Integration and assessment of a dual core chip - Atmel’s DIOPSIS 940 - for a flight control system.

Lukasz Majewski

Reg Nr: LIU-IDA/LITH-EX-A–09/040–SE
Linköping 2009
Master’s Thesis

Integration and assessment of a dual core chip - Atmel’s DIOPSIS 940 - for a flight control system.

Lukasz Majewski

Reg Nr: LIU-IDA/LITH-EX-A--09/040--SE
Linköping 2009

Supervisor: Professor Patrick Doherty
IDA, Linköpings universitet

Examiner: Professor Patrick Doherty
IDA, Linköpings universitet

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden
Integration and assessment of a dual core chip - Atmel’s DIOPSIS 940 - for a flight control system.

Lukasz Majewski

A dual core Atmel DIOPSIS 940 chip consists of a DSP and an ARM9 functional units in a single silicon die. This thesis presents the process of integration and assessment of using this processor in a flight control system. A complete design of the system is provided including a description of the DIOPSIS 940 from the perspective of requirements of the application. The integration of the processor with a typical set of components of a flight control system is provided. Additionally, a suite of programs required for developing software for the system is included. Capabilities of both cores of the processor are analysed in a series of experiments. Computational performance in typical tasks of a flight control system is analyzed and compared. The application of attitude stabilization for a micro-scale UAS is described.
Abstract

A dual core Atmel DIOPSIS 940 chip consists of a DSP and an ARM9 functional units in a single silicon die. This thesis presents the process of integration and assessment of using this processor in a flight control system. A complete design of the system is provided including a description of the DIOPSIS 940 from the perspective of requirements of the application. The integration of the processor with a typical set of components of a flight control system is provided. Additionally, a suite of programs required for developing software for the system is included. Capabilities of both cores of the processor are analysed in a series of experiments. Computational performance in typical tasks of a flight control system is analyzed and compared. The application of attitude stabilization for a micro-scale UAS is described.
Acknowledgments

I would like to express my gratitude to professor Patrick Doherty for giving me the opportunity to work on this interesting topic.

I would like to thank Alberto Dell’Olio and Antonio Costa from Atmel Roma DSP division for sharing their experience in programming the mAgicV DSP processor and the Linux operating system.

Last but not least, I would like to show my appreciation to Piotr Rudol and Mariusz Wzorek for sharing their experience in design, integration and programming of flight control systems.
Contents

1 Introduction 3
   1.1 Background .............................................. 3
   1.2 Goals ..................................................... 3
   1.3 Content ................................................... 4
   1.4 Contribution .............................................. 5

2 Overview of the existing flight control systems 7

3 DIOPSIS 940 flight control system overview 11

4 Hardware 13
   4.1 DIOPSIS 940 overview ..................................... 13
   4.2 DIOPSIS 940 CPU Module (DCM) ........................... 16
   4.3 DIOPSIS 940 Back Module (DBM) ........................... 17
   4.4 Servo Control System (SCS) ............................... 19
      4.4.1 Motivation ............................................. 19
      4.4.2 Architecture .......................................... 20
      4.4.3 Output data format .................................... 24
   4.5 System integration ......................................... 25

5 Software 27
   5.1 Development software ..................................... 27
   5.2 Real-time Linux ............................................ 29
      5.2.1 Control application .................................... 30
      5.2.2 Real-time device drivers ............................... 32

6 DIOPSIS 940 evaluation 35
   6.1 Software ................................................... 35
      6.1.1 Matrix multiplication .................................. 35
      6.1.2 Complementary filter ................................... 42
      6.1.3 State estimation based on a Kalman filter .......... 46
   6.2 Hardware ................................................... 52
      6.2.1 Interrupt latency evaluation .......................... 52
      6.2.2 DIOPSIS 940 flight control system power consumption . . 55
7 Summary

7.1 DIOPSIS 940 Advantages ........................................ 57
7.2 DIOPSIS 940 Disadvantages ...................................... 57
7.3 Final conclusion ...................................................... 58
7.4 Future work .......................................................... 59

Bibliography ............................................................... 61
List of Figures

1.1 The LinkMAV coaxial rotor helicopter [26]. .......................... 4
3.1 System block diagram. ................................................. 12
4.1 The AT572D940HF block diagram [7]. ............................. 14
4.2 The Atmel’s DIOPSIS 940 Development Board (DEB) [6]. ..... 16
4.3 DIOPSIS 940 Back Modules ........................................... 18
4.4 Servo Control System (SCS). .......................................... 19
4.5 Servo Control System block diagram. ............................... 21
4.6 Servo Control System - input block (s_in_block). ............... 22
4.7 Servo Control System - output block (s_out_block). ........... 23
4.8 Servo Control System - output data format. ...................... 24
4.9 The DIOPSIS 940 control system PCB boards combined together. 25
4.10 The DIOPSIS 940 control system integrated with the LinkMAV platform. .................................................. 26
5.1 Static cyclic schedule. .................................................. 30
5.2 The LinkMAV helicopter multiplexer and actuators. .......... 31
6.1 The mAgicV DSP 40 bits floating point format. .................... 36
6.2 Single precision 32 bits (ARM9) floating point format. ...... 36
6.3 Double precision 64 bits (ARM9) floating point format. ..... 37
6.4 Execution time of square matrices multiplication. ............ 38
6.5 The mAgicV DSP execution time improvement over ARM9. .. 39
6.6 Relative error of square matrices multiplication. ............... 40
6.7 Complementary filter - block diagram. ............................ 42
6.8 Complementary filter - pitch (φ) angle. .......................... 44
6.9 Complementary filter - roll (θ) angle. ............................ 45
6.10 Navigation system block diagram. .................................. 46
6.11 Kalman filter output data ............................................ 50

List of Tables

5.1 Control system tasks timing benchmarks. ......................... 32
6.1 Complementary filter timing benchmarks. ......................... 44
6.2 Kalman filter timing benchmarks. .................................. 47
6.3 The interrupt execution delay of the “PREEMPT_RT” patched Linux operating system. ................................. 53
6.4 The interrupt execution delay of the Linux operating system. 54
6.5 The comparison of flight control systems power consumption. 55
Chapter 1

Introduction

1.1 Background

Unmanned Aircraft Systems (UAS) are becoming more and more popular both in military and civilian applications, performing tasks such as reconnaissance, surveillance or rescue missions.

The Autonomous Unmanned Aircraft Systems Technologies Lab (UASTech-Lab) [21] at the Artificial Intelligence and Integrated Computer Systems Division (AIICS) [12] at the Department of Computer and Information Science (IDA) has been conducting research dealing with many aspects of UAS systems for over a decade. The areas of interest include a wide range of topics. From platform design, low-level control, development of control systems to high-level artificial intelligence techniques such as task and motion planning or execution monitoring. The results of the research have been verified on several UAS platforms in flight experiments. In large scale on the Yamaha R-MAX helicopter and in the micro-size on fixed-wing and rotor based airframes.

Small platforms pose a very challenging problem of miniaturisation of sensors and computer systems. The need for high computational capabilities and low power consumption requires evaluating new systems based on innovative processors. One of them, the DIOPSIS 940 from Atmel, has been evaluated in order to assess the feasibility of using it as a flight control system. Its features are very appealing for such a task since it combines an ARM9 core and a Digital Signal Processor (DSP) in a single chip.

1.2 Goals

The main goal of this thesis is to evaluate the Atmel’s DIOPSIS 940 dual core chip as the main processor of a flight control board. For this purpose, a new flight control board which integrates that chip was designed and tested. This included the design of a new Printed Circuit Board (PCB) and the development of the embedded software. The complete system was evaluated by measuring the
performance of typical tasks execution i.e. sensor data gathering, data fusion and controlling actuators.

The board has been integrated with the LinkMAV [26] - an in-house developed coaxial helicopter presented in figure 1.1. The complete system should be able to stabilize the attitude of the aircraft.

![Figure 1.1. The LinkMAV coaxial rotor helicopter [26].](image)

Detailed information about the DIOPSIS 940 and in particular mAgicV DSP cannot be presented in this final report. A Non-Disclosure Agreement (NDA) has been signed between Atmel Corporation and the UASTechLab.

### 1.3 Content

The report is structured as follows. First, a brief overview of the existing control systems is given in chapter 2. The new system hardware architecture is presented in chapter 3. Next, the hardware components are described in more detail in chapter 4. It is followed by the description of the software developed for the
1.4 Contribution

DIOPSIS 940 processor (chapter 5). The report concludes with the experimental results, analysis and the summary.

