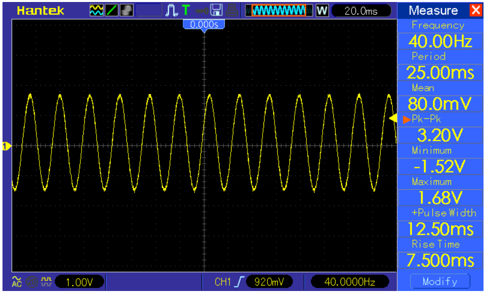
Waveform Generator implemented in FPGA using PWM DAC

Stefano Paracchino

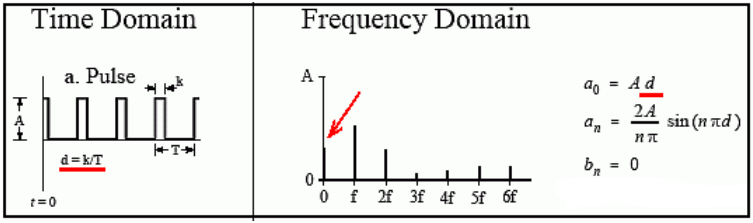
Follow-up of **Digital Electronics** course



Bill of materials:

- Artix-7 FPGA on Digilent Arty Evaluation Board
- IDE Xilinx Vivado (HDL, simulation, Syn & PnR, bitstream & download)
- Rs and Cs
- Oscilloscope (if you need debugging!)

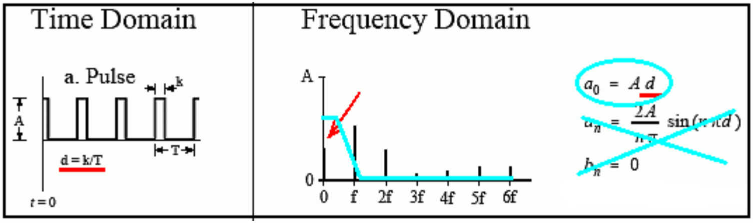
Pulse Width Modulation is a simple Digital to Analog Converter based on a square wave of some **duty cycle** (d)



$$a_0 = A\,d$$
$$a_n = \frac{2A}{n\pi}\sin(n\pi d)$$
$$b_n = 0$$

The Fourier Series (Transform) has a frequency zero term directly proportional to d

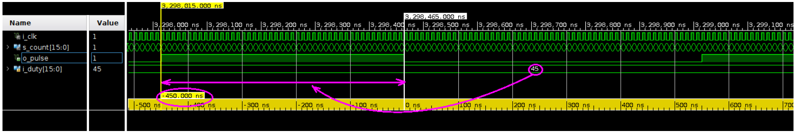
Therefore it is straightforward to get the only direct voltage value using a RC analog filter which cuts off the high frequency components



In this project there is a passive RC with $C = 1nF$ and $R = 45k\Omega \rightarrow f_c \approx 35kHz$

It is easy to generate a square wave of a defined duty cycle with Behavioral VHDL :

- (g_max)-module counter (s_count)
- i_duty comparator



In the present example firmware (g_max) is set to 100
$\rightarrow T_{pwm} = 100 \cdot T_{clk} = 1000ns \rightarrow 1MHz$

Once you get a single analog value (between 0 and 3.3 V), the next step is varying this value (so d) in order to obtain a generic waveform. A crucial question arises:

- Q1) How many different values can you get?
- A1) Just $g\_max + 1$ because we (d) are in a discrete world

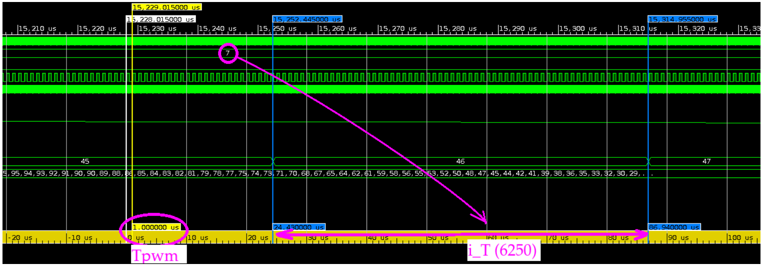
Now, let's suppose you want to build a triangle waveform. You are then supposed to vary the voltage value from minimum to maximum and then again to minimum. It is a **staircase** with i_T time step. The question now is :

- Q2) How much is i_T supposed to be to obtain a period T_w of the wave?
- A2) $T_w = 2 \cdot i\_T \cdot g\_max \cdot T_{clk} [ns]$

Conclusion: tradeoff between Q1 and Q2, namely between the **resolution** and the **maximum frequency** reachable

Example

$$T_w = 2 \cdot i\_T \cdot g\_max \cdot T_{clk} = 2 \cdot i\_T \cdot T_{pwm}\ [ns]$$

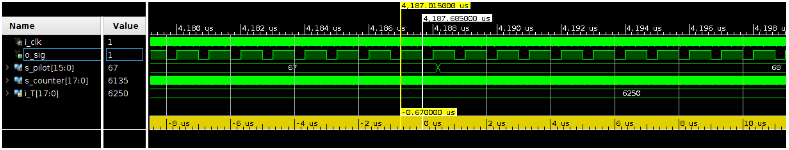


$$T_w = 2 \cdot 6250 \cdot 1000 = 12500000[ns] \ \rightarrow 80Hz$$

note1: 7 is the integer value of the code selected by the user through the switches to set a frequency
note2: the frequencies available in this example firmware are low...

