FPGAs used in Industrial Control Systems

Prof. Eric Monmasson,
Cergy-Pontoise University, Cergy-Pontoise, France
Email: eric.monmasson@u-cergy.fr

Dr. Marcian Cirstea,
Anglia Ruskin University, Cambridge, UK
Email: marcian@ieee.org

Overview

- Introduction, Presentation of the Current Trends (30 min, MC, EM)
- Description of FPGAs (30 min, EM)
- Holistic Modelling/Design Methodology (30 min, MC)
- Main Design Rules (30 min, EM)
  - Refinement of Control Algorithms by Simulation
  - Algorithm Architecture Adequation
  - Reusability, VHDL Coding
  - Hardware-In-the-Loop (HIL) Validation

Coffee break

- 1st case studies series: FPGA-based Current Controllers for AC Drives (40 min, EM)
  - Quasi-Analog Hysteresis Controller
  - Delta Modulator
  - PI – SVM Controller
  - Predictive Controller

- 2nd case studies series: FPGA-based Intelligent Controllers for AC Drives and AC Generators (40 min, MC)
  - Induction Motor Control Using Neural Networks
  - Stand Alone Generator Set Using Fuzzy-Logic and PWM

- Conclusions and Perspectives (10 min, EM, MC)
- Hands on practical demonstration on two simple examples (30 min, EM, MC)

ELECTRONIC SYSTEMS ON CHIP
Technical Committee of IEEE Industrial Electronics Society
http://vega.unitbv.ro/~ieee
Mission Statement:

This Committee aims to promote professional activities in the area of low power electronics used in the modern industry, with an important focus on the design, development, simulation, verification and testing of digital and analogue circuits integrated as Systems on Programmable Chips, targeting Field Programmable Gate Arrays / Application Specific Integrated Circuits for implementation, and including the use of Hardware Description Languages or high level programming languages hardware compilers, as well as embedded electronic systems and associated software.

Chair: Dr. Marcian Cirstea, Head of Department of Design & Technology, Anglia Ruskin University, Cambridge, UK.
Email: marcian@ieee.org

- Special conference sessions organisations subcommittee
- Coordinator and Committee Vice-Chair: Dr. Manus Henry, Deputy Director, Invensys University Technology Centre for Advanced Instrumentation at the Department of Engineering Science, the University of Oxford, UK.
- Dr. Vito Nardi, University of Cassino, Italy.
- Special Issues / Sections of Journals subcommittee
- Coordinator: Prof. Eric Monmasson, Head of the Institut Universitaire Professionnalisé de Génie Electrique et d’Informatique Industrielle (IUP GEII), University of Cergy-Pontoise (UCP), France.
- Prof. Josep M. Guerrero, Universitat Politècnica de Catalunya (UPC), Barcelona, Spain.
- Dr. Jeen G. Khor, Senior Design Engineer, INTEL, Penang, Malaysia.
- Web page subcommittee
- Coordinator: Dr. Andrei Dinu, Goodrich Engine Control Systems, Electromagnetic Systems Technical Centre, Birmingham, UK.
- Dr. Otilia Boaghe, Phillips Semiconductors, Zurich, Switzerland.
Introduction

- Traditionally, mathematical models were used to functionally evaluate engineering systems. The development of each system component used then to be separately addressed, often involving the use of other CAD tools and/or different software platforms.

- Traditional methods are not able to cope with increased complexity and demands of higher levels of systems integration / faster time to market. Recent advances in CAD methodologies/languages has brought the system’s functional description and hardware implementation closer.

- Modern Electronic Design Automation (EDA) tools are used to model, simulate and verify a complex engineering system fast, with high confidence in “right first time” correct operation, without producing a prototype.

- High performance electronic controllers can also be implemented.

- The presentation reveals recent work that was carried out in the area of holistic modelling of engineering systems using HDLs.

Activity Plan 2007

- Website development to support & promote committee’s work and to provide a point of reference on topics of interest.

- Guest-Editorship of a special issue of the Transactions on Industrial Electronics: FPGAs used in Industrial Control Systems. Dr. Eric Monmasson and Dr. Marcian Cirstea are joint Guest Editors.

- Organisation of a “best paper” prize of $500 for this Special Issue

- Organisation of special sessions at the forthcoming IEEE IES Conference: ISIE’07.

- Organising ISIE’08 in Cambridge


- Contributing to the organisation of other IES conferences by chairing Technical Tracks, refereeing papers, etc.

- Presenting tutorials

- Printing materials to advertise the committee, as well as its technical activities, as posters / leaflets.
Integrated Circuits

- Off-the-Shelf Logic - Function pre-set.
- PROM, PAL, FPLA - Programmed by fusible links / charge storage.
- FPGAs - User programmable Field Programmable Gate-Arrays.
- Gate Array Device - Function set at manufacture in the final stage of production (metallization).
- Cell Based Device - Function set at manufacture using CAD to speed up design and a library of optimised standard functions.
- Full Custom Device - Function set at manufacture - every circuit part is optimally designed. Long development time even with CAD.

More gates / chip ==> reduces cost but requires CAD.

Comparative Economics

- True cost formula shows that the final unit cost is:
  \[ \text{cost} = \frac{D}{N} + \text{chip} + F \]
  where D=total development cost, N=no. of chips manufactured, chip=unit chip costs in production, F=packaging, testing per chip

Application Specific Integrated Circuits

- Application Specific Integrated Circuits (ASICs) = any IC designed and built specifically for a particular application.
- ASICs allow tailoring the design during development stages of an IC.
  - Reduced Size and Cost
  - High Speed and Accuracy in Information Processing
  - Compact Structure
  - High Reliability of Circuit Operation
- Two major ASIC technologies: CMOS and BICMOS - millions gates.
- RISC and DSP cores are now offered by chip suppliers. They permit the design of single chip customised advanced integrated processors.
- Field-Programmable Gate Arrays (FPGAs) are a special class of ASIC's which differ from mask-programmed gate arrays in that the programming is done by end-users with no IC masking steps.

Introduction

- FPGAs have reached high density rate (> 10 millions gates)
- Performing Electronic Design Automation (EDA) Tools
- These components allow the programming of specific hardware architecture
- This leads to a flexible and an efficient solution (software development of dedicated hardware architecture that includes parallelism)
- System-on-a-Chip (SoC) scale
Introduction

- Many industrial applications
  - Telecom,
  - Video,
  - Signal Processing,
  - Medical Systems,
  - Embedded Systems (Aircraft, Automotive),
  - Electrical Systems:
    - PWM inverters,
    - Power factor correction AC/DC converters,
    - Multilevel converters,
    - Matrix converters,
    - Active filters,
    - Fault-detection on power grid,
    - Electrical machines control (induction machine drives, multi-machines systems),
    - Neural Network control of induction motors,
    - Fuzzy Logic control of power generators,
    - Speed measurement...

Advantages:

- The decrease of the cost
  - An architecture based only on the specific needs of the algorithm to implement,
  - Application of highly advanced and specific methodologies improving implementation time also called "time to market",
  - Expected development in VLSI design that will allow integrating a full control system with its analog interface in a single chip, SoC.

- The confidentiality
  - Specific architecture, integrating the know-how of a company, is not easily duplicable.

- The embedded systems
  - Many constraints as in aircraft applications, like limited power consumption, thermal consideration, reliability and Single Event Upset (SEU) protection.

- The improvement of control performance
  - Execution time can be dramatically reduced by designing dedicated parallel architectures, allowing FPGA-based controllers to reach the level of performance of their analog counterparts without their drawbacks (parameter drifts, lack of flexibility).
  - FPGA-based controller can also be adapted in run-time to the needs of the plant by dynamically reconfiguring it.

