TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

IMPLEMENTATION OF ALGORITHMS ON FPGAS

MATTIAS KARLSSON

THESIS WORK 2009

SUBJECT ELECTRICAL ENGINEERING

TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

IMPLEMENTATION OF ALGORITHMS ON FPGAS

Mattias Karlsson

This thesis work is performed at Jönköping Institute of Technology within the subject area Electrical Engineering. The work is part of the university's three-year engineering degree. The authors are responsible for the given opinions, conclusions and results.

Supervisor: Lennart Lindh

Credit points: 15 (C-level)

Date: May 28, 2009

Archive number:

Postal Address:
Box 1026
551 11 Jönköping

Visiting Address:
Gjuterigatan 5

Telephone:
036-10 10 00 (vx)

Abstract

This thesis describes how an algorithm is transferred from a digital signal processor to an embedded microprocessor in an FPGA using C to hardware program from Altera.

Saab Avitronics develops the secondary high lift control system for the Boeing 787 aircraft. The high lift system consists of electric motors controlling the trailing edge wing flaps and the leading edge wing slats. The high lift motors manage to control the Boeing 787 aircraft with full power even if half of each motor's stators are damaged. The motor is a PMDC brushless motor which is controlled by an advanced algorithm. The algorithm needs to be calculated by a fast special digital signal processor.

In this thesis I have tested if the algorithm can be transferred to an FPGA and still manage the time and safety demands. This was done by transferring an already working algorithm from the digital signal processor to an FPGA. The idea was to put the algorithm in an embedded NIOS II microprocessor and speed up the bottlenecks with Altera's C to hardware program.

The study shows that the C-code needs to be optimized for C to hardware to manage the up speeding part, as the tests showed that the calculation time for the algorithm actually became longer with C to hardware. This thesis also shows that it is highly probable to use an FPGA equipped with Altera's NIOS II safety critical microprocessor instead of a digital signal processor to control the electrical high lift motors in the Boeing 787 aircraft.

Keywords

C2H, FPGA, DSP, Algorithm, FOC, PMDC brushless motor and DO-254.

Abbreviations

ADC	Analog to Digital Converters
ANSI	American National Standards Institute
ASIC	Application Specific Integrated Circuit
C2H	C to Hardware
CPU	Central Processing Unit
DSP	Digital Signal Processor
FOC	Field Oriented Control
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IDE	Integrated Development Environment
IP	Intellectual Property
PI	Proportional Integral
PMAC	Permanent Magnetic Alternating Current
PMDC	Permanent Magnet Direct Current
SDD	Software Design Description
SOPC	System On a Programmable Chip
SRAM	Static Random Access Memory
SRS	Software Requirement Specification
SVPWM	State space Vector Pulse Width Modulation
VHDL	Very High Speed Integrated Circuit Hardware Language

Contents

1 Introduction 5

1.1 Background 5
1.2 Purpose 5
1.3 Limitations 6
1.4 Overview 6

2 Theoretical background 7

2.1 Permanent magnet direct current brushless motor 7
2.2 Field oriented control 8
2.3 Three phase inverter and space vector modulation 10
2.4 The standard DO-254 11
2.5 Examples of implementation methods 11
2.5.1 Hardware description language 11
2.5.2 Intellectual property 12
2.5.3 C to hardware 13

3 Implementation 15

3.1 Theoretical preparation and investigation 15
3.2 Choice of implementation method 16
3.3 The time demand and the testing of it 16

4 Result 18

5 Conclusions and future work 20

6 Reference 21

Table of figures

FIGURE 1. STATE FEEDBACK CONTROL ALGORITHM TO CONTROL A PMDC BRUSHLESS MOTOR. 8
FIGURE 2. SCHEMA ON THE DIFFERENT STATES OF THE TRANSISTORS IN THE TREE PHASE INVERTER. 10
FIGURE 3. A POSSIBLE TIMETABLE FOR THE ALGORITHM 16
FIGURE 4. RESULTS FROM THE TIMING TESTS. 18

1 Introduction

1.1 Background

Saab Avitronics develops the secondary high lift control system for the Boeing 787 aircraft. This high lift system consists of trailing edge wing flaps and leading edge wing slats which are controlled by both hydraulic and electric components. The electric motor which lifts and lowers the wing flaps and wing slats is itself controlled through a field oriented control (FOC). A FOC is an advanced feedback control system, which is used to control a permanent magnet direct current (PMDC) brushless motor. Today the algorithm of the FOC is placed in a digital signal processor (DSP). This solution is among other things expensive. It would be a more reliable and cheaper solution to place the algorithm in a field programmable gate array (FPGA).

