pane make this line equivalent to 2^3=8 lines in the expanded Truth Table. However, the second line overrides one line of the eight, forcing output signal to 1, when all inputs are 1.

The equivalent VHDL code is:

-- Line 1 in the truth table

3InputAndGate: process(i2,i1,i0)

begin

o3 <= ‘0’; -- default output is ‘0’

-- Line 2 in the truth table

if i2 = ‘1’ and i1 = ‘’ and i0 = ‘1’ then

o3 <= ‘1’;

end if;

end process;

Notice that Truth Table lines are “sequential”, with the same meaning as the lines within a process are: a line can override the effect of previous lines, in both cases.

To verify the behavior of the 3 input AND gate, Run the Pattern Generator and the Static IO. Set the DIO0…DIO2 as Switches and keep DIO3 al LED, in the Static IO. Notice that DIO3 is High only when all DIO2, DIO1 and DIO0 are High.

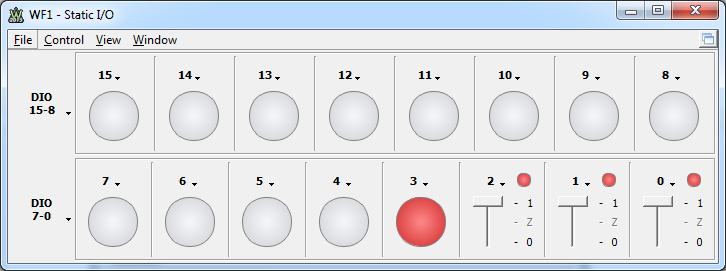


Figure 5 Using Static IO to verify the 3 input synchronized AND gate

Make sure you close the Static IO.

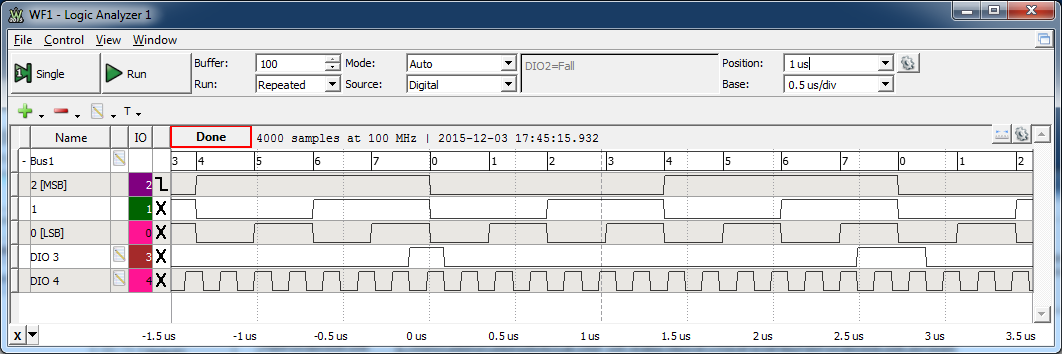
To observe the dynamic behavior of the synchronized 3-input AND gate, define a counter on DIO2…DIO0 in the Pattern Generator, as shown below. The clock frequency is set to 3MHz. However, the closest frequency WaveForms can generate is 2.941MHz. This frequency is chosen to not be integer multiple or submultiple of the ROM logic frequency (5MHz)

Also define a 5MHz clock on DIO4 (same as for the ROM logic). This clock is not used by the ROM logic, but will serve as reference for observation in the Logic Analyzer.

Open a Logic Analyzer and define a bus with DIO2…DIO0, and individual signals for DIO3 and DIO4. Set trigger on the falling edge of DIO2. Set convenient time base and position.

Run Pattern Generator and Logic analyzer. Observe instable image for DIO3 and DIO4 , as their frequency and phase is not correlated with Bus1.

Run a Single step in the Logic Analyzer. Repeat several times. Observe that DIO3 (the AND gate output) does not synchronize with the edges of BUS1 inputs, but is synchronous with the 5MHz clock DIO4.



