DYNAMIC PARTIAL RECONFIGURATION AS AN APPROACH TO MOTOR CONTROL DESIGN

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Paul Leeds Rogers III

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

February 2018

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

Title

DYNAMIC PARTIAL RECONFIGURATION AS AN APPROACH TO MOTOR CONTROL DESIGN

By

Paul Leeds Rogers III

The supervisory committee certifies that this thesis complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Scott C. Smith
Co-Chair

Rajesh Kavasseri
Co-Chair

Majura Selekwa

Approved:

February 28, 2018
Date

Benjamin Braaten
Department Chair

ABSTRACT

Recent work in motor controls has been utilizing the Field Programmable Gate Array (FPGA) instead of using the more standard microprocessors or digital signal processors. FPGAs have the advantage of flexibility and parallelization, allowing the platform to be used for multiple things. One aspect of the FPGA that has not been utilized in motor controls is Dynamic Partial Reconfiguration (DPR), which allows part of the FPGA to be reconfigured without disabling it.

In this thesis, two controllers, field-oriented control and voltage-by-frequency control, are implemented on an FPGA using DPR. Only one of the controllers is actually implemented on the FPGA at a time. This system trades FPGA area for memory requirements. This configuration was simulated and implemented in a physical test bench, showing that the transition between controllers at three different speeds is stable, indicating this is a potential direction for motor controls.

ACKNOWLEDGEMENTS

I would like to thank Dr. Scott Smith, Dr. Rajesh Kavasseri, and Dr. Majura Selekwa for their support in this thesis process. I thank Dr. Dong Cao and the SPEED Lab for their help in guiding me through the nuances of electric drives. For guiding me through some of my problems and assisting me while I was away from the lab, I thank Ashiq Adnan. I thank Dr. Michael Maassel and Jeffrey Erickson for their academic support during my time at NDSU.

I would also like to acknowledge John Deere, Inc. for providing financial assistance for this work.

DEDICATION

To my lovely and understanding fiancée, Rebecca Trubitt.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF APPENDIX FIGURES

Figure Page

1. INTRODUCTION

A recent shift away from fossil fuels has prompted increasing replacement of internal combustion engines with electric motors, which can better utilize sustainable energy sources. These replacements are not the DC motors that are in many applications already, but large three-phase AC machines similar to those in generators or industrial pumps. The industry standard for these large motors has been the induction machine, but recently the permanent magnet synchronous machine (PMSM) has gained traction due to its excellent performance and efficiency [1]. These motors are now being used in unmanned aerial vehicles [2], automobiles [3], and even in power generation [4].

1.1. Motor Controls

With their wide range of applications comes a large variety of control techniques, notably scalar and vector controls. Scalar type controllers have much simpler computational requirements when compared to vector controls, but suffer in their overall performance because of it. Voltage by frequency (V/f) control is the most common form of scalar control in industry, and is most commonly used for drives with low performance requirements. This scheme controls speed and torque by keeping the voltage-frequency ratio constant. One major advantage for V/f control is that it can be implemented in open-loop configuration, or closed-loop for added performance at the cost of complexity [5]. For higher performance drives, vector control schemes are used, the two most common being Direct Torque Control (DTC) and Field-Oriented Control (FOC). These methods are both closed-loop controlled and differ in how they obtain their control signals. DTC directly uses the torque and current parameters from the motor to select the optimum control signal to send the motor, whereas FOC manipulates the same inputs to generate a desired waveform signal, which is then generated by an inverter [6]. FOC is more computationally intensive than DTC and has generally better performance; but this makes FOC more difficult to design. The only major advantage of DTC over FOC is complexity; it is much easier to design and requires fewer resources than FOC [6]. Thus, while DTC is gaining favor in industry, this thesis focuses on V/f and FOC, which will both be discussed further in Chapter 2.

The standard way of implementing these controllers for high performance drives is on a digital signal processor (DSP), which is very good at performing the complex calculations needed for these control algorithms. Alternatives to DSPs are typically microprocessors, but these lack the computational speed required for high-performance motor drive control [7]. Field programmable gate arrays (FPGAs) are another alternative to DSPs and are now being used in research for high performance drive control [8, 9, 10, 11, 12]. They offer significantly higher performance in complex calculations over microprocessors; and while they may not be as fast as DSPs, they offer another advantage: flexibility. As an example, a single FPGA could be used to control the communications, motors, and some image processing for an autonomous car. FPGAs do have their disadvantages though; the design and implementation process can be more difficult than for DSPs, and the cost, both up front and maintenance, can be more expensive [13].

1.2. Previous Work

Most prior research into improving motor controls focuses on improving the algorithms or deriving new ones [9, 14, 15, 16]. If these new algorithms provide improved performance in many different conditions, they are typically very complex. An alternative explored by some researchers is to implement two different control schemes on the same FPGA and use a state machine or other similar function to switch which controller was active [13, 14]. This could allow for the selection of the appropriate simpler algorithm that best fits the present environment; however, implementing both controllers at the same time does not take full advantage of the FPGA and could still be done on a DSP [13].

To fully exploit the FPGA, other researchers have used one of the FPGA's most unique capabilities: dynamic partial reconfiguration (DPR). This allows a portion of the FPGA to be reconfigured while the rest of the FPGA continues to function as normal. This technology has seen much use in computer vision and communications, but recently power conversion researchers have become interested in this technology. After the authors of [17] used DPR to change the PWM scheme, they called for more research into DPR for motor drives, including control methods. This area has yet to be explored, only having a single reproduction of the original paper published [18].

The main advantage of DPR over standard FPGA usage is area. DPR allows access to many different controllers at the cost of slightly more area than the largest configuration, whereas implementing every one of those controllers would require significantly more area. This allows the

FPGA to maintain its flexibility while implementing multiple control schemes, allowing for the use of a smaller, cheaper FPGA, or the ability to switch to a simpler controller if any sensors required for advanced control fail. However, DPR directly trades FPGA area for memory, as the configuration files for each controller need to be stored. Additionally, DPR increases the complexity of the design by adding one more layer on top of the control methods.

1.3. Summary

In summary, this thesis utilizes DPR on an FPGA as a new approach to motor control development, specifically demonstrating the usefulness of this technology by implementing a DPR controller for a PMSM that switches between FOC and V/f control. This technique could improve performance by switching to a scheme better suited to the environment, while reducing the area for such an application, allowing for more functionality to be included on the FPGA, a smaller, cheaper FPGA to be used, or a simple fallback controller to be used in case of sensor failure. Chapter 2 describes in detail the FOC and V/f control methods used in this work, and how they were implemented on the FPGA. Chapter 3 describes the DPR design process; and Chapter 4 shows the experimental results of the DPR control implementation. Chapter 5 provides conclusions and outlines future work. Lastly, an Appendix of the files used in this work is included.

2. CONTROL SCHEME OVERVIEW

While DTC is being used more and more in industry, the standard method of control for three-phase motors is still FOC; however, because it uses sensors to obtain feedback, this method has additional points of failure. Sensorless, open-loop control is not typically desired in high performance drives, but if the operation of the motors is critical to safety, such as in a car, using it as a backup to retain control is preferable to a loss of control. Therefore, FOC was chosen due to its prevalence in industry with V/f as a fallback scheme. This chapter discusses how these algorithms work, how they were interfaced with the Pulse Width Modulation (PWM) generator, and how they were implemented on an FPGA.

2.1. FOC

The FOC method attempts to control the torque produced by a motor. This is typically accomplished by controlling the motor currents, which aligns the stator flux so that the torque is regulated. The configuration used in this work can be seen in Figure 2.1. This algorithm can be broken down into three main components: conversion to the control variables, control of these variables, and conversion to the control signals.

The purpose of the first stage is to convert the three-phase instantaneous current (i_{abc}) from the motor's rotating reference frame into values that are two-phase, time-independent, and in a stationary reference frame (i_{dq}). The input triplet is very difficult to control because of its constant changing, but the transformations create a more easily controlled pair of values. The

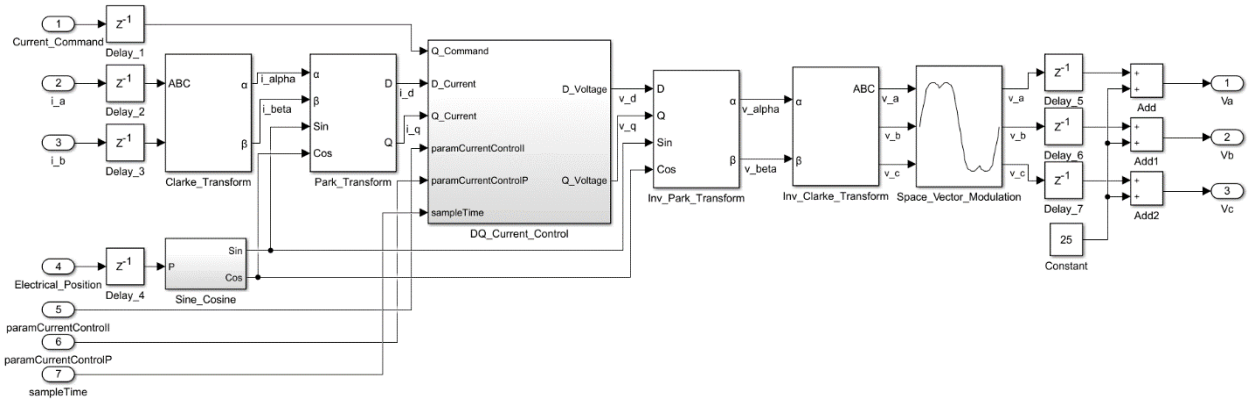

Figure 2.1. Block diagram for the FOC control method.

transformation begins with the Clarke and Park transformations, in that order, shown in Eqs. 2.1 and 2.2, respectively. To make the system simple, the system was assumed to be balanced, i.e., $i_a + i_b + i_c = 0$ and $i_\alpha = i_a$, which eliminates the need for one of the three phases for the transformation, reducing the number of sensors required in the final implementation.

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} i_d \\ i_q \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} \tag{2.2}$$

The θ in the Park transformation is the rotational position of the rotor. The resulting values (i_{dq}) are then passed to the PI controller. i_d effectively controls the motor flux and i_q effectively controls the motor torque. The PI controller is trying to keep the i_d value close to 0 and the i_q value at a specified value to produce the desired torque. The PI controller produces a pair of voltage values (v_{dq}), but they are still in the time-independent rotating reference frame. To obtain a three-phase, time-dependent, stationary reference frame set of control voltages (v_{abc}), they are passed through the inverse Park and Clarke transforms, in that order, shown in Eqs. 2.3 and 2.4, respectively. The resulting values are passed to the PWM generator, which is discussed in Section 2.3.

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} v_d \\ v_q \end{bmatrix} \tag{2.3}$$

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} \tag{2.4}$$

2.2. Scalar V/f

The V/f method controls the speed of the motor while attempting to keep the applied torque constant. One of the largest differences between this control method and FOC is that V/F

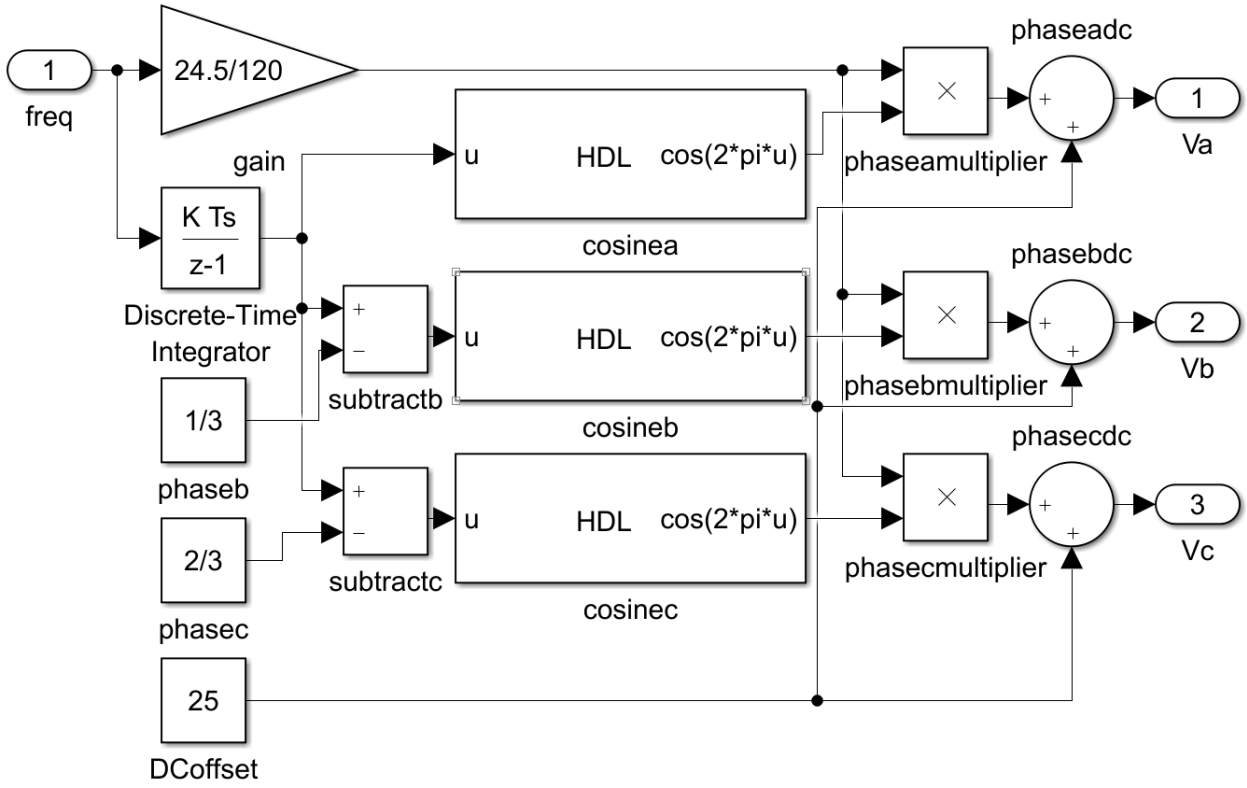

Figure 2.2. Block diagram for the V/f control method.

is an open-loop control scheme, meaning that it has no feedback loop. The block diagram for V/f can be seen in Fig. 2.2.

As can be seen from the block diagram, this method essentially keeps the voltage and frequency driving the motor proportional. The reason this works is as follows: assuming steady state operation of the motor, that the motor is operating in the linear magnetic region, and a few other simple things described in [7] and [19], then the phasor equation, Λ_m, for the magnetizing current is:

$$I_m = \frac{\Lambda_m}{L_m} = \frac{V_s}{2\pi f L_m} \rightarrow \Lambda_m = \frac{V_s}{2\pi f} \tag{2.5}$$

This shows that if the ratio V_s/f stays constant, the stator flux stays constant. Thus, the speed of the motor can change while maintaining constant flux if that ratio stays constant. This causes torque to only be determined by slip speed. As this added complexity without reason, a slip speed controller was not implemented.