The thesis incorporates information about the following issues:

• Electronic sensors and actuators needed for a flight control system.
• Usage of the DIOPSIS 940 processor - especially the mAgicV DSP in this system.
• Critical time requirements for the system design.
• Filtering techniques used in this type of a system.
• Usage of a real-time Linux for a flight control system.
• Comparison with other commercially available systems.

1.4 Contribution

The thesis contribution includes the following topics:

• Understanding of the theory behind flight control systems.
• Comparison of commercially available systems.
• Design of Printed Circuit Boards (PCB) to form a fully functional flight control system.
• Integration of the flight control system with the LinkMAV platform.
• Setting up software tools for the DIOPSIS 940 processor.
• Configuration, low and high level programming of an embedded real-time Linux system.
• Development of a real-time control application.
• Development of software for the mAgicV DSP processor.
• Variety of tests for assessment of the DIOPSIS 940 processor usage in the flight control application.

Only key points are presented above. More thorough description is given in the following chapters.
Chapter 2

Overview of the existing flight control systems

This chapter presents a brief overview of the existing flight control systems which are commercially available. It is intended to provide background information and a frame of reference for the system designed in this thesis. The introductory information will allow for better understanding of the chapters to follow.

The description of the existing flight control boards focuses on two major aspects. The type of architecture used (e.g. microcontrollers of different type, DSP, Field-Programmable Gate Arrays (FPGA)) and the number of Central Processing Units (CPUs) integrated in a system.

Additionally, a discussion about pros and cons of different configurations in terms of the two aspects is provided.

Single CPU systems

These systems utilize only one CPU which is responsible for all computations. Because of that, their PCB size and power consumption are smaller when compared to other solutions. Hence, they are especially suitable for a small UAS.

The Kerstrel Autopilot 2.2 [18] developed by Procerus Technologies is one example of such a solution. It is the smallest, commercially available system. It weighs only 16.65 grams and uses single 8-bit microcontroller which makes it computationally very weak. It is capable of basic navigation and uses an external Global Positioning System (GPS) module.

Another example is the MP2028g from MicroPilot [14]. This popular autopilot weighs 28 grams and includes a GPS receiver. It uses a 32-bit microcontroller. It is mostly used for fixed wing aircraft, although a helicopter version of the MP2028g is also available.
Overview of the existing flight control systems

Dual CPU systems

The most common configuration for dual CPU systems is the use of two processing units with computationally similar performance. Very often two identical processors are used in such systems.

For example, “Paparazzi” [15] is an open-source hardware and software project for flight control system development. Its documentation is available under the GNU General Public License (GPL). It consist of two ARM7 microcontrollers. Weight of the control board without sensors and a GPS receiver is 12 grams. Thanks to the use of two processors it separates “flight by wire” and “autopilot” functions in hardware. However, this control board is no longer actively developed. Instead, the “Paparazzi” project developers are focused on the design of a “Tiny”, single processor autopilot.

The “LinkBoard” from UAS Technologies Sweden AB [1] is another example of a system with multiple CPUs. In the basic configuration, it consists of two identical ARM “Cortex™-M3” core microcontrollers. The weight of the system ranges between 16 to 33 grams depending on the configuration.

One of processors of the LinkBoard is responsible for reading data from sensors and communication with a Remote Control (RC) radio receiver. Additionally, it is responsible for interfacing with actuators. The collected data is fused and used as input for control algorithms (e.g. attitude stabilization) running on the second processor.

The common problem of multi CPU systems is the need for communication between processors. On the other hand, the hardware separation of functionalities can be considered as an advantage from the safety (and redundancy) perspective.

The CPU architectures of the Papparazzi and the LinkBoard (in the basic configuration) lack the hardware Floating Point Units (FPU). This is a major drawback from the computational perspective. Dedicated hardware for high precision (single or double) operations is often required to achieve good computational performance for such tasks as sensor fusion.

It is possible to increase computational power in systems mentioned above by connecting another processor module - Triton [20] or Gumstix [11], which are complete computer systems. Their extremely small physical dimensions, make them a very good choice for robotics and UAS systems.

Mixed FPGA and DSP systems

The third group of flight control systems uses a combination of FPGA, DSP and microcontrollers.

The “FCS20” [10] from Adaptive Flight Inc. is one example of such a system. It consist of an Altera Stratix FPGA and a Texas Instrument DSP processor assembled on one board. The DSP gives the board its sequential processing power, while the FPGA is responsible for preprocessing low-level sensor data. The system weighs 65 grams.

The DIOPSIS 940 processor, evaluated in this thesis, should be included in this group. It has a unique combination of the mAgicV DSP and ARM9 core
embedded in one chip. The flight control system built around the DIOPSIS 940 requires an extra circuit for low-level input data preprocessing. In this thesis the circuit was built with use of a FPGA.

Summary

The choice of the type of architecture as well as the number of CPUs depends on the payload capability of a particular UAS platform and the application at hand. In this section, different aspects of using DSPs, microcontrollers and FPGAs for a flight control system are discussed.

Commonly used microcontrollers offer a good compromise between performance and cost. The unit cost of a CPU chip and the development cost is low. For example, the most commonly used compilers are free. They are developed and supported by large communities of enthusiasts. In contrast, the DSP and FPGA solutions require the use of specialized and costly development tools. The support is offered only by manufacturers of particular hardware. Even though those specialized compilers are tuned to specific architectures and produce extremely optimized machine code, they still require a deep knowledge of hardware (especially for FPGAs) which is to a lesser degree the case for a microcontroller based development.

Solutions based on microcontrollers, thanks to their simple architectures, are more power efficient when compared to FPGAs and DSPs. Unfortunately simple microcontrollers (i.e. 8-bit architecture) do not have enough computational power to perform tasks such as more advanced sensor fusion or image processing.

It is possible to use DSP processors for such computationally demanding tasks instead of microcontrollers. Their architecture focuses on the efficient data processing. This results in a significant performance boost for computationally heavy tasks.

Usage of a FPGA in a control system allows for larger flexibility, since its hardware configuration can be reprogrammed. New features can be implemented without the PCB alteration. Tasks like a Kalman filtering or image processing can be implemented on a FPGA and benefit from hardware-level optimization and parallelism.

The unique features of the DIOPSIS 940 processor (i.e. DSP and ARM9 cores in one chip) seem to combine the best of the two worlds. The DIOPSIS 940 processor provides the relative ease of software development for the ARM9 core and delivers a considerable amount of computational power of the DSP with a minimal size and small power consumption.

The power consumption of presented solutions is reported and compared with the DIOPSIS 940 based system in section 6.2.2.

The design of a system based on this processor is presented in the following chapter.
Chapter 3

DIOPSIS 940 flight control system overview

This chapter presents a brief overview of a flight control system designed with utilization of the DIOPSIS 940 processor.

Figure 3.1 presents the block diagram of the system. Blocks’ background colours and dotted lines denote printed circuit boards on which particular system components were assembled. With a radio receiver and LinkMAV actuators they are forming the complete flight control system.

Blocks with light blue background represent flight control system components assembled on the DIOPSIS 940 CPU Module (DCM). Its two main elements are the DIOPSIS 940 dual core processor and memory modules necessary for its operation. The DCM module was developed by the Atmel company and delivered with the DIOPSIS 940 evaluation kit [7].

The board described above is integrated with a second one, which includes inertial sensors, a power supply system and connectors for USB, Ethernet and a memory card. It is called the DIOPSIS 940 Back Module (DBM) and it is marked in a green colour on the block diagram. Elements enclosed with a dashed line box with the grey background are optional.

The radio receiver block (a separate device) collects servo control commands from a pilot. These are input data for the system, preprocessed by a separate board called the Servo Control System (SCS) and sent to the Atmel’s DIOPSIS 940 also referred to as the AT572D940HF or D940 in short.

The output data from the DIOPSIS 940 is transmitted to the SCS, which is labelled with the orange colour on the block diagram. In this circuit the data is converted to signals necessary for control servo mechanisms and rotational speed of electric motors. These actuators are mounted on the LinkMAV platform.

The introduced printed circuit boards are discussed in more detail in chapter 4.
Figure 3.1. System block diagram.
Chapter 4

Hardware

This chapter describes hardware components of the AT572D940HF based flight control board. The DIOPSIS 940 features are presented first, including the mAgicV and the ARM9 properties. Next, the DIOPSIS 940 CPU Module (DCM) and the DIOPSIS 940 Back Module (DBM) are discussed thoroughly. The motivation of using the Servo Control System (SCS) is presented along with its architecture and output data format. The chapter ends with a discussion about the complete flight control system hardware integration.