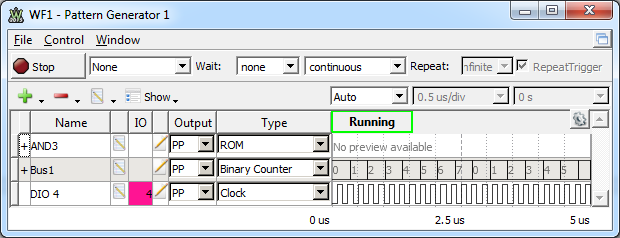
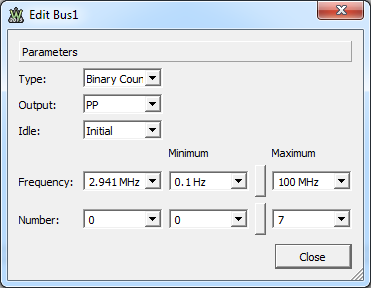


Figure 6. Generating input signals and reference clock for the 3 input synchronized AND gate

Figure 7Using Logic Analyzer to verify the ROM logic 3-input synchronized AND gate

## Example: Binary counter with Enable input.

In the figure below, DIO8 is renamed i8 and is used as Count Enable. DIO7…DIO0 are renamed s7…s0 and are the output bits of the 8-bit binary counter.

Notice how x-es, signal names, and negated signal names are used in the Truth Table.

When LSB s0=’0’, it will change to ‘1’ only if i8=’1’ (line 1 – equivalent to 2^8=256 expanded lines). All other lines cover the cases when s0=’1’.

When s1=’0’ and s0=’1’, both will swap if i8=’1’ or keep state if i8=’0’ (line 2 – equivalent to 2^7=128 expanded lines).

All nine liens are equivalent to 2^9=512 expanded lines and define all possible cases (8 bit+1 count Enable).

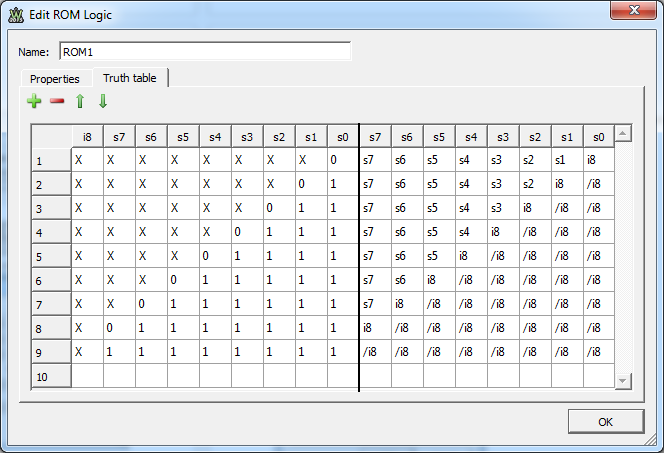


Figure 8. ROM logic defining an 8-bit binary counter with Enable

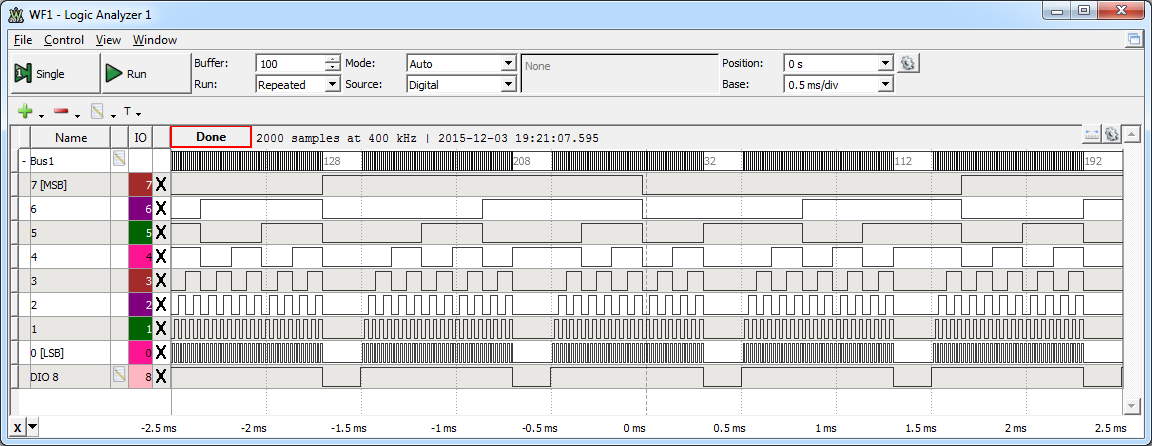


Figure 9. The behavior of the 8-bit binary counter with enable.