So far Saab Avitronics has not tried if it is possible to fulfill applicable standards and performance requirements by running the algorithm in an FPGA. Therefore this study will try to find out whether it is possible or not. It would be an advantage for the company if this thesis shows that there is a way to replace the DSP with an FPGA.

I will try to replace the DSP by going through two steps: First, the existing algorithm in the DSP will be translated to fit into a soft core central processing unit (CPU) called NIOS II from Altera. Next, to fulfill the performance demands, some portions of the algorithm will be replaced with hardware. This is done by using a program that translates C to hardware (C2H) directly. Most of the performance boost comes from the parallel nature of the hardware.

1.2 Purpose

The purpose of this thesis is to study an algorithm with focus on requirements and time demands. The purpose also includes finding out whether there is any usable solution to implement the algorithm in an FPGA. If there are several solutions, the purpose includes choosing one of the solutions and try if it works in practice.

1.3 Limitations

Since this thesis is based on an already existing system, which is not subject to change, it only includes one type of motor, a PMDC brushless motor, and one algorithm. The algorithm is given from the start. Because of this most of the theory around the PMDC brushless motor, e.g. mathematics of the dynamic model [1], is left out from this thesis. Some functions in the algorithm which are not important for this study, have also been left out. A further limitation is that I only make an effort to investigate one implementation method even if there might be several possible solutions available. The speed tests of the algorithm will only be done in one FPGA even if there are many suitable FPGAs on the market.

1.4 Overview

This thesis is organized as follows:

In Chapter 2 a theoretical background is given, which includes a brief explanation of the studied algorithm and an explanation on the space vector pulse width modulation technique. This chapter also briefly explains some of the requirements for electronics in airplanes according to the guidelines in RTCA/DO-254. The last section of chapter two gives six examples on how the algorithm can be implemented into an FPGA.

Chapter 3 starts with a description of the algorithm's requirements and continues with an explanation on the algorithm's time limits. This chapter also includes an explanation on why an Altera Cyclone II was chosen. Chapter 3 ends with the chosen solution and explains why that solution was chosen.

Chapter 4 goes through the time results of the algorithm implemented in the FPGA and briefly explains why the results turned out as they did.

2 Theoretical background

In this chapter a basic explanation of a PMDC brushless motor is given. To control a PMDC brushless motor an advanced algorithm is needed. The chapter gives a brief introduction to the algorithm called FOC. The chapter also includes an explanation on how a state space vector pulse width modulation (SVPWM) is working. The chapter continues with a basic explanation of the standard DO-254; Design Assurance Guidance for Airborne Electronic Hardware, which is used in the development of hardware, like FPGA, for airborne systems. The chapter ends with six descriptions on possible methods for implementing the algorithm in an FPGA.

2.1 Permanent magnet direct current brushless motor

A PMDC brushless motor is a motor with a permanent magnet as a rotor. The PMDC brushless motor is physically almost the same as a permanent magnetic alternating current (PMAC) motor. The big difference is that a PMDC brushless motor's rotor needs to be synchronized and controlled by a square wave current, which the PMAC motor's rotor doesn't need to be. The synchronization is given either from a sensor mounted on the shaft, which is measuring the rotor's angle, or a state space model, which estimates the angle [2]. The usual measuring sensor is a resolver, quadrature or encoder [3]. The state space model is a new technique offering higher reliability to a lower cost [2].

There are several benefits in using a PMDC brushless motor instead of another motor type e.g. higher torque, better dynamic performance, increased efficiency and less maintenance. The possibility of operating the motor over a large range of speed and still maintaining a good efficiency is also achieved [3]. The PMDC brushless motor has almost no power losses; because the rotor is a permanent magnet instead of an electromagnet field [2]. Cooling becomes a small problem when the electric magnetic field isn't in the rotor. This means that a PMDC brushless motor can be built smaller and with less weight but still remain the same strength [3]. The only disadvantage with using a PMDC brushless motor is that it needs to be controlled by an advanced FOC algorithm.