4.1 DIOPSIS 940 overview

The distinctive feature of the Atmel AT572D940HF is its dual core nature. It consists of a CPU and a DSP embedded in one package. The CPU is the ARM926EJ-S (abbr. ARM9) core, which is an intellectual property of ARM Ltd. [13], produced and manufactured by Atmel Corporation. The second major component of the processor is Atmel’s mAgicV Very Large Instruction Word (VLIW) DSP processor.

Figure 4.1 presents the DIOPSIS 940 block diagram. This processor, in addition to ARM9 and mAgicV cores, includes a number of peripheral blocks. The flight control system utilizes the Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI), General Purpose I/O (GPIO) pins and Timer/Counter (TC) blocks.

A cumulative feature list of the DIOPSIS 940 is presented below.

**ARM926EJ-S:**

- 220MIPS @ 200MHz.
- Memory Management Unit (MMU).
- 5 Stage Pipeline.
- Data and Instruction Cache.
**Figure 4.1.** The AT572D940HF block diagram [7].

- External Bus Interface (EBI).
- USB 2.0 Full Speed Interface.
- Ethernet MAC 10/100.
- JAVA support through ARM JAZZELLE.
- 3 x USART, 2 x TWI, 2 x CAN, 96 x GPIO, 4 x SSC.
- Debug Unit (DBGU).
- IEEE 1149.1 JTAG.
- Multimedia Card Interface (MCI).

**mAgicV VLIW DSP**

- 1 Billion Floating Point Operations per Second @ 100 MHz.
- Support for Complex Arithmetic and Vectorial SIMD Operations.
- 32-bit Integer and IEEE 40-bit Extended Precision Floating Point Numeric Format.
- AHB Master Port, Direct Memory Access (DMA).
- DMA Access to the External Program and Data Memory.
4.1 DIOPSIS 940 overview

- Operating modes: Run, Debug and Sleep Mode.
- User Mode and Privileged Interrupt Service Routines (ISR).
- Efficient Optimizing ASM and C-Oriented Architecture.

**ARM9 and mAgicV integration**

- The ARM9 can act as the master for the mAgicV VLIW processor. It controls operating modes of the mAgicV.
- The mAgicV has its own program memory, which makes it possible to execute its code in parallel with the ARM9.
- The mAgicV and ARM9 can exchange (at run time) synchronization signals.
4.2 DIOPSIS 940 CPU Module (DCM)

Figure 4.2 presents Atmel’s DIOPSIS 940 Development Board (DEB) [6]. The DIOPSIS 940 CPU Module (DCM) is mounted on top of it.

For evaluation purposes the DIOPSIS 940 is provided as a ready to use kit (DEB). It consists of two PCBs, namely, the main processor board (i.e. DCM) and a back module. This evaluation kit has been introduced by Atmel to demonstrate the DIOPSIS 940 performance in applications such as professional audio, radar and acoustic signal processing. Only the main board has been used for the development of the flight control system. The back module has been replaced with a customized PCB described in the following section.

The AT572D940HF chip is housed in a Thin Flat Ball Grid Array (TFBGA) package with 324 pins. It requires a very specialized and expensive assembly. The use of the main processor module allows for rapid hardware development and substantial cost reduction for a prototype design.
As presented in figure 3.1 the DCM module includes 256 MB (256 Mb x 8) of NAND flash, 16 MB (4 Mb x 32) of parallel flash and 64 MB (16 Mb x 32) of SDRAM memory. It also incorporates voltage regulators to provide power for the processor \((V_{DDcore} = 1.2\,\text{V} \text{ and } V_{DD} = 3.3\,\text{V})\). Moreover, it uses a specialized circuit for Ethernet connectivity. Physical dimensions of the DCM module (80 mm x 70 mm) and the location of connectors are standard for Atmel’s processor evaluation kits.

The DIOPSIS 940 CPU module can be easily connected to other boards thanks to three 120 pin connectors to build more sophisticated systems. This feature was used in this project to connect a specially designed back module, which includes all the components required for a flight control system. This module is described in detail in the following section. The DCM weighs 29 grams.

### 4.3 DIOPSIS 940 Back Module (DBM)

The custom designed DIOPSIS 940 Back Module (DBM) hosts inertial sensors, a pressure sensor and a very efficient power system (with over-current protected +3.3 V and +5 V outputs). Two revisions of the DBM have been designed and produced. The inertial sensors used on both DBMs include three gyroscopes (one for each axis) and three axis accelerometer sensors. A Serial Peripheral Interface (SPI) [25] bus is used for data transmission between the DIOPSIS 940 and the inertial sensors. The SPI operates at 7.5 MHz frequency and the D940 is used as the master device.

The two DBMs have been designed to fulfill different requirements. The basic DBM is a lightweight, two layer PCB (figure 4.3(a)). It hosts a minimal set of components needed for a flight control system, thus its weight is maximally reduced. The complete DBM weighs 40 grams. Communication with a host PC can be established only through a serial RS-232 connection, which makes the development of the software very cumbersome.

The second DBM features more hardware components and is based on a four layer PCB (figure 4.3(b)). This version of the board is useful for software development thanks to extensive connectivity options. The connection with a host PC can be established through Ethernet or USB links. Additionally a SD-Card can be used to host a Linux operating system. The total weight of this DBM is 51 grams.

Differences between both DBMs are described in chapter 3. In figure 3.1 components specific to the more advanced (“fully featured”) DBM are marked with a dashed line.
(a) The “Light weight” DIOPSIS 940 Back Module.

(b) The “Fully featured” DIOPSIS 940 Back Module.

Figure 4.3. DIOPSIS 940 Back Modules
4.4 Servo Control System (SCS)

4.4.1 Motivation

The third board designed for this thesis is the Servo Control System (SCS) board. It is presented in figure 4.4 and employs a non-volatile Xilinx Spartan 3AN FPGA. The FPGA resource usage is 99%. As presented in figure 4.5, the SCS consist of two separate blocks. The first block is responsible for reading data from a radio transmitter. The second one generates signals for UAS platform actuator (i.e. servos and motors) control.

In the initial phase of development, the AT572D940HF processor was intended to perform the tasks described above. First, the General Purpose Input/Output (GPIO) pins were employed. They were configured to trigger the D940 interrupt on its input state toggle. A rising edge started pulse duration measurement, whereas a falling edge stopped it. Afterwards, the pulse duration was calculated. This measurement was performed in the Linux Interrupt Service Routine (ISR) context.

However, experiments have shown that, such a solution was providing inaccurate results. This distorted input data was responsible for producing incorrect system output observed as unexpected movement of actuators. To solve this problem, a real-time “PREEMPT_RT” Linux kernel patch [4] was evaluated. Unfortunately, it also did not provide sufficient accuracy. Detailed descriptions of this experiment and measurement results are presented in section 6.2.1.

One possible solution for this problem is to employ the DIOPSIS 940 hardware timers (TC peripheral blocks) to measure signal durations from the radio receiver. This approach allows precise measurement, with a minimum load of the operating system. The control system has to calculate the duration of at least six signals.

One more TC block is necessary to generate output signals for a typical set
of actuators (with usage of the decade Johnson counter). Unfortunately, the AT572D940HF has only three such timers instead of the seven required. Therefore this solution has been disregarded.

The second solution was to rewrite the Linux kernel interrupt service routine code to assign the highest priority to tasks responsible for input signals measurement. It is not feasible, because it could break the current Linux kernel and the special “patch” would need to be applied to each next kernel release.

The arguments presented above have led to the conclusion that the best solution for this problem is to use a separate hardware component.

The hardware architecture implemented in the FPGA allows the DIOPSIS 940 processor to reduce its workload for this precision demanding task.

The data from the radio receiver is delivered to the control system as uncorrelated waveforms. The information is coded as an impulse duration with values between 1 ms and 2 ms. The same data format is used to control the LinkMAV’s servos and motors.

4.4.2 Architecture

The Servo Control System (SCS) block diagram is depicted in figure 4.5. It consists of a receiver, transmitter and control signals generation blocks. The transmitter and the receiver are working in parallel.

The receiver architecture, presented in figure 4.6 utilizes seven “servo_counter” blocks to measure the duration of input impulses. Subsequently, results are multiplexed in the “pulse_val_mux” block into the data representation convenient for a serial transmission, which is accomplished by a “uart_tx” block. The receiver supervisor - a “tx_control” block, is designed as a Finite State Machine (FSM).

The actuator control block architecture is depicted in figure 4.7. The idea of its operation is to perform receiver’s tasks in the inverse order, i.e. read the serial data, check its integrity and generate pulses with the duration corresponding to the received data.