To verify the behavior, a clock signal is added on DIO8 (frequency=1kHz, Duty Factor=80%).

In the Logic Analyzer, create a bus from DIO7…DIO0 and a signal DIO8.

Run Pattern Generator and Logic Analyzer (Single step for Logic Analyzer). Observe the behavior of the binary counter.

## Example: PWM modulator

In the figure below, an n-bit binary counter generates a digital sawtooth signal. The modulator samples are generated by a LookUp Table. The comparator drives a High when the sample value is higher than the counter content. The carrier frequency is:

and needs to be much higher than the sampling frequency.

>

n-bit counter

n-bit sample LUT

fcnt

DIOx

fsample

Figure 10. PWM block diagram

First, a 5 bit counter is implemented in a ROM logic object, as below, on DIO4…DIO0, renamed s4…s0.

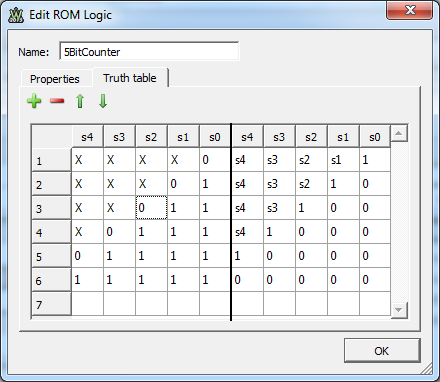
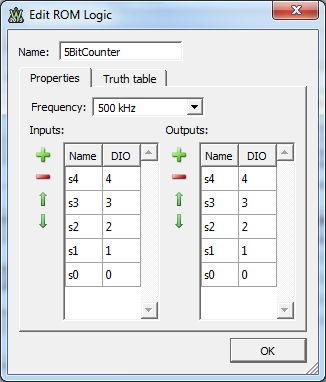


Figure 11. 5 bit counter ROM logic

A second ROM logic object is created to compare the DIO9…DIO5 word with DIO4…DIO0. DIO10 is High when DIO9…DIO5 is higher.

Notice again how higher order bits (later lines in the Truth Table) override the lower order bits (earlier lines in the Truth Table).

A 5-bit bus is created to generate the modulator samples (DIO9…DIO5).

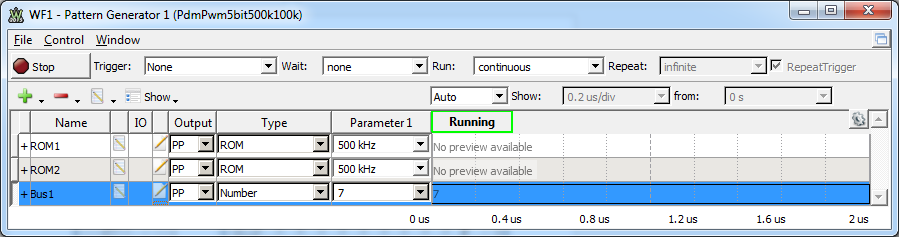


Figure 12.Pattern Generator prepared for PWM with static input

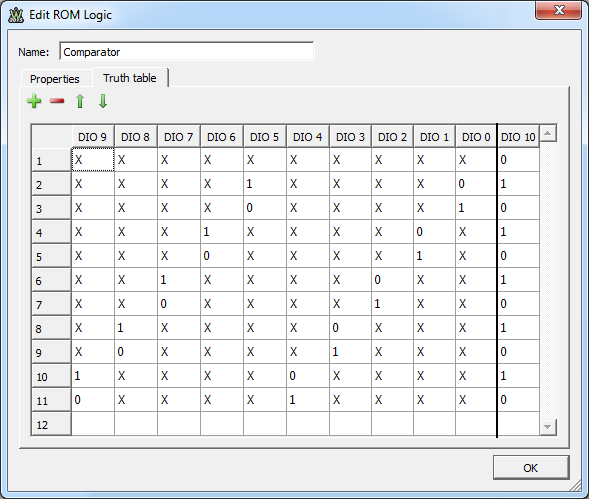
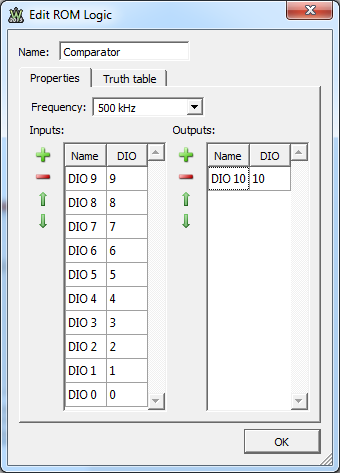


Figure 13. 5-by-5 bit Comparator ROM logic

For the first experiment, the bus type is set to Number, to generate a static value between 0 and 31.

