**Technische Universität Chemnitz**
**Fakultät für Elektrotechnik**
**und Informationstechnik**
Professur Schaltkreis- und Systementwurf
Prof. Dr.-Ing. Heinkel

# DESIGN AND SIMULATION OF A MOTOR CONTROLLER IN VHDL/VHDL-AMS

Manual

Practical course for the lecture "Design of Heterogeneous Systems (DHS)"
Date: April 7, 2021

# Contents

## Abbreviations

| | |
|---|---|
| **IC** | Integrated circuit |
| **PWM** | Pulse-width modulation |
| **SPI** | Serial Peripheral Interface |
| **MC** | Motor Controller |
| **MSB** | Most Significant Bit |
| **LSB** | Least Significant Bit |

## References

[1] Ashenden, Peter. The designer's guide to VHDL. Volume 3, Morgan Kaufmann. 2010.

[2] Bryan Mealy, Fabrizio Tappero. Free Range VHDL. 2013. `http://www.freerangefactory.org`

[3] The College of New Jersey. VHDL Quick Reference Card . 2016. `http://www.tcnj.edu/~hernande/r/VHDL_QRC__01.pdf`

[4] Bryan Mealy. VHDL Cheat-Sheet. 2005. `https://courseware.ee.calpoly.edu/cpe-169/Misc_stuff/cheat_sheet.pdf`

[5] TEXAS INSTRUMENTS. DRV883x Low-Voltage H-Bridge Driver Data sheet. 2019. `https://www.ti.com/lit/ds/symlink/drv8837.pdf`

[6] Ashenden, Peter and Peterson, Gregory and Teegarden, Darell. The system designer's guide to VHDL-AMS. Morgan Kaufmann. 2003.

# 1 Introduction

## 1.1 Objective of the Practical Course

Within the practical course the students should get familiar with basic VHDL/VHDL-AMS descriptive elements. Based on this knowledge, a Motor Controller (MC), which is a mixed-signal component, shall be designed and simulated. The MC takes commands and Pulse-width modulation (PWM) parameters from Serial Peripheral Interface (SPI) and translates them to electrical output signals. These output signals are driving one or two connected motors. These functionalities will be realised by the participating students in several design and simulation steps within this practical lab. Hereby, this lab is subdivided in three digital (part I to III) teaching units and one analogue (part IV).

## 1.2 Required Knowledge

For a successful participation in this course a regular and complete attending to the DHS lectures and exercises is necessary. The supervisors of this course will not repeat the lecture's content and they expect the previously taught knowledge about VHDL from every student! Deeper knowledge of VHDL can be additionally acquired by studying [1] or [2]. Furthermore the knowledge of the most important UNIX commands is also required for a successful work during the practical course. A short introduction can be found under: `https://ryanstutorials.net/linuxtutorial/`

## 1.3 Motor Controller Component

The MC, of these labs consists of a digital top level component called ”Digital PWM Generator” and an analogue ”Digital-to-Analog Motor Driver”.

The responsibilities of the digital part are the generation of PWM signals based on the given inputs from the SPI. Within the analogue part, the digital PWM signal gets converted to an electrical signal, which models the motor driver current and voltage. The specified behaviour of the analogue part is derived from a commercially available device, the DRV8837 of Texas Instruments [5]. Therefore, the timing and operational functionality of this teaching unit does not necessarily correspond with the device data sheet. In Figure 1 the top level components of the MC are shown.

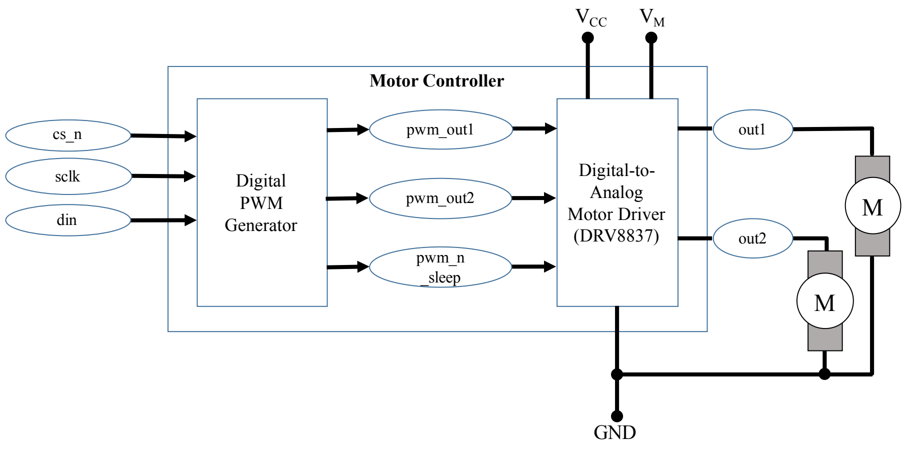

Figure 1: Top level view of the motor controller

The shown signals in Figure 1 have following meaning:

