FPGA Based UAV Flight Controller

Justin Young [1], Andrew Ross Price [2],

[1] *Aerobotics Group, Dept Electrical and Computer Systems Engineering, Monash University, Melbourne ,Victoria, 3800 Australia*
[2] *Aerobotics Group, Dept Electrical and Computer Systems Engineering, Monash University, Melbourne ,Victoria, 3800 Australia. Andrew.price@eng.monash.edu.au*

Summary: Small scale, light weight Unmanned Aerial Vehicles (UAVs) have very limited payload capacity. Yet the demands for greater mission complexity as well as increased flight performance tend towards increased onboard processing. This research examines the idea of designing a UAV control system in a low cost FPGA device using a custom designed processor core that has been specifically tailored to minimize the sensor interface. Data received by the sensor system is available directly to registers within the processor, and is updated by independent means. The processor is free of data gathering activities thanks to the parallel nature of FPGA design. This approach leads to a compact low cost system that has been optimized toward lightweight UAV control systems. This research has been conducted as part of an undergraduate thesis project at Monash University.

Keywords: UAV, Control, FPGA.

Introduction

As with most typical control systems, a flight controller consists of an input stage, control or processing stage and an output stage. The input stage is realised in the form of sensors measuring parameters such as static and dynamic air pressure, temperature, acceleration, etc. During the processing stage the input data can be used to determine the value of states such as altitude, spatial location as well as the pose of the aircraft. Further, the processing stage is then able to use the value of these states to determine an action to implement in the output stage where, in the case of a UAV, is usually control surface manipulation or throttle. The control processing stage is implemented in a Field Programmable Gate Array (FPGA). FPGAs can be advantageous over using commercial off the shelf processing platforms because they are easily expandable and configurable to allow for new features or different approaches to solving a problem. The flexibility of an FPGA allows for three abstract design philosophies: hardware implemented control, software implemented control on a processor core or a combination of hardware and software implementation. As outlined later in the paper, it was decided to use a combination of hardware and software. Control is mainly implemented in software run on a soft processor core with support for input and output coming from hardware modules interfaced to the core.

The processor core used was based loosely on a simple single instruction per clock cycle MIPS pipeline [1]. The example provided a simple 8-bit system with limited capabilities however beginning with a simple core made radical modifications to the architecture easier to perform. A new I/O system was developed to make data transfer to and from physical devices (or their controllers) easier for the user and less load on the Central Processing Unit (CPU). The output system consists of a series of 32-bit wide registers which the user can manipulate as easy as modifying the processors internal registers. Similarly the input data is provided to the processor on a number of 32-bit bus structures and can be accessed through register

transfer commands. This allows users to easily interact with the real world sensors without wasting CPU cycles. While the processor core is not a completely optimized processing solution, it is useful enough to run a simple flight control system and test the FPGA design philosophy selected.

Hardware modules take the responsibility of performing CPU intensive tasks in a faster fashion and without the need to be controlled by the CPU. This method lets the CPU concentrate on controlling the aircraft, with further CPU cycles still available for development of new features such as computer vision tasks. Hardware modules interact with the processor using the 32-bit I/O bus system described earlier, and a simple interrupt scheme. Examples of hardware modules include a servo pulse width modulator, and a serial to parallel converter for reading data from an SPI interface.

System Design

Before implementing the complete platform each philosophy was investigated to determine which to pursue. Table 1 shows the advantages and disadvantages of each of the philosophies.

Hardware Implementation	
Advantages • Fast, in some case immediate response to input changes • Easy to implement on FPGA • Adjusting an existing algorithm can be difficult	Disadvantages • As system complexity increases, interfacing of components becomes more difficult. • For asynchronous systems multiple instances of a component may be required for multiple control path, hence increasing FPGA resource usage.
Software implementation on soft processor core	
Advantages • Flexibility in design through changes in software. • Allows components to be instantiated once on the FPGA and used for multiple tasks sequentially, hence reducing FPGA resource usage.	Disadvantages • No better than use of a microprocessor chip
Combination of Hardware and Software	
Advantages • Can change hardware and software to achieve goal in a new way • CPU intensive tasks can be performed in hardware	Disadvantages • Time need to be spent developing interfaces between software and hardware.