An external RC Low Pass Filter (10nF, 1kΩ) is used to build the analog signal, removing the carrier from the PWM signal. The WaveForms scope shows the PWM signal (channel 1) and filtered signal (channel 2).



Figure 14. RC Low Pass Filter

Notice the channel 1 frequency and duty factor (both on the graphical view and measurement pane).

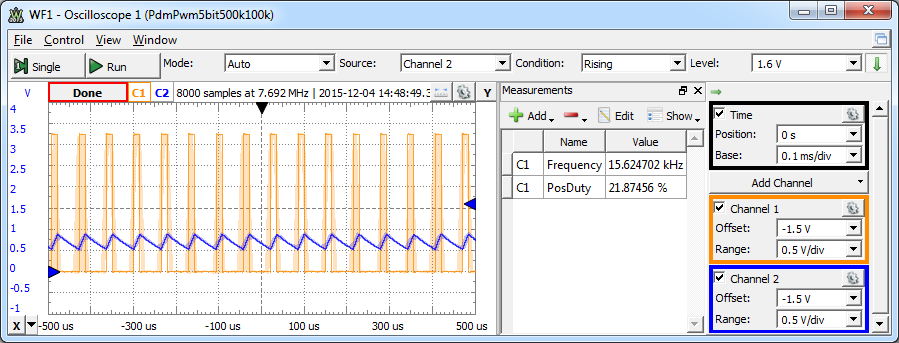


Figure 15. Static PWM Scope capture

Correlated with the values set in the Pattern Generator, they should be:

Modify the sample value in the Pattern Generator and observe the effect on the PWM and analog signals on the scope.

For the second experiment, Bus 1 in the Pattern Generator is set to custom, to drive a sinus signals with 100 samples per period.

First, a .csv file is generated with 100 values of form:

Then, Bus1 type is set to custom and imports the above described .csv file. The clock frequency is set to 100kHz, for a sampled sinus signal frequency of:

The scope shows variable Duty Factor PWM signal on channel 1, and an analog signal on channel 2, approximating a sinus signal. The carrier frequency is constant (same as above). The Duty Factor is variable with a mean value of 50%.

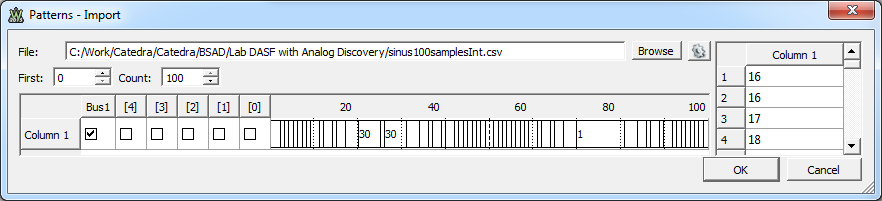
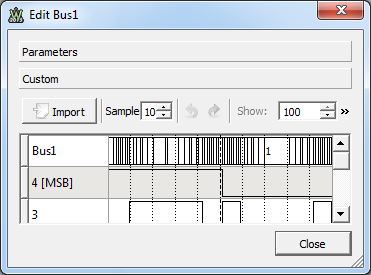
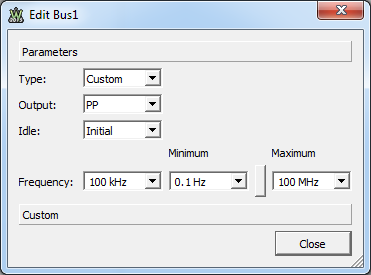
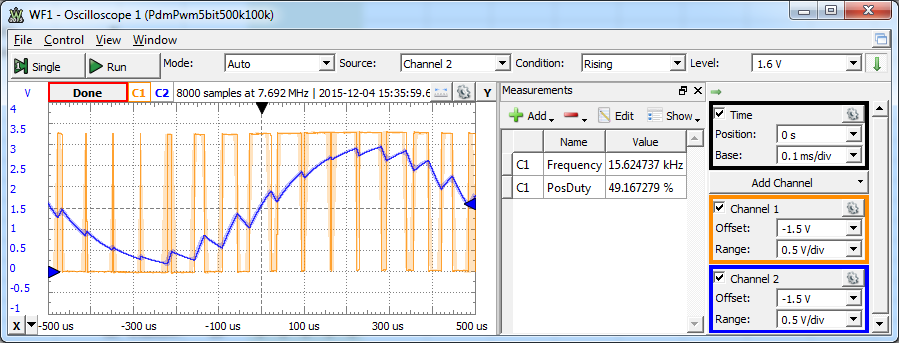