| Signal Name | Type | Function |
|---|---|---|
| $cs\_n$ | std_logic | Active low chip select |
| $sclk$ | std_logic | SPI clock |
| $din$ | std_logic | Serial data in port |
| $pwm\_out1$ | std_logic | PWM channel 1 output |
| $pwm\_out2$ | std_logic | PWM channel 2 output |
| $pwm\_n\_sleep$ | std_logic | Active low sleep enable signal |
| $out1$ | electrical | Motor driver channel 1 |
| $out2$ | electrical | Motor driver channel 2 |

Table 1: Port and internal signal definitions for the motor controller component

### 1.3.1 SPI Behaviour and Functionality

The shown SPI interface of the ”Digital PWM Generator” component (see Figure 1, left hand side) shall realise following timed interface behaviour:

1. With the falling edge of $cs\_n$ the $sclk$ clock signal is activated and the SPI communication is started.

2. The first two bits that are fed in at $din$ during two $sclk$ cycles represent the address bits ADR1 and ADR0 of the internal registers.

3. The bit of the next cycle (third cycle) represents the write bit and signalises the motor controller to write or not write out the bits of the corresponding register. When the RW bit is on low level the following data on $din$ is written to the motor controller. If RW bit is on high the following data is ignored.

4. During the following 14 cycles of $sclk$ 14 data bits are transmitted beginning with the Most Significant Bit (MSB). The meaning of these data bits depends on the selected register and is explained in the following sections.

5. After transmission of last bit, the Least Significant Bit (LSB), $cs\_n$ is rising again to high value and the $sclk$ stops cycling at high value level.

This communication cycle must be repeated to transfer all PWM parameters for the motor: base period, duty cycle and configuration (enable or disable, brake and rotation direction).

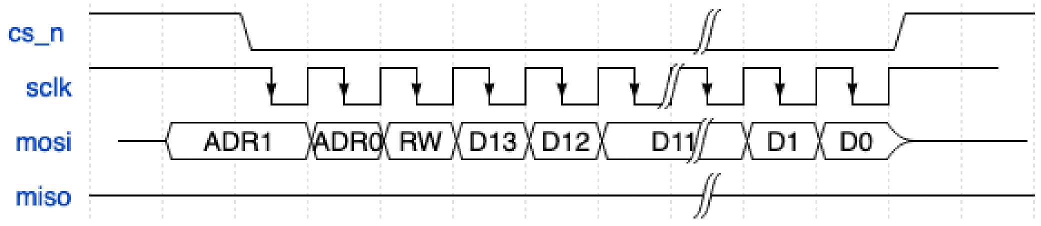

Figure 2: Timing and meaning of the SPI communication during a write operation to the motor controller.

#### 1.3.2 PWM Behavior and Functionality

The PWM outputs shall provide rectangle shaped pulses of variable length and variable duty cycle to the motors. Both parameters will influence the rotation rate of the motors. The intended PWM behaviour shall take the following requirements into account:

1. The PWM parameters must only be changed at the end of each PWM cycle.

2. The PWM duty cycle cannot be longer than the PWM base period.

3. After reset the motor should not rotate.

4. The PWM parameters must change continuously without larger jumps to protect the connected motors from huge changes of the electrical power and to provide a seamless acceleration and deceleration.

## 1.4 Setting up your digital environment

There are several files that are already available:

- spi.vhd contains the VHDL entity and the empty architecture for the SPI
- tb_spi.vhd contains the testbench for the SPI controller with basic test set
- pwm_controller.vhd contains the complete VHDL entity and architecture for the PWM controller
- tb_pwm_controller.vhd contains testbench for the PWM controller where you have to add test cases
- pwm_driver.vhd contains the VHDL entity and the empty architecture for the PWM driver

- tb_pwm_driver.vhd contains testbench for the PWM driver where you have to add test cases
- pwm_digital_top.vhd contains the VHDL entity and the empty architecture for the PWM generator top level
- tb_pwm_digital_top.vhd contains testbench for the PWM generator where you have to add test cases

To work on the files you need to type the following in a terminal window:

- `source /home/4all/tmp/DHS_ALL/start_labX.sh`

  where X depends on which lab you want to start (1 to 4)

This will load Questasim and opens one design file you have to edit. This must be done every time you want to work on your design! After you finished your work always close Questasim by click on File and on Quit!!