Table 1: Advantages and disadvantages of various FPGA design philosophies

As Table 1 shows, by combining a hardware and software based solution we are able to maximise the design flexibility, while maintaining a simple way to develop the algorithms. Further, the system can easily handle a large number of 32 bit I/O devices without adversely loading the CPU to control the I/O system.

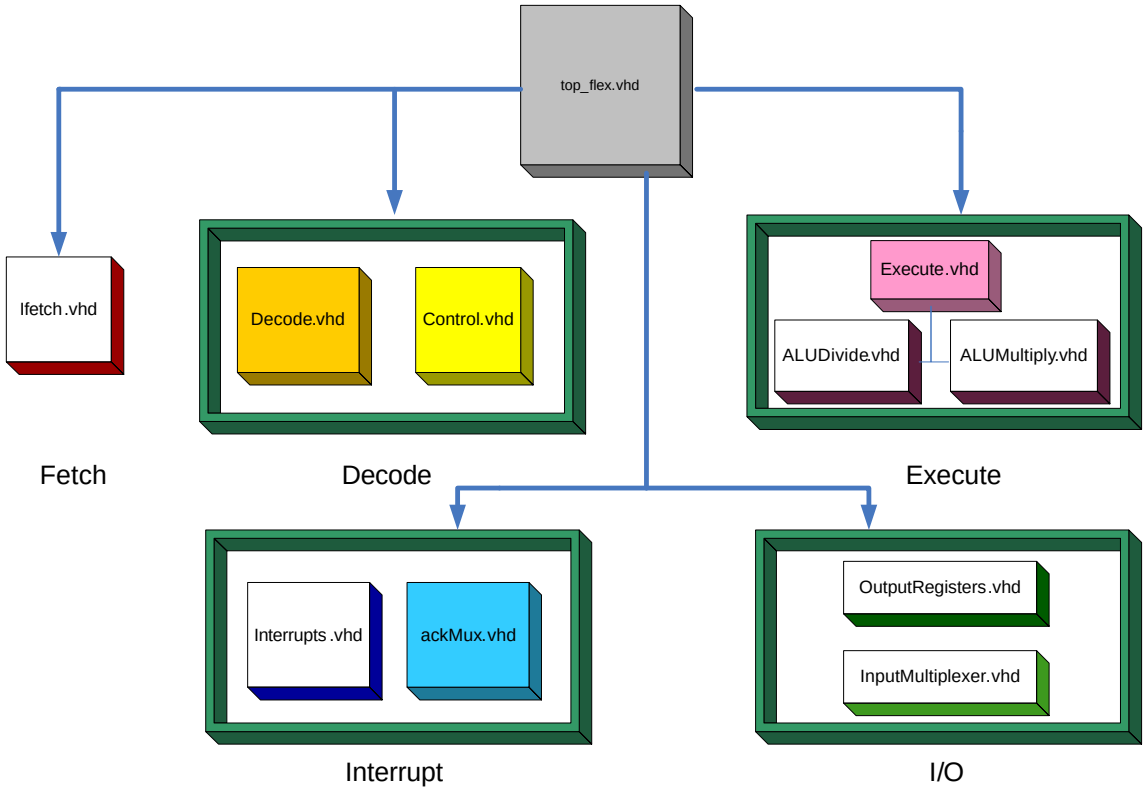

Fig. 1: MIPS architecture in VHDL

Fig. 1 shows the common 'MIPS' microprocessor architecture components: Fetch, Decode and Execute with the augmented I/O registers and interrupt scheme as they are implemented in VHDL. The Fetch and Decode stage have been extended to include enhanced interrupt and I/O instructions for the dedicated I/O system.