The complete board weighs 17 grams.
Figure 4.5. Servo Control System block diagram.
Figure 4.6. Servo Control System - input block (s_in_block).
Figure 4.7. Servo Control System - output block (s_out_block).
4.4.3 Output data format

This section describes the output data format of the Servo Control System (SCS) board. The communication is performed via the serial port. The transmission parameters are: baud rate 38400 bauds, 8 bits, no parity, one stop bit. The board has a signal level converter (compatible with RS-232C), which allows for a direct connection to a PC computer. It is also possible to transmit data with LVTTL signal levels. The FPGA based design allows to reassign different functions for the same pin.

Figure 4.8 depicts the structure of the output data packet. It includes eighteen bytes: two prefix, fourteen of data, one checksum and one indicating the end of packet. The checksum byte is calculated as byte-wise XOR operation for all data bytes.

![Figure 4.8. Servo Control System - output data format.](image)
4.5 System integration

Printed circuit boards described in previous sections can be stacked together, as presented in figure 4.9. The weight of the complete system is 100 grams.

![DIOPSIS 940 control system PCB boards combined together.](image)

Figure 4.9. The DIOPSIS 940 control system PCB boards combined together.

The flight control system hardware has been integrated with the LinkMAV platform. The complete UAS platform is shown in figure 4.10.
Figure 4.10. The DIOPSIS 940 control system integrated with the LinkMAV platform.
Chapter 5

Software

This chapter explains in detail the software tools used to develop the presented flight control system. Detailed information about use of programming tools for the mAgicV DSP and the ARM9 is included.

Additionally, the real-time Linux functionality used for software developed in this thesis is described. The emphasis is put on the motivation for using the real-time Linux for the flight control system application and the necessary real-time device drivers.

5.1 Development software

This section gives detailed information about the software used for application development for the DIOPSIS 940 processor. Specifically, description of the programming tools used for the ARM9 development is included as well as the mAgicV DSP compiler is introduced. Additionally, the embedded Linux used in the flight control system is described.

The source code of an application written for the ARM9 architecture needs to be compiled by a special set of programming tools (also called a toolchain), which for this project comprises the GNU Compiler Collection (GCC) [16], binutils and a C library. Code compilation can be performed on the target machine, which in this case is the control system or on the host computer, which can be an ordinary PC. The process of compiling on the host machine is often referred to as cross-compilation.

Compiling the source code on the DIOPSIS 940 target system is straightforward, because it has a preinstalled toolchain (included in the Linux distribution). This approach has one major disadvantage. The computational power and resources are very limited when compared to capabilities of a contemporary PC. It is more convenient for development to cross-compile programs on a PC host machine and then copy them to the target system.

In this project the second approach was employed. The toolchain on the host PC has been built with the ptxdist tool from Pengutronix [9]. It consists of the
GCC compiler (version 4.3.2), binutils utilities (version 2.18) and the glibc library (version 2.8) with the real-time routines support.

This toolchain was configured to support the -msoft-float option. Enabling this switch during compilation delivers performance improvement for operations executed on the floating point numbers. The DIOPSIS 940 evaluation kit, with the Debian 4.0 Linux system, has a preinstalled compiler (GCC version 3.4.6) without this option support. Therefore, floating point operations are executed with a huge performance penalty, since the ARM9 lacks the hardware FPU. The Linux kernel, in this case, emulates an FPU in software.

A different toolchain containing the uClibc C standard library (version 0.9.30.1), the GCC (version 4.3.3) and binutils (version 2.19.1) was also evaluated. It can be easily built using the buildroot utility. The process of downloading, patching and compiling is fully automated. The buildroot project has a user friendly interface for configuration, similar to the one of a vanilla Linux kernel. The toolchain has been built to support the -msoft-float compiler switch as well.

The uClibc C library included in the toolchain provides only the most commonly used features, therefore the size of a compiled program is much smaller when compared to the glibc version.

For example, the size of the gdbserver utility, which is a part of the GNU Debugger Collection (GDB) project, was compared after cross-compilation. Toolchains containing either the glibc or the uClibc were used. The executable code produced with the glibc toolchain had the size of 226 KB, whereas with the uClibc only 185 KB.

From the binary file size perspective a toolchain built around the uClibc is a better choice for use in an embedded system, where an efficient usage of storage media is crucial.

Unfortunately, the uClibc does not support real-time specific routines, like the clock_nanosleep function, which can suspend process execution for a specified amount of time. It is most commonly used for developing applications running on the Linux kernel with a real-time extension as an accurate time reference. The uClibc only supports the nanosleep function which is not designed for real-time usage (i.e. the exact sleep time is not guaranteed). The uClibc provides the gettimeofday function to obtain current system time with microsecond resolution, which is also not accurate enough for real-time applications.

The glibc is a superset of the uClibc library and hence, it supports the above mentioned routines. Moreover, it contains the clock_gettime, which has the necessary nanosecond resolution. It allows for specifying a real-time clock which can be used as a reference timer. Therefore this function is suggested for real-time applications.

The above arguments have led to the decision, that for the presented design the glibc toolchain will be used.

In the described control system static linking is used to maximize software portability. This results in including all used libraries into an executable file and therefore increases its size (comparing to dynamic linking). This approach has one major advantage. It allows for testing binaries on different root file systems without concern about availability and compatibility of libraries.
Portability and compatibility reasons presented above, have imposed support for the Embedded Application Binary Interface (EABI) in the Linux kernel and toolchains. Therefore, if two compilers support the EABI, libraries compiled with one compiler can be utilized with code generated by the other one.

Chapter 4 of [3] is an exhaustive source of information about building different toolchains for an embedded Linux system.

While the ARM9 is compatible with many compilers (e.g. GCC) the mAgicV DSP is supported only by the Target Compiler Technology (TCT) [19] programming tools. A different toolchain cannot be used. The tools provided with the mAgicV DSP were designed with a close cooperation with Atmel, therefore achieved code optimisation level is comparable to the hand optimised assembler code.

The most important elements of the TCT toolchain are: “Chess” - the C compiler, “Checkers” - the instruction set simulator and “Bridge” - the linker. These components are managed using the “ChessDE” Graphical User Interface (GUI).

The TCT toolchain is a proprietary and expensive solution. In this project, software necessary for evaluation was developed using a 60 day evaluation version of the software.

5.2 Real-time Linux

This section presents the utilization of the real-time Linux in the flight control system. It starts with the description of the used Linux distribution. Additionally, the control application executed in this system is described. The section ends with a discussion about real-time device drivers for the sensors and actuators used.

The Debian GNU/Linux 4.0 operating system runs on the ARM9 core. It creates a flexible and unified environment for software development. The Linux gives the opportunity to reuse code written for various processors. It supports a root file system mounted via Ethernet from a host PC. This machine is running the Network File System (NFS) server. This is a very convenient way to develop and test software. Nonetheless, for the “production” version of the control system, the root file system can be built as a RAM disk (initrd) and stored in a non-volatile memory of the DCM module.

The presented control system is a hard real-time application. This means that not only correct logical results are critical, but also the time when they are delivered is important. Missing an execution deadline is a serious fault, which can cause a crash of a controlled UAS platform.

Four different Linux kernels were tested for this project, namely linux-2.6.23.13-d940, linux-2.6.24.3-rt3-d940, linux-2.6.26.8-r14-d940 and linux-2.6.29.1-r8-d940. Suffix “d940” indicates that the vanilla kernel (the final release) has been patched with the DIOPSIS 940 specific patch (including the mAgicV driver). Moreover, for the three last kernels the “PREEMPT_RT” real-time patch has also been applied.

The real-time requirement for this design has imposed the use of the linux-2.6.29.1-rt8-d940. It has been tuned for the flight control system by removing all
unnecessary features.

5.2.1 Control application

This subsection describes the control application which is in charge of the flight control system.

The control application has been written in compliance with guidelines from the Real-Time Linux Wiki [4] and Alex’s van der Wal paper [22]. According to the cited publications, proper memory management is crucial for preserving real-time attributes of a system. Any page fault during execution of the real-time code can potentially cause missing of a deadline. To avoid such a situation, proper stack management and locking of used memory (with the `mlockall`), has been performed.

The control application is running with the second highest priority among all tasks executed by the real-time Linux. The highest priority is reserved for tasks essential for sustaining the operating system.

The control application is utilizing static cyclic schedule for task management. This method is simple and reliable, which results in very little processor overhead. Figure 5.1 presents the static schedule used in the system.

![Static Cyclic Schedule](image)

**Figure 5.1.** Static cyclic schedule.

The control application is divided into four different tasks - T1 to T4. Period T is equal to 20 ms. As depicted in figure 5.1, task T1 is responsible for reading data from the radio receiver and inertial sensors. Next, the collected data is fused
using either a Kalman or a complementary filter. This task is marked as T2 and discussed in section 6.1.

