Application Note: Multiple Pulse Width Modulation Signals

By: Group 5 W2016

Introduction

Pulse width modulation signals are commonly used to control servo motors. These signals vary in both period and duration, corresponding to different angles in the motor itself. By controlling the signal, you are able to move the motor to a desired location. This is done in this application note for 12 servo motors composing a single robot. A single VHDL custom component is used to control all of them, and it is interfaced with a C header file to allow for easy control. Inspiration for this was taken from [1].

Design

This application note assumes that you have a fully compiled DE0 Nano board with the stock components. If not, please do so before continuing with this tutorial.

We will first look at the VHDL, and indicate areas for modification should you choose to use this. Keep in mind that we used 12 servo motors, but the adjustments are quite minor if you require a different amount.

```
library altera;
use altera.altera_europa_support_lib.all;
-- Repurposed for use by Group 5, uAlberta ECE 2016
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity multi_pwm is
generic( W:integer :=15);
  port (
            -- Avalon MM-----------
    clk : in std_logic;
            reset_n : in std_logic;
            readas : in std_logic;
            writas : in std_logic;
            chipselect : in std_logic;
            address : in std_logic_vector(5 downto 0);
            readdata : out std_logic_vector(31 downto 0);
            writedata : in std_logic_vector(31 downto 0);
            PWM1, PWM2, PWM3, PWM4, PWM5, PWM6, PWM7, PWM8, PWM9, PWM10, PWM11, PWM12: out
std_logic;
            );

end multi_pwm;

architecture PWM of multi_pwm is
 signal  pwm_counter, pwm_value1, pwm_value2, pwm_value3,
 pwm_value4, pwm_value5,pwm_value6, pwm_value7, pwm_value8, pwm_value9, pwm_value10,
pwm_value11, pwm_value12 : std_logic_vector(W downto 0);
 signal control_reg:  std_logic_vector(7 downto 0);
begin
process (clk, reset_n, chipselect)
begin
if reset_n='0' then
pwm_counter<=(others=>'0');
pwm_value1<=(others=>'0');
pwm_value2<=(others=>'0');
pwm_value3<=(others=>'0');
pwm_value4<=(others=>'0');
pwm_value5<=(others=>'0');
pwm_value6<=(others=>'0');
pwm_value7<=(others=>'0');
pwm_value8<=(others=>'0');
pwm_value9<=(others=>'0');
pwm_value10<=(others=>'0');
pwm_value11<=(others=>'0');
pwm_value12<=(others=>'0');
elsif clk'event and clk='1' then
            ----------------------------------- PWM set -------------------------
            if address = "000000" and writas = '0' then -- PWM UPDATE COUNTER

                         pwm_value1<=writedata(W downto 0);
                        end if;

                        if  address = "000001" and writas = '0' then -- PWM UPDATE COUNTER
                         pwm_value2<=writedata(W downto 0);
                        end if;

                        if address = "000010" and writas = '0' then -- PWM UPDATE COUNTER
                         pwm_value3<=writedata(W downto 0);
                        end if;

                        if address = "000011" and writas = '0' then -- PWM UPDATE COUNTER
                         pwm_value4<=writedata(W downto 0);
                        end if;

                        if address = "000100" and writas = '0' then -- PWM UPDATE COUNTER
                         pwm_value5<=writedata(W downto 0);
                        end if;

                        if address = "000101" and writas = '0' then -- PWM UPDATE COUNTER
                         pwm_value6<=writedata(W downto 0);
                        end if;
                        if address = "000110" and writas = '0'  then -- PWM UPDATE COUNTER
                                            pwm_value7<=writedata(W downto 0);
                                           end if;
                        if address = "000111" and writas = '0' then -- PWM UPDATE COUNTER
                                            pwm_value8<=writedata(W downto 0);
                                           end if;
                        if address = "001000" and writas = '0'  then -- PWM UPDATE COUNTER
                                            pwm_value9<=writedata(W downto 0);
                                           end if;
                        if address = "001001" and writas = '0'  then -- PWM UPDATE COUNTER
                                            pwm_value10<=writedata(W downto 0);
                                           end if;
                        if address = "001010" and writas = '0'  then -- PWM UPDATE COUNTER
                                            pwm_value11<=writedata(W downto 0);
                                           end if;
                        if address = "001011" and writas = '0'  then -- PWM UPDATE COUNTER
                                            pwm_value12<=writedata(W downto 0);
                                           end if;

                       if address = "001100" and writas = '0'  then -- PWM UPDATE COUNTER
                        control_reg(7 downto 0)<=writedata(7 downto 0);
                       end if;

--and (control_reg(0)='1')
                       ----------------- PWM signal formation ------------

              pwm_counter<=pwm_counter+1;

              if (pwm_counter = "11110100001001000000") then
                                  pwm_counter <= "00000000000000000000";
                        end if;

                         if ((pwm_value1<pwm_counter)and (pwm_value1>0)) then
                                PWM1<='1';


                         else PWM1<='0'; end if;

                        if ((pwm_value2<pwm_counter)and (pwm_value2>0)) then
                                PWM2<='1';
                         else PWM2<='0'; end if;

                        if ((pwm_value3<pwm_counter)and (pwm_value3>0)) then
                                PWM3<='1';
                         else PWM3<='0'; end if;

                        if ((pwm_value4<pwm_counter)and (pwm_value4>0)) then
                                PWM4<='1';
                         else PWM4<='0'; end if;

                                if ((pwm_value5<pwm_counter)and (pwm_value5>0)) then
                                PWM5<='1';
                         else PWM5<='0'; end if;
```

```
                              if ((pwm_value6<pwm_counter)and (pwm_value6>0)) then
                           PWM6<='1';
               else PWM6<='0'; end if;

                           if ((pwm_value7<pwm_counter)and (pwm_value7>0)) then
                                                      PWM7<='1';
                                            else PWM7<='0'; end if;

                           if ((pwm_value8<pwm_counter)and (pwm_value8>0)) then
                                                      PWM8<='1';
                                            else PWM8<='0'; end if;

                           if ((pwm_value9<pwm_counter)and (pwm_value9>0)) then
                                                      PWM9<='1';
                                            else PWM9<='0'; end if;

                           if ((pwm_value10<pwm_counter)and (pwm_value10>0)) then
                                                      PWM10<='1';
                                            else PWM10<='0'; end if;

                           if ((pwm_value11<pwm_counter)and (pwm_value11>0)) then
                                                      PWM11<='1';
                                            else PWM11<='0'; end if;

                           if ((pwm_value12<pwm_counter)and (pwm_value12>0)) then
                                                      PWM12<='1';
                                            else PWM12<='0'; end if;

end if;

end process;
end PWM;
```