The speed given to the controller is initially in rotations per minute (rpm), but the controller requires a frequency to produce the controlling output waveforms. As a PMSM is synchronous, the rotational speed of the internal magnetic field matches the rotational speed of the motor. To calculate the rotational speed of the magnetic field, the number of pole pairs, P, and the desired speed of the motor, n_{ref}, in rpm are needed. As frequency is in Hertz, a conversion factor from rpm to Hz was included, producing:

$$f_{ref} = \frac{n_{ref}}{60} P \tag{2.6}$$

The reference frequency, f_{ref}, is then passed to the voltage calculator and the 3-phase calculator. Inside the voltage calculator is a scalar, K, defined as:

$$K = \frac{V_{rated}}{f_{rated}} \tag{2.7}$$

By multiplying f_{ref} by this ratio, the voltage and frequency ratio in Eq. 2.5 stays constant.

Inside the 3-phase calculator, the three reference waves are generated using the reference voltage and the reference frequency. Each wave is 120° apart, yielding a phase constant for the B and C phases of $(-2\pi/3)$ and $(-4\pi/3)$, respectively. To obtain the phase angle to calculate the instantaneous voltage required, the reference frequency is multiplied by $2\pi t$, where t is a loop counter iterating every clock cycle at a rate of $1/f_{sample}$, where f_{sample} is the sampling rate of the triangle wave generator in the PWM generator block. These waves are also shifted up by 25 V so that the center is the same as the center of the PWM generator. These waves are then sent to the PWM generator block.

2.3. PWM Generator

The PWM generator used in this work was designed to work with both the FOC and V/f algorithms. The basic block diagram for it can be seen in Fig. 2.3. This is a simple PWM generator in that it simply compares the incoming signals to a triangle waveform to generate the PWM. As the system was designed with binary in mind, the triangle waveform was generated as a counter that counts from 0 to 50 and then 50 to 0. The incoming waveforms were therefore centered at 25 before being compared to the triangle waveform. Before the comparators, the incoming waveforms were shifted up for the less than comparison and down for the greater than comparison to introduce

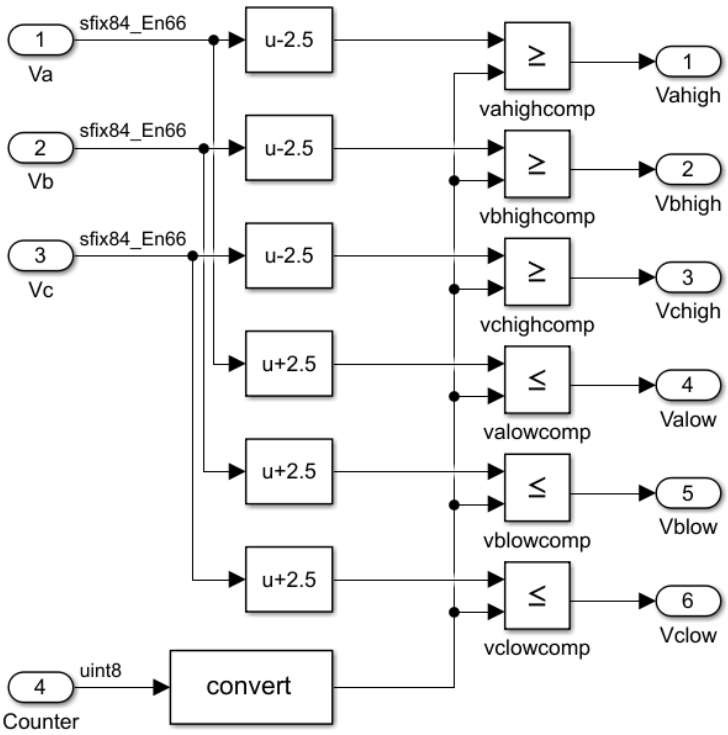

Figure 2.3. PWM Generator block diagram. The Va, Vb, and Vc inputs are offset by 2.5 to avoid static shoot-through current.

dead-time in the PWM to prevent shoot-through current in the inverter. The inclusion of the greater than and less than comparators was to ensure minimal output delay between comparator pairs.

2.4. MATLAB Implementation

The implementations for both the FOC and V/f were inspired by [12]. The design flow was: design the control algorithm in MATLAB Simulink using only HDL Coder compatible blocks, verify it works as intended in Simulink, convert to VHDL, and use as a standard VHDL component. This process allows for quickly piecing together a controller in Simulink to convert to VHDL instead of going through the painstaking process of writing the controller in VHDL from scratch. In addition, the control algorithm can be tested on an FPGA using MATLAB's HDL Verifier. The FPGA-in-the-loop (FIL) simulations allow MATLAB to configure an FPGA with the design and then run a Simulink testbench, which in this case included a PMSM model.

3. SYSTEM DESIGN

A complete overview of the system designed for this work is presented here, broken into three parts. The first is a full overview of the system and how each piece interacts. The second is the hardware description of the system, including component choices and printed circuit board (PCB) design descriptions. The third is the description of the VHDL implementations of the interfaces and algorithms that compose the system.

3.1. System Overview

The system can be broadly described as a motor controller that switches between two control algorithms using DPR. Required for this system are an FPGA, a PMSM (the motor), sensors, memory modules, and interfacing hardware. The FPGA is where all the control and DPR logic are held. Between the FPGA and the PMSM are optoisolators and the motor driver. The optoisolators are included to electrically decouple the sensitive FPGA from the high-voltage, high-current motor driver. The driver system converts the low-power FPGA signals to ones that can drive the motor. The sensors provide phase current information and motor position to the control algorithms. The memory modules hold the partial bit-files that describe the control algorithm configurations for the FPGA. There are also interface boards that provide logic level shifting to allow the FPGA and the sensors/memory modules to communicate.

3.2. Hardware

Before describing the VHDL design for the system, the hardware that makes up the system should be described. One of the major hardware limitations was the inverter, which was an IRAM136-3063B. This device was chosen due to limited accessibility of other devices, and it came with a limiting factor of its maximum PWM frequency of 20 kHz. This limitation impacted the PWM generation frequency in the VHDL section and the speed requirements of the sensor module components. The sensor module was comprised of a Hall-effect current sensor (ACS758LCB-050B) and an analog-to-digital converter (ADC) (AD7822BNZ). The isolator between the full bridge rectifier and the FPGA was comprised of several CPC5001 optical isolators configured so that one side was operating at 3.3 V and the other at 5 V, allowing for proper operation of the inverter. Similar operation for logic level shifting was obtained by using several 74LVCH2T45 translators on a sepa-

rate PCB. The FPGA itself was a Xilinx Virtex-5 (XC5VLX50-1FFG676) on an ML501 evaluation board. For the memory modules, a pair of SST39SF040-70 were used. These flash memory modules were selected due to their compatibility with the available programmer and their read speed of 70 ns. The speed of these components was important because the faster access allows the DPR to complete faster, potentially allowing the device to be completely reconfigured before a single PWM clock has finished. The motor was a PMSM with a rated power of 250 W, rated voltage of $42\,V_{dc}$, rated current of $5.7\,A_{rms}$, maximum speed of 4000 rpm, KT of $0.0757\,\frac{N\,m}{A}$, line-to-line (L-L) resistance of $0.19\,\Omega$, and rated L-L inductance of 0.49 mH. It had an integrated Timken M15 rotary encoder with a resolution of 1000 divisions per revolution.

These pieces of hardware were divided among several different PCB designs. The current sensor, isolator, memory module, and interfacing hardware each had separate PCBs, which can be seen in the Appendix.

3.3. VHDL Implementation

3.3.1. Current Sensor

The FPGA interface with the current sensors was designed based off the AD7822 Standalone Operation section of the AD7822 datasheet. In this mode, the device starts a conversion using a single signal and has only a single feedback signal indicating when the conversion is finished. The way this was implanted in VHDL was to have the conversion start control signal toggle on and off with the internal clock, and store the data in a register when triggered by the end of conversion signal.

3.3.2. Encoder

The encoder outputs consisted of A and B quadrature signals. These two signals both indicate when a magnet passes a position in the encoder. These positions are close to one another, but slightly offset physically. This allows for direction to be determined by which signal is asserted first. This information can be connected to an up/down counter to determine the position of the encoder. One way to extract the direction and enable signals using hardware is to use the following circuit shown in Fig. 3.1. This specific decoder is called a 4x decoder as it counts each transition of the inputs, thus counting 4 times per transition. What this does is

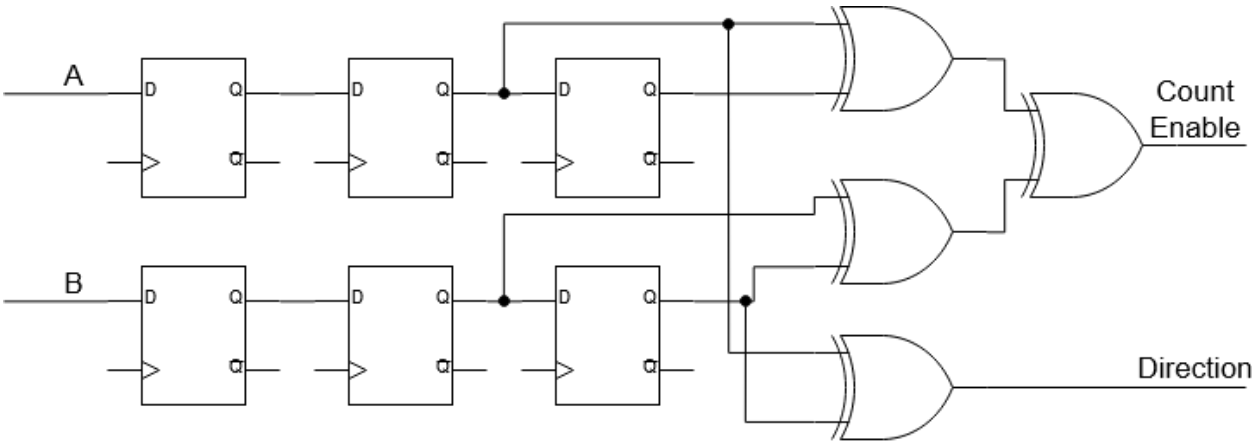

Figure 3.1. Decoder system. A and B are the quadrature inputs. Each D Flip Flop has the same clock signal.

1. check if either A or B have changed each clk cycle, to trigger the count enable signal, and

2. check which changed first, triggering the direction signal if A changed first.

The reason why each quadrature signal has 3 flip flops is to mitigate asynchronicity between the system clock and the quadrature signals.

3.3.3. Reconfiguration Management

To implement DPR in a Xilinx system, the top-level system needs to be implemented with a black box component where the changing system is located. This prevents synthesis in the standard Xilinx tools, limiting preliminary debugging to syntax checks and netlist creation. In addition, a buffer needs to be implemented around this black box to prevent signals from entering or leaving the DPR section during the configuration time. A pair of decouplers was implemented around the motor algorithm black box in the top-level design of the controller, which consisted of a series of registers that were disabled, by outputting all zeroes, during reconfiguration.

In Xilinx systems, partial reconfiguration is controlled by the internal configuration access port (ICAP). Standard FPGA configuration is accomplished by external hardware, typically a microprocessor. The ICAP allows the FPGA to configure itself, as the ICAP is accessible internally by the FPGA fabric. Interfacing with the ICAP is the same as the SelectMAP configuration protocol for full device configuration. Simply put, if the system is told to begin a configuration or that it is currently configuring a system, check if the memory access is complete. If the end of the partial bit file is reached, assert the done signal. If the end of the file is not reached, continue loading

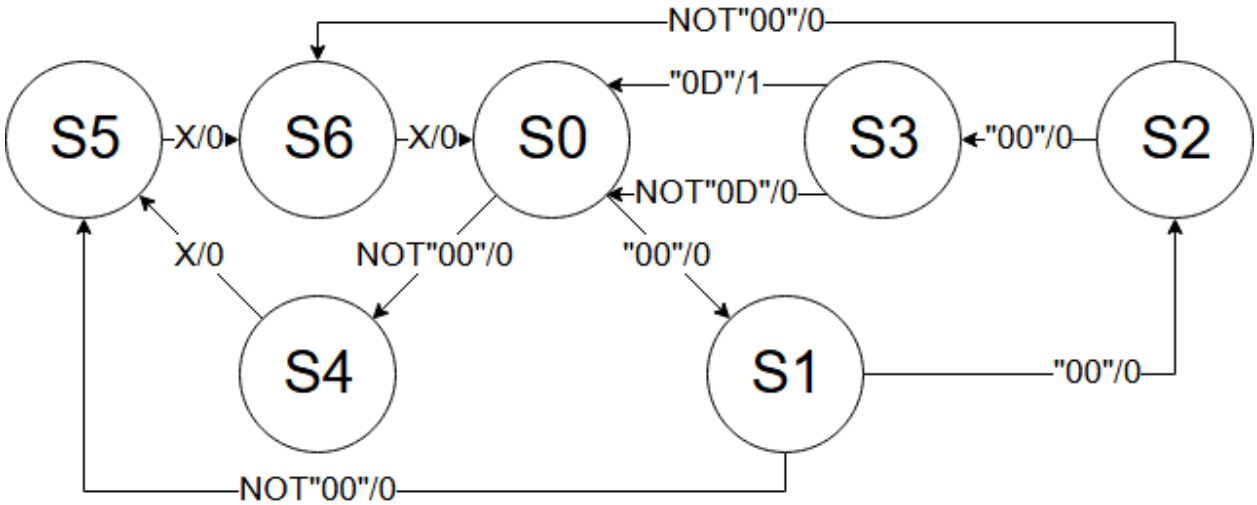

Figure 3.2. Sequence detector Mealy machine to detect "0x0000000D" by byte. The left of the slash is the input byte while the right digit is the output bit. All numbers in quotes are expressed in hexadecimal.

from the bit file and assert that the system is currently configuring. If the system is not currently configuring or being told to start a configuration, don't do anything. One additional thing to note is that the specific FPGA used here required that each byte from the partial bit file needed to be flipped. This was accomplished while reading from the memory.

Part of the partial configuration process is accessing the partial bit files. In this work, the partial bit files were stored on flash memory that used a standard EEPROM interface. To program the memory, the TL866CS Universal Programmer from XGautoelectric was used. To read from this memory was relatively simple: enable the chip and send it an address to read from, then wait until the chip asserts the done signal and store the data in a register. The address began at zero and would increment by one if the end of the file was not reached. Detecting the end of the file, which is noted by the number 0x0000000D, introduced some additional complexity to this system. This was accomplished by using the following Mealy based Finite State Machine (FSM), seen in Fig. 3.2. This FSM was simply a sequence detector with built in delay states so that it would only check every four bytes instead of always looking for the sequence. If the sequence was detected, the end of file flag would be asserted, and the read sequence would stop.