The fused data (i.e. the system state) is used by the control function, marked as T3. It executes the **Proportional Integral Derivative (PID)** controller for attitude stabilization. The structure of such a controller is explained in detail including useful pseudocode in [24].

The output data from the control function (i.e. pitch ($\phi$), roll ($\theta$) and yaw ($\psi$) control values) are transformed (“mixed”) into signals which are then commanded to actuators of a UAS platform. In the case of the LinkMAV platform, the attitude control is achieved by means of three servos and the rotational speed of two motors, as depicted in figure 5.2.

![Figure 5.2. The LinkMAV helicopter multiplexer and actuators.](image_url)

The last task, labelled as T4 is responsible for sending telemetry data to the ground control software for visualization. Thanks to that, the platform state can be observed in real-time in a graphical representation.

Execution times of all tasks are summarized in table 5.1. The timing of the data filtering functions (T2) is described in section 6.1. The time T2 depends on
used filter type and data representation.

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>588 µs</td>
</tr>
<tr>
<td>T2</td>
<td>134 µs - 10645 µs</td>
</tr>
<tr>
<td>T3</td>
<td>222 µs</td>
</tr>
<tr>
<td>T4</td>
<td>569 µs</td>
</tr>
</tbody>
</table>

Table 5.1. Control system tasks timing benchmarks.

The described control application, running on this real-time flight control system, requires constant update of the input data (task T1 on the schedule). The data is provided by device drivers whose reliability is crucial for the flight control system operation.

5.2.2 Real-time device drivers

This section describes development of device drivers necessary for communication with sensors and the Servo Control System (SCS). Two methods for accomplishing those tasks have been implemented and evaluated.

Inertial sensors are connected to the AT572D940HF processor via a SPI bus. A special Linux kernel driver has been developed to provide open, read and write standard interfaces (API) for the flight control system. Access to the data delivered by the API is available through /dev/uas_sensors character device. It provides the ioctl function for testing the correctness of sensor outputs, which is used during the initial system check. The kernel module was developed according to guidelines provided in the “Linux Device Drivers” book [17].

The data preprocessed by the SCS, presented in section 4.4, is transferred to the DIOPScs 940 via asynchronous serial transmission. On the Linux side, the driver, developed by Atmel and accessed via /dev/ttyS2 file is responsible for handling the transmission (with parameters described in 4.4.3).

Unfortunately, the performance of both drivers was insufficient. Test programs executed on the flight control system proved that handling standard API calls is causing too much kernel overhead. Therefore the execution of the real-time application is delayed and the task schedule cannot be met. This failure appearance was sporadic and therefore hard to debug. Attempts to use the gdbserver and strace were made, but both programs introduced their own delays and therefore were damaging the real-time schedule.

A different way of handling the data transmission was used instead. It does not rely on the read and write standard API. The data transmission is achieved by opening the /dev/mem device and then mapping this file with the mmap system call. Only the superuser (e.g. root) has appropriate system permissions to access the D940 physical registers that are required. Special care has to be taken not to write incorrect values to the registers as it may cause system instability or crash.
5.2 Real-time Linux

For inertial sensors, the device driver previously used as kernel module (the uas_sensors driver) has been rewritten for usage in the Linux userspace. In this solution the SPI communication is managed solely by the real-time control application, with no unnecessary system calls to the kernel. Therefore, the ARM9 workload is reduced.

The serial transmission driver has also been rewritten for the Linux userspace execution. Unfortunately, it uses kernel tasklets extensively, which cannot be easily rewritten to be used in the Linux userspace. This technique is very efficient on an ordinary Linux kernel since the data handling is split between both hardware and software interrupt contexts. Unfortunately this approach cannot be used in the Linux with the real-time extension because all drivers’ routines should be executed in the hardware interrupt context to preserve the real-time behaviour of an application such as in case of the flight control.

The new serial driver, written from scratch in the Linux userspace without tasklets usage, has to manage the serial transmitter (UART) and the Peripheral Data Controller (PDC) blocks. The PDC controller allows for a direct access to the D940 RAM memory and therefore is responsible for providing the data to the serial transmitter (UART). In the serial communication between the SCS and the AT572D940HF the latter is a slave.

Carrying out the data transmission to the SCS requires filling up proper RAM buffer for the PDC and initialization of the transmitter. Afterwards, the transmission is executed without any interaction with the control application.

The data from the SCS is read by the DIOPSIS 940 and stored in a ring buffer structure. The content of the ring buffer is analyzed to match with the packet pattern described in figure 4.8 and evaluated if the data was not corrupted during the transmission. Additionally, an eight bit checksum allows for a simple data integrity validation.

In order to maintain data integrity the described device driver uses 4KB region in the D940 RAM. This memory block is excluded from the Linux kernel management which prevents the data from being overwritten by a kernel process.

The control application developed with utilization of the kernel device drivers and the read and write system calls, consumed 13 ms of the 20 ms period. This is 65% of the available time.

Rewriting the control application with utilization of the new, userspace real-time drivers allows for a reduction of the execution time to 7.5 ms. Therefore the ARM9 core usage decreases to 37.5% for the same period. Thanks to that, the Linux operating system stability and robustness increased. All the shortcomings of the standard API based drivers have been overcome in the memory mapping based solution.

To conclude, the use of the /dev/mem character device and the mmap system call provide better performance, reliability and robustness for this particular flight control system.
Chapter 6

DIOPSIS 940 evaluation

6.1 Software

This section presents the results of using the DIOPSIS 940 processor in computational demanding applications. It compares the mAgicV DSP with the ARM9 in terms of computational performance. First, a matrix multiplication experiment is described. Then, two alternative filter solutions (complementary and Kalman) used in the flight control systems are presented and compared.

6.1.1 Matrix multiplication

This subsection describes square matrix multiplication performed with the mAgicV DSP and the ARM9 core. It starts with the motivation and a general information about the experimental setup. Thereafter, results including execution times, achieved speed-up and relative error values are reported. The section concludes with several final comments.

Introduction

The mAgicV DSP represents the float and the double C data types as 40 bit numbers, as presented in figure 6.1. This is a consequence of using 40 bits wide processor data bus. On the ARM9 architecture these data types are represented as 32 (single precision) and 64 (double precision) bits respectively. The two representations are depicted in figures 6.2 and 6.3. The float representations conform to the IEEE 754 standard.

The mAgicV 40 bits representation is a combination of a 32 bit form with 8 more bits for a mantissa (also called fraction). A transfer of the ARM9 float to the mAgicV requires adding extra 8 bits zero padding for the mantissa.

In the opposite direction the mAgicV Linux driver and the special wrapper library libmwrap.a provides routines for 40 bits mAgicV float conversion to ARM9 32 bits representation. This operation introduces rounding and truncation errors. To avoid this, it is possible to read the mAgicV 40 bits float “as is” and store it
Problem The evaluation of the mAgicV DSP utilization as a coprocessor for computationally demanding tasks is described. The problem of matrix multiplication is considered.

Experimental setup Square matrices, with sizes from 3 up to 70, have been multiplied using both the ARM9 core and the mAgicV DSP. This experiment was performed with the use of the float type (32 bits) as the input. The data was generated with the `rand` function, which produces pseudo random numbers in the range from 0 to \( \text{RAND\_MAX}=2147483647 \). Each element in the input matrices is between:

\[
0 \leq \frac{\text{rand}()}{\text{RAND\_MAX}} \leq 1
\]  

(6.1)

Calculations carried out on both cores used the same input data. For more reliable results, the experiment was performed several times with different
seeds for the `rand` function and average results have been used for further discussion.

The software for the ARM9 architecture was compiled with the `-msoft-float` switch enabled.

In the presented flight control system, the ARM9 core is working with 160 MHz frequency, while the mAgiC uses only 80 MHz clock frequency.

The execution time was measured with the `clock_gettime` function as the difference between the matrix multiplication routine stop and start times. In the mAgiC core, this also includes the time required to write and read data from the DSP internal memory. Moreover the synchronisation of both cores was taken into account. Therefore effective execution time was measured for the mAgiC.

The matrix multiplication performed on the ARM9 core was implemented in the C language following the definition described in [23]. The code listing 6.1, presents the routine used for this operation. This approach is intuitive and easy to implement, but it is not optimal in terms of the execution time.

