ECE 492: Group 3 2018w
Created By: Kelly Chin & Jessica Huynh

Application Notes: PWM L298N Driver

Objectives

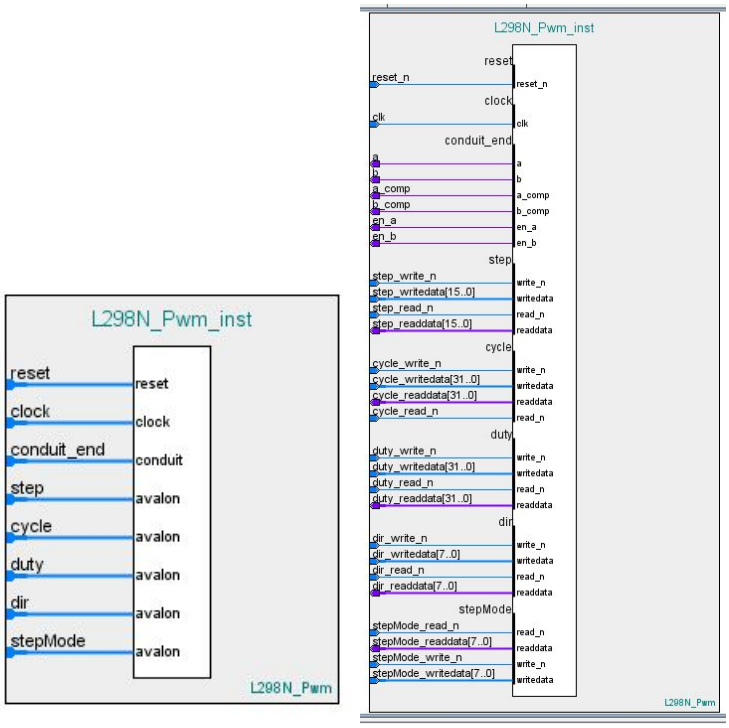
- To configure an L298N custom component
- To generate a PWM signals and step sequences
- To drive stepper motors using the L298N component and C software

Introduction

In this application note, creation of an L298N Qsys component and the code to drive simple stepper motors are explained. The platform used for development and testing is Altera's De1-SoC, Cyclone V FPGA board [1] with Quartus Prime 17.0. The stepper motors used for this application are from Adafruit Industries LLC [2] but any other simple stepper motor may be substituted with minimal modifications from the original notes presented.

Design

Hardware Design



There are five avalon memory mapped slave interfaces (step, dir, cycle, duty and stepMode). They each have an address so that their values can be changed from software.

From this equation for a frequency divider: [3]

$$f_{out} = \frac{f_{in}}{n}$$

Solving for n gives the number of clock cycles needed such that f_{out} can be generated from f_{in}

$$n = \frac{f_{in}}{f_{out}}$$

f_{in} is assumed to always be higher than f_{out} and is defaulted to 50MHz (as set by the DE1-SoC board.) This way, n is an integer always greater than 1. This value (n) is called cycle.

Aside: Cycle refers to the wavelength and the corresponding period (in seconds) can be calculated by :

$$period\ of\ f_{out} = period\ of\ f_{in} \times n = \frac{1}{f_{in}} \times n$$

Duty cycle is the amount of time the wave spends pulled high. Duty is calculated from a hard coded constant DUTY_CYCLE found in stepper.h which defines the percentage of high time. Cycle was previously calculated, so Duty is derived from that by multiplying it by the constant.

$$Duty = Cycle \times DUTYCYCLE$$

Dir indicates which way the motor will spin, clockwise/counter-clockwise. For our system, 0 means CW motor spin and 1 is CCW direction. Counter-clockwise traverses the step sequence backwards. [4]

Full-Step Sequence												
Steps	Winding Current A	Winding Current B	L 298 Outputs				L 298 Inputs					
			Out 1	Out 2	Out 3	Out 4	En A	En B	IN 1	In 2	In 3	In 4
			A	#A	B	#B			A	#A	B	#B
#1	Forward	Forward	Src	Sink	Src	Sink	1	1	1	0	1	0
#2	Forward	Reverse	Src	Sink	Sink	Src	1	1	1	0	0	1
#3	Reverse	Reverse	Sink	Src	Sink	Src	1	1	0	1	0	1
#4	Reverse	Forward	Sink	Src	Src	Sink	1	1	0	1	1	0
#1	Forward	Forward	Src	Sink	Src	Sink	1	1	1	0	1	0
#2	Forward	Reverse	Src	Sink	Sink	Src	1	1	1	0	0	1