To control when the system would start the partial reconfiguration process, and select the appropriate memory chip to read from, an Algorithmic State Machine (ASM) was created, seen in Fig. 3.3. The output "src_sel" controls which memory chip is enabled, i.e. which partial bit file to

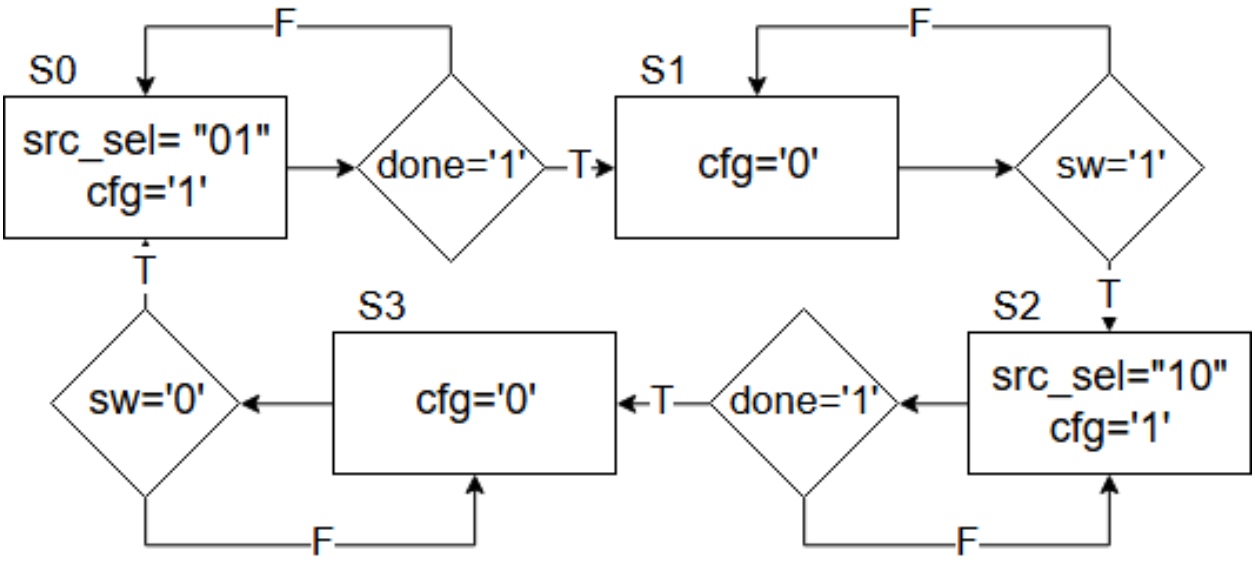

Figure 3.3. Controller selection ASM. Sates 1 and 3 are where configuration happens, while 2 and 4 are waiting for the change command.

access, and the "cfg" output indicates when to initiate a reconfiguration. The "sw" input controls which configuration is active at a specific time, but it cannot interrupt an ongoing configuration. The "done" input indicates when the reconfiguration is completed.

3.3.4. PWM Generator

The PWM generator function has already been discussed, but one piece of its function was not mentioned. The generator requires a triangle wave to compare incoming signals with. This triangle wave generator was implemented as an up/down counter with an upper limit that could be specified. This counter would count to the specified limit, for this work it was set to 50, count back to 0, and then repeat. This provided the carrier wave to generate the PWM.

3.3.5. Clock Management

These systems require clock management. The master clock is provided by the onboard 100 MHz clock on the Xilinx ML501 board. This clock is divided into 3 clocks: the ICAP clock, the buffered clock, and the controller clock. The ICAP clock is created by passing the master clock through a digital clock management (DCM) IP core set to divide the clock speed by 2, providing a 50 MHz clock that is used to run the sensors and reconfiguration controller. The buffered clock is the same frequency as the master clock, but taken from the frequency doubling output port of the DCM IP core. The controller clock is generated from the buffered clock, and generates a

20 kHz clock using a counter, which is then used for the motor controller and PWM generator. The controller clock needed to be generated from the buffered clock signal due to clock domain issues inside of the Xilinx PlanAhead synthesis tool.

3.3.6. Dynamic Partial Reconfiguration

When implementing a DPR system, the type of DPR must first be decided. There are two possible implementations of DPR in Xilinx systems: difference-based and module-based. In difference-based implementations, the partial bit file describes only the places where there are differences between the original configuration and the new one. This is typically used only when small changes are needed in a configuration. In module-based implementations, the partial bit file is a full description of a piece of the FPGA and is typically used when changing large pieces of a design. A module-based implementation of DPR was used in this work to change which motor controller was active. This required that the top-level HDL design be created separate from the controllers. In the top-level design, the controller was left as a black box with only inputs and outputs listed. The controllers were implemented, and the netlists generated separate from the top-level design, ensuring that both controllers had the same inputs and outputs even if extraneous ones were needed. Once the designs were complete, PlanAhead was used to allocate FPGA resources and impose clock conditions for final bit-file generation. Within PlanAhead, a black-box implementation of a module in a top-level design can be designated as a reconfigurable partition. The software then allows multiple netlists to be assigned to the black-box, in this case the FOC controller, the V/f controller, and an empty implementation. This empty implementation was used to allow for faster initial configuration and provide a known starting point for the reconfiguration controller. The software uses these implementations to provide the user with the minimum hardware requirements, for example look-up-tables (LUTs), for the reconfigurable partition. This partition is designated by the user, with the software indicating if the resource requirement has been met. The initial configuration is synthesized by the user, in this case the empty partition. The other configurations are then synthesized while associated with the initial configuration, allowing the new configurations to use the same static configuration. After this, the different synthesis results are compared against one another to ensure that the static portions of each are identical. Once completed, the bit-streams for each configuration are generated. This provides a full configuration image for each one,

i.e. empty, FOC, and V/f, and provides the partial bit-files for DPR as well. After this, the system is ready to be placed on an FPGA and tested in real life.

4. SIMULATION AND HARDWARE TESTING RESULTS

The physical experimental design was intended to mimic the simulation test bench as closely as possible. Both were designed with the motor being driven at a constant speed while controlling the torque. In the simulation, this was accomplished by connecting the motor block with a constant speed input. In the physical simulation, a DC motor was connected to the PMSM, with the DC motor being fed inputs intended to drive it at a constant speed. To drive the DC motor and to monitor the real time speed and torque characteristics of the system, a dSPACE DS1104 controller was used. Design of the dSPACE interface was accomplished through MATLAB Simulink, where a simple DC speed driver was implemented along with a system that would read the current, voltage, and speed of the DC motor to display each and to calculate torque using Eq. 4.1, where τ is torque in N m, P is power in W, and v is rotational speed in rpm.

$$\tau = \frac{60P}{2\pi v} \tag{4.1}$$

To get an accurate representation of the behavior of the switching controller, the system was tested at three speeds: 100 rpm, 500 rpm, and 950 rpm. Tests could have been done at faster speeds, but given the maximum rated speed of 4000 rpm and the potential for unknown behavior during transitions, a large margin of error was observed. Each speed was evaluated at a transition between the controllers to ensure behavior that matched the simulated results. The transitions from FOC to V/f at 100, 500, and 950 rpm can be seen in Figs. 4.1, 4.2, and 4.3, and for torque in Figs. 4.4, 4.5, and 4.6. For the transitions from V/f to FOC, the speed can be seen in Figs. 4.7, 4.8, and 4.9 and the torque in Figs. 4.10, 4.11, and 4.12.

These can be compared to the simulation behavior, which can be seen in Figs. 4.13-4.21. One major difference between the two is the noise level of the physical simulation. The main source of the noise was the DC motor being driven at a constant speed, as the actual effect of the driver was not a constant speed. The DC motor had oscillations in speed around the desired speed, so the measurements were taken with the average speed being the constant driver. This made it difficult to see the transition on the lower speed setting in Figs. 4.1, 4.4, 4.7, and 4.10, as the varying speed

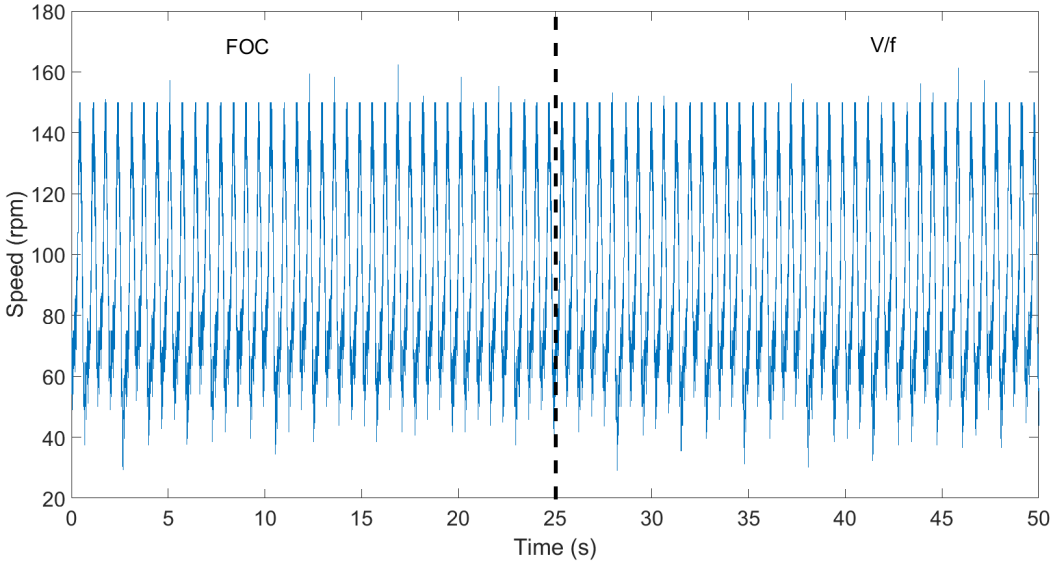

Figure 4.1. The speed curve of the transition between FOC and V/f at 100 rpm.

caused any transition effect to be drowned out by the oscillations. The results of the transition are more clearly seen at higher speeds. In most of these results, a drastic change follows very shortly after the transition was initiated, which indicates where the reconfiguration process finished, and the new controller takes over.

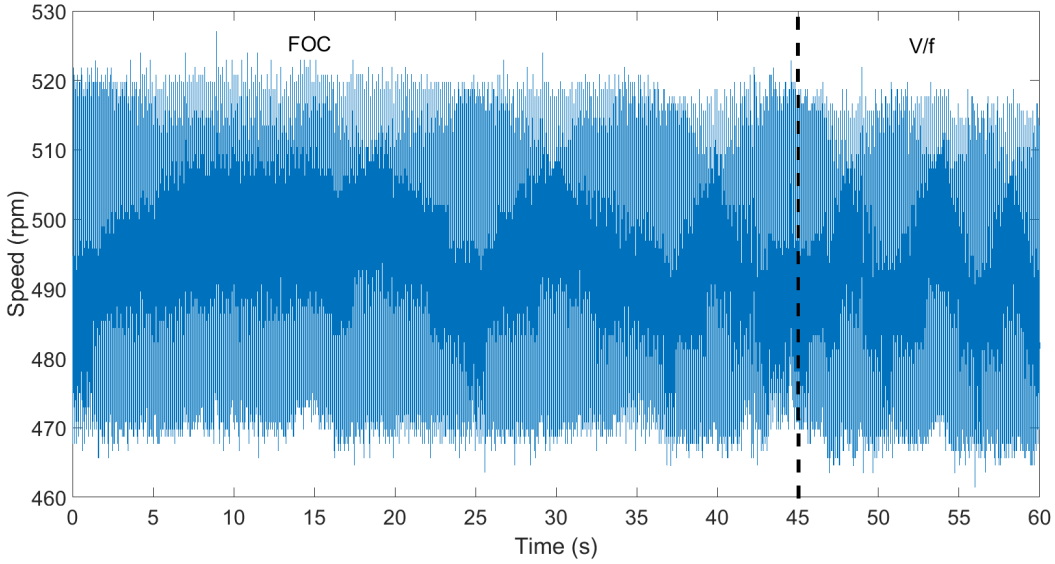

Figure 4.2. The speed curve of the transition between FOC and V/f at 500 rpm.

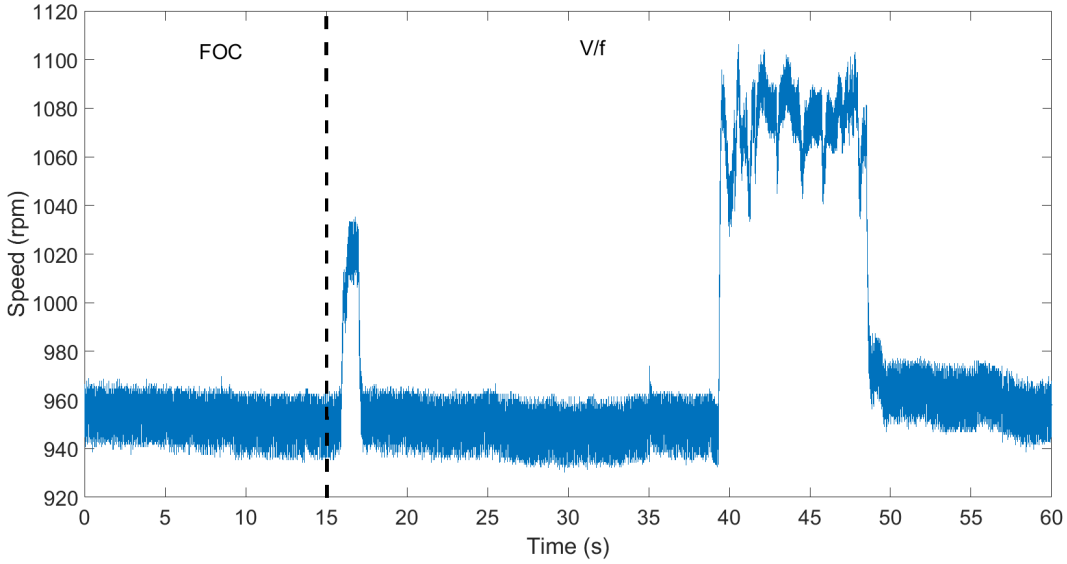

Figure 4.3. The speed curve of the transition between FOC and V/f at 950 rpm.