2.2 Field oriented control

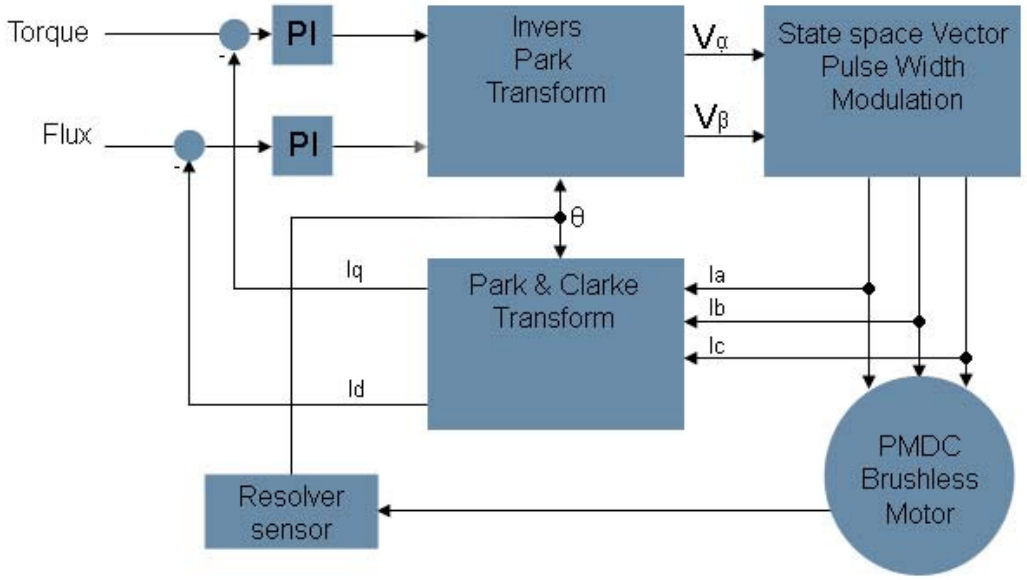

Figure 1. State feedback control algorithm to control a PMDC brushless motor.

An ordinary DC-motor is controlled by its torque i.e. more power gives more torque. To be able to control a PMDC brushless motor aligned with the torque an advanced algorithm called FOC is needed. The FOC, shown in Figure 1, converts the stators current vector from the three phase static frame coordinate system to a rotating flux and torque coordinate system. To convert between the coordinate system the FOC uses Clarke [equ1] and Park [equ2] equations. The equations demand special arithmetic in the processor.

The Park and Clarke transformation algorithms are:

Clarke [4]:

$$I_\alpha = I_a$$
$$I_\beta = (I_a + I_b \cdot 2) \cdot \frac{1}{\sqrt{3}} \qquad \text{[equ1]}$$

Park [4]:

$$I_d = I_\alpha \cdot \cos(\theta) + I_\beta \cdot \sin(\theta)$$
$$I_q = -I_\alpha \cdot \sin(\theta) + I_\beta \cdot \cos(\theta) \qquad \text{[equ2]}$$

Clarke transformation demands the processor to be able to calculate arithmetic like addition, multiplication, division and square root.

Park transformation demands the processor to be able to calculate arithmetic like addition, subtraction, multiplication, sine and cosine.

In the rotating coordinate system the flux and torque are measured by a reference value and in this case a feedback control. The feedback control used is a proportional integral (PI) regulator. The PI regulator algorithms are:

PI regulator [5]:

$$I_d = K_{pi} \cdot I_d + K_i \cdot I_d + \sum_{n=o}^{k-1} I_d$$
$$I_q = K_{pi} \cdot I_q + K_i \cdot I_q + \sum_{n=o}^{k-1} I_q \qquad \text{[equ3]}$$

The PI regulator demands the processor to be able to calculate arithmetic like addition and multiplication. The processor also needs to be able to store big values.

