https://www.imavs.org/pdf/imav.2010.26

FPGA-Based Onboard Computer System for Mini Aerial Vehicles

Florian Weimer,* Martin Trittler,† Alexander Joos,‡
Michael Gros,§ Andre Posch¶ and Walter Fichter‖
iFR, Universitaet Stuttgart, 70569 Stuttgart, Germany

In this paper the design and development of an FPGA-based onboard computer system for Mini Aerial Vehicles is presented. First, a short overview of current commercial and noncommercial onboard computer systems for Mini Aerial Vehicles is given and then, advantages and drawbacks of different platforms are outlined. Together with the knowledge about current activities concerning control tasks on FPGAs, the decision was made to develop an onboard computer system based on a commercially available FPGA module. Different system designs are considered and implementation strategies for algorithms on programmable hardware are discussed. According to the advantages and disadvantages of each implementation strategy, an optimal system design for a basic waypoint navigation of a fixed wing model aircraft is chosen and validated in flight tests.

http://www.imavs.org/

I. Introduction

A. Available systems versus custom developments

Systems currently available on the market can be divided roughly into two subclasses, namely open source and commercial products. Usually these systems are merged with sensors and Guidance, Navigation and Control (GNC) software to autopilot packages. In terms of sensors, most of the systems available are equipped with an Inertial Measurement Unit (IMU), GPS, dynamic and static pressure sensors and a magnetometer. Software ranges from adjustable control algorithms to waypoint navigation capabilities. Some of the systems offer interfaces for payload communication and/or additional sensors, e.g. optical or ultrasound sensors.

Commercial available products like Micropilot,[1] Kestrel (Procerus),[2] Piccolo (CloudCap)[3] and MINC (Mavionics)[4] as well as open source systems like Paparazzi[5] and ArduPilot[6] are all equipped with microcontrollers or CPUs. The computing power varies from Atmel ATMega328 (ArduPilot) with 16 MIPS to RISC Processors with 150 MIPS (MINC, Micropilot).

Commercial products usually demand less installation, integration and configuration effort into the existing system due to components tested and validated with each other. The major drawbacks on the other hand are restricted accessibility to included software, limited flexibility for code changes and payload interface limitations. Open source systems like Paprazzi[5] or ArduPilot[6] require more effort for installation and integration, but those systems offer increased flexibility through open source code. However, if significant changes of the open source hard- or software are required, the development of a custom onboard computer system could be more efficient than the modification of an existing open source system.

There are different types of computer platforms for custom developments such as processor-based systems, microcontrollers, or Field Programmable Gate Arrays (FPGAs). The most common processor-based onboard computer system is the PC/104. The advantage of the PC/104 is the strong computing performance comparable to a common desktop computer. When a operation system is used that is supported by the MATLAB Realtime Workshop, the onboard computer system can be directly programmed from a

*PhD Candidate, Universitaet Stuttgart, florian.weimer@ifr.uni-stuttgart.de
†PhD Candidate, Universitaet Stuttgart, martin.trittler@ifr.uni-stuttgart.de
‡PhD Candidate, Universitaet Stuttgart, alexander.joos@ifr.uni-stuttgart.de
§PhD Candidate, Universitaet Stuttgart, michael.gros@ifr.uni-stuttgart.de
¶Student, Universitaet Stuttgart, andre.posch@ifr.uni-stuttgart.de
‖Professor, Universitaet Stuttgart, fichter@ifr.uni-stuttgart.de

MATLAB/Simulink environment. These advantages come with the drawback that PC/104 based platforms are relatively heavy and power consuming.

Weight and power consumption of microcontrollers are significantly less than that for a PC/104. The fact that a single integrated circuit contains processor, memory and peripherals makes them even more interesting as computing units when space and maximum weight are limited. In terms of computing performance, microcontrollers cannot compete with CPU-based systems. Hardware interfaces are also limited by the integrated peripherals available.

Due to the capabilities of the recent generation of FPGAs, their field of application has been dramatically extended. In former times FPGAs have been used mainly as glue logic. In 1985, Xilinx invented the first commercially available FPGA which integrates the equivalent of 1200 logic gates. Recent Spartan-3A devices from Xilinx are rated with up to 3.4 million logic gates and the number is still increasing. Some of the recent FPGA families also offer embedded ressources like multipliers, high-speed-transceivers and PowerPC processors. These properties made FPGAs to serious competitors of microcontroller- or CPU-based systems which do not offer the flexibility to modify the system architecture without changing the hardware itself. The ability to process code in parallel is also a unique feature of FPGAs, because algorithms can be synthesized to independent electrical circuits. This opportunity also guarantees cycle-accurate timings. If code runs in time once on the FPGA, this code will always comply with timing constraints even if additional code is added. These benefits, on the other hand, come with limitations in space. Since algorithms are translated to electrical circuits, they also need a certain amount of logic cells. Furthermore, programming and handling of an FPGA is more complex than programming of microcontrollers or CPU-based systems.