- Algorithmic Constraints:
  - High data dependency
  - High level of parallelism of the algorithm
  - Few functions and/or homogenous functions
  - Lot of functions and/or heterogeneous functions

- Algorithm Architecture “Adequation” by means of FPGA
Introduction

How to easily implement a control algorithm on an FPGA-based optimized hardware architecture?

Simulink Model

FPGA board

This leads to follow up a design methodology

functional simulation

Fixed-point quantization & discrete model

Experimental board

FPGA target

VHDL coding

Generic FPGA Architecture

Logic Cell / Logic Element:

Configurable Input/Output Block

Configurable Logic Block

Interconnection Programmable Network

Combinatorial output

Flip-Flop output

Output carry

D Flip-Flop

LUT

Carry Path

Input carry

Clock
### Head-to-Head

<table>
<thead>
<tr>
<th>Xilinx Virtex-5</th>
<th>Altera Stratix II</th>
</tr>
</thead>
<tbody>
<tr>
<td>– 1v 65nm copper</td>
<td>– 1.2v 90nm copper</td>
</tr>
<tr>
<td>– 207,360 logic cells</td>
<td>– 179,400 logic elements</td>
</tr>
<tr>
<td>– 11.6 Mb RAM</td>
<td>– 9.4 Mb RAM</td>
</tr>
<tr>
<td>– 192 48-bit MAC Unit (25x18 multipliers, 550MHz)</td>
<td>– 96 36x36 multipliers (384 18x18 multipliers)</td>
</tr>
<tr>
<td>– Up to four PowerPC 405 cores</td>
<td>– 1,170 user I/O pins</td>
</tr>
<tr>
<td>– MicroBlaze 32-bit soft core</td>
<td>– Nios II 32-bit soft processor core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Altera Stratix II</th>
<th>Xilinx Virtex-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>– 192 48-bit MAC Unit</td>
<td>– 1,170 user I/O pins</td>
</tr>
<tr>
<td>– MicroBlaze 32-bit soft processor core</td>
<td>– Nios II 32-bit soft processor core</td>
</tr>
</tbody>
</table>

### Head-to-Head – Low Cost

<table>
<thead>
<tr>
<th>Spartan 3E</th>
<th>Altera Cyclone II</th>
</tr>
</thead>
<tbody>
<tr>
<td>– 1.2v 90nm copper</td>
<td>– 1.2v 90nm copper</td>
</tr>
<tr>
<td>– 33,192 logic elements</td>
<td>– 68,416 logic elements</td>
</tr>
<tr>
<td>– 0.65 Mb RAM</td>
<td>– 1.15 Mb RAM</td>
</tr>
<tr>
<td>– 36 18x18 multipliers</td>
<td>– 150 18x18 multipliers</td>
</tr>
<tr>
<td>– 376 user I/O pins</td>
<td>– 622 user I/O pins</td>
</tr>
<tr>
<td>– 8 DCMs</td>
<td>– 4 PLLs</td>
</tr>
<tr>
<td>– MicroBlaze 32-bit soft processor core</td>
<td>– Nios II 32-bit soft processor core</td>
</tr>
</tbody>
</table>

### Virtex-4 Architecture

- **RocketIO™**
  - Multi-Gigabit Transceivers
  - 622 Mbps–10.3 Gbps

- **Advanced CLBs**
  - 200K Logic Cells

- **XtremeDSP™**
  - Technology Slices
  - 256 18x18 GMACs

- **PowerPC™**
  - 405 with APU Interface
  - 450 MHz, 680 DMIPS

- **Smart RAM**
  - New block RAM/FIFO

- **Xesium Clocking Technology**
  - 500 MHz

- **Tri-Mode Ethernet MAC**
  - 10/100/1000 Mbps

- **1 Gbps SelectIO™**
  - ChipSync™ Source synch, XCITE Active Termination

### Virtex IV Platforms

<table>
<thead>
<tr>
<th>Resource</th>
<th>LX</th>
<th>FX</th>
<th>SX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logic</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14K–200K LCs</td>
<td>12K–140K LCs</td>
<td>23K–66K LCs</td>
<td></td>
</tr>
<tr>
<td>0.3–4 Mb</td>
<td>0.6–10 Mb</td>
<td>2.3–5.7 Mb</td>
<td></td>
</tr>
<tr>
<td>4–12</td>
<td>6–20</td>
<td>4–8</td>
<td></td>
</tr>
<tr>
<td>32–96</td>
<td>32–192</td>
<td>72–384</td>
<td></td>
</tr>
<tr>
<td>240–860</td>
<td>240–896</td>
<td>320–640</td>
<td></td>
</tr>
<tr>
<td>SelectIO</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>RocketIO</td>
<td>NA</td>
<td>8–24 Channels</td>
<td>NA</td>
</tr>
<tr>
<td>PowerPC</td>
<td>NA</td>
<td>1 or 2 Cores</td>
<td>2 or 4 Cores</td>
</tr>
<tr>
<td>Ethernet MAC</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>
**Altera Stratix**

- Logic Array Blocks (LABs)
- DSP Blocks
- MegaRAM™ Blocks
- I/O Elements (IOEs)
- M512 RAM Blocks
- Phase-Locked Loops (PLLs)

**Slices and CLBs**

- Each Virtex™-II CLB contains four slices
  - Local routing provides feedback between slices in the same CLB, and it provides routing to neighboring CLBs
  - A switch matrix provides access to general routing resources

**Distributed SelectRAM Resources**

- Uses a LUT in a slice as memory
- Synchronous write
- Asynchronous read
  - Accompanying flip-flops can be used to create synchronous read
- RAM and ROM are initialized during configuration
  - Data can be written to RAM after configuration
- Emulated dual-port RAM
  - One read/write port
  - One read-only port

**Simplified Slice Structure**

- Each slice has four outputs
  - Two registered outputs, two non-registered outputs
  - Two BUFTs associated with each CLB, accessible by all 16 CLB outputs
- Carry logic runs vertically up only
  - Two independent carry chains per CLB
Altera Stratix

Logic Array Blocks (LABs)

Logic Element

Embedded RAM

- Xilinx – Block SelectRAM
  - 18Kb dual-port RAM arranged in columns

- Altera – TriMatrix Dual-Port RAM
  - M512 – 512 x 1
  - M4K – 4096 x 1
  - M-RAM – 64K x 8
Xilinx: Embedded Multipliers

- 18-bit twos complement signed operation
- Optimized to implement Multiply and Accumulate functions
- Multipliers are physically located next to block SelectRAM™ memory

![](image)

Altera: Embedded DSP Blocks

- Two DSP Block columns per device
- Number varies by height of column
- Can implement:
  - Eight 9x9 multipliers
  - Four 18x18 multipliers
  - One 36x36 multiplier
- Contains adder/subtractor/accumulator
- Registered inputs can become shift register

![](image)

Altera Multiplier Sub-block

![](image)

Virtex: Active Interconnect

![](image)
Virtex Hierarchical Interconnect

- 24 Horizontal Long Lines
- 24 Vertical Long Lines
- 120 Horizontal Hex Lines
- 120 Vertical Hex Lines
- 40 Horizontal Double Lines
- 40 Vertical Double Lines
- 16 Direct Connections (total in all four directions)
- 8 Fast Connects

Altera: MultiTrack Interconnect

- Direct link between LABs and adjacent blocks
- Row interconnects
  - 4, 8, and 24 blocks left or right
- Column interconnects
  - 4, 8, and 16 blocks up or down

Stratix: R4 Interconnect

MicroBlaze Processor-Based Embedded Design

- 32-Bit RISC Core
- UART
- Memory Controller
- On-Chip Peripheral Bus
- Fast Simplex Link
  - 0,1,...,7
- Arbiter
- Local Memory
- Bus Bridge
- Processor Local Bus
- Dedicated Hard IP
- Flash/SRAM
- Off-Chip Memory
Embedded Development Tool Flow Overview

- Standard Embedded SW Development Flow
  - Code Entry
  - C/C++ Cross Compiler
  - Linker
  - Load Software Into FLASH
  - Debugger