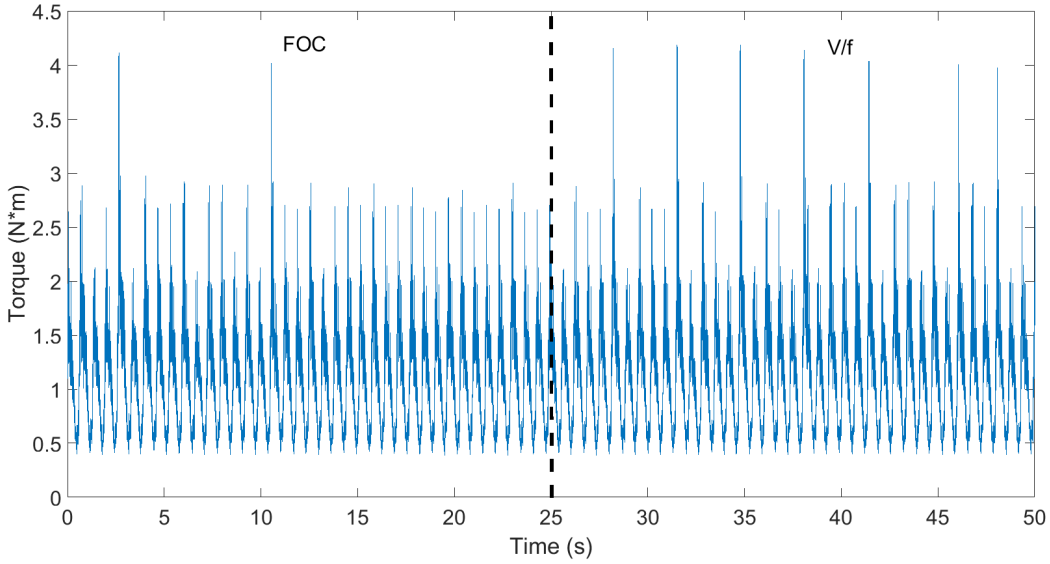

Figure 4.4. The torque curve of the transition between FOC and V/f at 100 rpm

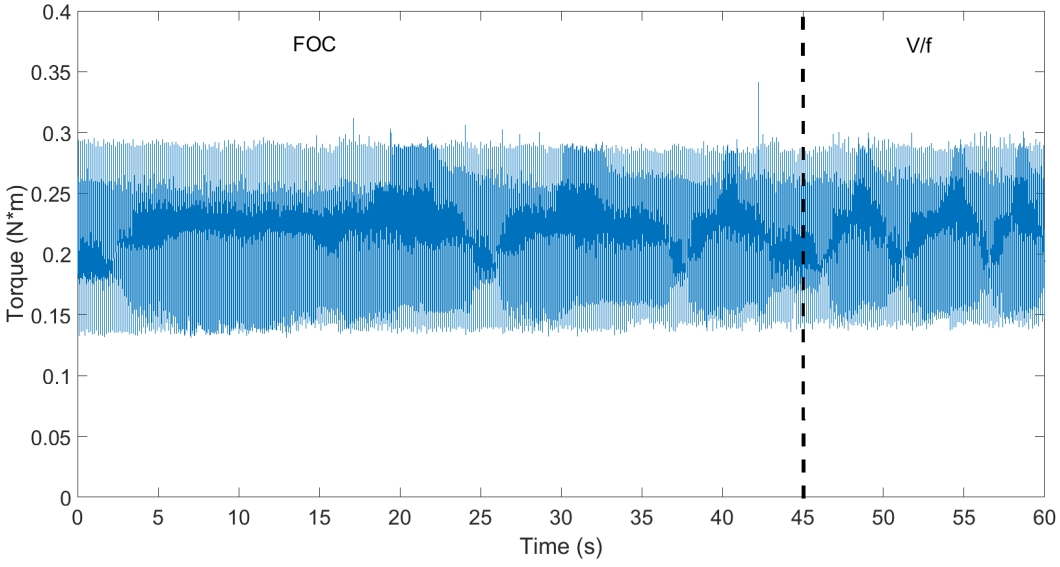

Figure 4.5. The torque curve of the transition between FOC and V/f at 500 rpm

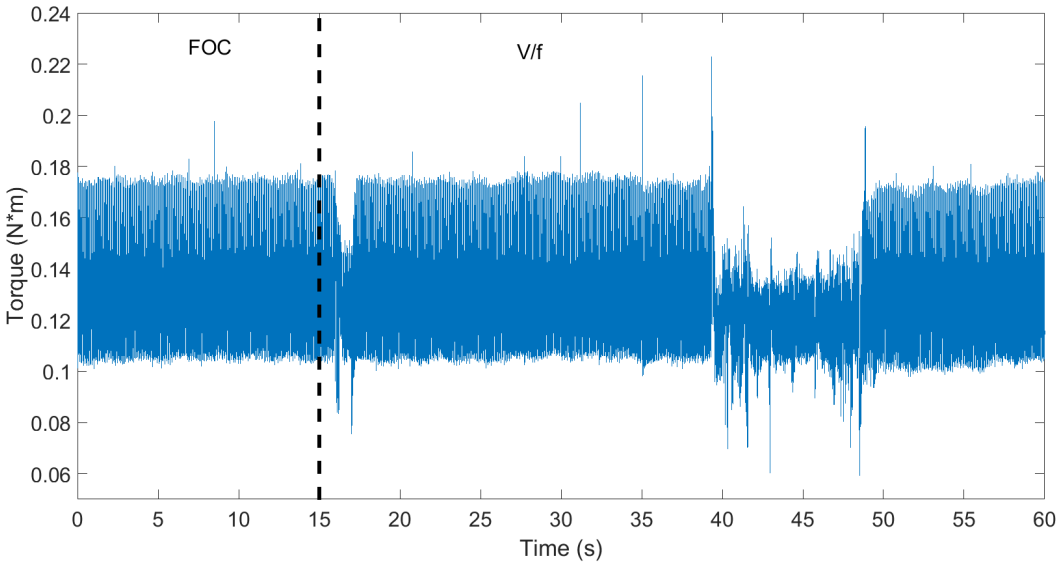

Figure 4.6. The torque curve of the transition between FOC and V/f at 950 rpm

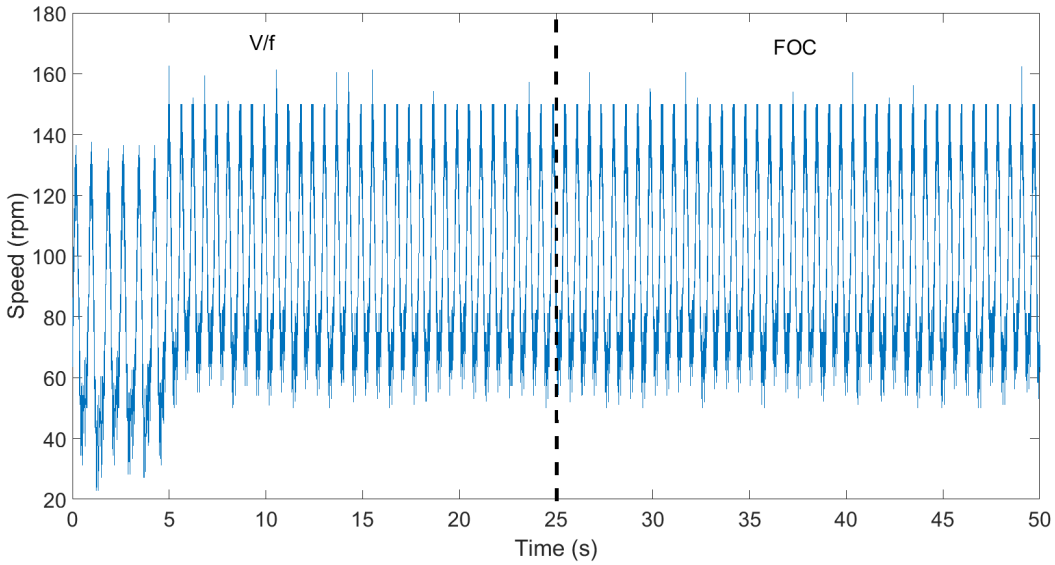

Figure 4.7. The speed curve of the transition between V/f and FOC at 100 rpm.

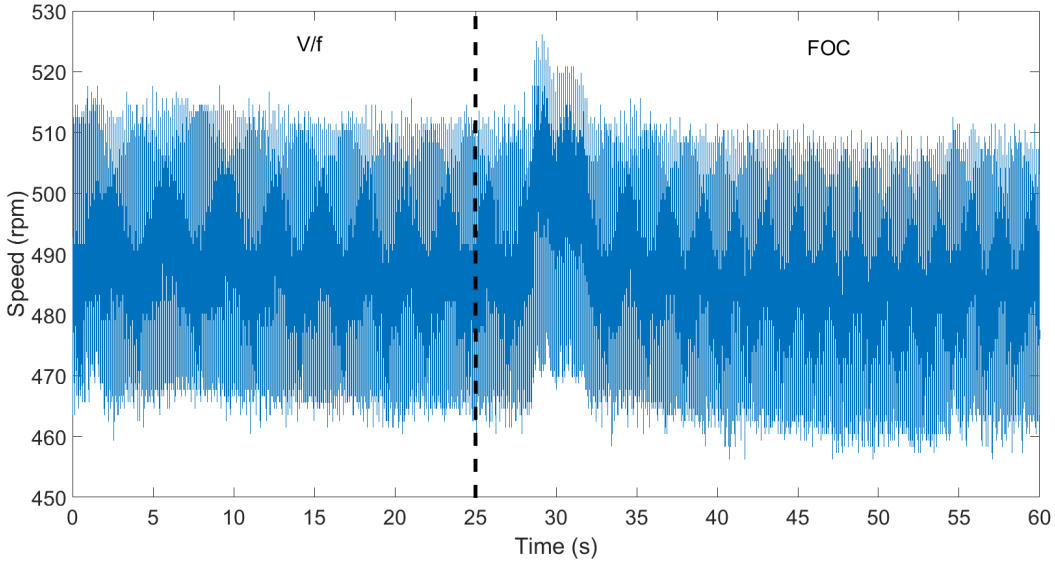

Figure 4.8. The speed curve of the transition between V/f and FOC at 500 rpm.

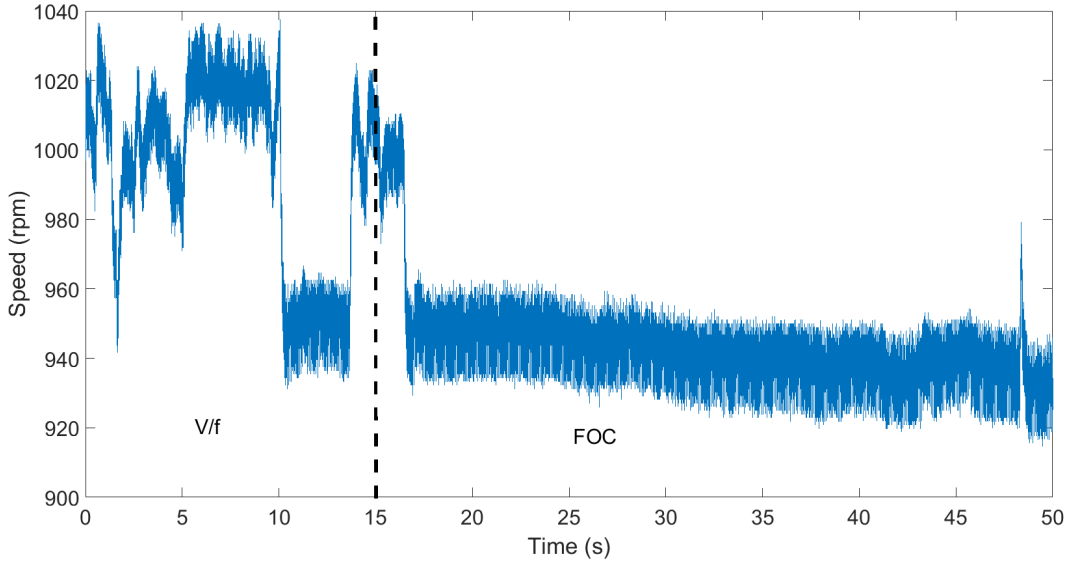

Figure 4.9. The speed curve of the transition between V/f and FOC at 950 rpm.

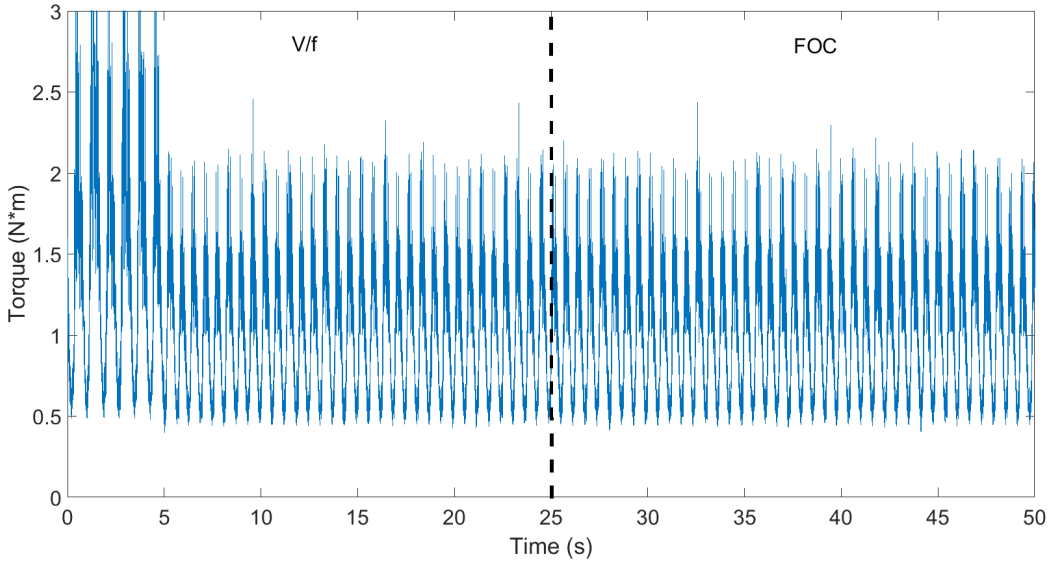

Figure 4.10. The torque curve of the transition between V/f and FOC at 100 rpm.

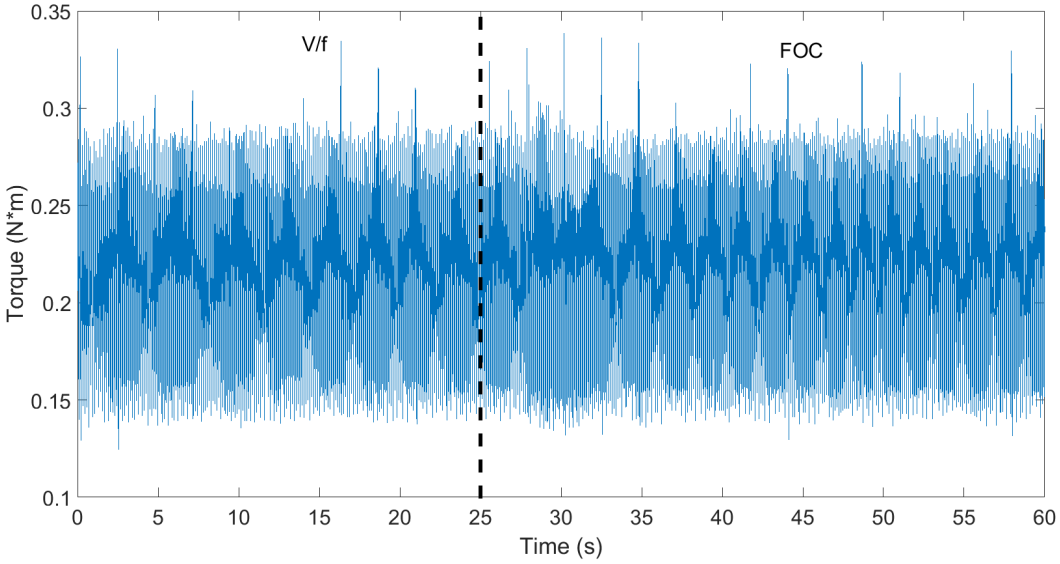

Figure 4.11. The torque curve of the transition between V/f and FOC at 500 rpm.