http://www.imavs.org/

B. Recent and current research activities

1. Onboard computer systems

The Institute of Space Systems at the Universitaet Stuttgart is working on an onboard computer system based on a FPGA for the "Stuttgart Small Satellite Program" with the final goal to launch a small spacecraft to the Moon.[7] The onboard computer system is based on a Xilinx Virtex-II Pro FPGA with an equivalent of approximately 3 million logic gates. On this satellite, the FPGA-based system is the only computing device and all functionalities need to be implemented on the FPGA chip.[7] Hardware-Implementation of the attitude control system and the on-board navigation system is described by Grillmayer.[8]

The Georgia Institute of Technology uses a FPGA based onboard computer system for flight testing a fixed wing aircraft.[9] The FPGA itself is used for sensor communication, data processing and sending sensor data to a DSP, in which the GNC algorithms are running.

Stepaniak[10] describes the attitude determination of a quadrotor vehicle implemented on a FPGA. Sensor interface, radio interface attitude update and control loop are implemented in hardware on a Xilinx Spartan-3 FPGA.

2. Control algorithms

Minghua He[11] demonstrates the implementation of a constrained model predictive control algorithm on a FPGA chip in hardware. The code is first prototyped in Matlab and translated into Handel-C code for implementation on the FPGA. Hardware verification was done in a hardware-in-the-loop simulation. An MPC with 128 inequality constraints with the dimension of $Q = 45$ was implemented on a Virtex II FPGA with an equivalent of one million gates.

Vouzis[12] presents another implementation of an MPC on an FPGA where computationally demanding parts of the algorithms are implemented in hardware and other parts are written in software. The algorithm is programmed in the hardware description language HDL. The FPGA used in this reference is a Xilinx Virtex-IV.

The implementation of a constrained model predictive control on an FPGA with parallel computing is described by Ling.[13] A sequential and a parallel implementation are compared in this paper where the parallel implementation outperforms the sequential implementation in computational speed, while hardware effort increases modest.

Johansen[14] presents the idea of implementing an explicit model predictive controller on an FPGA with direct hardware synthesis. Explicit model predictive controllers are precomputed and represented by piecewise linear functions that are evaluated in the control process without the need for optimization.

C. Objectives

This paper reports on the development and testing of an FPGA based onboard computer system for Mini Aerial Vehicles (MAV) with low weight and power consumption and the ability to act as a flexible computing platform for various MAV types. The chosen FPGA concept provides the freedom to optimize components of the algorithm for precisely timed parallel or sequential execution. Thus full control over hardware and software is assured.

Design and development of the onboard computer system is given in Chapter II. Special care is taken of the decision of software versus hardware implementation strategies to illustrate advantages and limitations of the FPGA. Chapter III presents vehicles currently equipped with the FPGA based onboard computer system. In Chapter IV the work is concluded and future steps in hardware development are outlined.

II. Design and Development

A. System design considerations

The current onboard computer system is based on a Spartan-3E FPGA with an equivalent of 1.6 million gates (XC3S1600E). The FPGA Module itself is a commercial of the shelf (COTS) component and is located on a custom made carrier board. The system architecture is depicted in Fig. 1.

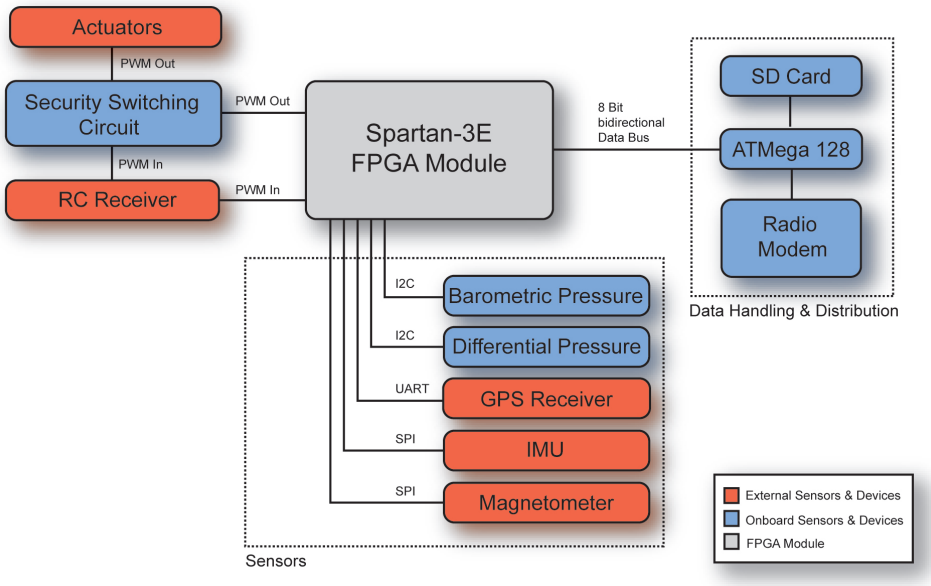


Figure 1: System architecture of onboard computer

The IMU, Magnetometer and GPS receiver are kept as external components because the placement of these sensors in the flight vehicle might have a big impact on their performance. To keep the IO flexibility, additional connectors for I2C and three analog signals are placed on the carrier board.