### 1.4.1 Compilation and Simulation

To compile your VHDL files use the shell command

- *vcom Filename.vhd*

You can compile multiple files with one command giving a list as argument. The file list is handled from left to right so basic components need to be written first.
A detailed introduction to the usage of Questasim will be given in the first lab!

## 1.5 Setting up your AMS environment

For Lab 4 you need to extend your simulator by analog capabilities. Your digital files will remain unchanged. In parallel to your digital directory *vhdl* a new directory named *vhdl_ams* is created. There are several files inside that directory already available:

- motor_controller_top.vhd is the top level of the AMS design which should contain the motor driver and the digital subsystem
- motormodel.vhd contains a very simple AMS model of a brushed DC motor. Please do not edit this file.
- motordriver.vhd is a template for your model of the TI8837 chip
- tb_motordriver.vhd is the testbench for the motordriver
- resistor.vhd is the behavioral analog model of a simple resistor
- tb_motor_controller.vhd is the testbench for the complete system (analog and digital parts)

To work on the files in AMS domain you need to type the following in a terminal window:

- `source /home/4all/tmp/DHS_ALL/start_lab4.sh`

This will load the AMS version of Questasim and opens the toplevel design file. This must be done every time you want to work on your design! After you finished your work always close Questasim by click on File and on Close!!

### 1.5.1 Compiling and Simulating

To compile your VHDL-AMS files use the shell command

- $vacom\ Filename.vhd$

You can compile multiple files with one command giving a list as argument. The file list is handled from left to right so basic components need to be written first.

## Online help

There is an online help for all the used software available. For the simulator (Questa SIM) you get the online help by typing `modelsimhelp`. This command shows the online help in the Acrobat reader as well.

## 2 Lab Part I - III: Design and Simulation of a PWM-Generator

Within lab part I to III a PWM-Generator has to be designed and simulated. This generator has three subcomponents, which are addressing different learning objectives of the DHS course. In each lab one of these subcomponents is used for a practical usage of specific VHDL language constructs. In summary following content shall be learned:

1. **PWM Controller:** Design of a testbench in VHDL, design and implementation of test cases and simulation of designs using QuestaSim.
2. **SPI:** Designing of behaviour using combinational and sequential logic and implementation of a Finite State Machine (FSM).
3. **PWM Driver and PWM Generator:** Reusing existing components to develop complex units by hierarchical design.

A graphical representation of the generator's architecture is given in Figure 3. The interface definitions of each subcomponent including the used data type description can be found in the next subsections.

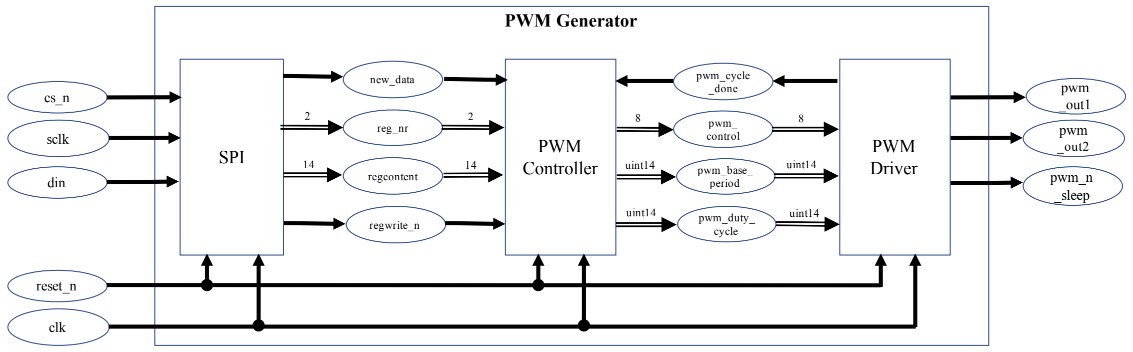

Figure 3: The subcomponents of the digital PWM generator.

## 2.1 Lab Part I: Simulation of the PWM-Controller

In this lab students shall learn to develop a VHDL testbench and write test cases to validate the behaviour of a given design. Subsequent to these steps the simulation can be started and the results be validated (comparison between expectations and actual results).

Therefore, the PWM-Controller is used in this lab as a design under test and an appropriate testbench for its validation has to be designed. First, the interface/port definitions are given in following table and the specification of the design, which has to be validated, is described afterwards.