The final control of the motor is done in the static coordinate system. As illustrated in Figure 1 the signals are transformed back by a Park^{-1} transform to the static frame coordinate system. The Park^{-1} transform algorithms are:

Park^{-1} transform [4]:

$$V_\alpha = V_d \cdot \cos(\theta) - V_q \cdot \sin(\theta)$$
$$V_\beta = V_d \cdot \sin(\theta) + V_q \cdot \cos(\theta) \qquad \text{[equ4]}$$

The inverse Park transformation demands the processor to be able to calculate arithmetic like addition, subtraction, multiplication, sine and cosine.

These formulas need to be calculated in a short period of time, in this case less than 100 µs [6], for the SVPWM to be updated as required. Because of the short time available and the difficult arithmetic to be calculated, it takes a special fast calculating processor like a DSP to manage the algorithm. Also an FPGA can be used, gaining some benefits against the DSP. The FPGA can e.g. use separate analog to digital converters (ADC), which work in serial in the DSP, to convert the current and it can read the current values from several ADCs at the same time. The FPGA can also calculate the currents in the formulas in parallel to each other.

Controlling a PMDC brushless motor instead of a PMAC motor through a FOC, differs only in the flux component. The flux in the PMDC brushless motor's rotor is a permanent magnet. Because of the permanent magnet the flux component is set to zero. In an ordinary FOC algorithm only two currents [equ5] are needed but the algorithm in this study needs all three because it has to work with full power even when half of the stator is damaged. To control the PMDC brushless motor in the dq-domain there are correct controls in all part of the motion i.e. from motionless to full speed [7]. For a more primary explanation I refer to [3],[5],[6],[7],[8],[9],[10].

$$I_a + I_b + I_c = 0 \text{ [5]} \qquad \text{[equ5]}$$

2.3 Three phase inverter and space vector modulation

The three phase inverter controlled by the SVPWM is the push/pulls driver stage. The three phase inverter consists of six IGBT or MOSFET transistors [3]. The transistors control both upper and lower part of the AC-current and are usually named after which part they control, i.e. upper and lower transistor.

The transistors can be controlled independently. The SVPWM function has full control over the three phase inverters behavior, but the SVPWM function can also short the current and break the transistors. This is extremely sensitive when the signal passes through zero because two transistors (upper and lower) are switched at the same time. To manage the signals passing through zero it is appropriate to have a delay time in a few µs [3]. It is also sensitive if two phases are switched at the same time. The later problem is solved by the switching order of the transistors.

Because the transistors control both upper and lower part of the AC-current the transistors can't make upper and lower active at the same time. This, as shown in Figure 2, leaves the SVPWM eight sector steps. In two of these steps, where all transistors are upper or lower, no current run in the stator of the motor. The accepted standard in which order the transistors are going to be controlled are the way that only one transistor is switch on at the time. This means that the null vectors need to alternate. When the SVPWM reaches the state V_7 it needs to go in reverse order until it reaches state V_0 [5].

Sector	X upper	X lower	Y upper	Y lower	Z upper	Z lower	V_{xy}	V_{yz}	V_{zx}
V_0 = {000}	OFF	ON	OFF	ON	OFF	ON	0	0	0
V_1 = {100}	ON	OFF	OFF	ON	OFF	ON	2/3 V_{dc}	−1/3V_{dc}	−1/3V_{dc}
V_2 = {110}	ON	OFF	ON	OFF	OFF	ON	1/3V_{dc}	1/3V_{dc}	−2/3V_{dc}
V_3 = {010}	OFF	ON	ON	OFF	OFF	ON	−1/3V_{dc}	2/3 V_{dc}	−1/3V_{dc}
V_4 = {011}	OFF	ON	ON	OFF	ON	OFF	−2/3V_{dc}	1/3V_{dc}	1/3V_{dc}
V_5 = {001}	OFF	ON	OFF	ON	ON	OFF	−1/3V_{dc}	−1/3V_{dc}	2/3 V_{dc}
V_6 = {101}	ON	OFF	OFF	ON	ON	OFF	1/3V_{dc}	−2/3V_{dc}	1/3V_{dc}
V_7 = {111}	ON	OFF	ON	OFF	ON	OFF	0	0	0

Figure 2. Schema on the different states of the transistors in the tree phase inverter.