Data logging and communication are done with the the help of an ATMega128 microcontroller running at 16 MHz. The ATMega128 manages the transmission and reception of dataframes to and from the ground-station as well as the logging of relevant information to an SD card. It is connected to the FPGA by an 8 bit bidirectional data bus which allows the ATMega128 to request data from the FPGA or to pass information to the flight controller. The downlink to the groundstation is established by a 2.4 GHz XBee Pro radio modem.

Up to 7 PWM-driven actuators can be connected to the flight computer. To guarantee a safe operation, a security mechanism has been designed which allows one to switch between a direct mode and a controller mode. For the direct mode, all actuator signals generated by the flight controller sofware are disabled and the inputs from the RC receiver are directly routed to the PWM outputs. In controller mode, the PWM

http://www.imavs.org/

outputs are generated by the flight controller software and the raw RC receiver commands are ignored. The security mechanism itself is a hardware circuit to avoid failures induced by the program code on the FPGA. The switching is done by means of a PWM signal which is generated by a standard remote control.

The FPGA itself holds the main part of the onboard computer system. Algorithms can be translated to electrical circuits and mapped directly to the logic cells of the FPGA without using any processor or microcontroller architecture at all. The advantage of such a design is its realtime behaviour. Timings within the system are cycle accurate and no jitter is involved. The parallel processing capabilities of the FPGA can be used to collect sensor data without any interrupts during the runtime of the algorithm itself. It is also possible to execute different parts of an algorithm in parallel, which decreases cycle times dramatically. Unfortunately there are also some drawbacks when using a pure hardware design. Because everything is mapped to electrical circuits, a certain amount of statically allocated logic is needed which grows with algorithm complexity.

If execution time of the algorithm has no hard constraint, it may be better to map a mirocontroller to the FPGA logic. Xilinx provides the Microblaze soft processor core, which is a 32 bit RISC Harvard architecture and has an optional IEEE-754 compatible Floating Point Unit (FPU) and a Memory Management Unit (MMU). The maximum clock frequency is always dependent on the configuration, the target FPGA architecture and the speed grade. The maximum clock frequency for the Spartan-3E architecture is rated at 106 MHz according to Xilinx. The maximum Dhrystone Performance is rated at 126 DMIPS when using a performance-optimized configuration with 5-Stage pipeline. The required space by the soft processor core is then fixed.

It is also possible to use a soft processor core and an additional piece of hardware implementation and split the task amongst them.[15] This allows to run time-critical applications or algorithms that can be parallelized in hardware while less time-critical code runs on the soft processor core. At least the major goal of the design process is to get a well-balanced System on a Programmable Chip (SOPC) with respect to the application.

The Microblaze soft processor core is fully customizable and the user has to decide its configuration according to the FPGA in use and the task which has to be performed. For the performance tests in subsection C, the Microblaze has been configured according to table 1.

Table 1: Microblaze settings used for performance tests. This configuration requires 4512 Slices(30%), 7 MULT18x18SIOs(19%) and 24 RAMB16s (66%) of the XC3S1600E device

Clock Frequency	83.33 MHz
Floating Point Unit	BASIC
Instruction Cache	8 kB
Data Cache	8 kB
DDR RAM	512 MBit

Setting the FPU to BASIC in the Microblaze settings increases the single-precision floating point performance for addition, subtraction, multiplication, division and compare operations. The pipeline is reduced from 5 to 3 stages for area optimization. The code and data section of the software running on the Microblaze are located in the DDR RAM of the FPGA Module. The instruction and data cache increase the performance of the soft processor core significantly when using DDR memory for code and data section.