A test circuit was designed consisting of pressure and temperature sensors, as well as a servo output. Sensor measurements were digitized using a 12-bit A/D converter which provides serial output data. A full measurement cycle of one sensor requires 16 clock cycles. As an example, if we are to sample 10 sensor inputs at 1kHz, then 160 000 clock cycles per second will be dedicated only to data sampling. To reduce this overhead, a hardware module was created that samples the serial input data when required. These modules are independent of the processor and can run in parallel, as such the only load on the CPU will be for the data to be transferred to a register after interrupt. With the current architecture such a transfer will require 3 clock cycles, hence reducing the number of clock cycles required to 30 000 clock cycles per second.

Other modules were produced to further decrease the CPU loading:

Timing module: this module is used to signal when a specified time has passed. Time is specified in number of clock cycles. A developer can use one of these modules to delay sampling of a sensor input or to count time. The timer can be set by connecting the timer input to one of the processors output registers. Multiple timers can be used to accurately count clock cycles while not using any CPU resources.

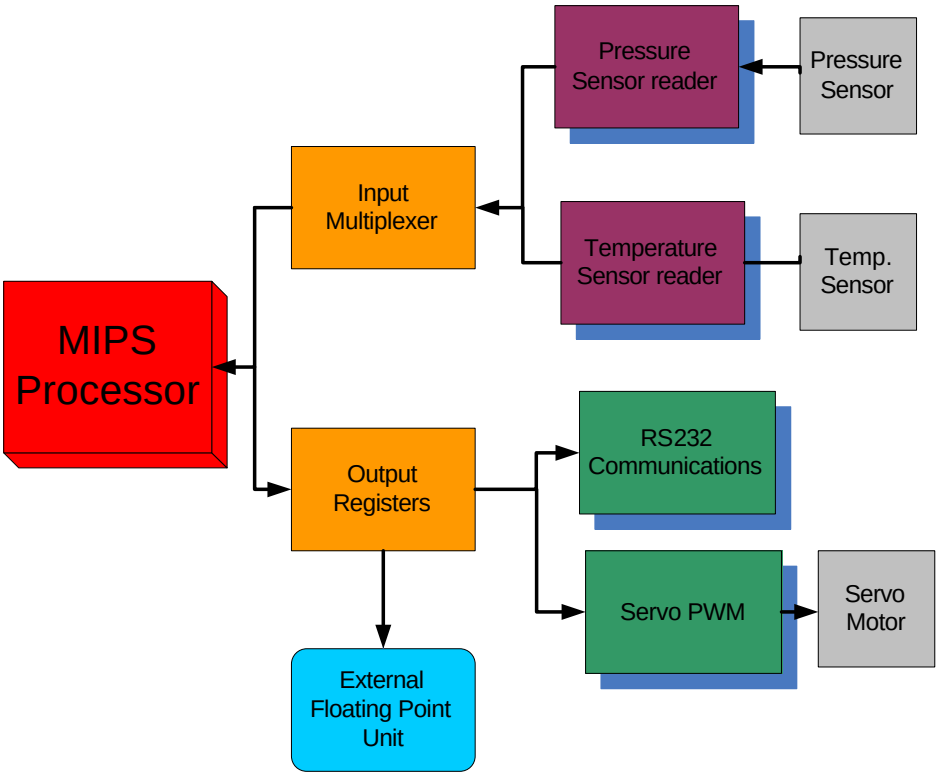

Fig. 2: Control platform architecture

Servo PWM module: a common output device for aircraft control is the servo motor. These motors are typically controlled with a pulse width modulated (PWM). The time the PWM signal is held high relates to the desired angle. The PWM module takes a number as specified from an output register, and converts it to a PWM signal for a servo motor. Again, the module uses no CPU resources and is designed for flexible use with a variety of servo motors.

RS232 Communication controller: for the system to report the state of parameters to another computer for example a communications system is required. A typically communications packet may contain header information, the amount of data being transmitted, the body data and a footer. This task of creating the packet of data and transferring it to the buffer is executed by the CPU. To alleviate the need for the CPU to do this, a module was designed that takes the body data straight from the CPU registers, creates the header and packet information and controls transmission. The software developer does not need to worry about controlling transmission with this architecture, and a large number of CPU cycles can be saved.

