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

# Application Notes: PWM EasyDriver

## Objectives

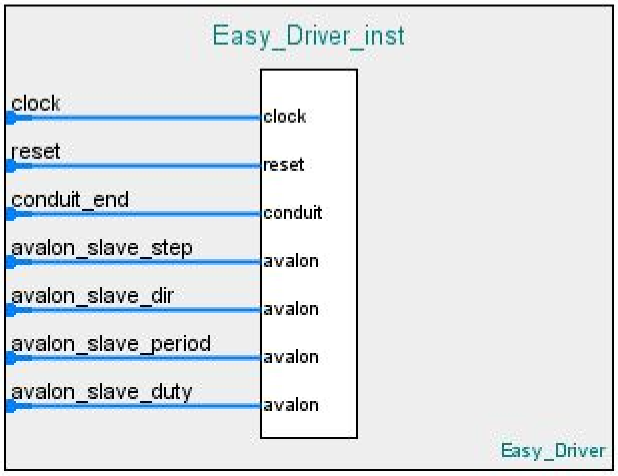
- To configure an EasyDriver custom component
- To generate a PWM signal to feed the driver
- To drive stepper motors using the EasyDriver component and C software

## Introduction

In this application note, creation of an EasyDriver 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 four avalon memory mapped slave interfaces (step, dir, period and duty). 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 'period'.

Aside: Period has the units of seconds, while here it is an integer. The reason we have called it period is because it refers to the wavelength and the corresponding measurement in seconds can be calculated by :

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

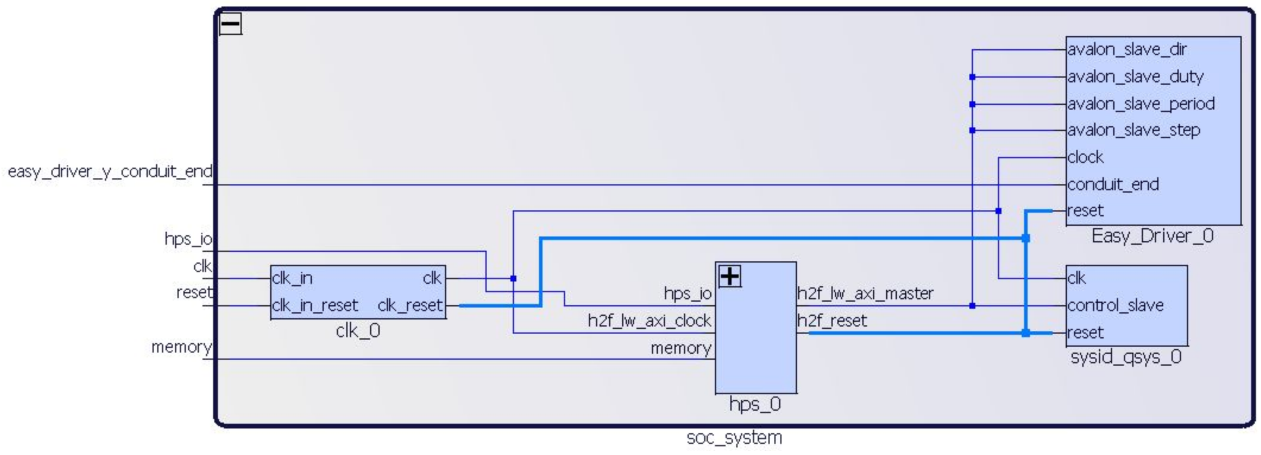
Duty cycle is the amount of time the wave spends pulled high. Duty is calculated from a hard coded constant DUTY_CYCLE found in motorConstants.h which defines the percentage of high time. Period was previously calculated, so Duty is derived from that by multiplying it by the constant.

$$Duty = Period \times DUTYCYCLE$$

Dir indicates which way the motor will spin, clockwise/counter-clockwise. For our system, 0 means CW motor spin and 1 is CCW direction.

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.

The conduit variables are used for connecting to the GPIO header. More information on how these variables are controlled in software is found below.



## Software Design

These four files should be included in the ‘HWLIBS’ folder within eclipse: stepper.c, stepper.h, motorConstants.c, motorConstants.h.

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

MotorConstants.c/MotorConstants.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.

Please Note that our project required two motors (x and y) called through enum Motor, you can add as many motors as you would like.

# 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, “Easy_Driver” by clicking on New in the IP Catalog panel.
5. 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 Easy Driver. More conduit pins can be set up (i.e. conduit signals to control the step mode, sleep, etc.).