B. Implementation strategies for programmable hardware

There are several tasks which have to be performed by MAV onboard computer systems like sensor data aquisition, communication, sensor data fusion and flight control. Especially the last two tasks mentioned require mathematical operations, such as trigonometric functions. Another requirement is an appropriate number representation since there are no floating- or fixed-point operations available by default. Usually algorithms used in the standard software approach do not map well into hardware. Therefore it is important to use hardware-efficient algorithms. Simple shift and add operations map especially well into hardware and most of the functionalities needed could be implemented just by these operations. Instead of Hardware Descripion Languages (HDL) like VHDL or Verilog, the programming language Handel-C was used throughout the development process. Handel-C provides a rich subset of C which includes all common C language features necessary to describe complex algorithms.

http://www.imavs.org/

https://www.imavs.org/pdf/imav.2010.26

1. Number representation

For the hardware implementation of navigation solution and control algorithm, an 18 bit fixed point format is used. The location of the decimal point is chosen according to the maximum absolute values expected for each data type in use. The reason for the 18 bit representation are the dedicated hardware multiplier blocks available on the Spartan-3E FPGA, which are able to perform 18 bit twos complement numerical multiplication. Wider multiplications require additional slice-based logic.[16]

2. Multiplication

For the XC3S1600E, the amount of available harware multipliers is limited to 36. If the number of parallel multiplications exceeds the number of dedicated multipliers available, additional multipliers are generated from slice-based logic. Unless the parallel processing power is needed, one has to avoid slice-based multipliers.

3. Division

The standard division operator in Handel-C performs an integer division in just one clock cycle. This behaviour may be good for high speed applications but it is not preferrable for the application discussed in this paper. The division is the most complicated arithmetic standard operation and there are no dedicated hardware elements that could handle it. If the division is carried out in just one clock cycle, it requires a lot more slice-based logic compared to a serialized implementation. The achievable system frequency is also reduced because of the signal progagation delay, which arises in that case. To overcome these problems, a digit-recurrence division algorithm is used. For an integer division of X by Y, the following equation can be written:

$$B^p X = QY + R \quad with \quad -Y \leq R < Y \quad and \quad sign(R) = sign(X) \tag{1}$$

Q and R are the quotient and remainder of the division. B^p defines the number p of fractional base-B digits for the fixed point representation and B^{-p} is the unit in the least significant position. Based on Eq.(1) the recurrence

$$r[i+1] = Br[i] - q[i+1]Y \ , \ i = 0, 1, .., (p-1) \tag{2}$$

can be formulated, where $r[0] = X$ and $q[i+1]$ is selected so that the residuals $r[i+1]$ remain bounded. After p iterations, the quotient and remainder of the division are given by

$$Q = \sum_{i=1}^{p} q[i]B^{p-i} \ , \ R = r[p]. \tag{3}$$

Equation (2) can be computed using different algorithms which can be distinguished according to the way $r[i+1]$ and $q[i+1]$ are chosen. In our case, we adopted a Base-2 nonrestoring algorithm from Deschamps et al.[17] The decision about $r[i+1]$ and $q[i+1]$ only depends on the sign of $r[i]$ and is defined as follows:

$$r[i] > 0 \rightarrow q[i+1] = +1 \ , \ r[i+1] = 2r[i] - Y \tag{4}$$
$$r[i] < 0 \rightarrow q[i+1] = -1 \ , \ r[i+1] = 2r[i] + Y \tag{5}$$

The overall procedure is shown in algorithm 1.

The Handel-C implementation of the algorithm requires $p + 4$ clock cycles to compute the result. By setting $B = 2$, the mathematical operations can be reduced to bitshifting, addition and subtraction.

4. Trigonometric functions

Trigonometric functions such as sine and cosine could also be efficiently implemented just by shift and add operations. There is a class of iterative solutions for trigonometric functions which is called COordinate

http://www.imavs.org/

Algorithm 1 Base-2 Nonrestoring Division

```
r(0)=X;
for i = 0 to p-1 do
  if r(i) < 0 then
    Q(i)=0; r(i+1) = 2*r(i)+Y;
  else
    Q(i)=1; r(i+1) = 2*r(i)-Y;
  end if
end for
Q(0)=1-Q(0); Q(p)=1; R=r(p);
if X ≥ 0 and R < 0 then
  R=R+Y; Q=Q-1;
else if X < 0 and R ≥ 0 then
  R=R-Y; Q=Q+1;
end if
```

Rotation DIgital Computer (CORDIC). The original work is credited to Jack Volder[18,19] and was originally developed for real-time navigation problems. Basically, the method consists of computing the new coordinates of a given vector after rotation by a given angle according to the standard rotation formula given by

$$\begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{6}$$

If the rotation angle ϕ is replaced by pseudorotations ϕ_i, Eq.(6) can be expressed as

$$\begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} = (\prod_{i=0}^{n-1} \cos\phi_i) \prod_{i=0}^{n-1} \begin{bmatrix} 1 & -\tan\phi_i \\ \tan\phi_i & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{7}$$

For the CORDIC algorithm, the pseudorotations ϕ_i are substituted by

$$\phi_i = \arctan d_i 2^{-i}\ ,\ d_i \in \{-1, 1\} \tag{8}$$

and the CORDIC iteration process in circular mode can be written to

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = K_i \begin{bmatrix} 1 & -d_i 2^{-i} \\ d_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}. \tag{9}$$

The CORDIC Algorithm involves an implicit scaling factor K_i. If the number of iterations goes to infinity, the scaling factor approaches

$$K = \lim_{n\to\infty} \cos\arctan 2^{-i} = \lim_{n\to\infty} \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}} \approx 0.607253. \tag{10}$$

Because of $cos(-\phi) = cos(\phi)$, K can be precomputed and isolated from the iterative process where only bitshifting, addition and subtraction are required.