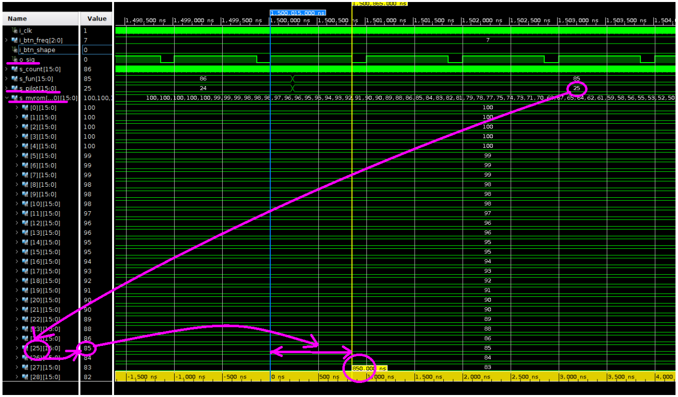
VHDL details:

s_pilot is the binary value of *d*, it is varied using the counter *s_counter* which is compared to i_T.
Finally a flag tells to *s_pilot* if it must increase or decrease.



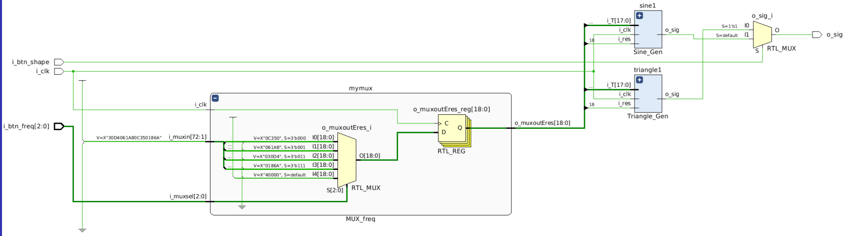
Name	Value
i_clk	1
o_sig	1
s_pilot[15:0]	67
s_counter[17:0]	6135
i_T[17:0]	6250

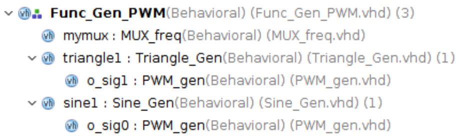
In order to obtain a Sine wave it is necessary to "modify" the Triangle wave.
s_pilot is not anymore the duty value but it is the the address of a ROM containing the duty values (*s_fun*) of a Sine.



Name	Value
i_clk	1
i_btn_freq[2:0]	7
i_btn_shape	0
o_sig	0
s_count[15:0]	86
s_fun[15:0]	85
s_pilot[15:0]	25
s_myrom[...0][15:0]	100,100,1
[0][15:0]	100
[1][15:0]	100
[2][15:0]	100
[3][15:0]	100
[4][15:0]	100
[5][15:0]	99
[6][15:0]	99
[7][15:0]	99
[8][15:0]	98
[9][15:0]	98
[10][15:0]	98
[11][15:0]	97
[12][15:0]	96
[13][15:0]	96
[14][15:0]	95
[15][15:0]	95
[16][15:0]	94
[17][15:0]	93
[18][15:0]	92
[19][15:0]	91
[20][15:0]	90
[21][15:0]	90
[22][15:0]	89
[23][15:0]	88
[24][15:0]	86
[25][15:0]	85
[26][15:0]	84
[27][15:0]	83
[28][15:0]	82

VHDL project has a typical *top-to-down* organization. Elementary blocks with simple tasks are instantiated and conveniently "wired" together in an a hierarchical structure to obtain a




Func_Gen_PWM(Behavioral) (Func_Gen_PWM.vhd) (3)
- mymux : MUX_freq(Behavioral) (MUX_freq.vhd)
- triangle1 : Triangle_Gen(Behavioral) (Triangle_Gen.vhd) (1)
  - o_sig1 : PWM_gen(Behavioral) (PWM_gen.vhd)
- sine1 : Sine_Gen(Behavioral) (Sine_Gen.vhd) (1)
  - o_sig0 : PWM_gen(Behavioral) (PWM_gen.vhd)