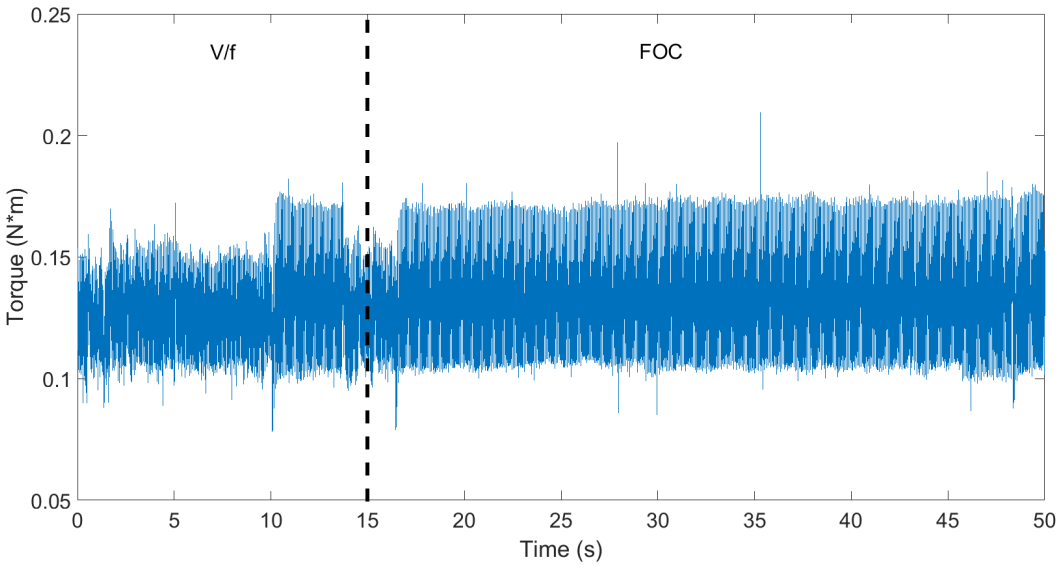

Figure 4.12. The torque curve of the transition between V/f and FOC at 950 rpm.

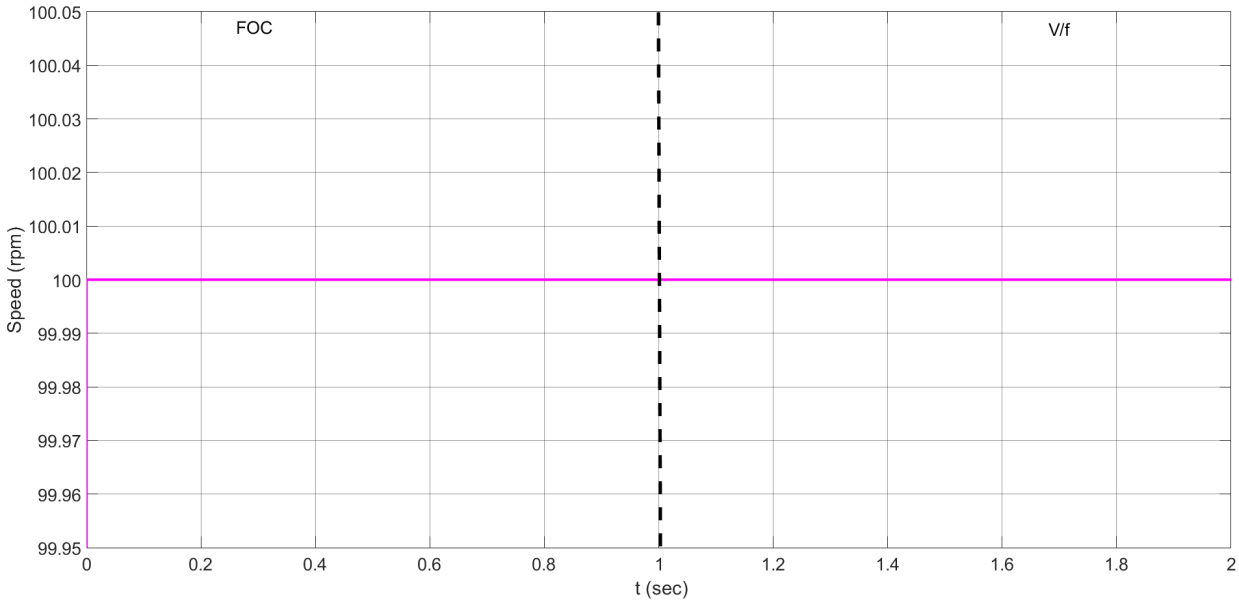

Figure 4.13. The simulated speed curve of the transition between FOC and V/f at 100 rpm.

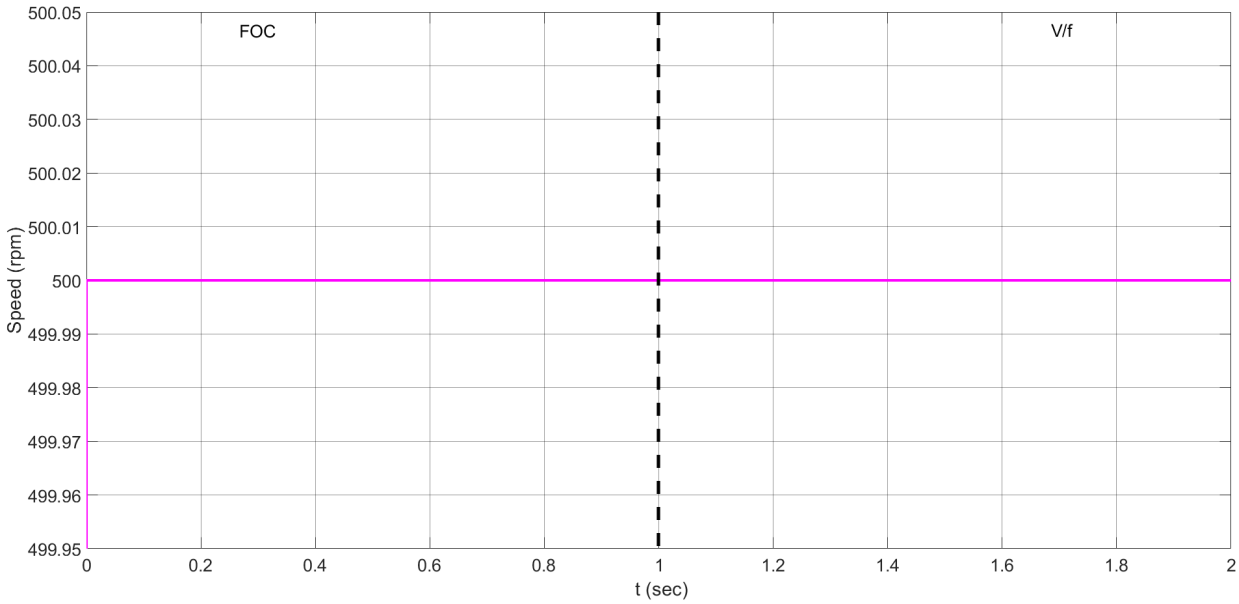

Figure 4.14. The simulated speed curve of the transition between FOC and V/f at 500 rpm.

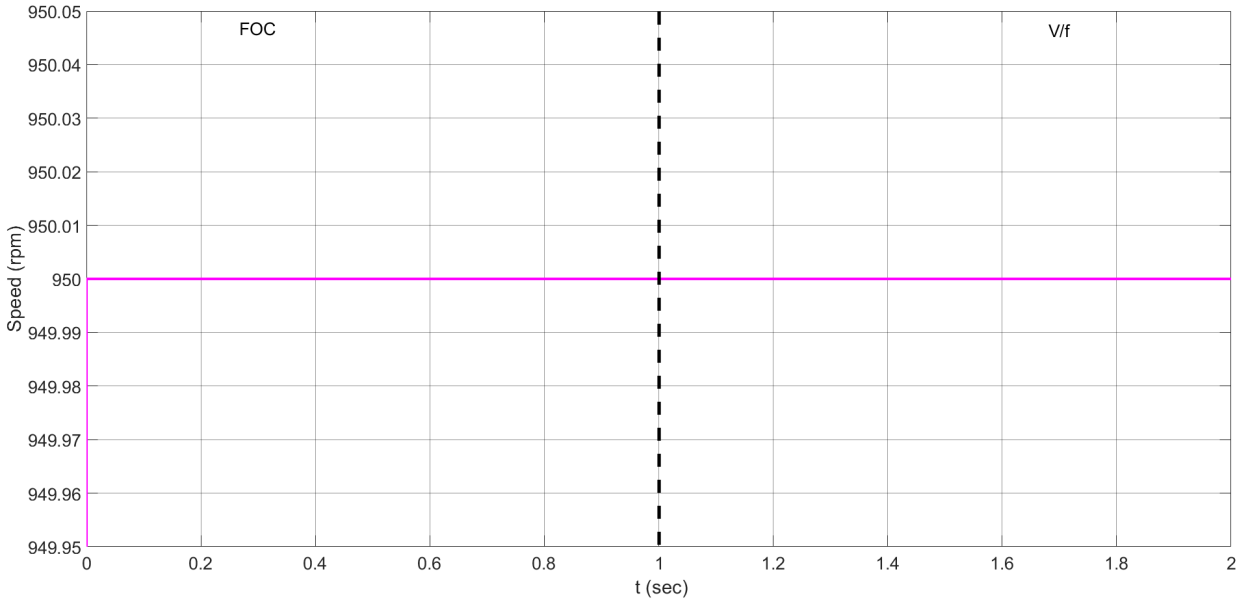

Figure 4.15. The simulated speed curve of the transition between FOC and V/f at 950 rpm.

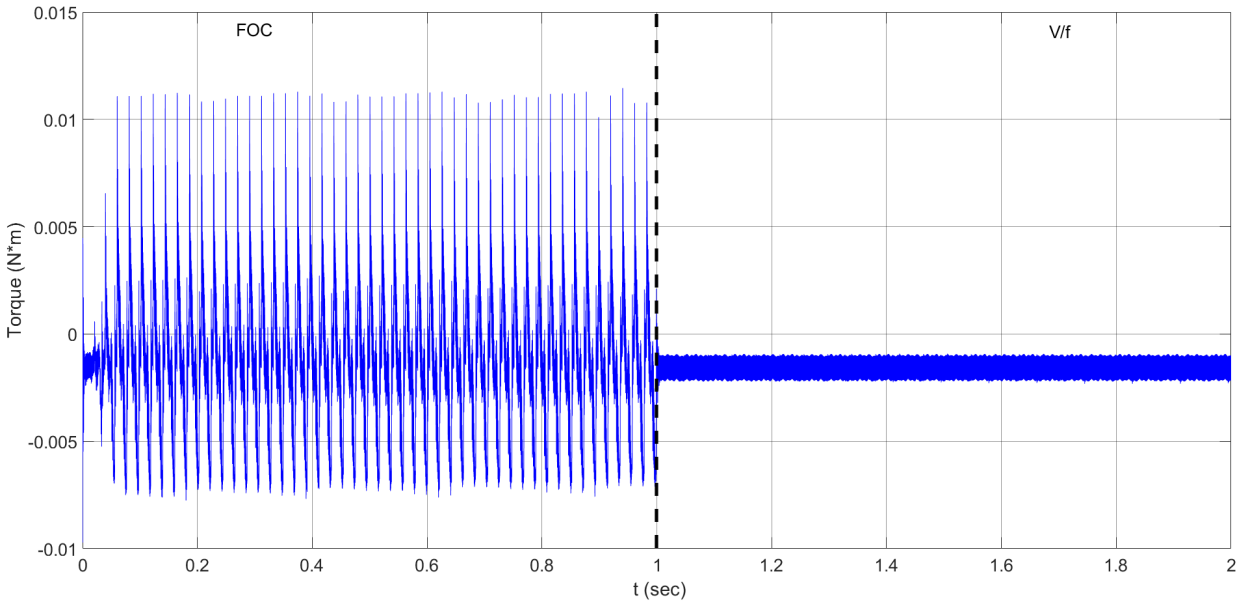

Figure 4.16. The simulated torque curve of the transition between FOC and V/f at 100 rpm

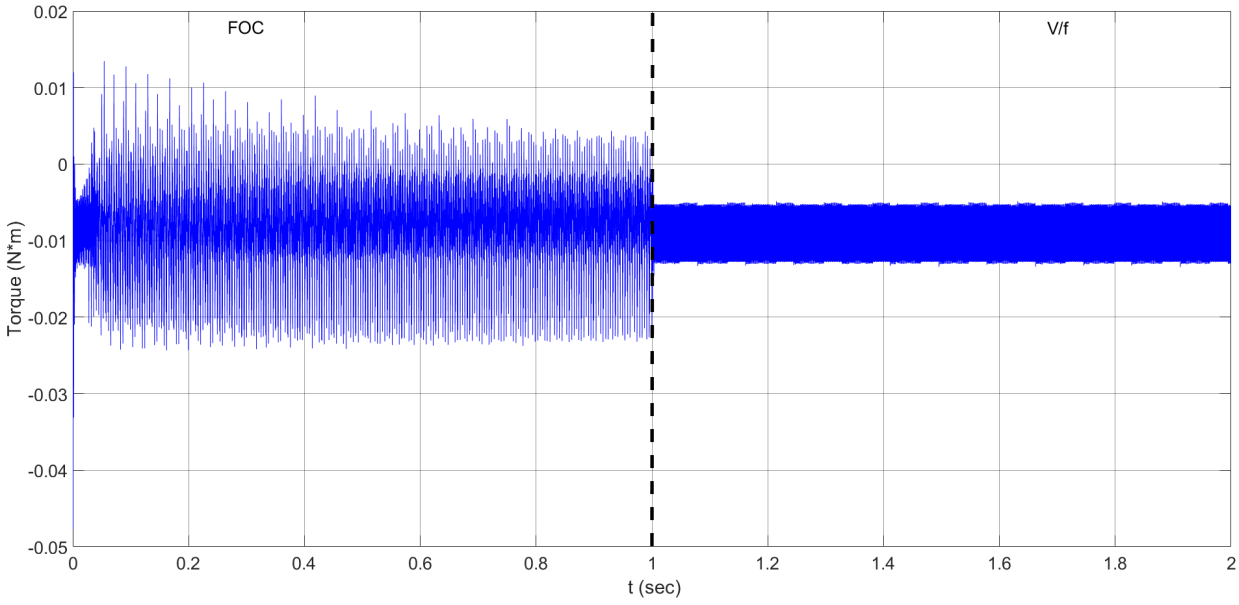

Figure 4.17. The simulated torque curve of the transition between FOC and V/f at 500 rpm