### 2.1.1 Port definitions

| Portname | Direction | Type | Width | Function |
|---|---|---|---|---|
| $reset\_n$ | in | std_logic | 1 | Low active asynchronous reset |
| $clk$ | in | std_logic | 1 | Clock signal (frequency: 100 MHz) |
| $new\_data$ | in | std_logic | 1 | Control signal: New data on $regcontent$ available |
| $regnr$ | in | std_logic | 2 | Address of controller's internal register: 01 addresses the PWM base period, 10 the duty cycle, 11 the control flags and 00 is not used |
| $regcontent$ | in | std_logic | 14 | Input register content |
| $regwrite\_n$ | in | std_logic | 1 | Control signal: Low value - Write access to controller's internal registers, High value - No write access |
| $pwm\_cycle\_done$ | in | std_logic | 1 | Control signal from PWM-Driver: High value - A PWM base period has finished and changes on PWM parameters are now allowed and taken by the driver |
| $pwm\_control$ | out | std_logic | 8 | Control bits for the PWM-Driver: $pwm\_control(2)$ - brake , $pwm\_control(1)$ - reverse PWM for pwm_out2 and $pwm\_control(0)$ - enable_all |
| $pwm\_base\_period$ | out | integer | range 0 to $2^{14}$ - 1 | Amount of clock cycles for one PWM period |
| $pwm\_duty\_cycle$ | out | integer | range 0 to $2^{14}$ - 1 | Amount of clock cycles in which the PWM output signals are active (driving) |

Table 2: Port definition for the PWM-Controller component.

### 2.1.2 Lab Task: Validating the Specification by Simulation

**Your task** is to write a testbench including all necessary testcases to validate the following specification:

1. The component's internal state and output ports are initialised by an active low asynchrobous reset.
2. The frequency of the design shall be 100 MHz and the design is synchronous to the rising edge of the clock.
3. New data is fetched from the $regcontent$ to the internal registers only when $regwrite\_n$ is low.
4. An internal register is addressed by the bit pattern provided on $regnr$. $01$ addresses the PWM base period, $10$ the duty cycle, $11$ the control flags and $00$ is not used.
5. When $pwm\_cycle\_done$ is on high value, the controller calculates and provides new values for $pwm\_control$, $pwm\_base\_period$ and $pwm\_duty\_cycle$. Signal $pwm\_cycle\_done$ wille be set to high value after each complete PWM cycle.
6. The lower 8 bits of $regcontent$ are used for the $pwm\_control$.
7. When the internal register of $pwm\_base\_period$ (corresponding address: 01) or $pwm\_duty\_cycle$ (corresponding address: 10) is changed, then the integer output values are slowly step-wise changed at the end of each PWM cycle. The increasing step height for the base cycle is 64 and 32 for the duty cycle. Decreasing step height for the base cycle is 31 and 31 for the duty cycle.
8. If the increasing step height exceeds the upper limit of the base or duty cycle parameters, then the parameter is set to its maximum value. Likewise, if the decreasing step heights exceeds the lower limit of these parameters, the lowest value is used.

As an example we are providing the first testcase of your testbench: The validation of the reset. Therefore, the reset port is set to low value ($'0'$) for 10 clock cycles. During simulation you should observe that all output ports are initialised by non-'U' values. The PWM parameters for instance are initialised by the PWM generator component. Please check this behaviour with the generated waveforms of your simulator! Please make sure that your test cases do not run longer than 200 ms!

The lab is successfully passed when your designed testcases are completely covering the specified behaviour which can be checked with the following script:

- `source check_ctrl.sh`

## 2.2 Lab Part II: Implementation and Simulation of the SPI

In this lab the main focus is on implementing a design based on a given specification which shall be followed. The design itself is sequential and combinational as well and in order to realise it a finite state machine has to be described. In this case the SPI acts as an interface to the PWM generator. It receives serial data and converts them into appropriate values that are set to the parallel outputs and further on will be written to the PWM controller.

### 2.2.1 Port definitions

In table 3 an overview of the ports of the SPI Controller is given. Based on them a synthesizable implementation shall be realised. Please note that the interface ports are fixed and must have the same name as described in the table!

| Port name | Direction | Type | Width | Function |
|---|---|---|---|---|
| $reset\_n$ | in | std_logic | 1 | Low active asynchronous reset |
| $clk$ | in | std_logic | 1 | Clock signal (frequency: 100 MHz) |
| $sclk$ | in | std_logic | 1 | Serial Clock for the SPI interface (frequency: 5 MHz) |
| $cs\_n$ | in | std_logic | 1 | active low chip select for the SPI |
| $din$ | in | std_logic | 1 | Serial data input |
| $new\_data$ | out | std_logic | 1 | Indicates that new data is available at $regcontent$ |
| $regnr$ | out | std_logic | 2 | Address of the PWM Controller registers |
| $regcontent$ | out | std_logic | 14 | Parallelized content that is written to the register which is specified in $regnr$ |
| $regwrite\_n$ | out | std_logic | 1 | Low active write access to the registers of the PWM controller |

Table 3: Port definition for the SPI controller component.