**Listing 6.1.** Matrices multiplication function source code

```c
static void mul_mat(fp_t *a, fp_t *b, fp_t * c,
                    int l, int m, int n)
{
    int i,j,k;
    /*
       a -> rows: l ; cols: m
       b -> rows: m ; cols: n
       c -> rows: l ; cols: n
    */
    for(i=0;i<l;i++)
        for(j=0;j<n;j++)
```
\[ c[i*n+j] = 0; \]
\[
\text{for } (k=0; k<m; k++){
\quad c[i*n+j] += a[i*m+k]*b[k*n+j];
\}
\]

The matrix multiplication performed on the mAgicV DSP used the highly optimised \texttt{fMatrixMul} floating point matrix multiplication function from the mAgicV (\texttt{DSPLib.a}) library. The library includes 115 functions for performing calculations such as FFT, convolutions, FIR filters. Operations may be performed on the float (40 bits) and the long (32 bits) mAgicV data types, with complex, vector or single arithmetic. Exhaustive description of the library usage and its functions is provided in the “DSPLib User Guide” [8].

**Results**

- Execution times

![Execution time of square matrices multiplication](image)

\textbf{Figure 6.4.} Execution time of square matrices multiplication.
Figure 6.4 presents matrix multiplication time benchmarks for the mAgicV and the ARM9 cores. The execution times for the ARM9 increases exponentially with the size of a input matrix. As expected, calculations with the double precision number representation are more time consuming, when compared to the single precision and the mAgicV 40 bits representations. The maximum value of 390819 $\mu$s for a single and 546928 $\mu$s for a double precision respectively was achieved for the matrix size of 70. The same computation performed with the mAgicV DSP core takes only 22587 $\mu$s.

- Performance improvement

![Figure 6.5. The mAgicV DSP execution time improvement over ARM9.](image)

The performance improvement due to the use of the mAgicV DSP over ARM9 core is presented in figure 6.5. It is clearly visible that the mAgicV delivers up to 17.3 times faster execution of the given task for the single precision and 24.7 for the double. The value of the improvement has been calculated according to equation 6.8.

The shape of curves shows, that for larger matrices, it is likely that the mAgicV will deliver even higher improvement.

The matrix multiplication implementation for the ARM9 (as presented on
listing 6.1) is not optimal. Utilization of other programming techniques (e.g. loop unrolling) may give performance improvement, therefore the mAgicV speed up over the ARM9 may be reduced.

- Calculation error

![Figure 6.6. Relative error of square matrices multiplication.](image)

As shown, the mAgicV DSP gives a considerable performance acceleration (figure 6.5) over the ARM9. However, the DSP uses 40 bits float representation and the conversion to a 32 bit form has an impact on the accuracy of the result. The influence of truncating the 8 bits was evaluated in this experiment by means of a relative error measurement. The error is depicted in figure 6.6 for the single and double precision ARM9 and the mAgicV calculations. Its value is very small. The relative error has been calculated according to equation 6.2.

$$|\delta_x| = \frac{|\Delta_x|}{|x_0|} = \frac{|x - x_0|}{|x_0|}$$  \hspace{1cm} (6.2)

where: $x$-measured value. $|\Delta_x|$-absolute error. $x_0$-reference value.
The reference value was calculated with MATLAB software using the same input data with a double precision arithmetic (figure 6.3) conforming to the IEEE 754 standard.

Figure 6.6 shows, that for the given input data, the smallest relative error was obtained for calculations executed on the mAgicV DSP. According to those results, the mAgicV DSP core (with DSPLib) produces more accurate results, than the ones from the ARM9 (with glibc, even for double precision). Therefore, the truncation performed by the mAgicV Linux driver, when results were transferred to the user program, can be neglected for this experiment.

Figure 6.6 also shows that calculations performed with use of the ARM9 double precision numbers have less relative error than the single precision. Such a result was expected.

From the perspective of the application developed in this thesis the results are satisfactory and further investigation was not performed. The accurate numerical analysis for this experiment is beyond the scope of this thesis.

Conclusion

Results acquired in this section confirm that the mAgicV DSP can work as a very efficient coprocessor for computationally expensive operations. However, one needs to verify if the 40 bits of mAgicV float representation is sufficient for a specific application.
6.1.2 Complementary filter

This subsection presents the results of the evaluation of the complementary filter used in the flight control system developed in this thesis. First, a short overview of a complementary filter is given. Afterwards, execution time comparison between the ARM9 and the mAgicV implementations is presented. Additionally, figures presenting complementary filter output from both cores compared to a reference value are provided. This section ends with a comment on the filter usage in the flight control system application.

Introduction

To perform the basic attitude stabilization of a UAS platform, values of three angles - pitch (\(\phi\)), roll (\(\theta\)) and yaw (\(\psi\)) have to be estimated. A complementary filter can be used to fuse the data from inertial sensors in order to estimate attitude angle values. This type of filter will work in near hovering condition since the angle estimation is also based on acceleration measurements. In case of, for example, accelerated turn, the centrifugal force will introduce an error. If such a manoeuvre is short the navigation system will still serve its purpose.

Figure 6.7 gives an overview of the implemented complementary filter. The filter uses data from two types of sensors, namely a gyroscope and an accelerometer to compute angle value for the pitch (\(\phi\)) and roll (\(\theta\)) axis. The yaw (\(\psi\)) angle cannot be estimated based on acceleration. Thus it is computed by integration of yaw axis gyroscope data. One possible solution is to use a magnetic compass sensor which would provide the complementary data to the rate measurement for calculating the yaw angle. The magnetic sensor was not integrated in this system revision.

The filter implementation uses a single precision floating point number representation (32 bits) for both mAgicV and ARM9 cores.

\[
\begin{align*}
\phi_{\text{gyro}} &\rightarrow \text{HPF} \rightarrow \phi_{\text{est}} \\
\phi_{\text{accel}} &\rightarrow \text{LPF} \rightarrow \phi_{\text{est}}
\end{align*}
\]

Figure 6.7. Complementary filter - block diagram.
where:

\[ \phi_{\text{gyro}} = \int (\text{angular velocity})dt \]  
\[ (6.3) \]

\[ \phi_{\text{accel}} = \sin \left( \frac{\text{accel}}{g} \right) \]  
\[ (6.4) \]

\[ H_g(s) = \frac{(\tau s^2 + 2\tau s)}{(\tau s^2 + 2\tau s + 1)} \]  
\[ (6.5) \]

\[ H_a(s) = \frac{1}{(\tau s^2 + 2\tau s + 1)} \]  
\[ (6.6) \]

\[ H_a(s) + sH_g(s) = 1 \]  
\[ (6.7) \]

The first input to the filter, the \( \phi_{\text{gyro}} \) (eq. 6.3) is the integration of the angular rate from a gyroscope (the \( \text{deg/sec} \) unit). The gyroscope sensor suffers from a long term drift which will accumulate during the integration. To alleviate this problem, High Pass Filtering (HPF) is performed.

The second filter input, the \( \phi_{\text{accel}} \) (eq. 6.4), is based on acceleration measurement of the gravitational force (the \( g \) unit). Unfortunately, this type of sensor is sensitive to vibration or rapid movements and therefore its output is filtered with a Low Pass Filter (LPF).

The filter transfer functions (HPF and LPF) have to comply with equation 6.7. This condition assures that the two inputs complement each other and provide the complete estimation.

The filter time constant (i.e. \( \tau = 0.6 \)) was determined empirically in the implemented flight control system taking into account the properties of the used sensors.

More detailed information about a complementary filter design and its usage in a UAS control system can be found in [2].

**Results**

The output from the complementary filter of the mAgicV, ARM9 and MATLAB is presented in figures 6.8 and 6.9 for pitch (\( \phi \)) and roll (\( \theta \)) angles respectively. The input data was identical. The filtering for different cores was performed with the same update rate. The MATLAB output should be considered as a reference, since the double precision (64 bits) representation was used.

Presented figures show no difference between the mAgicV and the ARM9 results. This leads to the conclusion that there is no clear advantage in using the mAgicV 40 bits float number representation over the ARM9 32 bits in this application.

However, there are differences of maximum values of angles between the reference MATLAB results and the filter outputs calculated on the two cores. The mismatch can be explained as the integration error caused by different floating
point number representations. Despite the inaccuracy of the on-board filter implementations the results are satisfactory for the application of a UAS platform attitude stabilization.

![Complementary filter output](image)

**Figure 6.8.** Complementary filter - pitch \( (\phi) \) angle.

<table>
<thead>
<tr>
<th>Core</th>
<th>MIN [( \mu s )]</th>
<th>MAX [( \mu s )]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM9 (32 bits)</td>
<td>134</td>
<td>263</td>
</tr>
<tr>
<td>mAgicV (40 bits)</td>
<td>453</td>
<td>750</td>
</tr>
</tbody>
</table>

**Table 6.1.** Complementary filter timing benchmarks.

Table 6.1 presents maximal execution times of the complementary filter for mAgicV and ARM9 cores. The time measurement procedure was identical as in the matrix multiplication experiment (subsection 6.1.1).