- Embedded Development Kit
  - System Netlist

- VHDL or Verilog

- HDL Entry
  - Simulation/Synthesis
  - Implementation

- Download Combined Image to FPGA
  - Compiled ELF
  - Compiled BIT

- RTOS, Board Support Package

- Instantiate the 'System Netlist' and Implement the FPGA

- HDL Entry Simulation/Synthesis Implementation

- Download Bitstream Into FPGA

- Chipscope

- Standard FPGA HW Development Flow

- VHDL or Verilog

- HDL Entry
  - Simulation/Synthesis
  - Implementation

- Download Bitstream Into FPGA

- Chipscope

- SoPC Builder

- Altera Nios II

- JTAG connection to software debugger

- SDRAM Memory
- Flash Memory
- SRAM Memory

- Nios II Processor Core
  - Program Controller & Address Generation
  - Exception Controller
  - Control Registers

- Instruction Cache
  - Tight Coupled Instruction Memory

- Data Cache
  - Tight Coupled Data Memory

- Arithmetic Logic Unit

- Custom I/O Signals

- Altera Nios II

- SoPC Builder
In trouble with a chip design?

**NOTHING EASIER!**

---

**Design Methodologies – EDA Tools**

**Traditional**

- **IDEA**
- **SYSTEM MODEL**
- **CIRCUIT DESIGN**
- **LAYOUT**
- **FABRICATION**
- **TEST**
- **MANUFACTURE**

**Modern**

- **IDEA**
- **SYSTEM MODEL**
- **Verification by Simulation**
- **CIRCUIT DESIGN**
- **LAYOUT**
- **FABRICATION**
- **TEST**
- **MANUFACTURE**

Everybody hates EDA tools at some stage!!!

---

**Design flow**

- **Design Entry** (schematic, HDL, state diagram).
- **Compilation**
- **Apply stimulus**
- **Simulation**
- **Implementation and Layout**
- **Timing Analysis / Verification**
- **Download design into silicon**
- **Testing the chip**

---

**Novel Systems Modelling Method**

- **- main features and context -**

- Extends the traditional use of Hardware Description Languages (HDLs) for electronic circuits design, to encompass holistic modelling of more complex engineering systems.
- Outcome: design environment that allows all aspects of the system to be simultaneously considered, therefore maximising performance.
- Proposed approach correlated with powerful international movement/leading edge research, directed towards system level modelling/design.
- The international EDA community, united under ACCELLERA (2000) (http://www.accellera.org/index.html), assumed the mission to drive the worldwide development and use of standards required by systems, semiconductors & design tools.
- Clear proof of the internationally identified need for the development of holistic models for complex engineering systems.
Specific Advantages Offered by VHDL

- Allows the functional/behavioural description of an engineering system to be combined with a detailed electronic design, on the same CAD platform.
- The mathematical aspects of systems and the electronic hardware design are simultaneously addressed, in a unique environment.
- It is supported by all major Computer Aided Design platforms.
- Ability to handle all levels of abstraction. The system can be simulated as an overall model during all stages of the electronic controller design, which can be subsequently targeted for “system on a chip” silicon implementation.
- Fast implementation & relatively short time to market of new designs.
- Hardware Implementation of Artificial Intelligence is facilitated.
- Versatile reusable models / design modules are generated, in accordance with modern principles of design reuse.

Advantages of FPGA Controller Prototyping

- A cheap & fast VHDL code validation is via a prototype board containing re-programmable devices - Field Programmable Gate Arrays (FPGAs).
- Allows electronic controllers’ hardware validation that provides significant information before the decision is taken to invest in an Application Specific Integrated Circuit (ASIC) = IC dedicated specifically to an application.
- It shortens the time to correct any design problem and it ensures an error free design before permanent ASIC implementation.
- The prototype board can be used for hardware testing other system components.
- The general benefits of holistic modelling of systems, combined with the advantages of VHDL and FPGAs, enable the efficient investigation of new engineering system topologies employing complex electronic controllers.

Engineering Systems Modelling Approach Summary

- Modelling / Development: VHDL
- Electronic Controller Hardware Prototyping: FPGA
- Advantages of using VHDL
  ⇒ Efficient design process
  ⇒ Single environment for modelling, simulation & electronic controller design.
  ⇒ Easy modifications and system integration of designs
  ⇒ EDA platform independence of VHDL designs (ASCII files)
  ⇒ Reusable IP block modelling/design style becomes possible
- Advantages of using FPGAs
  ⇒ Small, compact design
  ⇒ Fast, relatively cheap
  ⇒ Reusable hardware framework for testing a design
  ⇒ Short time to market of product, rapid prototyping

Top-Down Design

- VHDL allows the designer to develop and simulate ideas fast, without getting caught-up in the details of implementation.
- As the design evolves to completion, the language is able to support a complex detailed digital system description.
- Top-down design begins with modelling an idea at an abstract level, and proceeds through the iterative steps necessary to further refine this into a detailed system.
- A test environment is developed early in the design cycle. Concepts are tested before investment is made in implementation.
- As design evolves to new levels of detail, the test environment will check compliance with the original specification.
VHDL Description

- Due to increased demands of higher levels of integration / faster time-to-market, a standard language, that referenced a higher level of design abstraction was needed. This stand-alone specification is not dependent on any specific tool.
- An entire system, once consisting of many components/circuit boards, can be replaced by one/two integrated circuits.
- VHDL's flexibility and choice of modelling styles enable a natural progression from idea to implementation, giving the designer the ability to quickly create, simulate, and verify an abstract model.
- Thus, design concepts can be tested before the investment is made in the hardware implementation.
- A major feature of VHDL is its inherent ability to handle all levels of abstraction. The designer requires the use of only a single language, as well as a single simulator for all phases of design.

Design Units

Entity: describes the interface between the outside world and the design. The connection points (PORTs) to the design, the direction and type of data that flows through these points are defined here.

- For example, an AND gate with 3 connection points, 2 inputs and 1 output and data type “bit” (values '0' or '1') might look like:

```
ENTITY and2 IS
    PORT (in1, in2: IN bit;
          outp: OUT bit);
END and2;
```

Architecture: defines an entity's behaviour from a simulation point of view. It depends upon the information declared within an entity.

- Behavioural Design
  - In VHDL behavioural descriptions there is no reference to submodules within a specific VHDL architecture.
  - This does not preclude the use of subprograms within VHDL descriptions, but precludes the use of other VHDL components.
  - Behavioural descriptions are defining the design functionality.
  - A behavioural description of a multiply accumulate device (mac) is:

```
USE WORK.util.ALL;
ENTITY mac IS
    GENERIC(tco: time := 10 ns);
    PORT( in1, in2: IN bit_vector(15 DOWNTO 0);
          clk, reset: IN bit;
          out1: OUT bit_vector(31 DOWNTO 0));
END mac;
ARCHITECTURE behave OF mac IS
BEGIN
    PROCESS (clk, reset)
        VARIABLE reg_in1, reg_in2, reg_mul, accum: integer;
        ... (simulation logic) ...
    END PROCESS;
END behave;
```

- A structural architecture of a 3 input AND gate is:

```
ARCHITECTURE struct OF and3 IS
    COMPONENT and2
        PORT(sig1, sig2: IN bit;
             sig3: OUT bit);
    END COMPONENT;
    SIGNAL internal:bit;
BEGIN
    u1:and2 PORT MAP(sig1=>in1, sig2=> in2, sig3 => internal);
    u2:and2 PORT MAP(sig1=>in3, sig2=>internal, sig3=>output);
END struct;
```
BEGIN
    IF reset = '0' THEN
        reg_in1 := 0;
        reg_in2 := 0;
        reg_mul := 0;
        accum := 0;
    ELSIF rising_edge(clk) THEN
        accum := accum + reg_mul;
        reg_mul := reg_in1 * reg_in2;
        reg_in1 := vect_to_int(in1);
        reg_in2 := vect_to_int(in2);
    END IF;
    out1 <= int_to_vect(accum,32) AFTER tco;
END PROCESS;
END behave;

Bit_vector is a one dimensional array of bits. The width of the array is determined in the port declaration. The width of in1 and in2 is 16 bits while out1 is 32 bits. They can be visualised as buses.