2.4 The standard DO-254

For development and use of advanced complex electronics like FPGAs in safety-critical airborne systems there is a guideline (standard) called RTCA/DO-254 that must be followed. In Europe the same guideline is sometimes referred to as EUROCAE ED-80. Saab Avitronics is committed to follow RTCA/DO-254 in its work with safety-critical airborne systems. In this thesis RTCA/DO-254 and EUROCAE ED-80 are only referred to as DO-254. Guidelines for software considerations in airborne systems and equipment certification are called RTCA/DO-178B and EUROCAE ED-12B.

The purpose of DO-254 is to create guidelines so that electronics in airborne systems are free of design errors such that it can safely perform its functions under any circumstances. The purpose is also to be able to work in an environment with both old and new electronic technology.

In my study DO-254 limits the use of certain FPGAs and implementation methods.

2.5 Examples of implementation methods

There are several ways to implement the algorithm in an FPGA. One way is to completely rewrite the entire algorithm in a hardware description language (HDL) like Verilog or very high speed integrated circuit hardware language (VHDL) and implement it in the FPGA. To save time in the development of an FPGA the market offers a vast range of intellectual property (IP) cores. One type of IP is a soft core CPU which can be used to implement the algorithm into and use the advantage of an already designed code. Another method is to use a physical CPU on board the FPGA called hard core and then utilize that the CPU is designed and optimized for the FPGA. The final way is to use a program that converts C-code directly to hardware. In the following sections the different implementation methods are described more precisely.

2.5.1 Hardware description language

To implement gates in an FPGA there are several HDLs to be used in the market. The two most common languages are Verilog and VHDL. One way to implement the algorithm into the FPGA is by using an HDL. The HDL usually describes the construction on register transfer level. The advantage of constructing a circuit on lower levels is that the constructor gets knowledge about every part of the circuit and every function. This advantage is mainly what the standard DO-254 stands for. The two main disadvantages with using an HDL are that the construction is time consuming and demands extensive coding.

2.5.2 Intellectual property

To be able to do rapid prototyping and accelerate the development in FPGAs there is an advantage in using IP cores. The vendors of FPGAs usually offer a large library of tested HDL IP cores with common and complex functions like FFT-filter, PCI-bus interface and µP [8]. These functions take long time to develop in Verilog and VHDL, therefore it is an advantage to use already written and verified IP cores. An IP core can be at different stages between a piece of HDL code all the way down to physical hardware. Some examples are given below.

2.5.2.1 Soft intellectual property core

A soft IP core is an IP core heavily dependent on synthesis tools and it is often provided in Verilog or VHDL [8]. Because of the soft IP core is delivered in HDL the core is quite easy to change and modify (if the vendor allows it) to fit the project. The disadvantage with soft IP cores is the price and the time it takes to modify the IP core. The high price often depends on that it is hard to protect the soft IP core from being copied [3]. There are several soft IP cores which are matched to fit this project e.g. multipliers, sin & cos tables and PI-regulators.

2.5.2.2 Parameterized core

Achieving the advantage of a lower price often leads to less flexibility, i.e. the IP core only works with a certain branch of FPGAs or it is fewer parameters possible to change, in the core, to fit the FPGA. Another advantage besides lower price for parameterized cores is that they are more specialized to the chosen FPGA than soft IP cores. The core is often correct within 5 % [8]. To test the code it is often shipped with C or Matlab scripts. When choosing a parameterized core the FPGA is often automatically selected. Examples of parameterized cores that can be used in this project are coordinate rotation digital computer (Cordic) sine and cosine functions [5],[8].

2.5.2.3 Hard core

To achieve the advantage of a physical optimized core one should use hard cores. For these cores the physical design is already implemented. This makes timing and similarity independent from the synthesis result. This comes with the cost of a very precise core, optimized to a special FPGA or even to a dedicated function which can’t be changed e.g. embedded microprocessors.