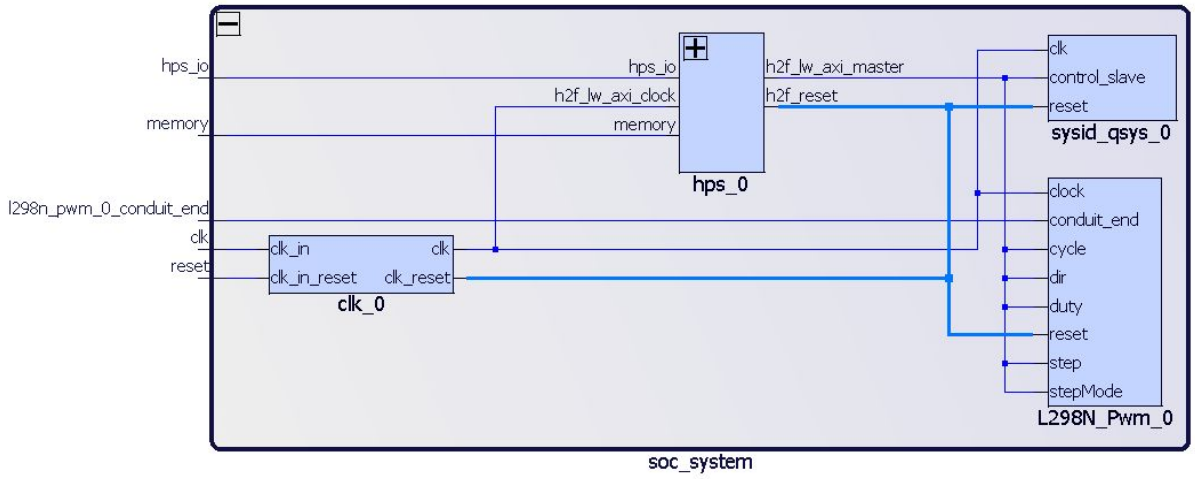
Table 1: Full-Step Sequence for a Bipolar Stepper Motor

Half-Step Sequence												
Steps	Winding Current A	Winding Current B	L 298 Outputs				L 298 Inputs					
			Out 1	Out 2	Out 3	Out 4	En	En	In 1	In 2	In 3	In 4
			A	#A	B	#B	A	B	A	#A	B	#B
#1	Forward	Forward	Src	Sink	Src	Sink	1	1	1	0	1	0
#2	Forward	Off	Src	Sink	Off	Off	1	0	1	0	DC	DC
#3	Forward	Reverse	Src	Sink	Sink	Src	1	1	1	0	0	1
#4	Off	Reverse	Off	Off	Sink	Src	0	1	DC	DC	0	1
#5	Reverse	Reverse	Sink	Src	Sink	Src	1	1	0	1	0	1
#6	Reverse	Off	Sink	Src	Off	Off	1	0	0	1	DC	DC
#7	Reverse	Forward	Sink	Src	Src	Sink	1	1	0	1	1	0
#8	Off	Forward	Off	Off	Src	Sink	0	1	DC	DC	1	0
#1	Forward	Forward	Src	Sink	Src	Sink	1	1	1	0	1	0
#2	Forward	Off	Src	Sink	Off	Off	1	0	1	0	DC	DC

Table 2: Half-Step Sequence for a Bipolar Stepper Motor

StepMode allows for the motor to be operated in full step or half step. Two functions are included inside the L298N_pwm.vhd that hard codes the sequence in the above table. Step mode decides which of these functions to use during operation.

Step refers to the steps of the stepper motor. The component will count down from this step value until it hits zero, at this point the motor will no longer rotate (but will stay stalled.) Our motor does 200 steps in full step mode which corresponds to 4 cm per revolution with our timing belt.



In the L298N_pwm.vhd file the step sequence is referenced and their values are always pushed to the GPIO pins via the conduit. Notice the enable pins are always pulled high (assuming during full step) regardless of duty cycle, this is because the generated frequency is built into the step sequence itself and does not need to be explicitly handled by the enable signals.

In half step, the two enable signals are not always high and actually change states, according to the above table. Their values are referenced in the hard coded function as previously mentioned and each of these six signals (en_a, en_b, a, a', b, b') is calculated and pushed to the GPIO pins on every pulse.