Depending on the side condition for the rotation direction d_i, either the angle z is accumulated to zero (circular rotation , $z_{i+1} \to 0$) or the result vector is aligned with the x-axis (circular vectoring, $y_{i+1} \to 0$). Based on the values chosen for x_0, y_0 and z_0, each mode of opperation leads to different results. In circular rotation mode e.g., sine and cosine can be obtained and in circular vectoring mode, one can get the inverse functions. The values for $\arctan 2^{-i}$ are prestored in a lookup table. Andraka[20] surveys existing CORDIC algorithms with an eye towards implementation in FPGAs in detail.

http://www.imavs.org/

https://www.imavs.org/pdf/imav.2010.26

CORDIC algorithms for $\sin, \cos, \arctan$ and $\arcsin$ have been implemented in Handel-C. Since the accuracy of the standard $\arcsin$ CORDIC suffers due to gain problems[20] as the input approaches ± 1, a so called double iteration algorithm[21] was used. All operations are carried out in 3.15 fixed point format.

C. Software versus hardware for typical UAV algorithms

In this section, the realization of typical algorithms for UAVs in either software or hardware are compared and discussed. Since these two kinds of implementations are of completely different nature it is not straightforward to do a meaningful comparison. After the introduction of the investigated algorithms, the performance criterions and their relevance are discussed and applied.

1. Benchmark definition

Almost every computer algorithm for UAV applications consists of some logic operations and basic algebra. Therefore we will first have a look at the performance of some basic algebraic functions like multiplication, division and trigonometric functions. After the examination of these basic functions, we will analyze the strapdown algorithm and a PID controller, which are exemplary for typical algorithms with relatively low complexity.

The strapdown algorithm calculates geodetic velocity ${}_g\mathbf{v}$, position ${}_g\mathbf{r}$ and attitude $\mathbf{q}$ from bodyfixed acceleration and rotational speed measurements, ${}_f\mathbf{a}$ and ${}_f\omega$ of an inertial measurement unit (IMU). Assuming a stationary flat earth and a small range UAV the differential equations for translational degrees of freedom are

$$ {}_g\dot{\boldsymbol{r}} =_g \boldsymbol{v} \tag{11} $$

and

$$ {}_g\dot{\boldsymbol{v}} = \boldsymbol{T}_{gf} \cdot_f \boldsymbol{a} +_g \boldsymbol{g}_l, \tag{12} $$

with ${}_g\boldsymbol{g}_l$ the local gravity acceleration. The direction cosine matrix to transform measured bodyfixed acceleration to the geodetic frame is defined as follows:

$$ \boldsymbol{T}_{gf} = \begin{bmatrix} (q_0^2+q_1^2-q_2^2-q_3^2) & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & (q_0^2-q_1^2+q_2^2-q_3^2) & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & (q_0^2-q_1^2-q_2^2+q_3^2) \end{bmatrix}, \tag{13} $$

with the attiude quaternion $\boldsymbol{q} = (q_0,q_1,q_2,q_3)^T$. The attitude dynamics is descibed by the quaternion differential equation

$$ \dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{W} \cdot \boldsymbol{q} \tag{14} $$

with

$$ \boldsymbol{W} = \begin{bmatrix} 0 & -\boldsymbol{\omega}^T \\ \boldsymbol{\omega} & [\boldsymbol{\omega}]_\times \end{bmatrix}. \tag{15} $$

The numerical integration is done with the simple Euler forward method. More detailed information on strapdown navigation systems could be found in.[22] In combination with the used low cost IMU it is necessary to correct the strapdown navigation solution by e.g. GPS measurements using a Kalman Filter to get satisfactory results. Such a filter is used in the application described in chapter III but not investigated in this paper. Algorithm 2 gives the details of the examined strapdown implementation.