COMPONENT multiply
    PORT( port1, port2: IN bit_vector(15 DOWNTO 0);
           output: OUT bit_vector(31 DOWNTO 0));
END COMPONENT;
COMPONENT buf
    PORT( input: IN bit_vector(31 DOWNTO 0);
           output: OUT bit_vector(31 DOWNTO 0));
END COMPONENT;
SIGNAL reg_in1, reg_in2: bit_vector(15 DOWNTO 0);
SIGNAL mul, reg_mul, adder, accum: bit_vector(31 DOWNTO 0);
BEGIN
u1: reg GENERIC MAP(16) PORT MAP(in1, clk, reg_in1);
u2: reg GENERIC MAP(16) PORT MAP(in2, clk, reg_in2);
u3: multiply PORT MAP(reg_in1, reg_in2,mul);
u4: reg GENERIC MAP(32) PORT MAP(mul, clk, reg_mul);
u5: adder PORT MAP(reg_mul, accum, adder);
u6: reg GENERIC MAP(32) PORT MAP(add, clk, accum);
u7: buf PORT MAP(accum, out1);
END structure;

Structural Design
- Structural descriptions are categorised by the instantiation & interconnection of VHDL components. They can be viewed as VHDL netlists.
- The architecture's body instantiates as many declared components as needed, and connects those components by the use of the PORT MAP construct.
- A structural architecture for the mac entity is:

ARCHITECTURE structure OF mac IS
    COMPONENT reg
        GENERIC(width: integer := 16);
        PORT(d: IN bit_vector(width-1 DOWNTO 0);
             clk: IN bit;
             q: OUT bit_vector(width-1 DOWNTO 0));
    END COMPONENT;
    COMPONENT adder
        PORT(port1, port2: IN bit_vector(31 DOWNTO 0);
             output: OUT bit_vector(31 DOWNTO 0));
    END COMPONENT;
    BEGIN
        u1: reg GENERIC MAP(16) PORT MAP(in1, clk, reg_in1);
        u2: reg GENERIC MAP(16) PORT MAP(in2, clk, reg_in2);
        u3: multiply PORT MAP(reg_in1, reg_in2,mul);
        u4: reg GENERIC MAP(32) PORT MAP(mul, clk, reg_mul);
        u5: adder PORT MAP(reg_mul, accum, adder);
        u6: reg GENERIC MAP(32) PORT MAP(add, clk, accum);
        u7: buf PORT MAP(accum, out1);
    END structure;

Library/Use statement
- The Library Statement: A library can be referenced by the identifier. The name of the library must be made visible using the LIBRARY statement:

LIBRARY IEEE;
ENTITY test IS
END test;

- VHDL implicitly provides two library statements before every design unit, making the libraries STD and WORK available:

LIBRARY STD;
LIBRARY WORK;

- To make a package in a library visible to a design unit, the package must be specified with a USE statement including: name of a library followed by a dot ',', package name followed by a dot ',', and reference to a package element (type, constant, signal, function, etc.). It ends with semicolon ';'.

USE ieee.std_logic_1164.ALL;

- Provided implicitly: a USE statement that makes the STANDARD package from the STD library available to all design units:

USE STD.STANDARD.ALL;
The VHDL model is converted into a hardware structure with the help of synthesis tools.

First the VHDL model is mapped to a hardware structure described using cells from a technology library. Then the netlist is “placed and routed”.

Usually an optimizer is involved in generating the final result based on silicon area minimisation or speed considerations.

The VHDL code has to be written in a style that is implementable and generates reliable circuits.

Synchronous circuits are preferred.

Clock Buffering

- In FPGAs the clock tree is already designed and the clock distribution is dealt with automatically by the synthesis tool.
- For ASIC design it may be necessary to design the clock tree manually.
- It is important to avoid:
  - Clock skew generated by unequal depth of clock buffering.
  - Unequal load-dependent delays generated by unbalanced clock buffers fan-out.
  - Slow clock edges due to excessive buffer loading.

Correct Clock Buffering

- The circuit provides the same buffering depth at all clocked points;
- All buffers have the same fan-out;
- The buffers are lightly loaded (less than 50% of maximum fan-out).

Incorrect Clock Buffering
Shift Registers and Clock Buffering

- Shift registers are particularly sensitive to clock skew. The register operation could be incorrect due to set-up and hold problems.

Situations to be Avoided when Operating with the Asynchronous Reset

- It is recommended to avoid driving the asynchronous reset input of one flip-flop using the output of the other.

The Recommended Solution for flip-flops with Synchronous Reset

Synchronizing Asynchronous Inputs

- A signal conflict may arise at the interface between a synchronous circuit and an external asynchronous input. It is recommended to synchronize an asynchronous input by passing it through one or more flip-flops.

Adders and Multiplexers

```vhdl
architecture arch of entity1 is
begin
  outp <= a + b when sel = '0' else a + c;
end arch;
```
Modelling Sequential Circuits

- To model sequential logic for synthesis, the clock signal must be used, following some recommendations:
  - Designate a signal as a clock through its behavioural description (using IF or WAIT UNTIL statements).
  - Use at most one clock with at most one active edge.
  - All procedural statements must completed into a single clock cycle.
  - Data-dependent loops must be synchronised by clock.

### Modelling Clock Signals with WAIT Statements

<table>
<thead>
<tr>
<th></th>
<th>Positive clock edge</th>
<th>Negative clock edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait until</td>
<td>Wait until clk'event and clk = '1' ;</td>
<td>Wait until not clk'stable and clk = '0' ;</td>
</tr>
<tr>
<td></td>
<td>Wait until clk'event and clk = '0' and clk'last_value = '0' ;</td>
<td>Wait until clk'event and clk = '0' and clk'last_value = '1' ;</td>
</tr>
<tr>
<td></td>
<td>Wait until not clk'stable and clk = '1' ;</td>
<td>Wait until clk'event and clk = '0' and clk'last_value = '1' ;</td>
</tr>
<tr>
<td></td>
<td>Wait until clk'event and clk = '1' ;</td>
<td>Wait until clk'event and clk = '0' and clk'last_value = '1' ;</td>
</tr>
</tbody>
</table>

Some Remarks on VHDL Design for Synthesis

- VHDL elements which are not synthesizable: AFTER, WAIT FOR, ASSERT, File operations, REAL signals.
- Write all input signals in the sensitivity list, otherwise they will be latched.
- CASE and IF statements must be complete. For std_logic signals the CASE statement needs “OTHERS” and IF needs ELSE.
- Avoid instantiating too many components because this worsens the size of optimised implementation.
- Avoid large combinational multipliers and dividers. When complicated equations are used, design the circuit so that all the operands share a single multiplier or divider.
- Do not use INTEGER without RANGE:
  ```vhdl
  SIGNAL x: INTEGER RANGE 0 TO 255;
  ```
**Objectives**

- To ensure a more automated and less intuitive approach for the design of FPGA-based control systems
- Reduction of the development time
- Development of a specific library of reusable modules dedicated to the control of electrical systems
- First attempt success guarantee of the designed architecture

**Different steps**

1. Modular partitioning of the algorithm
2. Simulation procedure
3. Optimization of the consumed resources
4. Architecture design
5. Validation of the architecture