### 2.2.2 Lab task: Design a SPI Controller

**Your task** is to write a design that realises the conversion of a serial SPI communication to a parallel access to register content of the subsequently following PWM controller. Take especially care that there are two clock domains (system clock and SPI clock)! The component shall fulfil the following specification:

1. The component acts as a SPI slave.
2. Only the access to SPI interface shall be synchronous to the falling edge of the SPI clock ($sclk$) with a clock frequency of 5 MHz.
3. The access for the remaining parts of the design including I/O ports shall be synchronous to the rising edge of the system clock ($clk$) with a clock frequency of 100MHz.
4. The component's internal state and output ports are initialised by an active low reset.

5. The default value for $regwrite\_n$ after reset shall be 1.

6. The control of the SPI interface shall be realised with a finite state machine that is synchronous to the system clock.

7. Only fetching data to $DIN$ is supported

8. Each write access to the SPI consists of exactly 17 cycles where the first two indicate the address to which data shall be sent, the third indicates a read/write access where RW=0 is write and RW=1 shall not be implemented (no read access!). The remaining 14 cycles are the actual data which is transferred to the register selected by the first two bit. Refer also to Figure 2 on page 6.

9. When the input $cs\_n$ is set from 1 to 0 a new data transmission is initiated and with the next falling edge of $sclk$ the data at $din$ shall be read for the next 17 clock cycles with every falling clock edge of $sclk$ as long as $cs\_n$ is 0.

10. After the last bit has been fetched the port $new\_data$ shall be set to 1 for one system clock cycle ($clk$) to indicate that new data is available.

11. At the same time when $new\_data$ is set the received value must be written to the outputs $regnr$ (the first 2 received bit), $regwrite\_n$ (the third bit) and to the remaining data to $regcontent$.

Please check this behaviour with the generated waveforms of your simulator! An indication of the correct behaviour is given with the following script:

- `source check_spi.sh`

Be aware that this only checks a subset of the overall behaviour! Final results can be seen after lab 4 and will be checked by the supervisors!

## 2.3 Lab Part III: Implementation and Simulation of the PWM Driver and the top level design

This lab consists actually of two parts where the first will be the design of a PWM driver that gets values from the PWM controller like the base period and the according duty cycle and depending on them generates appropriate PWM signals at its outputs. In the second part all the designed components from lab part I to III will be used and connected together to form the complete PWM generator from its SPI inputs to the PWM outputs (refer to figure 3 on page 9).

### 2.3.1 Port definitions for the PWM Driver

| Port name | Direction | Type | Width | Function |
|---|---|---|---|---|
| $reset\_n$ | in | std_logic | 1 | Low active asynchronous reset |
| $clk$ | in | std_logic | 1 | Clock signal (frequency: 100 MHz) |
| $pwm\_cycle\_done$ | out | std_logic | 1 | Indicates that one PWM cycle finished. |
| $pwm\_control$ | in | std_logic | 8 | Control bits for the PWM-Driver: $pwm\_control(2)$ - brake , $pwm\_control(1)$ - reverse PWM for pwm_out2 and $pwm\_control(0)$ - enable_all |
| $pwm\_base\_period$ | in | integer | range 0 to $2^{14}$ - 1 | Amount of clock cycles for one PWM period |
| $pwm\_duty\_cycle$ | in | integer | range 0 to $2^{14}$ - 1 | Amount of clock cycles in which the PWM output signals are active (driving) |
| $pwm\_out1$ | out | std_logic | 1 | PWM channel 1 output (forward direction) |
| $pwm\_out2$ | out | std_logic | 1 | PWM channel 2 output (reverse direction) |
| $pwm\_n\_sleep$ | out | std_logic | 1 | Low active sleep enable |

Table 4: Port definition for the PWM-Controller component.