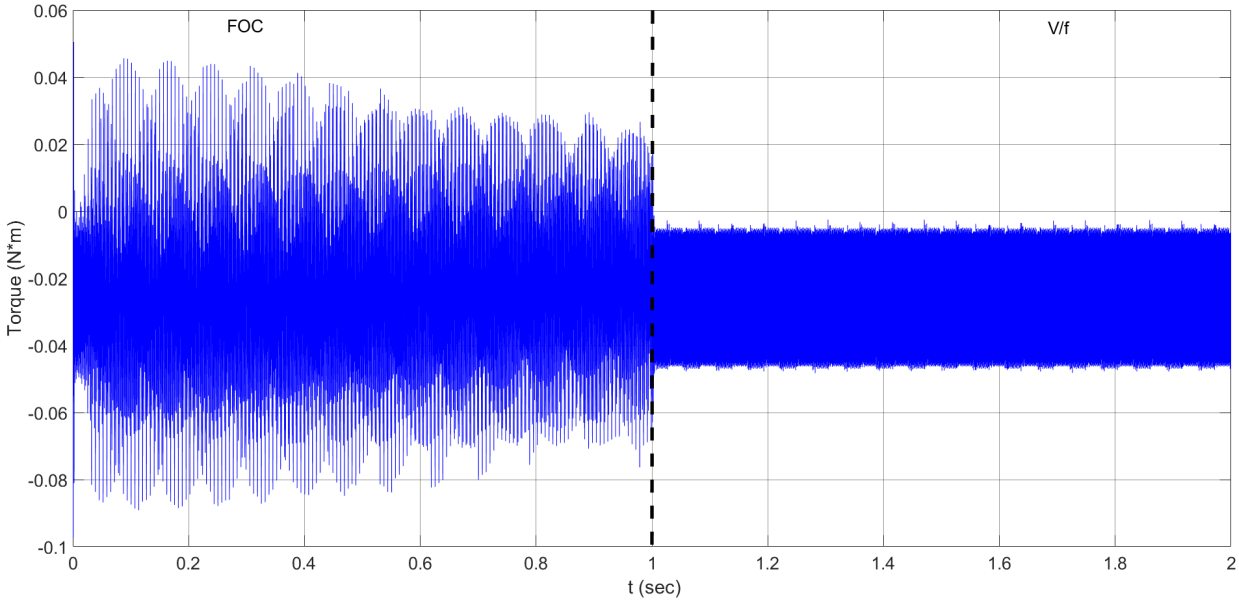

Figure 4.18. The simulated torque curve of the transition between FOC and V/f at 950 rpm

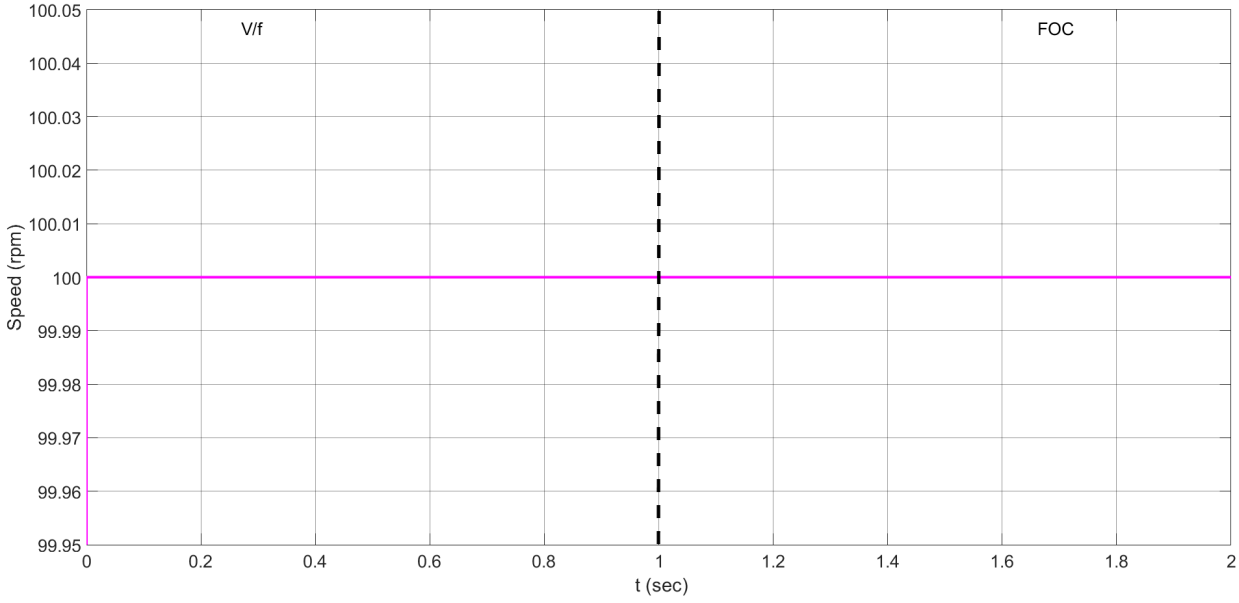

Figure 4.19. The simulated speed curve of the transition between V/f and FOC at 100 rpm.

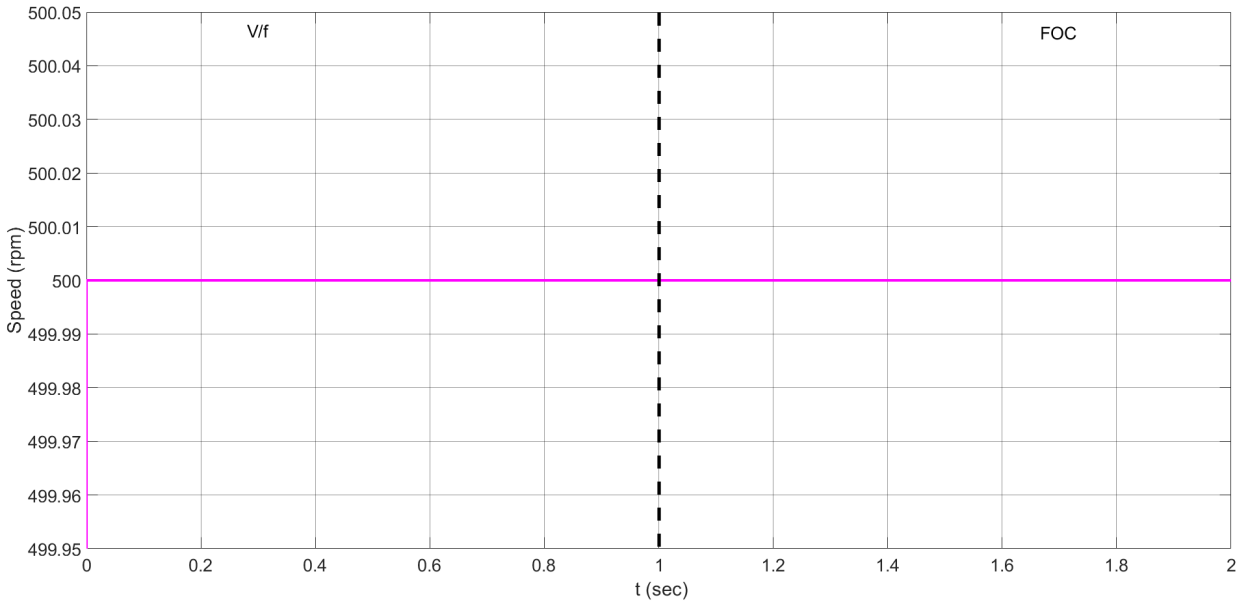

Figure 4.20. The simulated speed curve of the transition between V/f and FOC at 500 rpm.

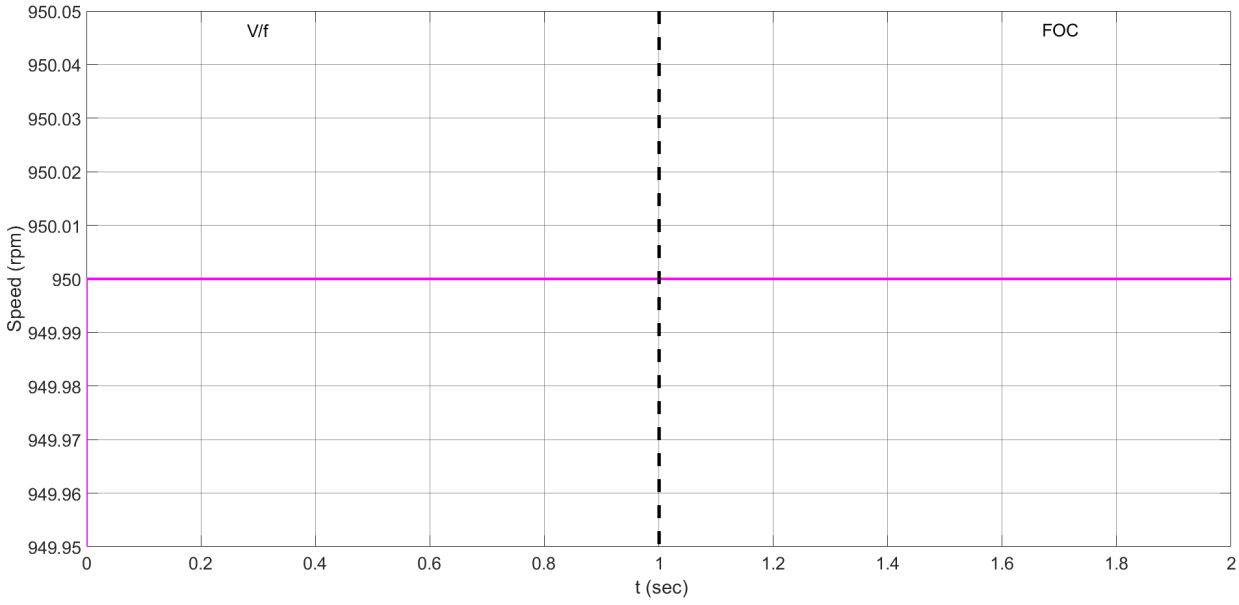

Figure 4.21. The simulated speed curve of the transition between V/f and FOC at 950 rpm.

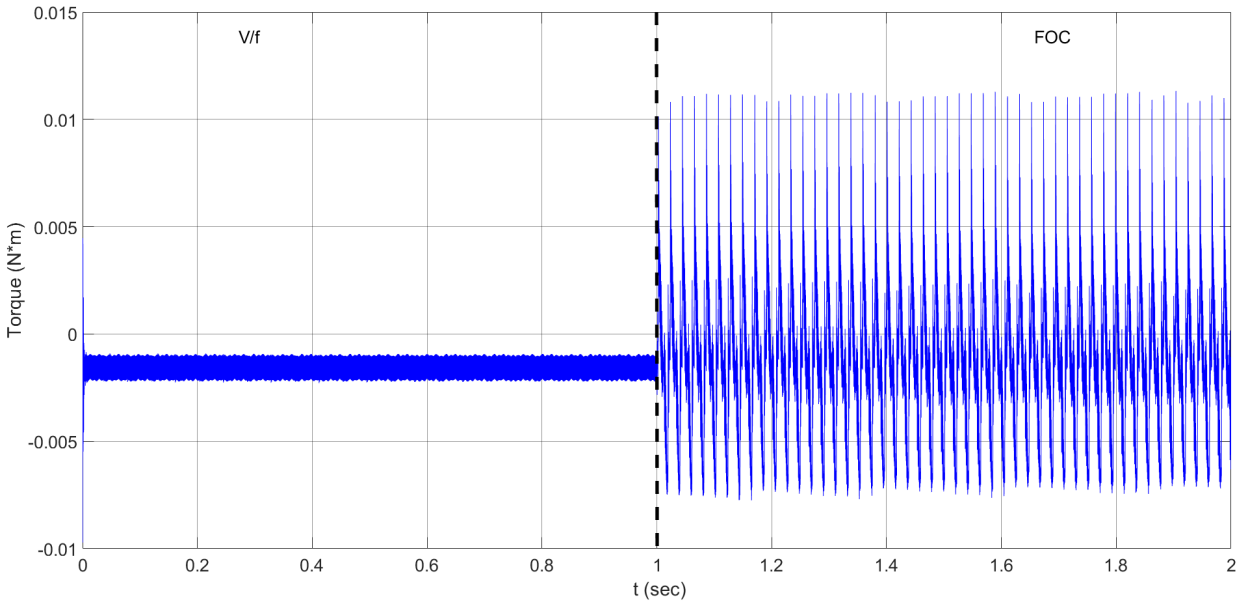

Figure 4.22. The simulated torque curve of the transition between V/f and FOC at 100 rpm.

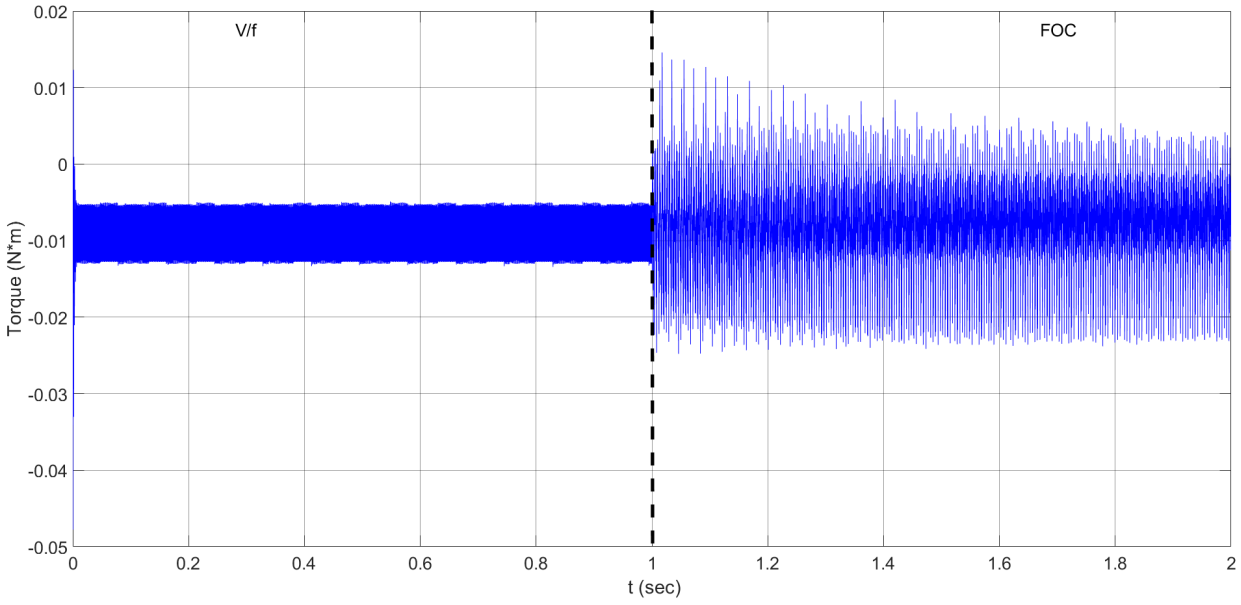

Figure 4.23. The simulated torque curve of the transition between V/f and FOC at 500 rpm.