Note the PWM1-12 outputs declared from the component. These are direct bit outputs, firing quick pulses out to 12 different named areas. Define however many signals you wish to generate (equal to your number of motors or other devices controlled in this method) as output bits, and use the method shown to create additional signals if needed. Note the line:

```
if (pwm_counter = "11110100001001000000") then
                    pwm_counter <= "00000000000000000000";
end if;
```

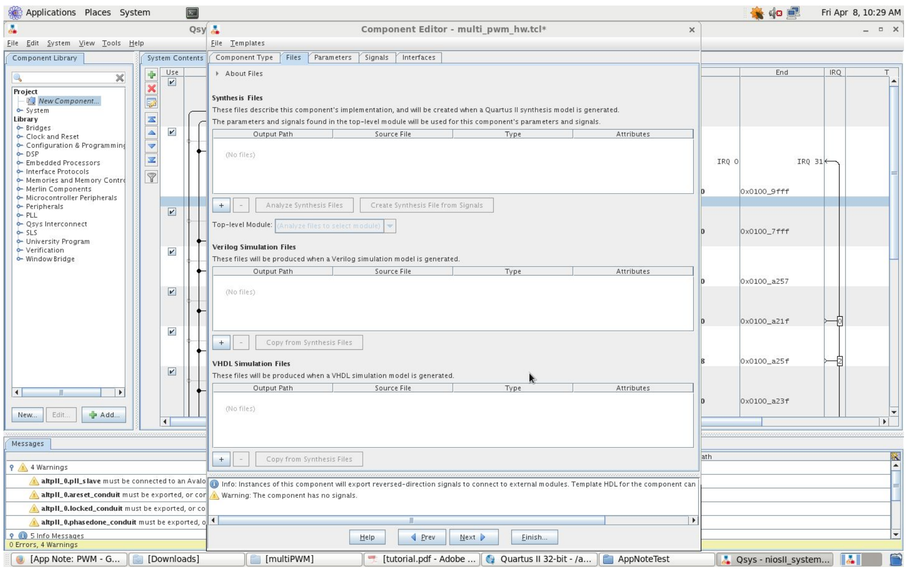
This number will change based on the frequency required for your devices. We used servo motors with a 20 Hz refresh rate, which corresponded to 1 million in binary. Take note that this value will likely change based on your use case. One other line that is important to note is this:

```
if address = "001100" and writas = '0'  then -- PWM UPDATE COUNTER  PWM12
```