**Continuous Model of a FOC Estimator**

Example: \( \frac{dX}{dt} \approx \frac{(X[k+1]-X[k])}{Ts} \)
Continuous Model of a FOC Estimator

\[ i_{sd} = \frac{2}{\sqrt{3}} \sin(\theta + 60) i_{1s} + \sin(\theta) i_{2s} \]

\[ i_{sq} = \frac{2}{\sqrt{3}} \sin(\theta + 150) i_{1s} + \cos(\theta) i_{2s} \]

\[ \Phi_r = \frac{L_m}{(1 + T_r s)} i_{sd} \]

\[ W_{dq} = w + L_m i_{sq} / (T_r \Phi_r) \]

\[ C_{em} = \frac{3}{2} p \frac{L_m}{L_r} \Phi_r i_{sq} \]

\[ \Phi_{sq} = L_s \sigma_{sq} + (L_m / L_r) \Phi_r \]

\[ \Phi_{eq} = L_s \sigma_{eq} \]

\[ d\theta_{dq}/dt = w_{dq} \]

Digital Model of a FOC Estimator

\[ i_{sd}[k] = a_0 (\sin(\theta[k] + 60) i_{1s}[k] + \sin(\theta[k]) i_{2s}[k]) \]

\[ i_{sq}[k] = a_0 (\sin(\theta[k] + 150) i_{1s}[k] + \cos(\theta[k]) i_{2s}[k]) \]

\[ \Phi_r[k] = a_2 i_{sd}[k-1] + a_3 \Phi_r[k-1] \]

\[ W_{dq}[k] = a_4 w[k] + a_6 i_{sq}[k] / \Phi_r[k] \]

\[ C_{em}[k] = a_6 \Phi_r[k] i_{sq}[k] \]

\[ \Phi_{sq}[k] = a_7 i_{sd}[k] + a_8 \Phi_r[k] \]

\[ \Phi_{eq}[k] = a_9 i_{sq}[k] \]

\[ \theta_{dq}[k] = \theta_{dq}[k-1] + a_{10} w_{dq}[k-1] \]

Design Methodology

Continuous Model

Discrete Model

Per Unit Discrete Model

Euler: \[ dX/dt \approx (X[k+1] - X[k])/Ts \]
Per Unit Digital Model of a FOC Estimator

\[ i_{sd}[k] = A_0 (sin(\theta[k]+150) + \frac{i_{is}[k]}{\Phi_b}) + i_{is}[k] \]
\[ i_{sq}[k] = A_1 (sin(\theta[k]+60) + \frac{i_{is}[k]}{\Phi_b}) + i_{is}[k] \]
\[ \Phi_r[k] = A_2 i_{sd}[k-1] + A_3 \Phi_r[k-1] \]
\[ W_{sq}[k] = A_4 w[k] + A_5 i_{sq}[k] \]
\[ C_{en}[k] = A_6 \Phi_r[k] i_{sq}[k] \]
\[ \Phi_{sd}[k] = A_7 i_{sd}[k] + A_8 \Phi_r[k] \]
\[ \Phi_{sq}[k] = A_9 i_{sq}[k] \]
\[ \theta_{dq}[k] = \theta_{dq}[k-1] + A_{10} w_{dq}[k-1] \]

Example 1: FOC Estimator algorithm

abc to dq Transformation

\[ i_s[k] \rightarrow \theta_{dq} \rightarrow \omega_{dq} \rightarrow \Phi_{dq} \rightarrow \Phi_{sq} \]

Low Pass Filter

Integrator

Example 2: Sliding Mode Torque Control algorithm

Hysteresis Controller

Current Control, Torque Control

Full Control Algorithms

Level 3

Level 2

Level 1

Basic Operators

Arithmetic Operators
Design Methodology

Library of IP modules dedicated to the Control of Electrical Systems
- VHDL Programs
- Matlab Simulink Models
- Data-sheets

Features:
- Name of the module / PV50
- Generic VHDL model for use with different types of IFO (VSI)

Functional Description
This IP is a Space Vector Modulation SVM with additional zero sequence signal (ZSS) module. It is used for the control of a three phase inverter. The additional zero sequence value in a 3-phase and multiple of 20 harmonic, which does not produce phase voltage distortion and set affect load average currents. The SVM module accepts signed input reference values $V_{ref}$ and $V_{qref}$ with a generic fixed point format and it generates the three phase switching states $C_{a}$, $C_{b}$, and $C_{c}$ for the control of the power converter. Figure 1 shows the corresponding functional description.

Example 1: Per Unit FOC Estimator functional model

Design Methodology

Modular partitioning of the algorithm
- Verification of the algorithm functionality
- Choice of the suitable sampling period and fixed-point format

Verification of the algorithm functionality

Example 1: Per Unit FOC Estimator functional model

Induction motor

Estimator algorithm

IFO Controller +
VSI

Simulation procedure

First level library
- Flux
- Flux vector estimation
- Park transformation

Module blocks
- Space Vector Modulation
- Park Transformations

Transformation blocks
- Power to digital
- Digital to analog
- Volt-Ampere transformation

Extension blocks
- Flux and torque estimation of an induction motor based voltage model
- Flux or current based voltage model
Example 2: Per Unit Sliding Mode Torque Control functional model

Choice of the sampling period and fixed-point format of the digital algorithm

Example 1: FOC Estimator specification model

Example 2: Sliding Mode Torque Control specification model
Design Methodology

**dq-to-abc transformation**

**specification model**

Lots of possibilities in terms of parallelism

**Factors**

• Hardware resources
  • Execution time

**Generation of optimized hardware architecture (A³ methodology)**

<table>
<thead>
<tr>
<th>Operation</th>
<th>DFG</th>
<th>FDFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Multiplication</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Sine</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

**abc-to-dq transformation**

**Factorized Data Flow Graph (FDFG)**
Design Methodology

1) Modular Hardware Architecture Design

2) Design of the whole algorithm architecture

Example: FOC Estimator algorithm architecture
Design Methodology

3) VHDL coding of the architecture

```vhdl
library IEEE;
useIEEE.STD_LOGIC_1164.ALL;
useIEEE.STD_LOGIC_ARITH.ALL;
useIEEE.STD_LOGIC_UNSIGNED.ALL;

entity test is
generic(integer_n: integer);
    Port ( clk : IN std_logic;
        reset : IN std_logic;
        en : IN std_logic;
        t : OUT std_logic);
end test;

architecture Behavioral of test is
    component dir_tin
        generic (n: integer :=4; integer_c:=16);
        Port ( clk : IN std_logic;
            reset : IN std_logic;
            clk_out : INOUT std_logic)
    end component;
    component dir_out
        generic (n: integer :=4; integer_c:=16);
        Port ( din : IN std_logic_vector(n-1 downto 0);
            en : IN std_logic;
            sum : OUT std_logic;
            clk : IN std_logic;
            dout : OUT std_logic_vector(n-1 downto 0)
        )
    end component;
end Behavioral;
```

Design Methodology

3) VHDL coding – TOP DOWN Approach

- System Level
  - Simulation Environment
  - Analog HDL
  - Behavioral HDL
  - Synthesis

- Behavioral Level
  - RTL or Synthesis Level

- RTL or Synthesize Level
  - Reuse and IP RTL or Synthesize Model Blocks

- Physical Level
  - FPGA
  - ASC

Design Methodology

3) VHDL coding - Reusability

- System Level
- Behavioral Level
- RTL or Synthesize Level
- Physical Level

- Reuse and IP Behavioral Model Blocks
- Reuse and IP RTL or Synthesize Model Blocks

- Modular partitioning of the algorithm
- Simulation procedure
- Optimization procedure
- Architecture Design
- Validation of the architecture
Design Methodology

1) Hardware in the loop test

First attempt success guarantee

Example: Hardware in the loop results of the FOC Estimator

Start-up and a speed reversal at 0.27s of a 1 Kw induction machine controlled by a classical Indirect Field Oriented Strategy.