Software Design

These two files should be included in the 'HWLIBS' folder within eclipse: stepper.c, stepper.h,

Stepper.c
This file contains functions that perform the calculations described above for the five memory mapped variables that the vhdl requires to do the pwm signal.

Stepper.h
This file contains the addresses of the memory mapped variables which you will have to change. Other constants for the stepper motor itself are included here and may have to be changed according to your stepper motor.

Procedure

Hardware Procedure

1. Open Quartus (Quartus Prime 17.0) and create a new project.
2. Open Qys by clicking on Tools > Qsys.
3. Add the following components and set them up as you would from the lab tutorials:
    a. Arria V/ Cyclone C Hard Processor System
    b. System ID Peripheral
4. Create a new custom component, "L298N_Pwm" by clicking on New in the IP Catalog panel.
5. If using the provided Easy_Driver.vhd, please do the following:
    a. Files > Add file > .. > Analyze Synthesis Files.
    b. Ensure the correct signals/ interfaces were added (i.e. often the conduit loads itself as an avalon slave and so needs to be changed, associate resets/clocks)
6. If creating a new component from scratch, add the following interfaces and signals:

Interface: Clock Input		
Signal	**Width**	**Direction**
clk	1	input

Interface: Reset Input		
Signal	**Width**	**Direction**

reset_n	1	input

The conduit will allow us to write to GPIO pins that will be fed to the pins of the L298N H Bridge. These pins will be written to with VHDL to complete a proper step sequence.

Interface: Conduit		
Signal	**Width**	**Direction**
a	1	output
a_comp	1	output
b	1	output
b_comp	1	output
en_a	1	output
en_b	1	output

The following Avalon MM Slave will allow users to write a value for the number of steps the motor should do. The width of the data can be changed depending on what range of steps you wish to step your motors to.

Interface: Avalon MM Slave (Step)		
Signal	**Width**	**Direction**
step_read_n	1	input
step_readdata	16	output
step_write_n	1	input
step_writedata	16	input

The following Avalon MM Slave will allow users to switch between full step or half step mode. Although, only one bit is required (i.e. 0 = full step and 1 = half step), this cannot be done as the minimum width of data for any Avalon MM slave has to be a multiple of 8.

Interface: Avalon MM Slave (StepMode)		
Signal	**Width**	**Direction**
stepmode_read_n	1	input
stepmode_readdata	8	output

stepmode_write_n	1	input
stepmode_writedata	8	input

The following Avalon MM Slave will allow users to write a value for the direction the motor should spin.Similarly to the above slave, only one bit is required (i.e. 0 = CW and 1 = CCW) even though 8 bits are used.

Interface: Avalon MM Slave (Direction)		
Signal	**Width**	**Direction**
dir_read_n	1	input
dir_readdata	8	output
dir_write_n	1	input
dir_writedata	8	input