2.5.2.4 Open intellectual property cores

To achieve the flexibility of being able to change the IP core and to save time there are communities on the Internet who develop many different projects and share them for free. The change or improvement of the IP core is usually fed back to the community in return. Some open IP core projects which can be found on the Internet that are suitable for this project are:

- PWM projects (www.opencores.org)
- Microprocessor projects (www.opencores.org)
- C to hardware projects (www.c-to-verilog.com)
- Cordic projects (www.opencores.org)
- Motor control project MotionFire (www.motionfire.eu)

2.5.3 C to hardware

Making hardware of C-code is a new way of programming an FPGA. This can be done in several ways. One way is by using an embedded microprocessor like NIOS II and American national standards institute (ANSI) written C-code. Altera offers a C2H tool for the NIOS II microprocessor which is integrated in the integrated development environment (IDE). The C2H tool converts C-functions into hardware making them coprocessors, which relieves the NIOS II microprocessor with time consuming applications [11]. The C-functions become dedicated hardware which accelerates the C-functions and the program. The target is to convert the time consuming bottlenecks which are located in the program. Once the bottlenecks are located they can be put into own C-functions and then converted into hardware. The C2H makes a C driver which is compiled into the software and integrated into the hardware, which is either working in parallel to the processor or in serial depending on the constructor's choice [11].

A NIOS II microprocessor is configured with needed functions, e.g. inputs, outputs and SPI, in the system on a programmable chip (SOPC) builder environment. All functions in the NIOS II are connected through an Avalon bus [11]. If an application needs a certain function and it is missing, it can be made in HDL and imported into the SOPC builder. By doing so an alternative way to achieve the DO-254 standard in C2H is given.

The communication between the hardware and the microprocessor's Avalon bus is difficult [12]. To use an Avalon bus vendors developed a tool called wrapper. The wrapper is an interface between hardware and the Avalon bus. Because of this solution the wrapper and the functions need to be analyzed and verified according to the DO-254 standard [11].

There is one version of the NIOS II called NIOS II safety critical that is developed using the DO-254 standard and therefore ease the use in airborne systems built all over the world, including the US [13].

In my project the algorithm is designed for a DSP. The code is written in a limited set of standard ANSI C-code [6],[14]. Therefore it is a natural step to try out the C2H alternative. Even for Saab Avitronics it would be a proof of concept to see the C2H alternative.

3 Implementation

The following chapter describes how the theoretical preparations and practical testing was carried out.

3.1 Theoretical preparation and investigation

The theoretical preparations for this thesis included reading the software requirement specification (SRS) documentation [6] and the software design description (SDD) documentation [14] for Saab Avitronics B787HL project. The SRS documentation is a description of the algorithm and is written to secure that all the demands in the DO-178B standard are met, as it sets the rules for the algorithm and the C-code. The SDD documentation describes the C-code as it is determined by the SRS documentation. Several requirements were located throughout the documentation, which all are necessary to make the algorithm function properly. About half of these requirements are basic arithmetic whereas the rest are more advanced and time demanding arithmetic.

The reader shall observe that these arithmetic demands are not the only demands on the algorithm. Other demands are set by the application, but these demands are not subject for this thesis.

Also requirements set by other standards including DO-254 were identified throughout the above mentioned documentation, e.g. that it isn't possible to use static random access memory (SRAM) based FPGAs. By using the already existing C-code it is secured that the DO-178 requirements assigned to the code are fulfilled in this project. The Cyclone 2 FPGA that is used for this study is SRAM based and can therefore not be used in a real project. The same functions can however be offered by Altera in an application specific integrated circuit (ASIC) model, which meets the safety requirements. The FPGA and the ASIC are identical as to the physical performances and therefore the results of this study won't be affected if the FPGA is replaced by an ASIC in the application.

To understand the SRS and SDD documents and their meaning an extensive theoretical investigation was made. The findings of the extensive theoretical investigation are included in the chapter Theoretical background, which contains e.g. information about the FOC and PMDC brushless motors.

3.2 Choice of implementation method

After the extensive theoretical investigation I located the C-code and H-code files for the project of Boeing 787. These code files contained a FOC with several algorithms. Except one file the code files were written in standard ANSI C-code. The natural choice of implementation method was therefore to test the algorithm in Altera's embedded NIOS II microprocessor and if the speed wasn't satisfying try to convert the algorithm to hardware. Before this the only non-standard ANSI written C-code was translated from DSP specific assembler to ANSI C-code which fits the NIOS II microprocessor.