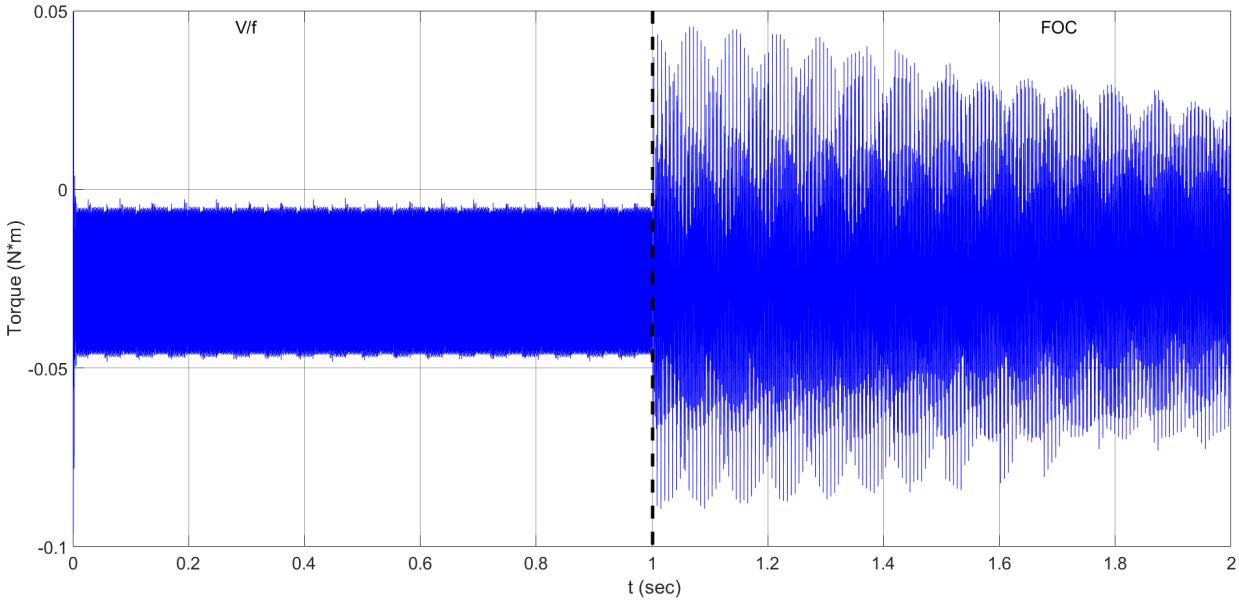

Figure 4.24. The simulated torque curve of the transition between V/f and FOC at 950 rpm.

5. CONCLUSION

Much of the motor controls research focusing on FPGA uses them as another form of DSP or microcontroller to implement their work. In addition, much of the work on motor controls in general is in making large and complex motor controllers that work well for many different conditions. This work proposed utilizing DPR to improve upon both approaches. DPR is a unique feature to FPGAs that allows a piece of their fabric to be reconfigured while the FPGA continues to run. Regarding motor controls, there is potential for DPR to allow simpler, more focused controllers to be developed that work well in different conditions. For example, a simple controller may be designed to work well for a motor experiencing dynamic torque conditions, indicating perhaps a slippery city road, which could be switched out using DPR for another simple controller that works well under constant torque conditions, such as a clean highway. This work showed that DPR can provide a stable transition between the FOC and V/f controllers, and implies that this will carry over to other transitions between different controllers.

5.1. Future Work

One major area of exploration this work suggests is the study of simple motor controller performance under different conditions. If looking at a car motor, for example, perhaps there is a controller that works best under conditions like the city and a different one that works best on a highway. Additionally, the conditions under which the controllers would switch should be considered, such as what is being monitored to trigger controller switching. Even though this work implies that smooth transitions will occur between other controllers, this area still needs confirmation. Additionally, this work did not focus on performance of the individual controllers, so improvement of the FOC and V/f controllers could also be an area of future research.

REFERENCES

[1] S. Morimoto, "Trend of permanent magnet synchronous machines," *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 2, pp. 101–108, 2007.

[2] R. Nukki, A. Kilk, A. Kallaste, T. Vaimann, and K. T. Sr, "Exterior-rotor permanent magnet synchronous machine with toroidal windings for unmanned aerial vehicles," in *Electric Power Quality and Supply Reliability Conference (PQ), 2014*, June 2014, pp. 215–220.

[3] G. Friedrich, "Experimental comparison between wound rotor and permanent magnet synchronous machine for integrated starter generator applications," in *2010 IEEE Energy Conversion Congress and Exposition*, Sept 2010, pp. 1731–1736.

[4] M. Bash, S. Pekarek, S. Sudhoff, J. Whitmore, and M. Frantzen, "A comparison of permanent magnet and wound rotor synchronous machines for portable power generation," in *Power and Energy Conference at Illinois (PECI), 2010*, Feb 2010, pp. 1–6.

[5] B. K. Bose, *Power Electronics and Variable Frequency Drives: Technology and Applications.* John Wiley & Sons, Inc., 1996, ch. Variable Frequency Permanent Magnet AC Machine Drives, pp. 277–331. [Online]. Available: http://dx.doi.org/10.1002/9780470547113.ch6

[6] R. Krishnan, *Permanent Magnet Synchronous and Brushless DC Motor Drives.* CRC Press, 2009.

[7] R. M. Crowder, *Electric Drives and Electromechanical Systems*, 1st ed. Elsevier/Butterworth-Heinemann, 2006.

[8] E. Monmasson and M. Cirstea, "Guest editorial special section on industrial control applications of FPGAs," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1250–1252, Aug 2013.

[9] A. Darba, F. De Belie, P. D'haese, and J. Melkebeek, "Improved dynamic behavior in BLDC drives using model predictive speed and current control," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 2, pp. 728–740, Feb 2016.

[10] P. Zhang, A. Mills, J. Zambreno, and P. H. Jones, "A software configurable and parallelized co-processor architecture for LQR control," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2015, pp. 1–8.

[11] B. R. Mutlu and M. Dolen, "Implementations of state-space controllers using field programmable gate arrays," in *SPEEDAM 2010*, June 2010, pp. 1436–1441.

[12] P. Rogers, R. Kavasseri, and S. C. Smith, "An fpga-based design for joint control and monitoring of permanent magnet synchronous motors," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Nov 2016, pp. 1–6.

[13] M. A. Zare, "FPGA-based simulation and implementation of induction motor torque control systems based on direct torque control (DTC)," Master's thesis, North Dakota State University, Jun. 2014.

[14] E. Maurelia, J. R. Espinoza, C. A. Silva, C. A. Rojas, P. E. Melín, and E. E. Espinosa, "An operating condition-based scheme to alternate between control strategies for improved steady-state and transient behavior," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 6, pp. 1246–1254, Dec 2015.

[15] T. Vyncke, S. Thielemans, and J. Melkebeek, "Finite-set model-based predictive control for flying-capacitor converters: Cost function design and efficient FPGA implementation," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 2, pp. 1113–1121, May 2013.

[16] J. Holtz and X. Qi, "Optimal control of medium-voltage drives - an overview," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 12, pp. 5472–5481, Dec 2013.

[17] R. K. Pongiannan, S. Paramasivam, and N. Yadaiah, "Dynamically reconfigurable PWM controller for three-phase voltage-source inverters," *IEEE Transactions on Power Electronics*, vol. 26, no. 6, pp. 1790–1799, June 2011.

[18] E. E. Garibay, D. T. Lucio, and S. Weber, "An approach: FPGA based dynamically reconfigurable architecture to enable several scheme controls for power converters," in *2012 9th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, Sept 2012, pp. 1–6.

[19] B. Akin and N. Garg, "Scalar (V/f) control of 3-phase induction motors," Texas Instruments, Application Report SPRABQ8, Jul. 2013.

APPENDIX. RESOURCES

For the VHDL and MATLAB code used in this work, access the following repository: https://github.com/TheOnceFreeMan/dpr-thesis.

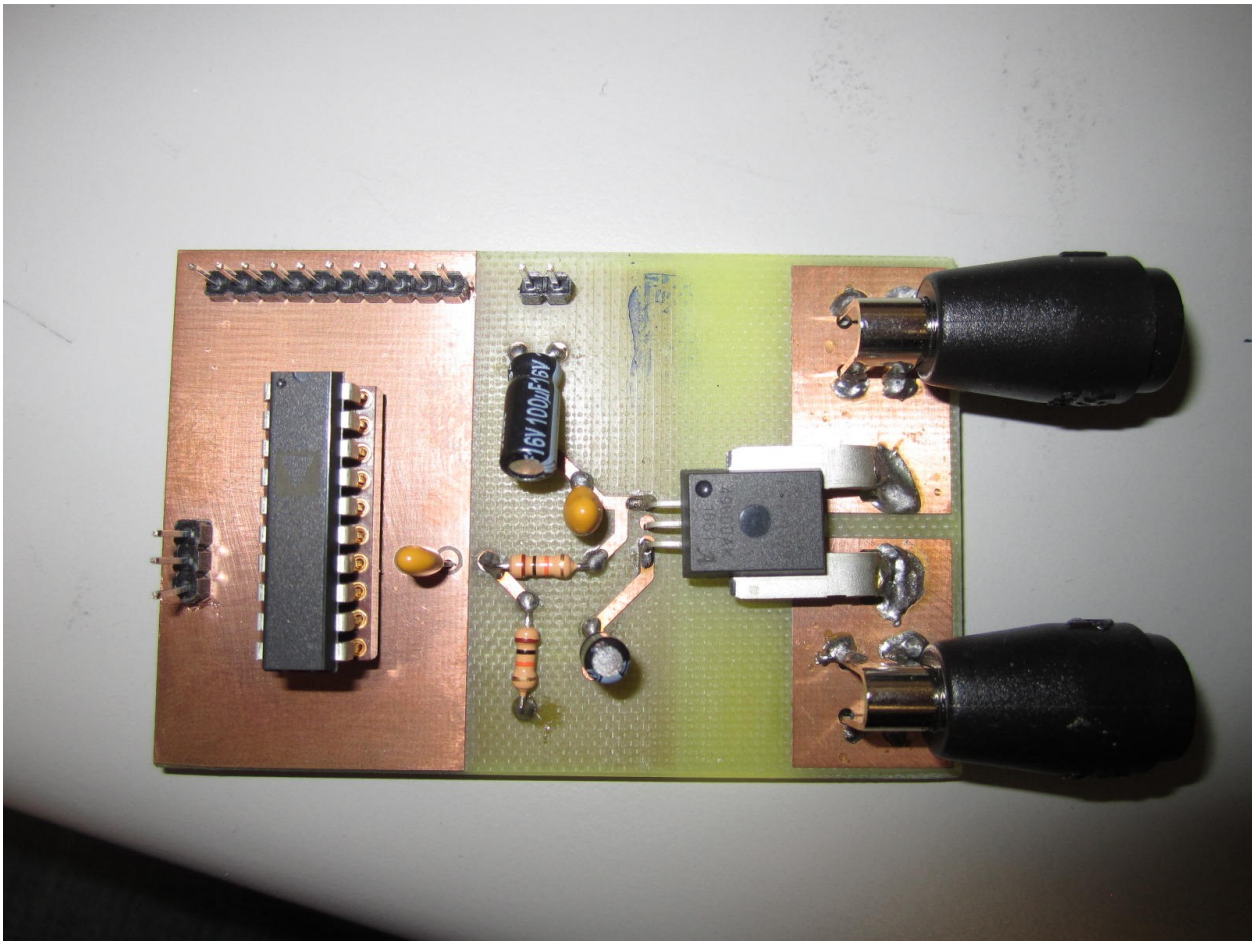

Figure A.1. Front side of the current sensor PCB.

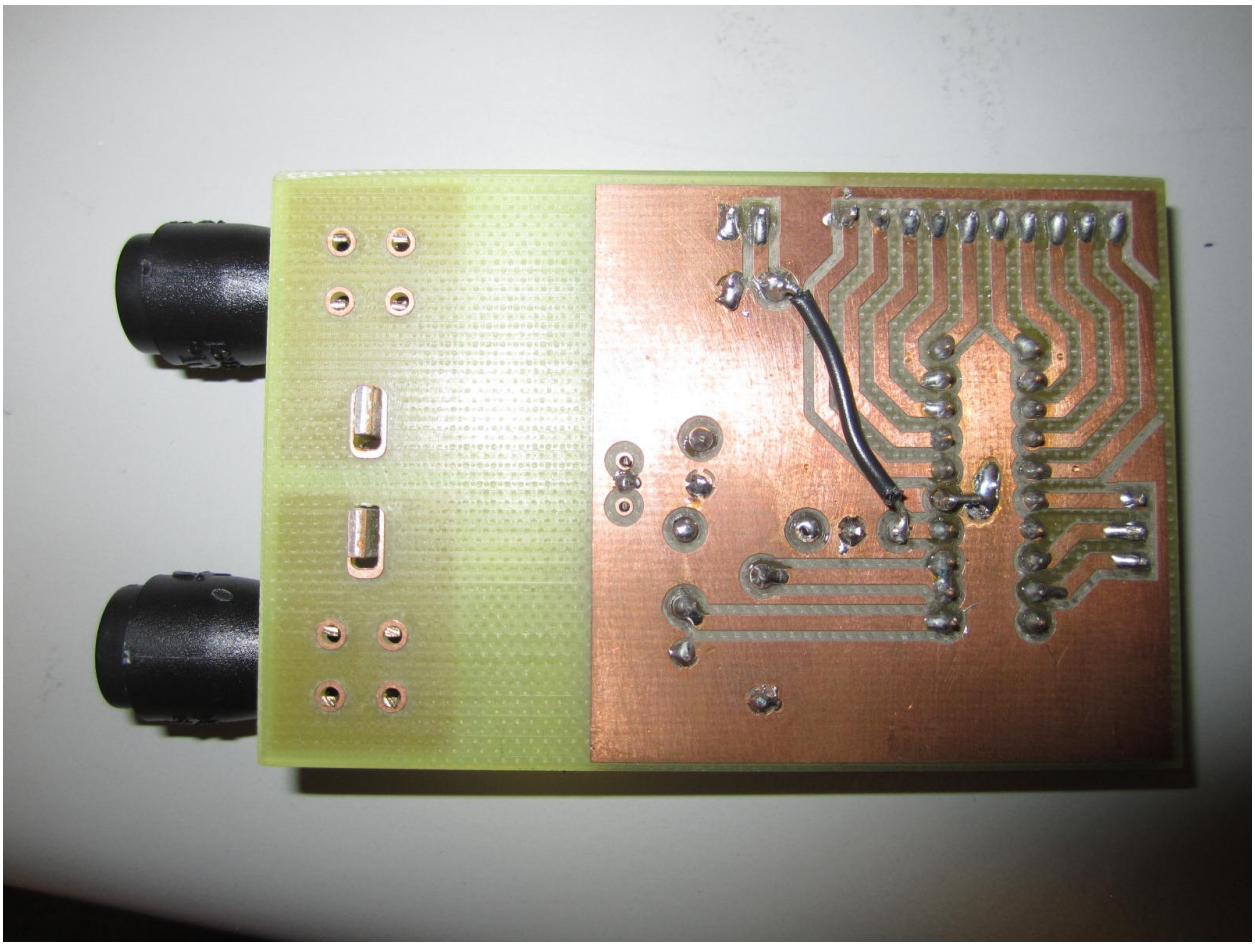

Figure A.2. Reverse side of the current sensor PCB.