Simulation results ($i_{sd}(A)$ and $i_{sq}(A)$)

Hardware in the loop results ($i_{sd}(A)$ and $i_{sq}(A)$)

Slight difference between simulation and hardware in the loop results
Design Methodology

2) Experimental test

FPGA (Actel Fusion)

Conclusion

- **Advantages**: 
  - Less intuitive and more automatic approach 
  - Reduction of the development time 
  - Optimization of the consumed resources 
  - Reusability of the design 
  - Development of a specific library 
  - First attempt success guarantee 

1st Case Studies Series:

FPGA-Based Current Controllers for Synchronous Machine Drive

Advantages & Features
Control Algorithm Execution Time

(a) General purpose microcontroller: 
- μc limitations!

(b) DSP controller: 
- VSI limitations

(c) FPGA-based controller: 
- Quasi-analog behavior

Experimental Set-up

- Voltage Source Inverter
- Current sensors
- SM
- Encoder
- Host PC
- RS232
- VSI Interface Board
- FPGA Spartan3
- 400,000 Gates
- AD Conversion Board
Current Controllers Based on ON-OFF Regulators

Two Main groups

- Variable switching frequency ON-OFF regulators
  - Well adapted for analog controls

- Limited switching frequency ON-OFF regulators
  - Well adapted for analog & digital controls

Example 1: Independent three phase free running hysteresis regulators

\[ T_S = T_{AD} = 2.4 \mu s \]
\[ T_{ec} = T_{AD} + t_{IP} + t_{H2} = 2.74 \mu s \]

Controller computation time
Variable Switching Frequency ON-OFF regulators

Low execution time
Effects of sampling and delays are very negligible

Example 2: Space vector based regulator with three level hysteresis comparators and look-up table working in the \( \alpha-\beta \) reference frame

\[ T_S = T_{AD} = 2.4 \mu s \]
\[ T_c = T_{AD} + t_{IP} + t_c + t_{H3} + t_T = 2.92 \mu s \]

Controller computation time

Global control unit FSM

Hardware Architecture

\[ \Delta \text{H} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]

\[ \Delta \text{h} \]
Variable Switching Frequency ON-OFF regulators

Low execution time
Effects of sampling and delays are very negligible

Execution time = 50 µs
Execution time = 2.92 µs

Limited Switching Frequency ON-OFF regulators

✓ Example 1: Independent three phase free running hysteresis regulators

Global control unit FSM

Hardware Architecture

✓ $T_S = T_{AD} = 100$ µs
✓ $T_{ox} = T_{AD} + T_IP + T_H2 = 2.74$ µs

Controller computation time
Limited Switching Frequency ON-OFF regulators

Low execution time
Effects of sampling and delays are very negligible

Execution time = 50 µs
Execution time = 2.92 µs

Example 2: Space vector based regulator with three level hysteresis comparators and look-up table working in the α-β reference frame

THD=14.9%
THD=8.9%

Square current vector error ($\Delta i_\alpha^2 + \Delta i_\beta^2$)

THD=11.1%
THD=8.9%

Square current vector error ($\Delta i_\alpha^2 + \Delta i_\beta^2$)
Current Controller Based on PI Controllers

Hardware Architecture

Case 1: Synchronized PWM

\[ T_S = \frac{T_{PWM}}{2} \]

\[ T_{ex} = t_{ID} + t_{VC} = 3.28 \mu s \]

Vector Control computation time
**Current Controller Based on PI Controllers**

**Case 1: Synchronized PWM**

1. Ch 1: 200 mVolt 250 µs
2. Ch 2: 2 Volt 250 µs

**Case 2: Non-Synchronized PWM**

1. Ch 1: 200 mVolt 10 µs
2. Ch 2: 2 Volt 10 µs

**Vector Control computation time**

- $T_S = 5$ µs
- $T_{ex} = t_{AD} + t_{VC} = 3.28$ µs

**Experimental Results**

- Carrier Frequency = 1KHz
  - THD = 11.1%

- Carrier Frequency = 3KHz
  - THD = 4.1%
Predictive Current Controller

Analytical State model of the synchronous machine in the \(dq\) rotor reference frame:

\[
\begin{bmatrix}
\frac{di_{sd}}{dt} \\
\frac{di_{sq}}{dt}
\end{bmatrix} = \begin{bmatrix}
\frac{1}{T_{sd}} & \frac{L_{sd}}{L_{sq}} \omega(t) \\
\frac{L_{sq}}{L_{sd}} \omega(t) & -\frac{1}{T_{sq}}
\end{bmatrix} \begin{bmatrix}
i_{sd} \\
i_{sq}
\end{bmatrix} + \begin{bmatrix}
1/T_{sf} & 0 \\
0 & 1/T_{sf}
\end{bmatrix} \begin{bmatrix}
M_{sq} \omega(t) \\
M_{sd} \omega(t)
\end{bmatrix} \begin{bmatrix}
V_{sd} \\
V_{sq}
\end{bmatrix}
\]

Digital Prediction Equations:

\[
i_{sd}[k+1] = \frac{T_{sf}}{L_{sd}} (V_{sd}[k] - e_{sd}[k]) + (1 - \frac{T_{sf}}{T_{sd}}) i_{sd}[k]
\]

\[
i_{sq}[k+1] = \frac{T_{sf}}{L_{sq}} (V_{sq}[k] - e_{sq}[k]) + (1 - \frac{T_{sf}}{T_{sq}}) i_{sq}[k]
\]

where

\[
e_{sd}[k] = -L_{sd} \omega \phi_{sd}[k]
\]

\[
e_{sq}[k] = L_{sq} \omega \phi_{sq}[k] + M_{sq} \omega \phi_{sq}[k]
\]

7 different stator voltage vectors \(V_{sdq} = [V_{sd} V_{sq}]^T\) \(j=0..7\):

\[
\begin{bmatrix}
V_{sd} \\
V_{sq}
\end{bmatrix} = \begin{bmatrix}
\cos(\theta_{sd}) & \sin(\theta_{sd}) \\
-\sin(\theta_{sd}) & \cos(\theta_{sd})
\end{bmatrix} \begin{bmatrix}
V_{sd}^* \\
V_{sq}^*
\end{bmatrix}
\]

7 different directions \(t_j (j=0..7)\) and errors \(\Delta t_j (j=0..7)\):

\[
i_{sd}[k+1] = \tilde{i}_{sd}[k+1] - \tilde{i}_{sd}[k]
\]

\[
\Delta i_{sd}[k+1] = \tilde{i}_{sd}^*[k] - \tilde{i}_{sd}[k+1]
\]

\[
i_{sq}[k+1] = \tilde{i}_{sq}[k+1] - \tilde{i}_{sq}[k]
\]

\[
\Delta i_{sq}[k+1] = \tilde{i}_{sq}^*[k] - \tilde{i}_{sq}[k+1]
\]

where

\[
e_{sd}[k] = -L_{sd} \omega \phi_{sd}[k]
\]

\[
e_{sq}[k] = L_{sq} \omega \phi_{sq}[k] + M_{sq} \omega \phi_{sq}[k]
\]

Example of different prediction possibilities.
Predictive Current Controller

**Predictive Controller Principle**

- Predictive Controller Principle
- Limited Switching Frequency ON-OFF regulators
- Hardware architecture
- Global control unit FSM
- \( T_s = 100 \mu s \)
- \( T_{ex} = t_{AD} + t_{Pr} = 4.52 \mu s \)
- Predictive Controller computation time
- Current Controller Based on PI Controllers
- Experimental Results

**Experimental Results**

- Experimental Results
- THD=8.8%
- THD=8.8%
- THD=8.8%
FPGA-Based speed control for Synchronous Machine Drive using PPI controller

Problem Positioning

Objective: Development of a high performance FPGA-based speed controller

Most important criteria for the speed control:
- Fast speed dynamic
- Accurate speed response
- Quick speed recovery from disturbances