To meet the hardware requirements in the DO-254 standard Altera and HCELL Engineering offer the microprocessor NIOS II safety critical. It wasn't used in this study instead it was represented by the NIOS II fast, which is functionally equivalent with the NIOS II safety critical.

A DE2-70 card with a Cyclone 2C70F896C6 FPGA was ordered and delivered from Altera. The FPGA was equipped with e.g. 150 embedded 18*18 multipliers.

3.3 The time demand and the testing of it

One of the identified demands on the algorithm is time. The time demand is to be able to update the SVPWM signal once every 100 µs [6]. To be able to do this the algorithm needs to be calculated. To calculate the algorithm, the algorithm needs input from the ADC and the resolver. Because the FPGA is working in parallel the FPGA manages to control three different ADCs simultaneously with the resolver. This saves time to the FOC calculations. Figure 3 describes (1) how the SVPWM cycle starts, (2) the time that the ADC needs to generate ADC signals for all three faces and the FPGA needs to import the ADC values into the FPGA, (3-6) the time the FPGA needs to calculate the FOC and import the values to the SVPWM function.

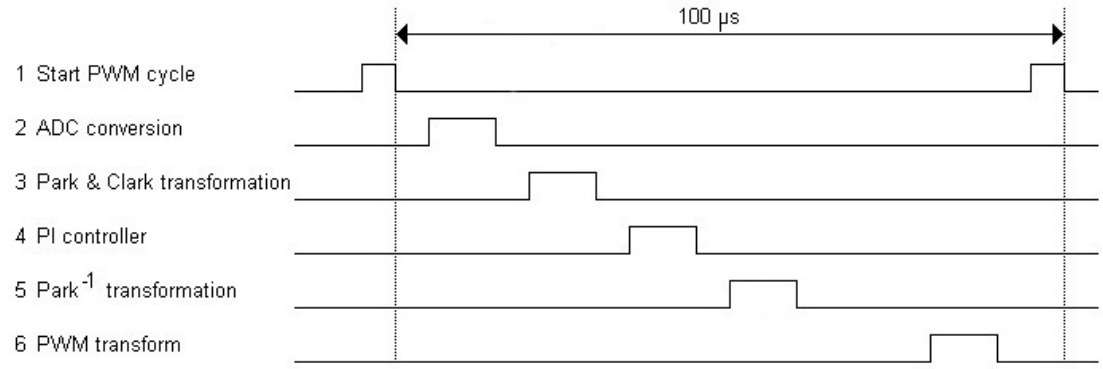

Figure 3. A possible timetable for the algorithm.

To determine the time consumption for each part of the algorithm the parts were put in three different C-functions. These are illustrated as 3-5 in Figure 3. These functions represent the different parts of the FOC and are timed one by one. To time each function a special timer was built. The special timer counts cycles in the processor. The microprocessor's frequency is 50 MHz which makes the time calculation simple. The timer was imported into the microprocessor by the SOPC environment and connected to the Avalon bus by a wrapper. Suitable macros were built to ease the control of the timer in the C-code.

4 Result

In total I tested the calculation time for the algorithm twelve times, to see if it's possible to fulfill the time demand of the algorithm when it's transferred from a DSP to an FPGA. The results are illustrated in Figure 4, which includes the maximum time it took to run the algorithm in different configured NIOS II microprocessors. Figure 4 also shows that a NIOS II microprocessor doesn't manage the timing if the NIOS II isn't equipped with inbuilt multipliers.

I used C2H as implementation method when implementing the algorithm into the FPGA. Using C2H was supposed to speed up the calculation time for the algorithm, but as shown in Figure 4 some unexpected timings became the result when the algorithm was speeded up by hardware. The timing in the different NIOS II microprocessors was actually more often slower with then without the hardware. The tests show that the C2H doesn't manage the time demands by any performances of the microprocessor. This depends on that the original and unchanged C-code isn't optimized for the C2H tool.

When I optimized the code for C2H, even the NIOS II economic microprocessor managed the algorithm within the time demand. This is shown in Figure 4. I'm prohibited to describe the optimizing due to confidentiality.