### 2.3.2 Lab task A: Design the PWM Driver

**Your task** is to write a design that realises the conversion of the integer values coming from the PWM Controller and depending on the control input set the outputs following this specification:

1. The component's internal state and output ports are initialised by an active low reset.
2. The frequency of the design shall be 100 MHz and the design is synchronous to the rising edge.
3. The base period of the PWM outputs is set by the value of $pwm\_base\_period$ where its value indicates the number of clock cycles for one PWM period.
4. The duty cycle of the PWM outputs is set by the value of $pwm\_duty\_cycle$ where its value indicates the number of clock cycles for which PWM output is active.

5. The behaviour of the PWM Driver outputs is controlled by the input $pwm\_control$ where the value of bit position 0 indicates the enable (1=enable outputs, 0=disable outputs), bit position 1 indicates whether out_pin2 is inverted compared to out_pin1 (0=same behavior as pin1, 1=inverted output) and bit position 2 indicates brake (0=normal operation, 1=brake).

6. When enable is set to 0 the outputs $pwm\_out1$ and $pwm\_out2$ shall be high impedance and the output $pwm\_n\_sleep$ shall be set to 0, when enable is 1 the following shall be realised:

   (a) $pwm\_n\_sleep$ shall be set to 1 and the not active output shall be set to 0.

   (b) $pwm\_out1$ shall output the PWM value directly

   (c) $pwm\_out2$ shall output the inverted PWM value if $pwm\_control(1)$ is 1, othwerwise it should output the same as $pwm\_out1$ .

   (d) Each PWM period shall be $base\_period$ clock cycles.

   (e) The duty cycle (high value at coressponding output) shall be $duty\_cycle$ clock cycles.

   (f) When brake is 1 then both PWM outputs shall be set to 1.

#### 2.3.3 Port definitions for the PWM Generator

| Port name | Direction | Type | Function |
|---|---|---|---|
| $reset\_n$ | in | std_logic | Low active reset |
| $clk$ | in | std_logic | System clock signal |
| $sclk$ | in | std_logic | Serial Clock for the SPI interface |
| $cs\_n$ | in | std_logic | active low chip select for the SPI |
| $dinl$ | in | std_logic | Serial data input |
| $pwm\_out1$ | out | std_logic | PWM channel 1 output |
| $pwm\_out2$ | out | std_logic | PWM channel 2 output |
| $pwm\_n\_sleep$ | out | std_logic | Active low sleep enable |

Table 5: Port definition for the PWM-Controller component.

Please check this behaviour with the generated waveforms of your simulator! An indication of the correct behaviour is given with the following script:

- `source check_pwm_drive.sh`

Be aware that this only checks a subset of the overall behaviour! Final results can be seen after lab 4 and will be checked by the supervisors!

#### 2.3.4 Lab task B: Design the PWM Generator

**Your task** is to create out of the components you have designed in the labs a top level design including all digital parts of the PWM generator. Refer to figure 3 on page 9. In this case no additional logic is needed, the whole description shall a structural description of the design. Please check this behaviour with the generated waveforms of your simulator! An indication of the correct behaviour is given with the following script:

- `source check_top.sh`

Be aware that this only checks a subset of the overall behaviour! Final results can be seen after lab 4 and will be checked by the supervisors!

## 3 Lab Part IV: Implementation and Simulation of the Analog and Mixed-Signal part of the Motor Controller

The Analog/Mixed-Signal part of the motor controller is represented by a simplified model of the TI DRV 8837 chip. It contains the conversion logic and the driver circuit. In this lab you should implement only some parts of this chip to reduce complexity. The connected motor model is provided by the supervisors. Those modules operate in time-continuous domain, so it is necessary to switch the toolchain to VHDL-AMS [6]. Within the DHS lab you shall create a simulation model of the driver stage and finally combine it with the digital part to a complete AMS simulation model of the motor controller.

### 3.1 Setting up your AMS environment

For Lab 4 you need to extend your simulator by analog capabilities. Your digital files will remain unchanged. In parallel to your digital directory *vhdl* a new directory named *vhdl_ams* is created. There are several files inside that directory already available:

- motor_controller_top.vhd is the top level of the AMS design which should contain the motor driver and the digital subsystem
- motormodel.vhd contains a very simple AMS model of a brushed DC motor. Please do not edit this file.
- motordriver.vhd is a template for your model of the TI8837 chip
- tb_motordriver.vhd is the testbench for the motordriver
- resistor.vhd is the behavioral analog model of a simple resistor
- tb_motor_controller.vhd is the testbench for the complete system (analog and digital parts)

To work on the files in AMS domain you need to type the following in a terminal window:

- `source /home/4all/tmp/DHS_ALL/start_lab4.sh`

This will load the AMS version of Questasim and opens the toplevel design file. This must be done <u>every time</u> you want to work on your design! After you finished your work <u>always</u> close Questasim by click on File and on Close!!