http://www.imavs.org/

https://www.imavs.org/pdf/imav.2010.26

Algorithm 2 Strapdown Algorithm

Calculate $\boldsymbol{T}_{gf}$, Eq. 13 [10 Mult, 15 Add/sub]
Calculate ${}_g\dot{\boldsymbol{v}}$, Eq. 12 [9 Mult, 7 Add/sub]
Calculate ${}_g\boldsymbol{v}$, solve Eq. 12 [3 Mult, 3 Add/sub]
Calculate ${}_g\boldsymbol{r}$, solve Eq. 11 [3 Mult, 3 Add/sub]
Calculate $\dot{\boldsymbol{q}}$, Eq. 14 [12 Mult, 8 Add/sub]
Calculate $\boldsymbol{q}$, solve Eq. 14 [3 Mult, 3 Add/sub]

The realization of a PID controller of the form

$$\boldsymbol{u} = k_P \cdot \boldsymbol{e} + k_I \cdot \int \boldsymbol{e} + k_D \cdot \frac{d}{dt}\boldsymbol{e} \tag{16}$$

with a simple anti-windup scheme and output limitation is given in algorithm 3. The summation of the integral part of the control deviation $e_I(k)$ is stopped if the control output $u(k)$ reaches its allowed maximum u_{max} or minimum u_{min}. Integral and differential control deviation $e_D(k)$ are calculated from the actual and last control deviation $e(k)$ and $e(k-1)$.

Algorithm 3 PID Controller

if $u(k-1) < u_{max}$ and $u(k-1) > u_{min}$ **then**
  $e_I(k) += dT/2 \cdot (e(k) + e(k-1))$
end if
$e_D(k) = e(k) - e(k-1)$
$u(k) = k_P \cdot e(k) + k_I \cdot e_I(k) + k_D \cdot e_D(k)$
if $u(k) > u_{max}$ **then**
  $u(k) = u_{max}$
else if $u(k) < u_{min}$ **then**
  $u(k) = u_{min}$
end if

2. Results and discussion

Even at the relatively low clock rate of 24 MHz, the hardware implementation of algorithms or algebraic functions is significantly faster than its software counterpart, except for the base-2 nonrestoring division. If the Handel-C standard division operator is used instead, the consumption of slice based logic increases, because the operation is carried out in just one clock cycle. Furthermore, the length of the critical path increases which on the other hand decreases the maximum achievable clock frequency of the synthesized algorithm.

If the algorithm involves parallel multiplications, the usage of the avilable MULT18x18SIOs reduces the consumption of slice based logic significantly. If multiplications are carried out sequentially, there is no big benefit when using hardware multipliers.

The number of occupied slices in general is dependent on the FPGA family chosen, because the logic elements of a slice could differ. Also, a slice is not necessarily fully occupied, but it seems well established to evaluate the implementation of logic for FPGAs. Table 2 shows the consumption of hardware slices and execution cycles of the algebraic functions introduced in subsection B for the target FPGA family.

The serial nature of the Microblaze soft processor core makes execution time of the needed operations the main performance criterion. The Microblaze software implementation is mainly based on single precision floating point operations which makes software developement very convenient in contrast to the fixed point implementation in Handel-C. Table 2 shows the execution time of some algebraic functions for the implemented Microblaze soft processor core.

In table 3 the investigated algorithms are summarized. The hardware realization of the strapdown algorithm consumes about 9% of the available slices and 6 of 36 hardware multipliers on the target FPGA. The

http://www.imavs.org/

Table 2: Hardware slices and multiplier consumption and execution cycles for hardware plus execution time for Microblaze software implementation of algebraic functions. Hardware operations working on 18 bit fixed point numbers, Microblaze software on single precision floating point numbers.

	Hardware			**Microblaze**
Algebraic Function	**Slices**	**MULT18x18SIOs**	**Cycles**	**Ex.time**
Multiplication (ALU Mapping)	1	1	1	–
Multiplication (Logic)	105	0	1	290 ns
Division (Serialized)	163	0	24	580 ns
Division (Single Step)	290	0	1	580 ns
Sine/Cosine	239	1	36	2/2.56 μs
Atan2	216	0	34	6.01 μs
Asin	267	0	49	9.23 μs

algorithm is not parallelized because this would unnecessarily increase the hardware consumption since execution time for the sequential implementation is fast enough. The need for 6 hardware multipliers originates in the need of multiplying 18 bit fixed point numbers of different formats. Every multiplication of e.g. 3.15 and 5.13 format numbers needs its own multiplier, which is then used sequentially. The computation of the strapdown algorithm in hardware needs 63 cycles. With a clockrate of 24 MHz, this results in a execution time of 2.625μs which is well an order of magnitude faster than the Microblaze software implementation.

The PID controller consumes 230 slices of the FPGA which is basically because not so many variables are involved that have to be preserved. Again, the hardware implementation is signifficantly faster than its software counterpart.

Table 3: Hardware slices and multiplier consumption and execution cycles for hardware plus execution time for Microblaze software implementation of strapdown and PID controller. Hardware operations working on 18 bit fixed point numbers, Microblaze software on single precision floating point numbers.