Speed Controller Design

Synchronous machine model

\[ V_{sd} = R_{sd}i_{sd} + \frac{d\phi_{sd}}{dt} - \omega \phi_{sq} \]
\[ V_{sq} = R_{sq}i_{sq} + \frac{d\phi_{sq}}{dt} + \omega \phi_{sd} \]
\[ \phi_{sd} = L_{sd}i_{sd} + M_{sr}i_{rd} \]
\[ \phi_{sq} = L_{sq}i_{sq} \]
\[ \phi_{rd} = L_{rd}i_{rd} + M_{sr}i_{sd} \]
\[ T_e = \frac{3}{2} p(\phi_{sd}i_{sq} - \phi_{sq}i_{sd}) \]

Speed Controller Design

Current controller
**Current controller**

Current Controller Timing Diagram

- $T_s = 100 \mu s$
- $T_{ex} = 2.74 \mu s$

Sample
- $i_{sa,b}[k]$
- $\theta_{sa,b}[k]$

Application
- $S_{sa,b}[k]$
- $S_{sa,b}[k+1]$

**Speed controller synthesis**

- **Internal Loop (Proportional controller)**
  - Impose the controlled system poles at the desired positions
  - Internal speed control loop transfer function

**PI controller**

- Zero steady-state error
- Impose the shape and the dynamic of the speed response

**External Loop (PI controller)**

- $\frac{\omega_s}{\omega_s} = \frac{1.5K_p M_s i_{sa}}{s + \frac{1}{T_m}}$
- $\xi = 1$

Roots Locus

- $\frac{\omega_s}{\omega_s} = \frac{1}{s + \frac{1}{2T_m}}$

**speed**
Speed Controller Design

- **Speed controller synthesis**

- **Speed estimator design**
  - Backward difference
  
  \[ \omega[k] = \frac{1}{1024} \frac{\theta[k] - \theta[k-1]}{T_k} \]

  - Variable sampling period
  - Operating mode synchronized with state changes of the LSB of the encoder

  Determined via the state changes of the two LSB P0 and P1

  \[ \omega[k] = \text{Sense} \frac{4\pi}{1024} \frac{1}{T_k} \]

  Denotes the time spent for one unit displacement of the encoder

---

**Speed estimator design**

**Sense Computation**

<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Sense</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Positive</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Negative</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Positive</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Negative</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Positive</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Negative</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Positive</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Negative</td>
</tr>
</tbody>
</table>

**Tk Computation**

\[ T_k = \frac{n_c}{F_c} \]

Determined via the state changes of the two LSB P0 and P1

\[ \omega[k] = \text{Sense} \frac{4\pi}{1024} \frac{1}{T_k} \]

Denotes the time spent for one unit displacement of the encoder

---

**Speed Estimator Hardware Architecture**

**Data-Path**

**Control Unit**

**Reset**

**Sense Computation**

**Variable sampling period**

**Operating mode synchronized with state changes of the LSB of the encoder**

**Denotes the time spent for one unit displacement of the encoder**
**Speed Controller Design**

**Speed controller architecture**

![FPGA-based Speed controller diagram](image)

**Speed Controller Timing Diagram**

- $T_s = 100 \mu s$
- $T_{\text{ex}} = 3.45 \mu s$

Sample $i_{s_a}[k]$ Application $S_{s_a,[k]}$ Sample $i_{s_b}[k+1]$ Application $S_{s_b,[k+1]}$

Note: The speed estimator works independently from the other modules and is synchronized to the state changes of the LSB of the encoder.

**Experimental Results**

- **Step Speed Response**
  - $\omega = 200 \text{ rad/s}$
  - $\omega = 0 \text{ rad/s}$

- **Speed tracking performance**
  - $\omega = 200 \text{ rad/s}$
  - $\omega = 0 \text{ rad/s}$

**Experimental Set-up**

- 400V/50Hz
- **Encoder**
  - Interface
  - Amplification
  - A/D
- **Speed Controller**
  - Interface
  - Speed Control unit
  - AD Interface
- **FPGA-based Speed controller**
  - Global control unit
  - AD Interface

**Experimental Results**

- **Current waveforms**
  - Response to a step of a rated load torque
    - $T_L = 5 \text{ Nm}$

- **Current waveforms for a reversal speed operation**

**References**

- Spartan3 Xc3s400 (400,000 gates)
- RS232
Conclusions

- A full FPGA-based speed controller for SM drive has been presented
- A very efficient P-PI speed regulator has been synthesized
- An original speed estimator has been developed, it allows to obtain the best accuracy
- An original speed estimator has been developed, it allows to obtain the best accuracy
- The obtained experimental results give proof of the ability of the developed speed control system to achieve an efficient and robust speed control under different operating conditions

Induction Motor Experimental Set-up

- Interface and Control Boards
- IM
- VSI
- AD Converters Board
- FPGA Spartan 2
- 100.000 Gates
- References RS232
- UART
- FPGA
- VSI Interface
- AD Interface
2nd Case Studies Series:

FPGA-based Intelligent Controllers for AC Drives and AC Generators:

* A PWM control system modelling / design / FPGA implementation using VHDL

✓ Modelling an induction motor drive system using an FPGA PWM neural controller

✓ Modelling a diesel driven generator employing fuzzy-logic/PWM FPGA control
Complete VHDL Code - 1 Phase PWM Generator

```
library ieee;
use ieee.std_logic_1164.all;
entity pwm is
  port(out_signal: out std_logic;
       clock,start: in std_logic;
       Max_count: in integer);
end pwm;
architecture behav of pwm is
  signal counter_out_bus : integer;
  signal next_pulse,reset: std_logic;
  signal val_max, adr, data: integer;
  begin
    counter_rev: process(clock,reset)
      variable direction: std_logic := '1';
      variable v: integer := 0;
      begin
        if reset'event and reset='1' then
          v:=1;
        elsif clock'event and clock='1' then
          if direction='1' then
            if v<val_max then
              v:=v+1;
            else
              v:=v-1;
              direction:='0';
            end if;
          else
            if v>-val_max then
              v:=v-1;
            else
              v:=v+1;direction:='1';
            end if;
          end if;
          counter_out_bus<=v;
        end if;
    adr_gen: process (next_pulse,reset)
      begin
        if adr >=0 and adr<18 then
          data<=d(adr);
        else
          data<=0;
        end if;
      end process;
    control: process(start,clock,counter_out_bus,
                        Max_count)
      variable temp_Max_count: integer;
      begin if Max_count'event then
        temp_Max_count:=Max_count;
      end if;
      if start='1' and start'event then
        temp_Max_count:=Max_count;
        val_max<=temp_Max_count;
        reset<='1';
        next_pulse<='0';
      elsif start='1' and clock='0' and
            clock'event then
        if reset='1' then
          adr<=0;
        elsif next_pulse'event and
               next_pulse='1' then
          adr<=(adr+1) mod 18;
        end if;
      end if;
      end process;
    comparator:
      process(counter_out_bus,data)
      begin
        if data<counter_out_bus then
          out_signal <= '0';
        else
          out_signal <= '1';
        end if;
      end process;
    memory: process(adr)
      type mem_data is array (0 to 17) of integer;
      variable d : mem_data := (-250,-230,-190,-100,0,100,190,230,250,250,230, 90,100,0,-100,-190,-230,-250);
      begin
        if direction='1' then
          if v<val_max then
            v:=v+1;
          else
            v:=v-1;
            direction:='0';
          end if;
        else
          if v>-val_max then
            v:=v-1;
          else
            v:=v+1;direction:='1';
          end if;
        end if;
      end process;
    adr_gen: process (next_pulse,reset)
      begin
        if reset='1' then
          adr<=0;
        elsif next_pulse'event and
               next_pulse='1' then
          adr<=(adr+1) mod 18;
        end if;
      end process;
```
Architecture arch_test of test is
component pwm
  port(out_signal : out std_logic; clock,start: in std_logic; Max_count: in integer);
end component;
signal clock: std_logic := '0';
signal start: std_logic := '0';
signal Max_count: integer;
signal out_signal: std_logic;
begin
  Max_count<=300, 500 after 40 ms;
clock <= not clock after 1 us;
start <= '1' after 5 ns;
x: pwm port map(clock=>clock,start=>start, Max_count=>Max_count,
  out_signal=>out_signal);
end arch_test;
configuration conf_test of test is
for arch_testend for;
end conf_test;