External Floating Point Unit (EFPU): to perform advanced functions on a UAV some intensive mathematics may be required. Examples of common CPU intensive tasks for UAV control include numerical integration and the Kalman filter algorithm [2]. These tasks can be performed by a small hardware controller and floating point unit. The idea is that the CPU can pass parameters to the Kalman filter controller for example in which the EFPU can process the resulting states and return them to the CPU when the calculation is complete.

To demonstrate the ability of the new architecture the proportional control system in Fig. 3 was designed and implemented in MIPS assembly. The control system measures pressure and aircraft roll and determines servo outputs according to the control signal required to get the craft to the desired states.

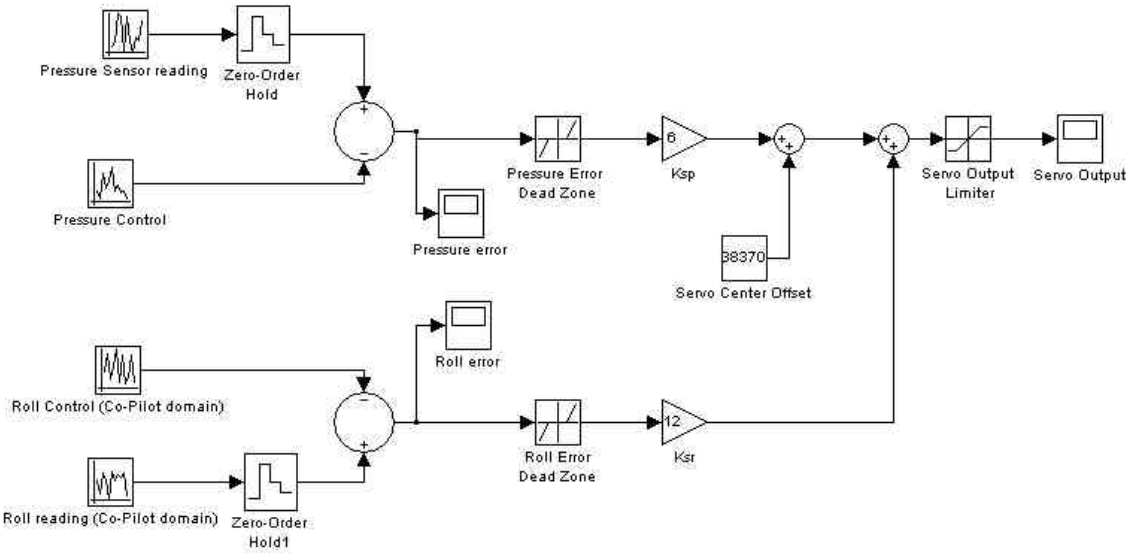

Fig. 3: Simple proportional flight controller system

The final design has a control loop that uses only 11 instructions for calculation, with additional cycles used when data is updated from inputs. With such a small loop the control parameters can be updated at a high frequency with only a small number of CPU cycles utilized. This leaves room for more advanced features to be implemented on the CPU and also in hardware.

Conclusions

By utilizing an FPGA's flexibility a different design philosophy was explored for use in UAV flight control. Combining a software based controller with supporting hardware modules the designer is able to use a larger number of I/Os without adversely affecting CPU performance. With a control system capable of sustaining roll angle and altitude the design philosophy was shown to use a small number of clock cycles leaving room for advanced features and additional pay load for a UAV mission. Further, the design is easily expanded using hardware modules or by modifying the control software.

References

1. Hamblen, J.O. and Furman, M.D.., "Rapid Prototyping of Digital Systems" (Second Edition), Kluwer Academic Publishers, 2001.
2. University of North Carolina, Dept. computer science, Web Based Tutorial on Kalman Filter. http://www.cs.unc.edu/~welch/kalman/