#### 3.1.1 Compiling

To compile your VHDL-AMS files use the shell command
*vacom src/Filename.vhd*

You can compile multiple files with one command giving a list as argument. The file list is handled from left to right so basic components need to be written first.

### 3.1.2 Simulating

To simulate your VHDL-AMS files please use the scripts given in the verification section of the single tasks. Please always close the simulator before starting the scripts again!

## 3.2 Implementation of the driver stage (lab 4 part A)

### 3.2.1 Overview

The sub parts of the TI DRV 8837 are visualised in Figure 4.

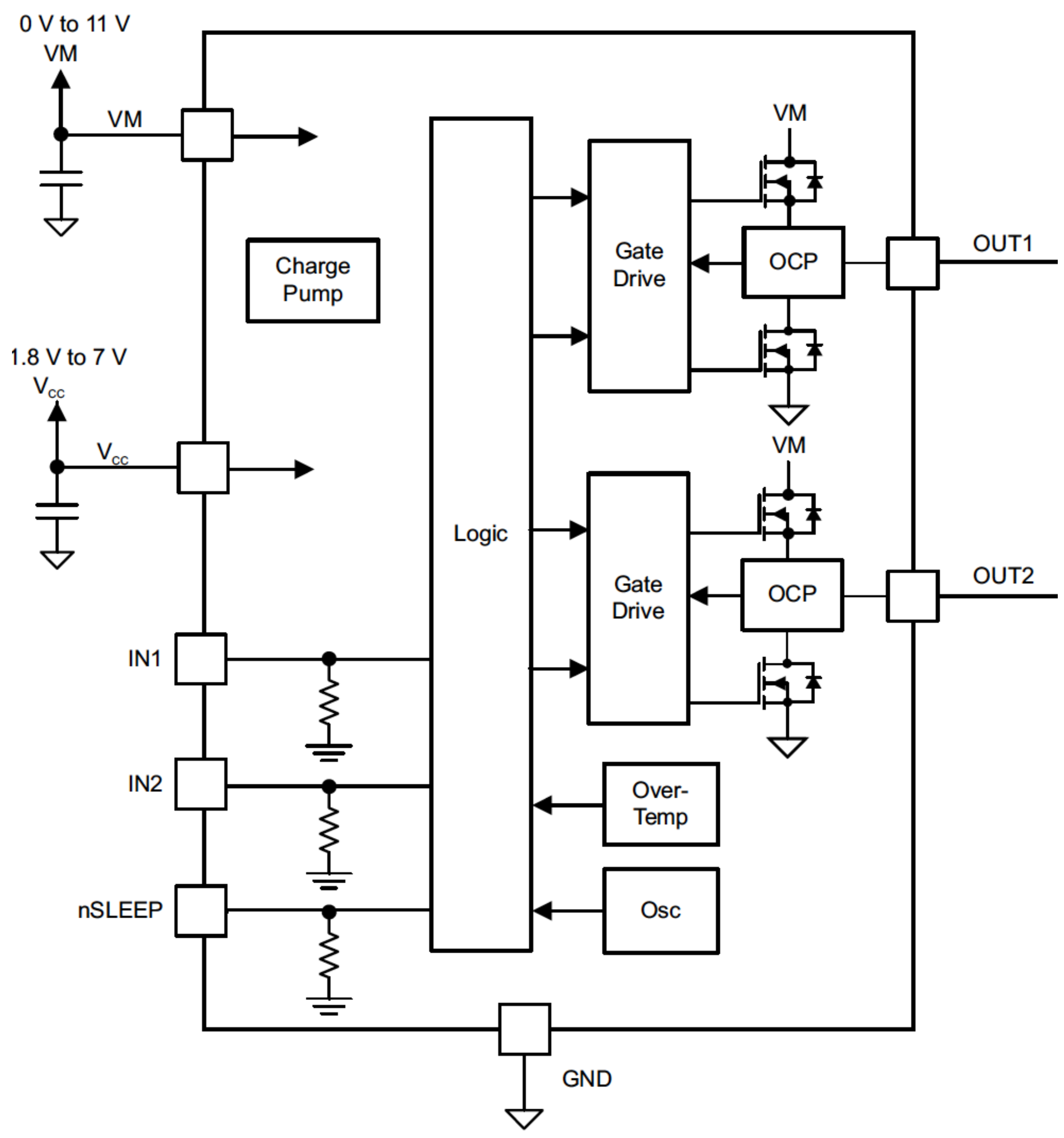

Figure 4: The sub parts of the TI DRV 8837 chip [5])

In analog domain on element level all interfaces must be represented by **terminals**. So all nodes beyond the Gate Drive units must be modeled using AMS constructs. The blocks Charge Pump, OCP, Over-Temp and Osc as well as the internals of Gate Drive are ignored in the model.