Each PWM signal contains this line in its definition, which allows us to map accordingly to it. The address is of particular importance, as it gives us easy access to the specific signal in the pre-defined header file. For additional PWM signals, simply add to each address in sequential order (for example, if you added a PWM13 signal the address for the check would be 001101). Pretty straightforward.

Now, we will use this file to create a custom component in our system and implement it in C code. As stated before, you must have a basic, fully compiled DE0 project before continuing.

First up, open up Qsys and create a new component on the left hand side.



Component Editor - multi_pwm_hw.tcl*

Synthesis Files
These files describe this component's implementation, and will be created when a Quartus II synthesis model is generated.
The parameters and signals found in the top-level module will be used for this component's parameters and signals.

Output Path	Source File	Type	Attributes
(No files)			

Top-level Module: (Analyze files to select module)

Verilog Simulation Files
These files will be produced when a Verilog simulation model is generated.

Output Path	Source File	Type	Attributes
(No files)			

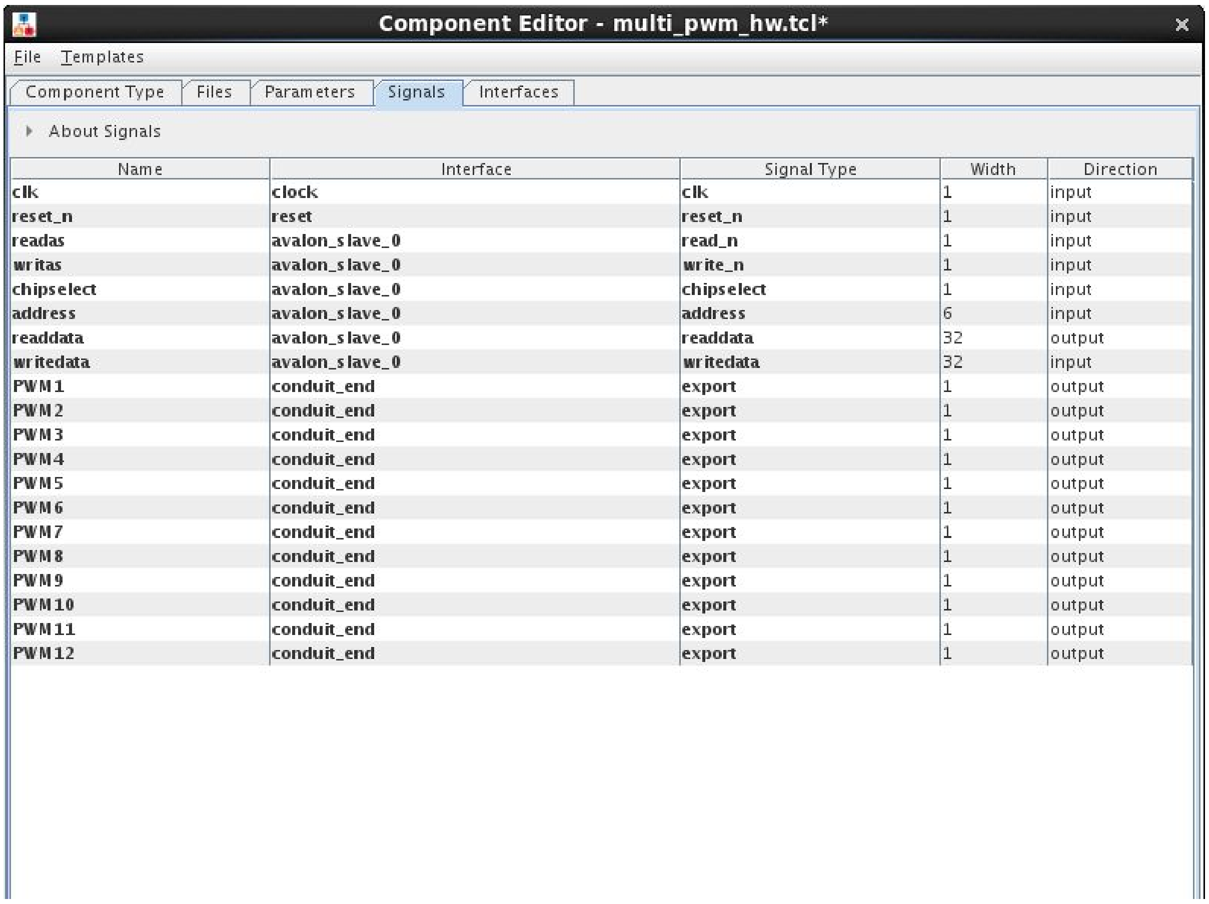
VHDL Simulation Files
These files will be produced when a VHDL simulation model is generated.