Figure 16. Custom Bus1 for 100 samples sinus

Figure 17. Sinus PWM and analog signal



Example: PDM modulator

In the figure below, an n-bit accumulator register is used to add the sample value over and over. The Carry Out bit is the Pulse Density Modulated (PDM) signal. The accumulator is refreshed with fcnt frequency, while samples succeed with fsample frequency. Same frequencies and names are used as for PWM, for comparison.

Figure 18. PDM modulator block diagram

+

n+1 bit Carry Out

accumulator

register Sum Out

n-bit sample LUT

DIOx

fsample

fcnt

For being able to switch very fast from PWM to PDM and back, a second pattern Generator instance is open from the already existing one:

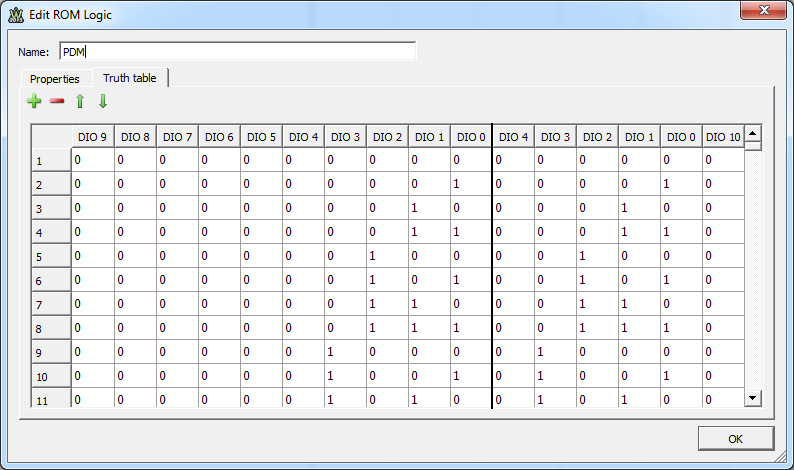
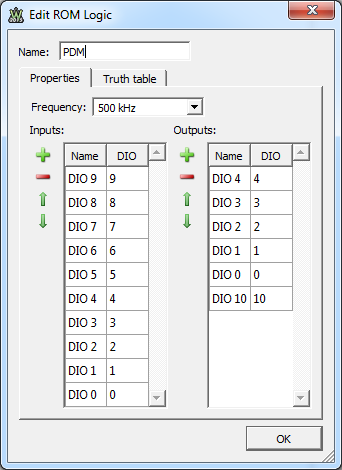


Figure 19. Implementing a PDM in ROM logic

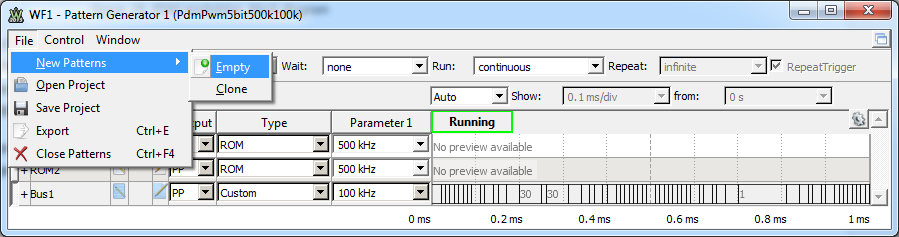


Figure 20. Open a second pattern generator

A 5-bit PDM modulator is implemented in PDM ROM logic object, with DIO 9…DIO5 as “Sample”, DIO4…DIO0 as “Sum” and DIO10 as “Carry” which is the PDM signal.