We focus on the electrical behavior of the output stage controlled by the Logic Block fed by IN1, IN2 and nSLEEP.

Within the lab you should create:

- the logic block driving the four MOSFETs at the output
- the MOSFETs as switchable resistors (ON = 0.01 Ohm, OFF = 10 MOhm)
- the diodes across the MOSFETs. Their parameters are:
  - Thermal voltage $25\,\mathrm{mV}$
  - Saturation current $1\,\mathrm{nA}$
  - Ideality factor $1.1$

### 3.2.2 Components

VHDL-AMS does not contain predefined electrical elements. So the first step is to create the necessary basic elements. In the directory you will find a resistor in the file resistor.vhd. You shall create a switchable resistor (sw_resistor.vhd) and a generic diode (diode.vhd) based on this resistor.
The switchable resistor shall have two terminals A and B of type electrical, and one digital input to control the switch. Please remember that a break statement might be necessary at each switch timepoint.

The diode shall have two terminals A and C of type electrical and three generic parameters thermalVoltage, idealityFactor and saturationCurrent of type real. It shall use the modelling equation

$$I_d = saturationCurrent \cdot e^{(\frac{U_d}{idealityFactor \cdot thermalVoltage} - 1.0)} \tag{1}$$

### 3.2.3 Driver Model

After creating diode and switch model you should now create the driver model. Please open the file motordriver.vhd. Here you will find a predefined entity. You must now add the architecture including:

- a digital process to control your switchable resistors
- four instances of the switch and the diode

The logic table pf the driver model is taken from the component data sheet [5]. Please note that in analog domain the high impedance state cannot be represented by using a 'Z' like in std_logic data type. Use the switchable resistors for the according analog characteristics.

### 3.2.4 Verification

Now please test your driver model using the provided testbench. The testbench currently only checks the driving of motor1 and motor2. Please extend the testbench to check also the other requirements from table 6.
To simulate your motordriver please use the script $./sim\_motordriver$
A rough check is possible using $./check\_motordriver$
If the model works as expected you may continue with part B.

| nSLEEP | In1 | In2 | Out1 | Out2 | Function |
|---|---|---|---|---|---|
| 0 | X | X | Z | Z | Coast |
| 1 | 0 | 0 | Z | Z | Coast |
| 1 | 1 | 0 | H | L | Drive motor 1 |
| 1 | 0 | 1 | L | H | Drive motor 2 |
| 1 | 1 | 1 | L | L | Brake |

Table 6: Logic output definition of motor driver.

## 3.3 AMS Model of the complete Motor controller (lab 4 part B)

The digital model from lab 1 to 3 shall now be combined with the analog driver part to a complete model of the Motor Controller. In the file motorcontroller.vhd you will find the toplevel entity. Now include your component models and connect them properly based on the block diagrams given in the introduction 1.3.

### 3.3.1 Verification

In the directory you will find an AMS testbench file named tb_motorcontroller.vhd. Run the simulation to prove that all components are working properly.
The testbench is currently checking only one parameter set. Now please extend the testbench by a second testcase (the first one must remain active!): Extract the lower 4 digits (0-9999) from your own matriculation number and the number of your partner. Now send the larger of those two numbers to the base register and the lower number to the duty cycle register. If you work alone please use 234567 as second matriculation number.

To simulate your design please use the script *./sim_all*

**The provided scripts and testbenches will not check all requirements given in this manual. You must extend the testbenches of lab2-4 to cover 100 % of the specification.**

## 3.4 Final Submission

Before you submit your files please

- enter your names and matriculation numbers as comment in each file you modified
- check that the files you plan to submit are the very last tested version
- collect all .vhd files that you modified (testbenches of lab1 and lab4, model files of lab2-4)
- do not submit old file versions. We will not search multiple files for a correct version.
- submit only .vhd files. We are not interested in simulation plots or work library contents.
- make sure that the files have the correct extension! the submission script will only submit .vhd and .vhdl files
- make sure the correct files are in the /src folder! Only these files will be copied

**A script is used to submit your design. In order to execute this script make sure, that you are in folder /dhs there you have to enter the command** *./copy_design.sh*
**Your solution can only be submitted one time!**