Processor	Inbuilt Multiplicator	Hardware	Logic elements	Total register	Memory bits	Antal multiplicerare	ParkeClarke	PI Regulator	InversePark	Full		Info
			Gates in the FPGA				Timing results					
NIOS II economic			1 554	851	141 312	0	150	170	37	357	µs	More memory in the µp was needed
NIOS II economic		V	5 854	4 155	176 146	13	1 814	10	1 811	3 635	µs	
NIOS II standard			2 063	1 161	176 640	0	42	47	12	102	µs	
NIOS II standard	V		2 600	1 521	176 640	4	9	8	5	22	µs	
NIOS II standard		V	6 322	4 472	309 906	13	458	5	454	917	µs	More memory in the µp was needed
NIOS II standard	V	V	10 486	7 772	311 972	26	458	5	454	917	µs	More memory in the µp was needed
NIOS II fast			2 612	1 536	194 240	0	41	47	12	99	µs	
NIOS II fast	V		3 189	1 916	194 240	4	10	6	4	20	µs	
NIOS II fast		V	6 937	4 851	229 226	13	41	47	11	99	µs	More memory in the µp was needed
NIOS II fast	V	V	11 617	8 531	231 332	30	355	29	353	738	µs	More memory in the µp was needed
NIOS II economic							270	136	51	456	µs	C-code optimized for C2H
NIOS II economic		V					5	6	5	16	µs	C-code optimized for C2H

Figure 4. Results from the timing tests.

The result of the practical tests done in this study is that all the NIOS II equipped with multipliers are able to run the algorithm within the time demand. But as mentioned before the only way to achieve the DO-254 standard is by using a NIOS II safe critical microprocessor. I haven't had the opportunity to test this microprocessor, because the IP core is highly secured and only available for buyers. The NIOS II fast with embedded multipliers, which I tested and that managed the algorithm within the time demand, has the same performances as the NIOS II safe critical microprocessor [13]. This shows that the NIOS II safe critical microprocessor also ought to fulfill the time demand.

The final result of my study is that a NIOS II safe critical microprocessor is capable to calculate the algorithm within the time demand and to achieve the demands set by the DO-254 standard.

5 Conclusions and future work

Twelve tests of the calculation time for the algorithm have been done through this study. The results of the tests are put together in Figure 4 in Chapter 4. The main result is that the algorithm can be calculated within the stipulated time of 100 µs using a NIOS II microprocessor with multipliers. The conclusion is therefore that it is possible for Saab Avitronics to replace the DSP with an FPGA.

Although the conclusion of this thesis is positive there is more work that might be carried out. Below I give some examples of possible future work.

1. Test the NIOS II safety critical microprocessor.
2. Build the algorithm in HDL code.
3. Build the different parts of the algorithm in HDL and import them in SOPC through a wrapper.
4. Use open IP cores.
5. Replace the resolver with a sensorless estimator.

6 Reference

[1] Gieras, Jacek F Wing, Mitchell (2002) Permanent magnet motor technology (second edition). ISBN 0-8247-0739-7.

[2] Dr Flint, Tom Analog Devices: High efficiency, low cost, sensorless motor control.

[3] Valentine, Richard (1998) Motor control Electronics handbook. ISBN 0-07-066810-8.

[4] http://embeddedstudio.com/FOC.aspx (Acc. 7/15/2009).

[5] http://en.wikipedia.org (Acc. 7/15/2009).

[6] B787HL Software Requirement Specification (SRS).

[7] Texas Instruments application note, “Field Orientated Control of 3-Phase AC-Motors” Literature Number: BPRA073.

[8] U. Meyer-Baese Digital signal processing with field programmable gate arrays (third edition). ISBN 978-3-540-72612-8.

[9] Johansson, Andreas FPGA baserad PWM-styrning av BLDC-motorer LiTH-ISY-EX-3341-2003.

[10] http://www.mathworks.com (Acc. 7/15/2009).

[11] White paper Automated generation of Hardware Accelerators With Direct Memory Access From ANSI/ISO Standard C Functions.

[12] Korsnes, Eirik Nios II C-to-Hardware Acceleration Compiler.

[13] PowerPoint: HCELL Engineering NIOSII_SC Safety Critical Processor Soft IP DO-254.

[14] B787HL Software Design Description (SDD).