The truth table is fully expanded, with 1024 lines. The full pane contains all binary values of DIO9…DIO0. The right pane contains the 5-bit + carry accumulated sum:

(DIO10,DIO4…DIO0) <= (DO9…DO5) + (DIO4…DIO0)

To compare with PWM, similar examples are used. The same external RC filter, the clock frequency of the modulator, the sinus sample LUT and sample frequency are all the same as in the corresponding PWM experiments.

In the static experiment, the same duty factor is generated, with a much higher frequency, resulting in a much smaller ripple of the analog signal.

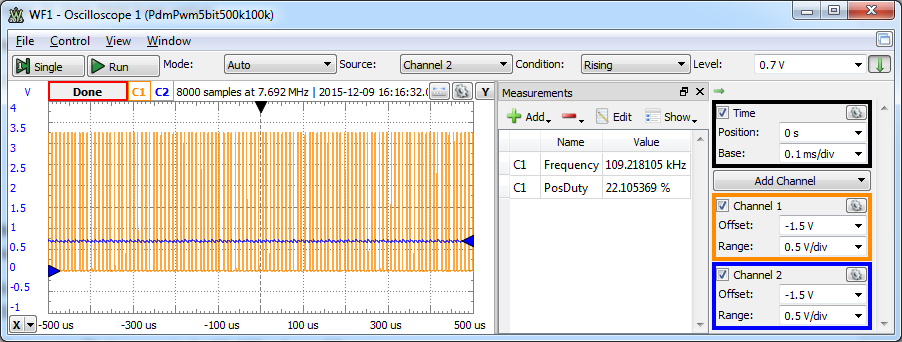


Figure 21. Static PDM scope capture

In the dynamic experiment, the same sinus custom Bus1 was used, as for the PWM. The average Duty Factor is again around 50%, the frequency is variable and the average frequency is much higher compared to PWM, for the same clock frequency. The ripple of the analog signal is also much smaller.

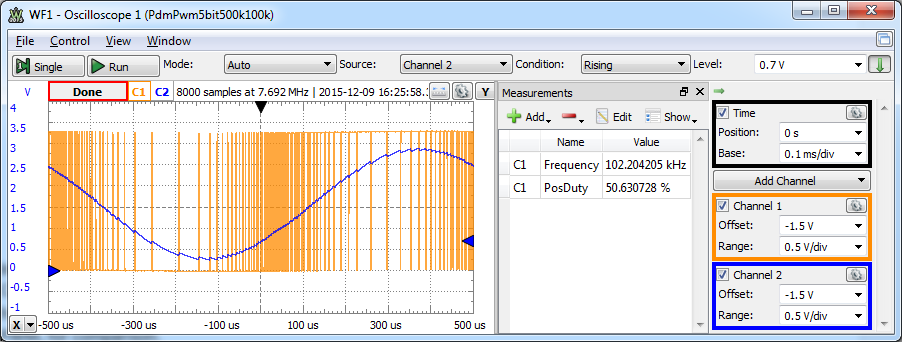


Figure 22. Sinus PDM and analog signal

The lowest instantaneous frequency of the PDM signal happens when the sample value is either 1LSB or 1-1LSB, and is the same as the constant frequency of the PWM signal. For all other values of the samples, the resulting instantaneous frequency is higher, up to half of the clock frequency, when the sample value is at half of the range.

Having both instances of the Pattern generator open, is very easy to run them alternatively for easy comparison on the scope instrument. Hitting Run on one instance automatically turns the other instance in Busy state.

For dynamic measurements ad comparison, the modulator clock frequency is increased to 4 MHz in the next experiment, both for PWM and PDM. The PWM carrier frequency, as well as lowest frequency component in the spread PDM spectrum is:

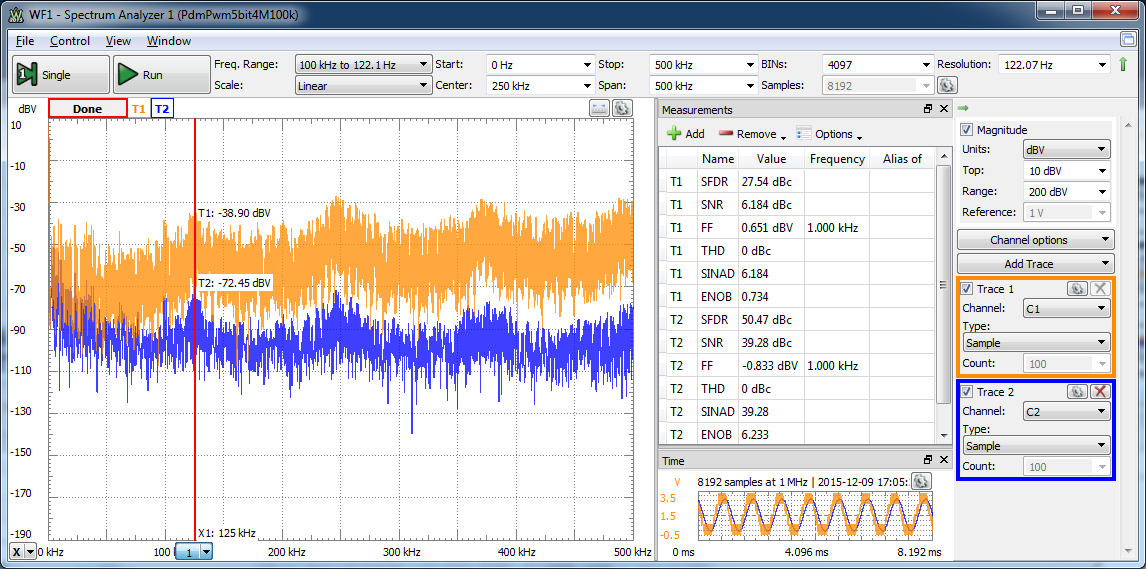
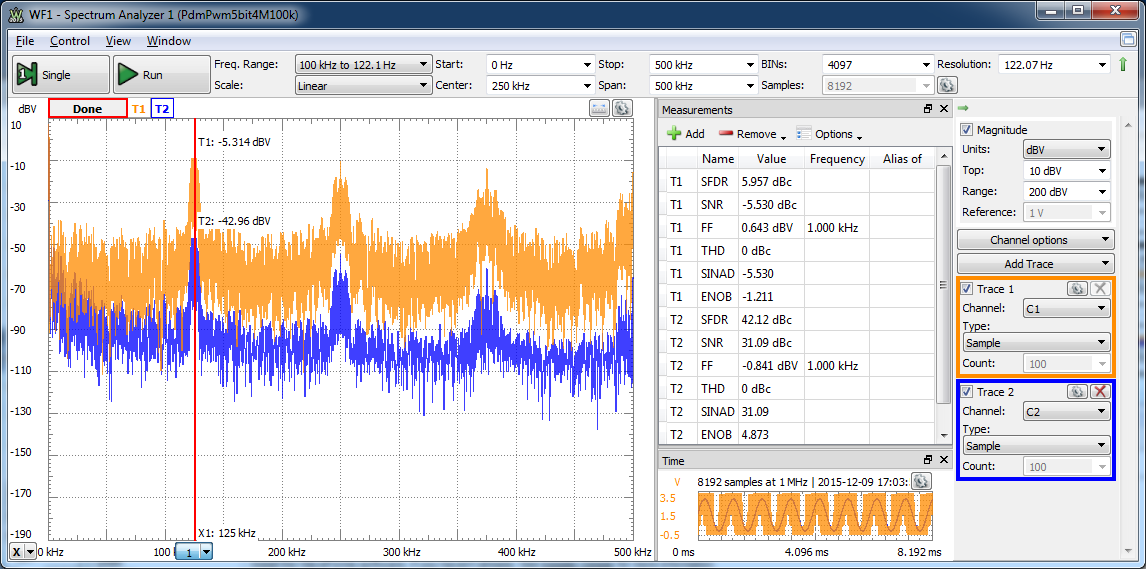


Figure 23. Spectral comparison: PWM (up), vs. PDM (down)

Same sinus signal is generated as before, with the frequency of:

The spectrum analyzer instrument is used to analyze data. The two traces correspond to the two scope channels in the experiments above. In the spectral view, a cursor is set at *fcarrier*. Some parameters of the modulated and analog signals are shown in the measurements window.

The carrier fundamental and harmonics are significant for PWM, while PDM spread those components more uniformly in the spectrum.

The measurements have much better values for SFDR, SNR, SINAD and ENOB in the case of PDM.