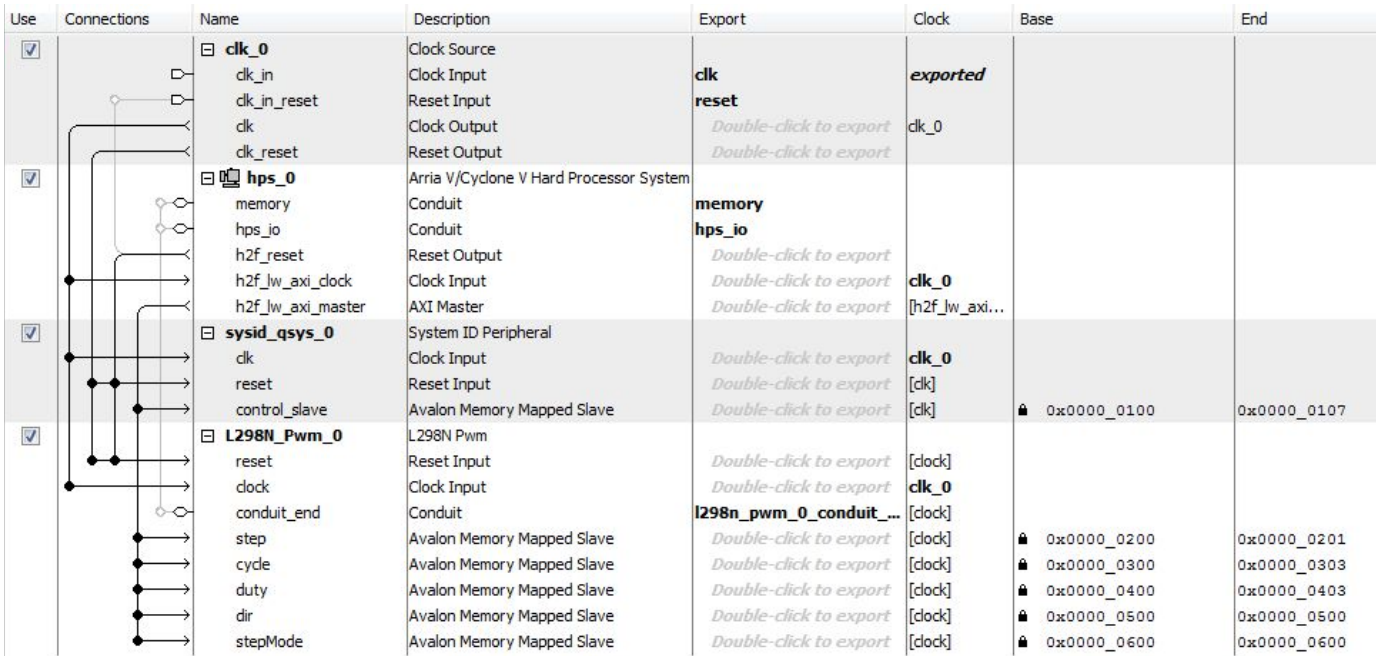
The following Avalon MM Slave will allow users to write a value for the cycle of the signal. Through software, this is calculated based on the frequency desired in PPS.

Interface: Avalon MM Slave (Cycle)		
Signal	**Width**	**Direction**
cycle_read_n	1	input
cycle_readdata	32	output
cycle_write_n	1	input
cycle_writedata	32	input

The following Avalon MM Slave will allow users to write a value for the duty cycle of the signal. Through software, this is calculated as a fractional value of the period.

Interface: Avalon MM Slave (Duty)		
Signal	**Width**	**Direction**
duty_read_n	1	input
duty_readdata	32	output
duty_write_n	1	input
duty_writedata	32	input

7. Save this component and create a synthesis file: Files > Create Synthesis File From Signals > VHDL
8. Add this new component to the system.
9. Connect all components as shown below:



Use	Connections	Name	Description	Export	Clock	Base	End
☑		**clk_0**	Clock Source				
		clk_in	Clock Input	**clk**	***exported***		
		clk_in_reset	Reset Input	**reset**			
		clk	Clock Output	*Double-click to export*	clk_0		
		clk_reset	Reset Output	*Double-click to export*			
☑		**hps_0**	Arria V/Cyclone V Hard Processor System				
		memory	Conduit	**memory**			
		hps_io	Conduit	**hps_io**			
		h2f_reset	Reset Output	*Double-click to export*			
		h2f_lw_axi_clock	Clock Input	*Double-click to export*	**clk_0**		
		h2f_lw_axi_master	AXI Master	*Double-click to export*	[h2f_lw_axi...		
☑		**sysid_qsys_0**	System ID Peripheral				
		clk	Clock Input	*Double-click to export*	**clk_0**		
		reset	Reset Input	*Double-click to export*	[clk]		
		control_slave	Avalon Memory Mapped Slave	*Double-click to export*	[clk]	0x0000_0100	0x0000_0107
☑		**L298N_Pwm_0**	L298N Pwm				
		reset	Reset Input	*Double-click to export*	[clock]		
		clock	Clock Input	*Double-click to export*	**clk_0**		
		conduit_end	Conduit	**l298n_pwm_0_conduit_...**	[clock]		
		step	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0000_0200	0x0000_0201
		cycle	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0000_0300	0x0000_0303
		duty	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0000_0400	0x0000_0403
		dir	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0000_0500	0x0000_0500
		stepMode	Avalon Memory Mapped Slave	*Double-click to export*	[clock]	0x0000_0600	0x0000_0600

10. Ensure that all conduit signals have a unique locked memory address.
11. Connect all reset signals: System > Create Global Reset System
12. Ensure there are no errors or warnings.
13. Generate HDL > VHDL
14. In Quartus add the .qip, Analyze & Synthesize, run TCL scripts, and compile.

If using the tutorial1.vhd file that was provided in the labs, you will need to modify the top level file to include the new components and pins. The following is an example of the pin assignments for the top level; this is available in the zip file as “motorPwm.vhd”. Please ensure the signal names match the ones created for your project (i.e. the conduit names).