Achievements

- Original design of a 3-phase PWM generator, successfully modelled, designed and simulated using VHDL.
- Important original aspect = the (64x2) ROM, efficiently targeted by a three-way routing circuit.
- The functional simulation results proved the correct controller operation, followed by practical tests that validated the circuit.
- A plethora of other synchronous and asynchronous modes can be tested, since the circuit is very flexible in producing a range of carrier frequencies between 5 KHz to 427 KHz.
- The amplitude modulation index, which was set to 1 for this particular test, can be varied in the full range between 1 and 0.
- In terms of silicon usage, the circuit proposed is the optimum choice. The memory size is the minimum one required to efficiently describe the 0-phase sinewave.

Simulation Results of the 3-phase PWM circuit
### The RLe Equivalent Circuit of the Induction Motor

\[
L = L_s - L_m^2/L_r
\]

\[
R = R_s
\]

\[
e = \frac{L_m}{L_r} \left( R_r i_r^s + j\omega_r \left( L_r i_r^s + L_m i_m^s \right) \right) = \frac{L_m}{L_r} \left( -R_r i_r^s + j\omega_r \Psi_r^s \right)
\]

\[
u = u_s
\]

\[
i = i_s
\]

---

### Speed Control Principles

- The control is achieved in polar coordinates (module and angle).
- The rotor speed is controlled by compensating the slip frequency.
- Slip frequency is kept constant for any load torque & any rotor speed.
- The slip frequency depends on the angle \( \alpha \) between \( e \) and \( i_s^r \); controlled by means of:
  - stator frequency
  - stator current amplitude
- The current amplitude \( I_s \) is corrected according to the position of vector \( e \) in the complex plane.
- The stator frequency \( f_s \) follows the reference speed profile.
- Very fast stator frequency changes have to be avoided because they cause slow transient response.

---

### The Neural PWM Controller

#### Basic Control Algorithm

#### Improved Control Algorithm

#### Simulation
Neural PWM Controller – VHDL Design for Implementation

Test Results - Motor

Achievements

- Induction motor drives can be controlled using neural algorithms, implying a smaller number of calculations than vector control.
- The proposed speed control algorithm can be expressed as a set of mathematical equations written in polar co-ordinates.
- The angle and sector calculations are carried out by hardware implemented neural networks.
- The entire control scheme has been modelled and designed in VHDL, synthesised and implemented into Xilinx XC4010 FPGA.
- The implementation offers a cost-effective solution for industrial applications without high dynamic requirements.
- Test results have confirmed correct operation of the controller.
2. MODELLING A STAND ALONE DIESEL DRIVEN GENERATOR SET USING FUZZY-LOGIC AND PWM CONTROL

Project Background

♦ In a given synchronous machine the operational speed is dependent on the desired output frequency.

♦ Variable speed operation of generators increases design freedom: speed is not determined by the desired electrical frequency.

♦ It allows engine-generators systems to be operated at speeds which optimise desired parameters such as noise, vibrations, fuel efficiency, engine emissions.

♦ The research aim is to design and build a control system for a stand alone variable speed PM synchronous generator.

♦ This has been developed on the basis of fuzzy logic, using VHDL and is implemented in Xilinx FPGA.

Fuzzy Variable Speed Governor

• Fuzzy Variable Speed Governor (FVSG) - controller based on fuzzy logic.
• Designed using VHDL for easy correction and future integration with other components to extend the system.
• System configuration allows variable speed operation of the generator.
DC Voltage Fuzzy Controlled Response to Load Current Step Increase

Experimental Test Results

Without controller
With controller

Step Change in a.c. Load Current – d.c. Voltage response

Achievements

PWM controller
• voltage control using PWM is a simple and effective strategy for obtaining and maintaining the desired output voltage parameters.

Fuzzy logic
• an effective design solution for the speed governor.
• able to produce a competent control system without the need for a precise mathematical model of the plant.
• the controller is reconfigurable by changing the rule base.
• design can be easily extended to include more parameters.

VHDL
• design, modelling & simulation performed on a single platform
• the same design tool can be used for hardware implementation
• reusable design modules are produced
• new developments of the design can easily be performed

General Conclusions

• A novel modelling technique is proposed for the holistic investigation of engineering systems. This is based on Hardware Description Languages (VHDL).
• The sample systems were developed from idea, through modelling / simulation, to complete systems commissioning, in short time, giving further advantages:
  ✓ easy integration of electronic controllers in complex engineering system models.
  ✓ reliable framework for design verification
  ✓ high confidence in correct first time operation
  ✓ allows rapid FPGA prototyping of electronic controllers
  ✓ gives multiple choices for controller’s final implementation technology
  ✓ high degree of flexibility

• A CAD platform independent model & design are developed and therefore valuable IPs can be produced, in co-relation with the modern principles of design reuse.
• Concurrent engineering basic rules (unique EDA environment and common design database) are fulfilled.
• Estimation: HDL based holistic modelling methodologies will be increasingly used in the future and expanded to encompass other areas of engineering systems.
Recent Developments

- **Handel-C** – a novel compiler for Hardware-Software co-design from Celoxica.

![Diagram of Handel-C compiler]

- **Rosetta** is a language for modelling/describing engineering systems
  - Presently the focus is on complex **electronic** systems -> SOC
  - Being explored for complex **mechanical** systems
  - Defines systems by writing and composing models with respect to domains.
  - Consists of a **syntax** *(a set of legal descriptions)* and a **semantics** *(a meaning associated with each description)*

- **Millenium Machine** – new EPSRC (UK) funding initiative for holistic modelling of engineering systems (systems of systems).

General Conclusions

- The simultaneous increase of the control algorithm complexity and the chip density implies the use an efficient design methodology.
- A modeling technique is proposed for the holistic investigation of power electronic systems. This is based on System Level Modeling Languages or HDL and allows rapid FPGA prototyping of the control systems.
- Three main design rules are presented.
  - the algorithm refinement,
  - the modularity,
  - the systematic search for the best compromise between the control performances and the architectural constraints (see A3 section).
- Full and timely examples are presented to illustrate the benefits of FPGA implementation when using the proposed design approach.
- It is demonstrated that in both cases a low cost FPGA-based controller can greatly improve the control performance, especially due to the reduction of execution time, while keeping a high level of flexibility.

Perspectives

- In the near future, the complexity of the control systems will continue to grow.
- The tasks devoted to the control algorithm will no longer be limited to regulation but will have to manage diagnosis and fault-adaptive on line control.
- The research effort on the theory and the applications of dynamic reconfiguration is crucial.
- **Network-on-a-Chip (NoC)**
- **SoC** design that can include digital control and its analog interface (sensors, ADC, power drivers, etc.).
- Co-design issue must be addressed, since the borders between software and hardware are rapidly vanishing. The main problem in this case is to propose automatic rules of partitioning, based on relevant quantitative indicators.
- Another interesting direction of research is based on the following observation: a control algorithm, when implemented in an FPGA, can have a very short execution time due to the high degree of parallelism of its architecture. At the same time, the constraints imposed by the power electronic components imply a sampling period that is much higher than the execution time. The resulting "wasted time" could be advantageously employed.
- Several examples of relevant FPGA utilizations in this context were presented. They consist of predictive control, over-sampling strategies, multi-plants control, etc. All these very promising control paradigms must still be improved.

Bibliography

Bibliography