Surprisingly, the ARM9 filter execution time is shorter than on the mAgicV DSP. This can be a result of a DMA data transfer overhead to and from the mAgicV internal memory. Additionally, the time needed for the synchronisation with the ARM9 core has to be included in the execution time measurement.
Conclusion

The complementary filter despite its shortcomings (e.g. limited flight envelope) provides an accurate state estimate for the attitude stabilization of a UAS platform. The main advantage is a short execution time as presented in table 6.1 which makes it a particularly interesting solution for computationally limited systems. The timing benchmarks revealed that the ARM9 has a sufficient performance and the use of the mAgicV DSP is not beneficial.
6.1.3 State estimation based on a Kalman filter

A different type of state estimation commonly used for navigation purposes takes advantage of a Kalman filter. This subsection describes such a system implemented and evaluated in this thesis. First, a short overview is given. Afterwards, execution benchmark comparison between the ARM9 and the mAgicV DSP is presented. Additionally, the filter results for attitude angle estimation are compared with a MATLAB reference. The section ends with comments about the filter usage in the flight control system developed in this thesis.

Introduction

The Kalman filter is a recursive algorithm, which is used to estimate a state or an error of a linear system in the presence of noisy sensor data. It is widely used in many applications e.g. navigation, radar and computer vision. The filter uses a sensor noise and a process model to deal with uncertain inputs. The filter described in this section has a specific structure suitable for UAS navigation purposes.

![Figure 6.10. Navigation system block diagram.](image)

Figure 6.10 presents the architecture of a Kalman filter based state estimation. The filter estimates nine system states, namely the attitude angles, horizontal position and altitude, horizontal and vertical velocities. The inertial data (i.e. accelerations and angular rates) is used by the Inertial Navigation System (INS) to deliver the full state estimate. The process of computing the state, also referred to as mechanization suffers from a long term drift caused by the sensors’ inaccuracies accumulating over the time in the process of the data integration. The role of the Kalman filter in this architecture is to estimate the errors of the INS outputs. This is achieved by using an additional source of measurement, i.e. position information which is commonly delivered by a GPS receiver. Detailed description of the state
estimation architecture used in this thesis can be found in [5].

The process of porting of the existing filter implementation to the mAgicV DSP was not straightforward due to certain limitations of the TCT C compiler. For example, commonly used pointer casting techniques which speed up access to matrix elements were not allowed. Alternative solutions to the shortcomings were implemented and the same functionality was achieved.

The described system was not equipped with a GPS receiver, so an equivalent (in terms of noise characteristic) artificially generated data was used to perform the evaluation.

**Results**

As previously described in section 6.1.2 the basis for the attitude stabilization of a UAS platform are the pitch ($\phi$), roll ($\theta$) and yaw ($\psi$) angles. For this reason the evaluation of the filter focuses on the results of the attitude estimation. Figures 6.11(a)-6.11(f) present the example of a pitch ($\phi$) and roll ($\theta$) attitude angle estimation computed on-board the flight control system. The MATLAB implementation of the filter should be considered as a reference, since it utilizes the double precision representation. As one can see from the plots, angle values calculated by the mAgicV DSP and the ARM9 cores are identical with the MATLAB results. This confirms correctness of the calculated results.

Additionally, tests over longer periods of time were performed to check the solution stability of the described filtering technique within a typical flight endurance of a UAS platform. The test case was to simulate the flight control system movement for approximately 20 minutes. No unexpected results appeared.

The Target’s C compiler for the mAgicV DSP provides settings for the executable code optimization in respect to a compilation time and the output code size. An excessively aggressive optimization results in the code which is unstable and causes the DSP software to freeze. In order to avoid such a situation the default compiler settings were used.

<table>
<thead>
<tr>
<th>Core</th>
<th>MIN [\mu s]</th>
<th>MAX [\mu s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM9 (32 bits)</td>
<td>5601</td>
<td>8543</td>
</tr>
<tr>
<td>ARM9 (64 bits)</td>
<td>6592</td>
<td>10645</td>
</tr>
<tr>
<td>mAgicV (40 bits)</td>
<td>788</td>
<td>1113</td>
</tr>
</tbody>
</table>

Table 6.2. Kalman filter timing benchmarks.

Table 6.2 presents maximum execution times of the Kalman filter. The mAgicV DSP delivers results 7.67 times faster than the ARM9 processor working with a single precision floating point representation and 9.56 times faster for a double representation - calculated according to equation 6.8.

$$m\text{AgicV speedup} = \frac{\text{ARM9 exec time}}{\text{mAgicV exec time}}$$ (6.8)
(a) Kalman filter ARM9 float - pitch ($\phi$) angle.

(b) Kalman filter ARM9 float - roll ($\theta$) angle.
(c) Kalman filter ARM9 double - pitch ($\phi$) angle.

(d) Kalman filter ARM9 double - roll ($\theta$) angle.
(e) Kalman filter mAgicV - pitch ($\phi$) angle.

(f) Kalman filter mAgicV - roll ($\theta$) angle.

Figure 6.11. Kalman filter output data
For the mAgicV DSP core, the execution time also includes the time needed to transfer the data to and from the internal memory. The synchronization with the ARM9 is taken into account as well.

Conclusion

The state estimation based on a Kalman filter presented in the thesis demands higher computational power to operate compared to a complementary filter (as presented in table 6.2). On the other hand, the technique provides more state estimates, i.e. position and velocities when using an additional position sensor (e.g. a GPS receiver). The estimation of the position and velocities is required for implementing position stabilization (i.e. hovering control mode) thus making it more versatile than a system based on a complementary filter.

The use of the mAgicV DSP core gives 7.67 and 9.30 times faster execution when compared to the ARM9 float and double, respectively. The heavy computational problems (like the one described in this section) should be computed on the mAgicV working as an efficient coprocessor.
6.2 Hardware

This section describes two aspects of the DIOPSIS 940 evaluation influenced by specific hardware features of this processor. First, an interrupt latency problem is discussed. Then, the power consumption of the flight control system, including a comparison with other existing systems, is described.

6.2.1 Interrupt latency evaluation

Introduction

This section describes the experience with using the linux-2.6.24.3-rt3-d940 (with applied the “PREEMPT_RT” extension) and the linux-2.6.23.13-d940 as an actuator interface. It is responsible for both reading the data from a radio receiver (RC receiver) and generating of servo control signals by using the Johnson’s decade counter. A RC receiver usually provides 4 to 6 outputs which are in the form of a Pulse Width Modulation (PWM) signal. The typical PWM signal width is from 1 to 2 ms. Each servo channel is sampled with a 20 ms period. The most suitable technique for measuring a PWM signal from a RC receiver is to use the DIOPSIS 940 hardware interrupts and hardware Timer/Counter (TC) blocks. As described in subsection 4.4.1 this approach was not possible to realise due to the limited number of TC blocks and the interrupt service latency.

• Problem

The accuracy of the measurement of a PWM signal depends on a latency of the interrupt handling. The latency between rising hardware interrupt triggered by the detection of a state change (toggle) on a GPIO pin and execution of its handler code (written by a programmer) in the Linux (running on the ARM9 core) have been evaluated. It includes the Interrupt Service Routine (ISR) response time and a dispatch latency of the ISR. The first is the time between rising of the interrupt and the operating system response (the ISR low-level code execution). The second is the time necessary to start execution of the interrupt handler (high-level code) in the kernel process.

• Test setup

A digital oscilloscope was used to assess the latency by measuring the time difference between two signals. The first one was generated by the oscilloscope’s calibration output which was connected to a DIOPSIS 940 GPIO pin. The signal invoked an interrupt handler routine and generated a signal on a different GPIO pin. This signal was used as a second input for measuring the latency.

In order to conduct the experiment in a realistic setup, the operating system load was simulated by execution of a script which caused 70 percent CPU usage.
6.2 Hardware

- **D940 hardware priorities**
  
The DIOPSIS 940 hardware priorities configuration was set up to reduce the measured latency. The *Programmable Interrupt Controller (PIC)* was configured as follows. The highest hardware priority (7) was assigned to a system bus and the *Advanced Interrupt Controller (AIC)* peripherals. The one level lower priority (6) was set to the PIO (Peripheral IO) block, which controls GPIO pins. The same priorities were assigned to *Timer/Counter (TC)* (0 to 2) peripheral blocks. They have been responsible for measuring the duration of impulses from the *Radio Control (RC)* receiver and controlling a UAS platform’s actuators using the Johnson’s decade counter. Other peripheral blocks were assigned lower priority levels.