	Hardware			**Microblaze**
Algorithm	**Slices**	**MULT18x18SIOs**	**Cycles**	**Ex.time**
Strapdown	1375	6	63	12.75 μs
PID Controller	230	1	18	2.80 μs

In our case, hardware implementation is at minimum an order of magnitude faster than software implementation although the algorithms are basically designed to work sequentially to save hardware consumption. Software implementation is more convenient to program since existing libraries and floating point numbers can be used. The limitation is that all necessary functions have to be executed in a certain time to keep the desired bandwith of navigation and conrol algorithms.

D. Proposed system design

Based on the results of the comparison between software and hardware implementation, we decided to use the Microblaze soft processor core as the main processing unit for navigation and control tasks. The sensor data aqusition and the signal routing is done by a custom made intellectual property (IP) core which is connected to the Processor Local Bus (PLB) of the Microblaze. The design is shown in Fig. 2.

Since the navigation and control algorithms run at a frequency of 100 Hz, there is no need for the significantly faster hardware implementation. The Microblaze performs all tasks required in approximately 4 ms, which still leaves 6 ms for future modifications. The implementation effort for the software development is also much lower than for hardware designs because of the floating point operations and the standard libraries available. Sensor data aquisition on the other hand is best done in hardware. This allows to aquire the data of each sensor in parallel without using interrupts. Hardware interfaces are also kept independent

http://www.imavs.org/

from the software design. The replacement of sensors has no impact on the software-side of the system as long as the specifications are similar.

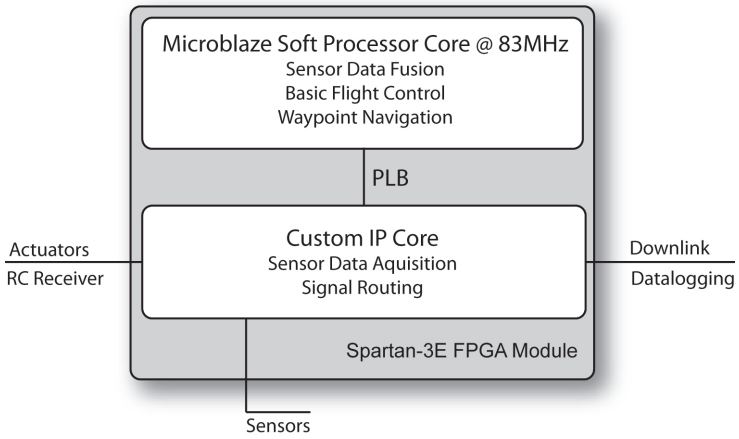
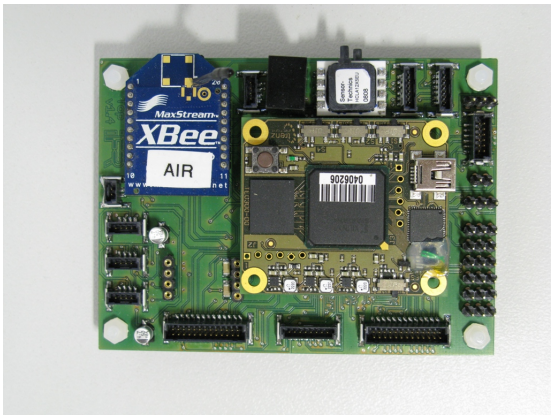


Figure 2: System design

III. Current Applications

The basic idea behind the onboard computer system developed, is to keep it independent from a specific platform. One of the platforms in use is a standard fixed-wing model aircraft with a wingspan of $1.6m$. The airplane is able to do automatic takeoffs as well as an automatic waypoint navigation so far. Attitude, position and additional system information is transmitted to a groundstation running on a notebook computer. It is also possible to send certain comands to the onboard flight computer, e.g. to change waypoints or to finetune parameters of the controllers. At fist, software and hardware has been testet in the loop using Mathworks XPC Target realtime operating system and a 6-DOF model of the aircraft dynamics. Afterwards the algorithms have successfully been tested on the real airplane.

(a) Bo Monsun model aircraft



(b) Flight computer hardware

Figure 3: Fixed-wing demonstrator

The flight computer has also been used during flight tests of an unmanned blimp. The main task was to collect flight data for system identification and to monitor the overpressure valves. Due to the flexibility of the hardware interfaces, the flight computer could quickly be adopted to the new vehicle without any hardware modifications.

http://www.imavs.org/

IV. Conclusion

The design and devolpment of an FPGA-based onboard computer system has been presented and different aproaches in algorithm implementation have been evaluated. While a pure hardware implementation of the algorithms offers great performance and cycle accurate timings it also consumes a lot of logic on the target FPGA. The fixed point format offers limited precision and the effort in applying a correct scaling to all values and avoiding overflows is very high. The soft processor core on the other hand is easy to handle thanks to the common programming techniques for microcontrollers, the ability to use floating point operations and the availability of ANSI C standard libraries. The performance of the Microblaze soft processor core is way below a pure hardware implementation in terms of throughput but as long as the desired cycle time is met it is the better way to go. The hardware consumption of the Microblaze is relatively low and mainly driven by the required size of storage elements like cache or RAM.