**Important Note:** The enable conduit should be connected to the RST pin on the EasyDriver. This is active high and will allow current to be pulled. On low, no current flows through the chip. More can be explain in this guide [4]. The enable pin in the EasyDriver may be used (controls current to the motor), however this is active low and the code should be modified respectively.

| **Interface: Conduit** | | |
|---|---|---|
| **Signal** | **Width** | **Direction** |
| conduit_end_dir | 1 | output |
| conduit_end_step | 1 | output |
| conduit_end_enable | 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 write a value for the direction the motor should spin. Although, only one bit is required (i.e. 0 = CW and 1 = CCW), 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 (Direction)** | | |
|---|---|---|
| **Signal** | **Width** | **Direction** |
| dir_read_n | 1 | input |
| dir_readdata | 8 | output |
| dir_write_n | 1 | input |
| dir_writedata | 8 | input |

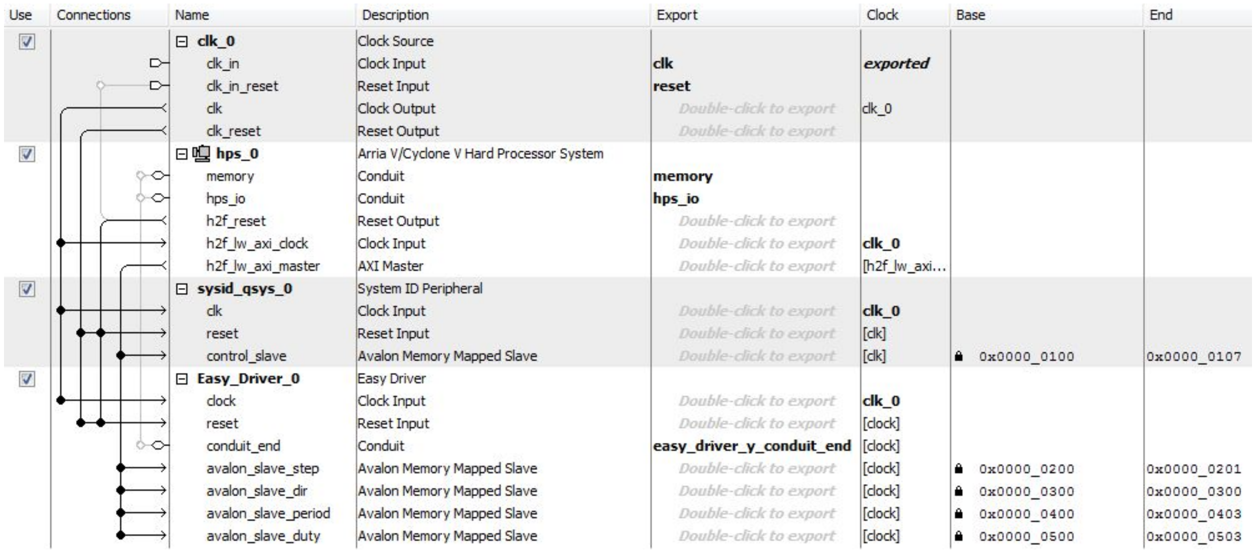
The following Avalon MM Slave will allow users to write a value for the period of the signal. Through software, this is calculated based on the frequency desired in PPS.

| Interface: Avalon MM Slave (Period) | | |
|---|---|---|
| **Signal** | **Width** | **Direction** |
| period_read_n | 1 | input |
| period_readdata | 32 | output |
| period_write_n | 1 | input |
| period_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 |

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



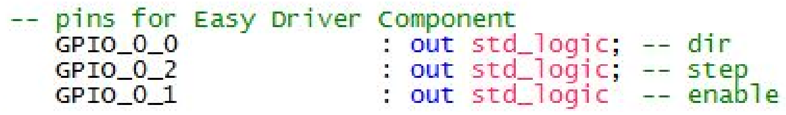
| Use | Connections | Name | Description | Export | Clock | Base | End |
|---|---|---|---|---|---|---|---|
| ✓ | | **clk_0** | Clock Source | | | | |
| | | clk_in | Clock Input | **clk** | ***exported*** | | |
| | | clk_in_reset | Reset Input | **reset** | | | |
| | | clk | Clock Output | Double-click 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 |
| ✓ | | **Easy_Driver_0** | Easy Driver | | | | |
| | | clock | Clock Input | Double-click to export | **clk_0** | | |
| | | reset | Reset Input | Double-click to export | [clock] | | |
| | | conduit_end | Conduit | **easy_driver_y_conduit_end** | [clock] | | |
| | | avalon_slave_step | Avalon Memory Mapped Slave | Double-click to export | [clock] | 0x0000_0200 | 0x0000_0201 |
| | | avalon_slave_dir | Avalon Memory Mapped Slave | Double-click to export | [clock] | 0x0000_0300 | 0x0000_0300 |
| | | avalon_slave_period | Avalon Memory Mapped Slave | Double-click to export | [clock] | 0x0000_0400 | 0x0000_0403 |
| | | avalon_slave_duty | Avalon Memory Mapped Slave | Double-click to export | [clock] | 0x0000_0500 | 0x0000_0503 |