Please ensure your top level entity and project name are the same.

Entity Port:

```
-- pins for hBridge
    GPIO_0_0              : out std_logic; -- A
    GPIO_0_2              : out std_logic; -- #A

    GPIO_0_1              : out std_logic; -- B
    GPIO_0_3              : out std_logic; -- #B

    GPIO_0_4              : out std_logic; -- en A
    GPIO_0_5              : out std_logic  -- en B
```

Architecture Component Port:

```
-- Motor L298N Conduit Signals
l298n_pwm_0_conduit_end_en_a              : out   std_logic;
l298n_pwm_0_conduit_end_en_b              : out   std_logic;
l298n_pwm_0_conduit_end_a                 : out   std_logic;
l298n_pwm_0_conduit_end_b                 : out   std_logic;
l298n_pwm_0_conduit_end_a_comp            : out   std_logic;
l298n_pwm_0_conduit_end_b_comp            : out   std_logic
```

Port Map:

```
-- Port Mapping L298N to GPIO Pins
      l298n_pwm_0_conduit_end_a               => GPIO_0_0,
      l298n_pwm_0_conduit_end_a_comp          => GPIO_0_2,

      l298n_pwm_0_conduit_end_b               => GPIO_0_1,
      l298n_pwm_0_conduit_end_b_comp          => GPIO_0_3,

      l298n_pwm_0_conduit_end_en_a            => GPIO_0_4,
      l298n_pwm_0_conduit_end_en_b            => GPIO_0_5
```

The VHDL code for the PWM signal can be found in full in the zip as “L298N_Pwm.vhd”. In summary, the code will obtain the values written to (all Avalon MM Slave signals) on a rising clock edge. Once there is a value in step_writedata that is greater than 0, the PWM signal will be generated based upon and period and duty cycle. The direction and step mode are all set in the software and effects the VHDL code (i.e. step sequence and/or traversal direction) Note, these values are all controlled in software and are not hardcoded in the VHDL code.

Software Procedure

```
194 static  void  RunStepperMotorTask(void *p_arg)
195 {
196     BSP_OS_TmrTickInit(OS_TICKS_PER_SEC);              /* Configure and enable OS tick interrupt.             */
197     BSP_WatchDog_Reset();                              /* Reset the watchdog.                                 */
198
199     InitMotor(200);
200     StepMotor(0); // no steps
201     OSTimeDlyHMSM(0, 0, 0, 500);
202     SetDirection(cw);
203
204     for(;;)
205     {
206         StepMotor(200);
207         OSTimeDlyHMSM(0, 0, 2, 0);
208     }
209 }
210
```

To operate the motors, place the above code snippet within a task (inside app.c) to initialize the pwm component via the InitMotor(...) function. A direction must be specified before operation using ‘SetDirection’. StepMotor(...) can be given a hard coded amount of steps to spin the motor that many steps. Stepping 0 steps ensures that the pwm component sends no signals (as a smoke test). After setting direction and frequency, StepMotor(...) can be called as many times as desired to rotate the motor that many steps.

Source Documentation and References

[1] Terasic Technologies Inc., “DE1-SoC User Manual Rev F” [Online]. Available: https://eclass.srv.ualberta.ca/pluginfile.php/3468647/mod_resource/content/1/DE1-SoC_User_manual_REV_F.pdf [Accessed Mar. 15, 2018]

[2] Digi-Key Electronics., “324 Adafruit Industries LLC | Motors, Solenoids, Driver Boards/ Modules | DigiKey” [Online]. Available: https://www.digikey.ca/product-detail/en/adafruit-industries-llc/324/1528-1062-ND/5022791 [Accessed Mar. 15, 2018]

[3] Wikipedia, “Frequency Divider” [Online]. Available: https://en.wikipedia.org/wiki/Frequency_divider [Accessed Mar.15, 2018]

[4] N. Minderman, ”ECE315 – Lab #5 Stepper Motor Control Using the ColdFire Enhanced Time Processor Unit (eTPU)”, 2016.

Attached Files

Stepper.c: code to write and read the motors
Stepper.h: memory mapped addresses and motor constants

L298N_Pwm.vhd: step sequence functions in PWM
motorPwm.vhd: top level vhdl code for port mapping

Application Notes by: Kelly Chin & Jessica Huynh