The posibility to attach custom IP cores to the Microblaze allows the use of FPGA ressources for more specific and timing-critical tasks, which are well suited for hardware implementation. In general this could be a function that has to be evaluated several times but with different input parameters or the parallel aquisition of sensor data. In future hardware developments, a CPU-based computer module will be added to the onboard computer system to free FPGA logic for more specific tasks. The main work will be related to the communication between both computing devices since a low bandwidth in communication could neutralize the benefits of the FPGAs processing power.

References

[1]http://www.micropilot.com [cited 01 February 2010]

[2]http://www.procerusuav.com [cited 01 February 2010]

[3]http://www.cloudcaptech.com [cited 01 February 2010]

[4]http://mavionics.de [cited 01 February 2010]

[5]http://paparazzi.enac.fr [cited 01 February 2010]

[6]http://diydrones.com/profiles/blogs/ardupilot-main-page [cited 01 February 2010]

[7]Kuwahara, T., Boehringer, F.,Falke, A., Eickhoff J., Huber, F. "FPGA-based operational concept and payload data processing for the Flying Laptop satellite", *Acta Astronautica*, Vol. 65, 2009, pp.1616-1627.

[8]Grillmayer, G., Hirth, M., Huber, F., Wolter, V. "Development of an FPGA Based Attitude Control System for a Micro-Satellite", *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 21-24 August 2006, Keystone, Colorado, AIAA 2006-6522.

[9]Johnson, E.N., Turbe, M.A., Wu, A.D., Kannan, S.K., Neidhoefer, J.C., "Flight Test Results of Autonomous Fixed-Wing UAV Transitions to and from Stationary Hover", *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 21-24 August 2006, Keystone, Colorado, AIAA 2006-6775.

[10]Stepaniak, M.J., de Haag, M.U., van Graas, F., "Field Programmable Gate Array-Based Attitude Stabilization", *Journal of Aerospace Computing, Information, and Communication*, Vol.6, Juli 2009, pp. 451-463.

[11]He, M., Ling, K.-V., "Model Predictive Control On A Chip", *IEEE ISIE 2006*, July 9-12, 2006, Montreal, Quebec, Canada pp. 228-233.

[12]Vouzis, P.D., Bleris, L.G., Arnold, M.G., Kothare, M.V., "A Custom-made Algorithm-Specific Processor for Model Predictive Control", *2005 Intemational Conference on Control and Automation (ICCA2005)*, June 27-29, 2005, Budapest, Hungary, pp. 528-532.

[13]Ling, K.V., Wu, B.F., Maciejowski, J.M., "Embedded Model Predictive Control (MPC) using a FPGA", *The International Federation of Automatic Control*, Seoul, Korea, July 6-11, 2008, pp. 15250-15255.

[14]Johansen, T.A., Jackson, W., Schreiber, R., Tondel, P., "Hardware Synthesis of Explicit Model Predictive Controllers", *IEEE Transactions on Control Systems Technology*, 2007, Vol. 15, No.1, pp. 191-197.

[15]Xilinx, *MicroBlaze "System Performance Tuning"*, WP348, v1.1, August 2008.

[16]Xilinx, *"Spartan-3E FPGA Family: Datasheet"*, DS320 (v3.8), August 2009.

[17]Deschamps, J.-P., Bioul, G. J. A., Sutter, G. D., *"Synthesis of arithmetic circuits: FPGA, ASIC and embedded systems"*,Wiley, New Jersey, 2006.

[18]Volder, J., *"Binary computation algorithms for coordinate rotation and function generation"*, Convair Report IAR-1 148 Aeroelectrics Group, June 1956.

[19]Volder, J., *"The CORDIC Trigonometric Computing Technique"*, IRE Trans. Electronic Computing, Vol EC-8, pp330-334, September 1959.

[20]Andraka, R., *"A survey of CORDIC algorithms for FPGA based computers"*, Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, February 1998, Monterey,CA. pp191-200.

[21]Mazenc, C., Merrheim, X., Muller, J.-M. *"Computing Functions cos^{-1} and sin^{-1} Using Cordic"*, IEEE Transactions on Computers, Vol.42, No. 1, January 1993.

[22]Titterton, D.H., Weston J.L., "Strapdown Inertial Navigation Technology - 2nd edition", *Institution of Engineering and Technology and AIAA*, 2004.

http://www.imavs.org/