9. Ensure that all conduit signals have a unique locked memory address.

10. Connect all reset signals: System > Create Global Reset System
11. Ensure there are no errors or warnings.
12. Generate HDL > VHDL

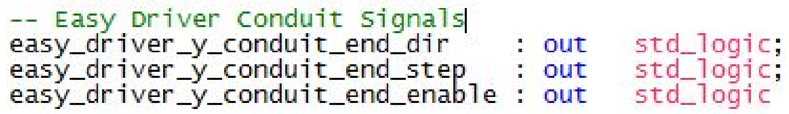
In Quartus, add the top level file, custom component file, and qip system to the project. If using the tutorial1.vhd file that was provided in the labs, you will need to modify the top level file to include the new components and pins. The following is an example of the pin assignments for the top level;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

Entity Port:



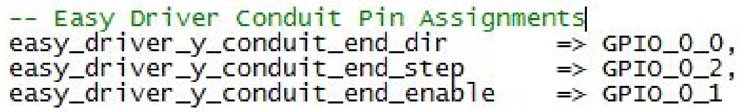
```
-- pins for Easy Driver Component
   GPIO_0_0              : out std_logic; -- dir
   GPIO_0_2              : out std_logic; -- step
   GPIO_0_1              : out std_logic  -- enable
```

Architecture Component Port:



```
-- Easy Driver Conduit Signals
easy_driver_y_conduit_end_dir    : out   std_logic;
easy_driver_y_conduit_end_step   : out   std_logic;
easy_driver_y_conduit_end_enable : out   std_logic
```

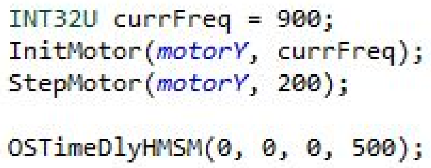
Port Map:



```
-- Easy Driver Conduit Pin Assignments
easy_driver_y_conduit_end_dir       => GPIO_0_0,
easy_driver_y_conduit_end_step      => GPIO_0_2,
easy_driver_y_conduit_end_enable    => GPIO_0_1
```

The VHDL code for the PWM signal can be found in full in the zip as “Easy_Driver.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. Note, these values are all controlled in software and are not hardcoded in the VHDL code.

## Software Procedure



```
INT32U currFreq = 900;
InitMotor(motorY, currFreq);
StepMotor(motorY, 200);

OSTimeDlyHMSM(0, 0, 0, 500);
```

To operate the motors, place the above code snippet within a task (inside app.c) to initialize the pwm component via the InitMotor(...) function. StepMotor(...) can be given a hard coded amount of steps to spin the motor that many steps. To convert from distance (cm) to steps, the functions found in Stepper.c can be used. XYCoor2Steps(int y) calculates the number of steps given a distance in cm (- is left/down and + is right/up). StepMotor(...) can now be called with this converted number of 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] Allegro MicroSystems, LLC, “A3967 Microstepping Driver with Translator” [Online] Available: https://cdn.sparkfun.com/datasheets/Robotics/A3967-Datasheet.pdf [Accessed Mar. 15, 2018]

[5] Toni_K. SparkFun Electronics ®, “Easy Driver Hook-up Guide” [Online] Available: https://learn.sparkfun.com/tutorials/easy-driver-hook-up-guide [Accessed Mar. 15, 2018]

## Attached Files

Stepper.c: code to write and read the motors
Stepper.h: header file for stepper code
motorConstants.c: code that returns the proper addresses based on motor enum
motorConstants.h: contains all motor addresses and constants

Easy_Driver.vhd: easy driver component, with PWM
motorPwm.vhd: top level vhdl code for port mapping

**Application Notes by: Kelly Chin & Jessica Huynh**