**Results**

- **linux-2.6.24.3-rt3-d940 (with PREEMPT_RT patch)**
  
The interrupt service routines are implemented as separate kernel threads in the PREEMPT_RT patched Linux. The *chrt* utility was used to change their real-time scheduling policy (to the SCHED_FIFO in this case) and priorities. The highest available value for a real-time task priority is 99 and the lowest is 1.
  
  For the experiment described above the high-resolution system timer had the highest priority. The kernel task responsible for generating signals to control a UAS platform actuators had priority of 80. The priority of 70 was assigned to routines responsible for reading input data from the RC receiver. Other tasks and interrupts had lower priorities.
  
  Measurement results are presented in table 6.3.

<table>
<thead>
<tr>
<th>Interrupt execution delay</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>without load</td>
<td>23 µs</td>
</tr>
<tr>
<td>with load [min]</td>
<td>27 µs</td>
</tr>
<tr>
<td>with load [max]</td>
<td>142 µs</td>
</tr>
</tbody>
</table>

**Table 6.3.** The interrupt execution delay of the “PREEMPT_RT” patched Linux operating system.

The maximum value of the relative error normalized to 1000 µs was calculated according to equation 6.9.

\[
e = \frac{MAX_{delay} - MIN_{delay}}{Trigger\ pulse\ time} = \frac{142 - 27}{1000} * 100% = 11.5% \quad (6.9)
\]

The error of 11.5 percent does not provide sufficient accuracy for the hard real-time flight control system. However its value should not increase due to the characteristic of the real-time Linux (PREEMPT_RT), used in this experiment. The real-time Linux assures bounded interrupt execution delay.
• linux-2.6.23.13-d940

Measurement results are presented in table 6.4.

<table>
<thead>
<tr>
<th>Interrupt execution delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>without load</td>
</tr>
<tr>
<td>with load [min]</td>
</tr>
<tr>
<td>with load [max]</td>
</tr>
</tbody>
</table>

Table 6.4. The interrupt execution delay of the Linux operating system.

The maximum value of the relative error normalized to 1000 µs was calculated according to equation 6.10.

\[
e = \frac{\text{MAX}_{\text{delay}} - \text{MIN}_{\text{delay}}}{\text{Trigger pulse time}} = \frac{82 - 7}{1000} \times 100\% = 7.7\%
\]  

\[(6.10)\]

The relative error of 7.7 percent is smaller than for a real-time Linux kernel (PREEMPT_RT). Even though the accuracy measurement was better it is still not acceptable for a UAS control. Additionally, the task responsible for the control of a UAS platform actuators had been occasionally delayed. Therefore, appropriate signals were not generated due to the absence of the preemption and real-time priorities support.

Conclusion

The DIOPSIS 940 is not particularly suitable for a RC receiver signal measurement and to control UAS actuators. The number of hardware timer (TC) blocks is insufficient. Moreover, the interrupt latency on tested Linux versions is unacceptable for interfacing to a UAS platform RC receiver and actuators in software.

Therefore a separate hardware circuit - the Servo Control System (SCS) has been developed. It replaces the Johnson’s decade counter. Description of the circuit is presented in section 4.4.
6.2 Hardware

6.2.2 DIOPSIS 940 flight control system power consumption

This subsection presents power consumption characteristics of the flight control system developed in this thesis.

The system utilizes 1.75 W (0.25 A * 7 V) for its operation. It is powered by a Li-Polymer battery with the nominal capacity of 1320 mAh and the voltage of 11.1 V.

<table>
<thead>
<tr>
<th>Flight control system</th>
<th>Power consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kestrel Autopilot 2.2</td>
<td>0.77 W @ 3.3 V &amp; 5 V</td>
</tr>
<tr>
<td>Micropilot MP2028g (with GPS receiver)</td>
<td>0.91 W (0.14 A @ 6.5 V)</td>
</tr>
<tr>
<td>Paparazzi Clasix Control Board</td>
<td>0.8 W (0.16 A @ 5 V)</td>
</tr>
<tr>
<td>LinkBoard</td>
<td>0.70 W (0.1 A @ 7 V)</td>
</tr>
<tr>
<td>DIOPSIS 940</td>
<td>1.75 W (0.25 A @ 7 V)</td>
</tr>
<tr>
<td>FCS20</td>
<td>1-3 W @ 7-20 V</td>
</tr>
</tbody>
</table>

Table 6.5. The comparison of flight control systems power consumption.

The power consumption of an on-board system will influence a UAV flight endurance. Table 6.5 shows the comparison of power consumption for systems presented in chapter 2.

The flight control boards equipped with microcontrollers (single or dual CPU solutions) need substantially less power to operate when compared to more complicated, mixed FPGA and DSP solutions. Therefore, microcontroller based systems can be used in wider range of platforms where power budget and weight is limited.

The DIOPSIS 940 based control system keeps a very good balance between the high mAgicV DSP performance and relatively small power consumption. This processor is a reasonable choice for computationally demanding, battery powered robotic and control systems.
Chapter 7

Summary

This chapter presents a summary based on the data gathered in previous chapters. It starts with the list of the DIOPSIS 940 advantages and disadvantages. Then, the final conclusion about usage of the DIOPSIS 940 processor in the flight control system is given. The chapter ends with possible future work on the flight control system project.

7.1 DIOPSIS 940 Advantages

• A performance improvement for computationally demanding tasks (i.e. Kalman filtering).

• The DSPLib library for the mAgieV. It contains many optimized functions.

• DSP(mAgieV) and ARM9 cores on the same chip - easy and efficient communication between the two.

• Usage of the Linux operating system on the ARM9 core - ease of development due to well documented API. The system functionality can be easily extended.

• A very good balance between the high mAgieV DSP performance and the relatively small power consumption.

7.2 DIOPSIS 940 Disadvantages

• Only three Timer/Counter (TC) blocks embedded in the AT572D940HF processor - not enough for a UAS control system application. Therefore, a separate Servo Control System (SCS) is needed.

• The Target Compiler toolchain cost - for this project time limited, 60 days evaluation version has been used.
• No native support for 64 bits double precision format on the mAgicV. Only 40 bits, extended floating point format support.

7.3 Final conclusion

The completion of this thesis required accomplishing of following tasks:

• Design, production, assembly and validation of two versions of the DIOPSIS 940 Back Module (DBM) and the Servo Control System (SCS) boards.

• Integration of the DBM, the DIOPSIS 940 CPU Module (DCM) and the SCS with the LinkMAV coaxial helicopter platform.

• Setting a convenient environment for embedded system development and building of the appropriate toolchain for the ARM9 core.

• Compilation of the Linux kernel version tailored to a UAS platform application. Specifically by applying kernel patches both real-time and Atmel specific and removing all unnecessary features.

• The design of the real-time drivers for communication with inertial sensors and the SCS.

• Understanding of the theory behind the flight control systems.

• Porting of the Kalman and the complementary filter code for the mAgicV DSP.

• Conducting the matrix multiplication experiment for the mAgicV performance analysis.

• The real-time control application development.

• The interrupt latency and a power consumption measurement.

There were two important turning points during work on this project:

• A design of the FPGA based Servo Control System (SCS) (described in the section 4.4).

• A development of new device drivers, which are not violating real-time features of the system as described in section 5.2.2.

Arguments presented in the previous chapters allow the author to conclude, that the AT572D940HF processor needs a specialised Servo Control System (SCS), as for example the one described in section 4.4, to control a UAS platform.

The maiden flight has not been performed, since this flight control system still requires some tuning and adjustments in the real-time control application. Nonetheless all necessary parts of the application (real-time drivers, the Kalman or the complementary filter, the PID control loop) have already been developed
and tested. The complete, ready to fly hardware platform has been built for this thesis.

A very important trade-off between the mAgicV usefulness and the Target Compiler toolchain cost has to be considered. Experimental results presented in section 6.1.3 shown that the DSP delivers significant performance improvement (up to 9.56 times faster than the ARM9 core). However, the mAgicV DSP development toolchain cost is very high compared to other available solutions. Therefore, each potential user of the DIOPSIS 940 has to calculate if the mAgicV performance will be worth the money invested in the Target Compiler purchase.

7.4 Future work

The following list presents possible future work for the flight control system described in this thesis.

- Outer PID control loop - position hold control mode.
- Integration of three previously described boards (i.e. DBM, DCM and SCS) into a single one.
- Evaluation of the system with other UAS platforms e.g. quadrotor configuration.
- Porting of the whole control application into the mAgicV DSP processor.
Bibliography


[22] Alex van der Wal. Combining real time with non real time software on a single Linux image, 2007.


Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/

© Lukasz Majewski