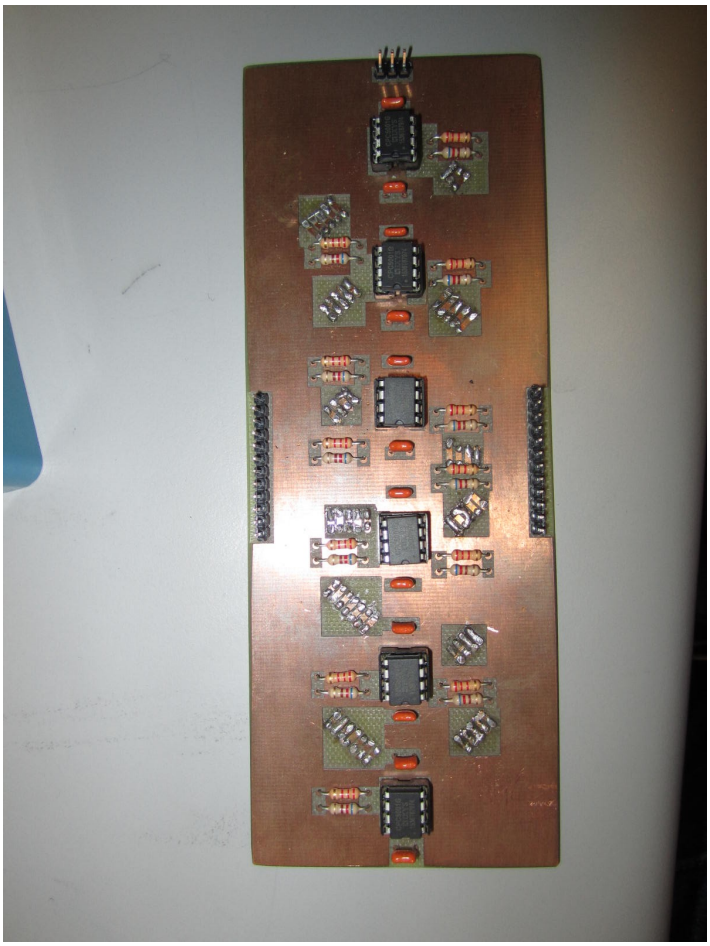

Figure A.3. Front side of the isolator PCB.

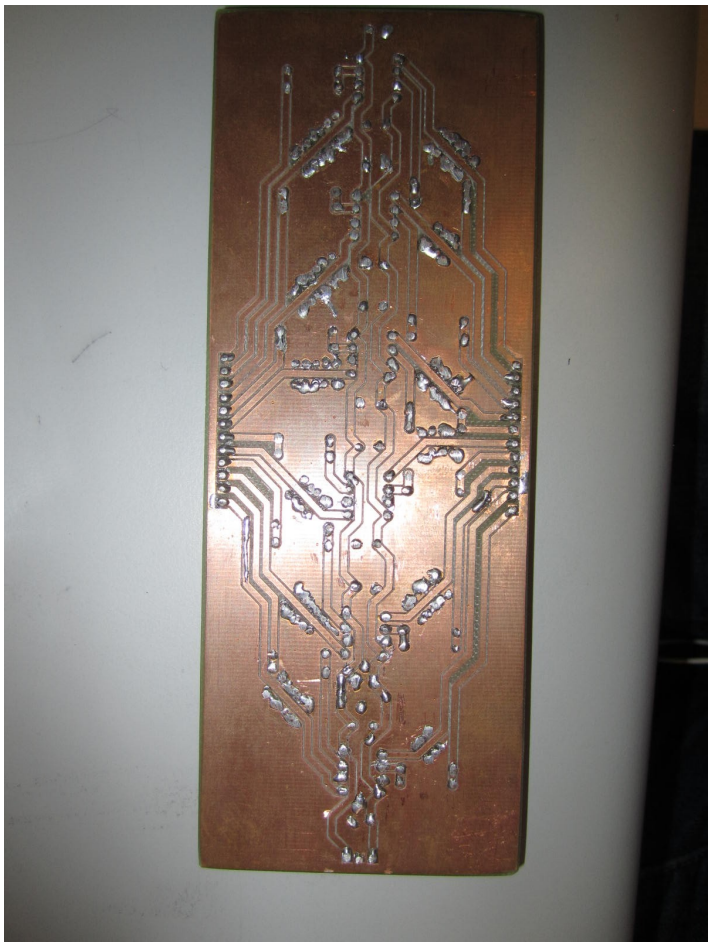

Figure A.4. Reverse side of the isolator PCB.

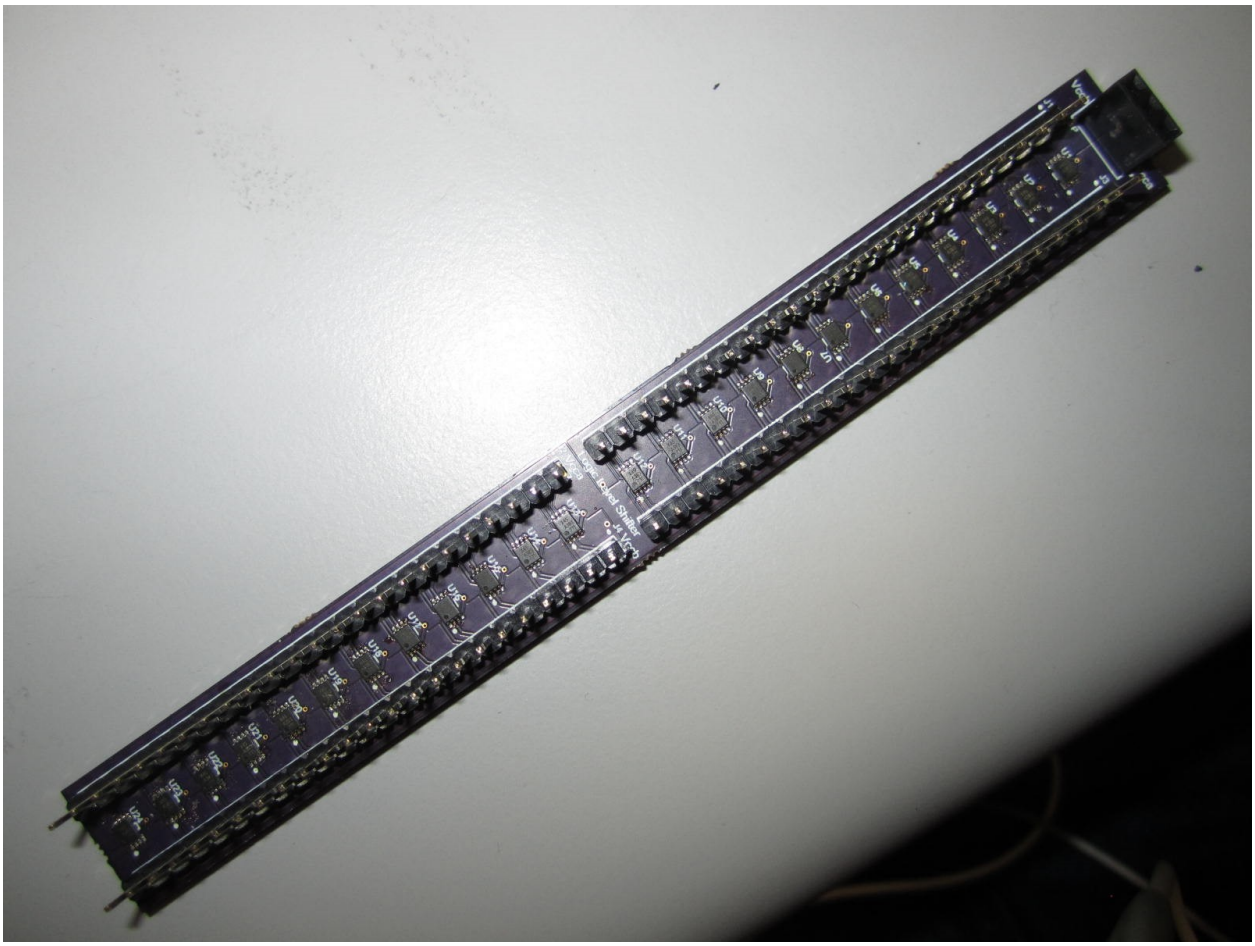

Figure A.5. Front side of the level shifter PCB.

Figure A.6. Reverse side of the level shifter PCB.

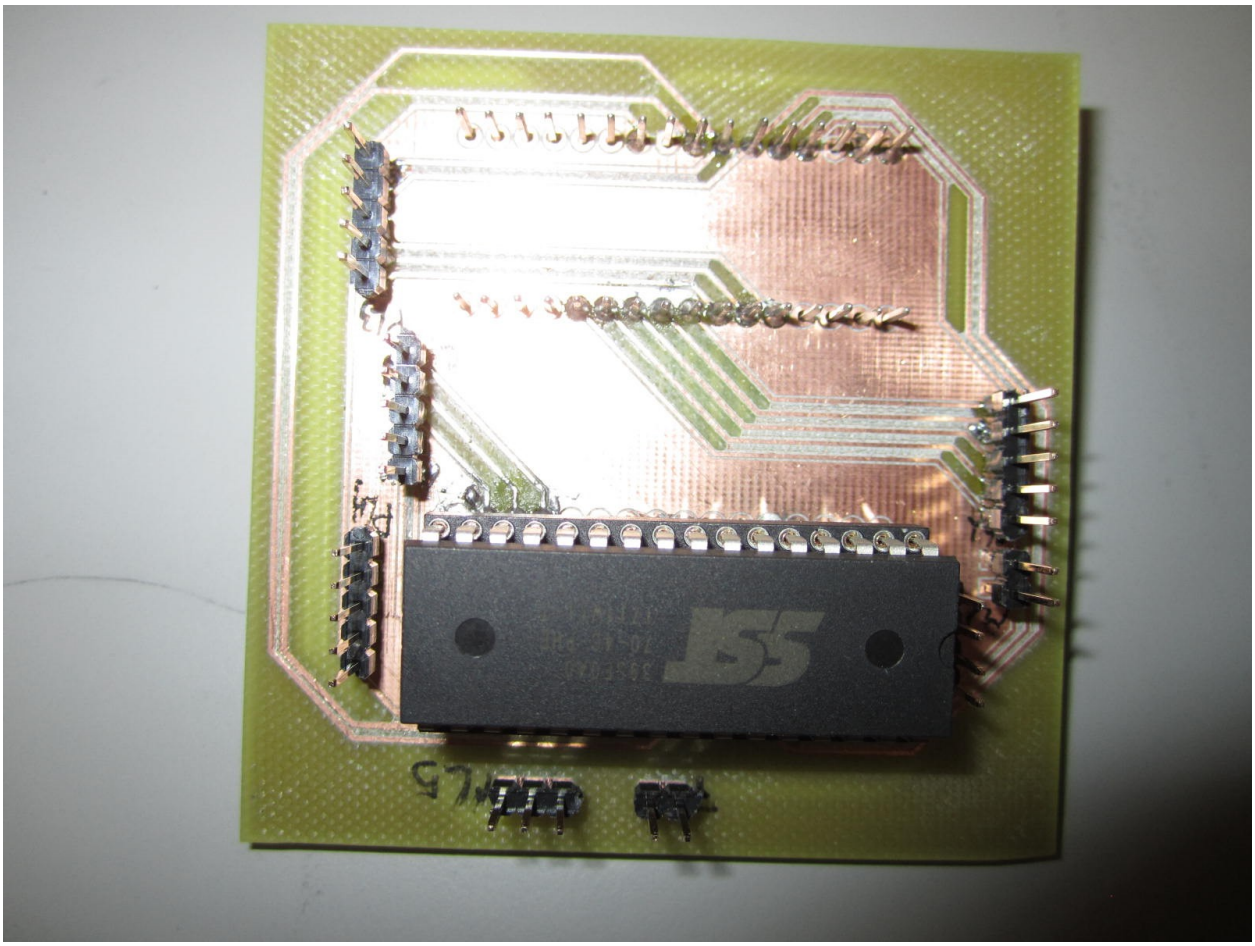

Figure A.7. Front side of the memory module PCB.

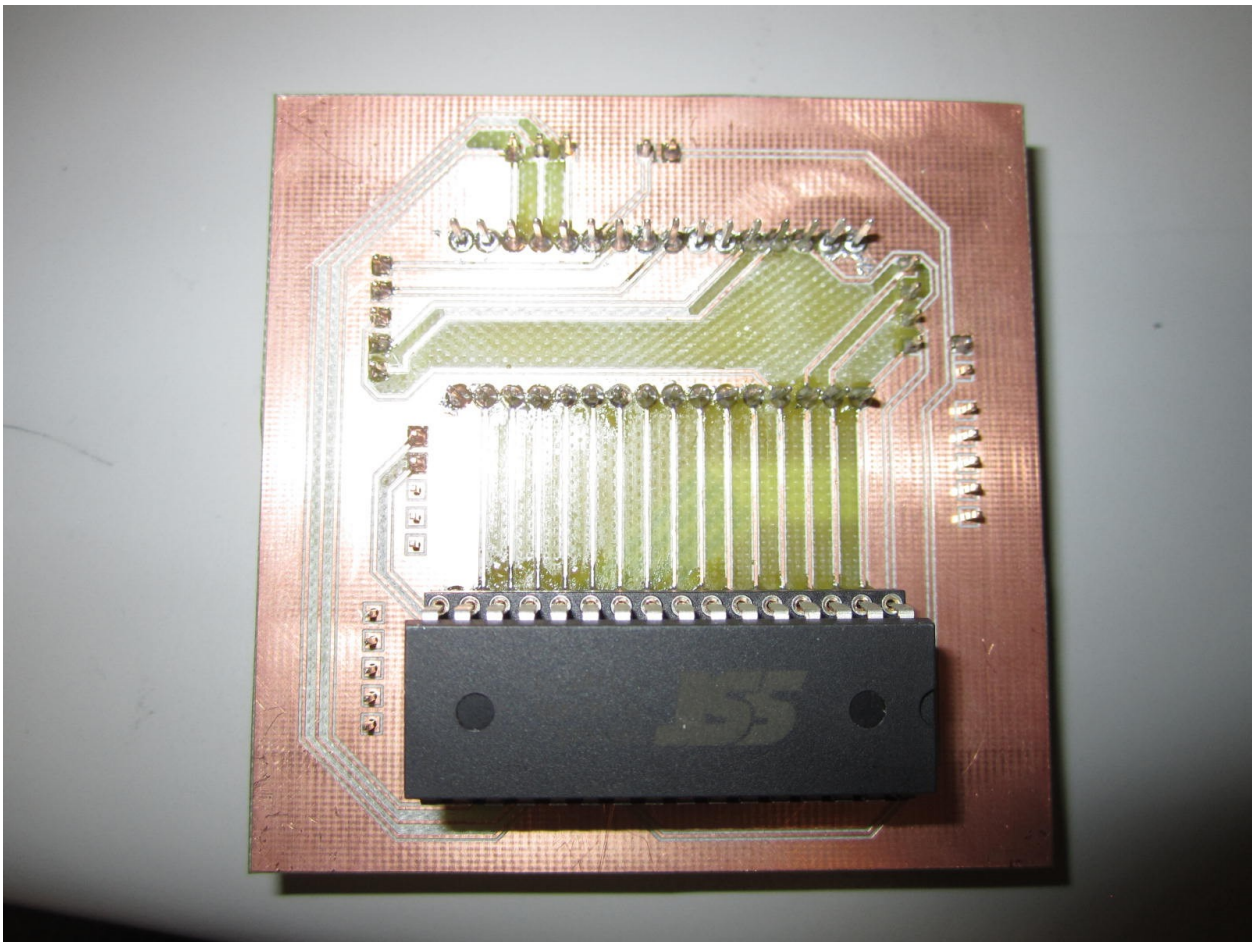

Figure A.8. Reverse side of the memory module PCB.