Output Path	Source File	Type	Attributes
(No files)			

Info: Instances of this component will export reversed-direction signals to connect to external modules. Template HDL for the component can
Warning: The component has no signals.

```
4 Warnings
  altpll_0.pll_slave must be connected to an Avalo
  altpll_0.areset_conduit must be exported, or cor
  altpll_0.locked_conduit must be exported, or co
  altpll_0.phasedone_conduit must be exported, o
5 Info Messages
0 Errors, 4 Warnings
```

Add the VHDL file we defined earlier as the synthesis file, and click Analyze Synthesis Files to check and make sure that no errors exist. This should give us our signal definitions for later.


Component Editor - multi_pwm_hw.tcl*

Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
reset_n	reset	reset_n	1	input
readas	avalon_slave_0	read_n	1	input
writas	avalon_slave_0	write_n	1	input
chipselect	avalon_slave_0	chipselect	1	input
address	avalon_slave_0	address	6	input
readdata	avalon_slave_0	readdata	32	output
writedata	avalon_slave_0	writedata	32	input
PWM1	conduit_end	export	1	output
PWM2	conduit_end	export	1	output
PWM3	conduit_end	export	1	output
PWM4	conduit_end	export	1	output
PWM5	conduit_end	export	1	output
PWM6	conduit_end	export	1	output
PWM7	conduit_end	export	1	output
PWM8	conduit_end	export	1	output
PWM9	conduit_end	export	1	output
PWM10	conduit_end	export	1	output
PWM11	conduit_end	export	1	output
PWM12	conduit_end	export	1	output

Match your signals to the above screenshot. You may need to declare a conduit_end for your exports, as well as set the reset of the Avalon slave in the Interfaces tab. Following that, ensure that you have correctly mapped the interfaces for proper Avalon MM useage. Basically change things until it doesn't complain anymore.

Now, add the component to your system and hook up the clock and reset sources. Export the conduit as well, assign base addresses and generate the system.

After this, add the top level VHDL file shown below, as well as the Qsys file to your project. Compile your design, and pray that everything works. You should have some warnings. This is normal. Program your device as normal, then open Eclipse.

Create a new NiosII project with associated BSP, and point it to your SOPC file. We will use this file to generate PWM signals. Import the associated multi_pwm.h file that we have provided. It provides useful C wrappers to access the outputs. See the attached C files for sample code.

Getting the Base Project

Download the qar file and open it in Quartus, compile and program the board. Download and the C and header files included, create an Eclipse project and open the files from there. Running this on a DE0 Nano should result in different PWM signals on GPIO 0 through 11. Check pin alignment files to check pin mappings if you are unsure which GPIO pins align with physical pins.

References

[1]: http://www.grigaitis.eu/?p=566