Reengineering the Software of an Autopilot for Unmanned Aerial Vehicles using TDL

Peter Hintenaus

C. Doppler Laboratory
Embedded Software Systems
University of Salzburg
Austria

Technical Report T030
October 28, 2008

Abstract

This note describes the ongoing reenigneering of the software of an autopilot for unmanned aerial vehicles using TDL. In the first part we cover the autopilot hardware, our modelling effort and a slight extension we propose to TDL. In the second part we talk about porting the E-Machine to the autopilot, the tools we use and their setup.

1 Reenigineering an Autopilot

The autopilot used in this note has been developed by the Aerospy company [7] for the last two years. As most of the tasks performed by the autopilot require hard real-time behavior, it is a perfect “guinea pig” for verifying TDL concepts. The Hardware consists of two ARM7 based microcontrollers [1] connected via a SPI (synchronous peripheral interface) serial bus, see e.g. [4]. The first processor is responsible for navigation. It is equipped with three gyros and three accelerometers, a GPS, a pressure sensor and an electronic compass. The second processor is responsible for controlling the vehicle using six pulse width modulated outputs, recording data for later analysis on a secure digital card, and for some auxiliary functions like an ultrasound based altitude sensor.

In the near future a redesign of the hardware is planned. To increase the computational power of the platform the replacement of one of the two ARM processors with a floating-point signal processor is investigated. In the new hardware, the assignment of peripheral components to the processors might change. To keep changes to the source code to a minimum during this transition we plan to use the transparent distribution [3] feature of the TDL [8] methodology. As TDL modules are the basic unit of distribution, our model consists of a number of small TDL modules with only a few modes and tasks per module. In our reengineering effort we try to preserve the timing behavior of the original system.

1.1 Navigation

The three gyros and the three accelerometers are used to form a strapdown inertial measurement unit, see e.g. [9]. The equations of motion are solved to provide the position in earth coordinates and its orientation describes by Euler angles. The sensordata is sampled with a rate of 2 kHz and down-sampled to a rate of 250Hz, before it is transfered into the TDL environment. A position update is calculated every four milliseconds. This position is corrected with the data from the GPS receiver, the altitude sensor (based on atmospheric air pressure) and the electronic compass using Kalman filters [6]. The TDL module contains a single task for computing the position update and performing the corrections. Two modes where used, one during the initial calibration phase, the other for actual navigation.

1.2 The GPS Receiver

The GPS receiver is connected via an asynchronous serial interface. Every 250 milliseconds it delivers a record of data containing the receivers position, its velocity and information about the quality of this data. In the original software this data is handled by a top half –bottom half scheme, see e.g. [2]. First the data is received by an interrupt service routine on a byte per byte basis and placed into a buffer. This interrupt service routine, the top half, implements a state machine, which synchronizes with this data stream, stores the data and checks a checksum. Also the top half does not consume a lot of time, it has to run with a rather low latency, or else single bytes of data will be lost, and it will be impossible to receive complete records. Once reception of a complete record has been recognized by the top half, the data is passed on to the bottom half, which parses the record using the processor time that is left by the hard real-time tasks.

When describing such behavior in TDL one might be tempted to make the top half an asynchronous task triggered by the devices interrupt. Such an approach is doomed, as asynchronous tasks are scheduled by the E-machine [5] *after* all synchronous tasks have been executed, and thus the latency requirements cannot be met. We kept the top half outside the TDL and describe the bottom half as an asynchronous task. To make it possible for the top half to trigger execution of the bottom half, a small extension to the E-machine is required. An application program interface will be added that allows for signalling “soft” interrupts.

1.3 Pressure Sensor and Electronic Compass

The pressure sensor and the electronic compass are sampled every 100 milliseconds. TDL slotselection is used to make sure the data is available to the rest of the system 4 milliseconds after being sampled.

1.4 Processor – Processor Communication

Processor – Processor communication has not been describes explicitly yet. We plan to investigate how the SPI bus can be managed by the E-machine to make transparent distribution of modules onto different processors possible.

1.5 Datalogging

The datalogging module consists of a single asynchronous task, that is triggered whenever a new position is provided by the navigation task.

1.6 Flight Control

Flight is controlled by a position and an attitude controller. The reference position for the aerial vehicle is received by a radio uplink. The position controller computes a position correction which is passed on to attitude controller. The attitude controller computes a new attitude and derives control inputs for the engines and the servos for the control surfaces of the vehicle, which are output via six pulse width modulated channels. In our TDL model the uplink forms a separate module. The controllers for position and attitude reside as separate tasks in a single module. At the time of writing two modes are realized, one for flying low, aided by the ultrasound altimeter, and one for flying at higher altitudes where the ultrasound altimeter is useless.

2 Porting the E-Machine

Our ongoing port of the E-machine to the ARM platform does not make use of an underlying operating system. Insted the E-code interpreter will be called from a timer interrupt. Scheduling of the realtime tasks will be non-preemtive. The tasks that are ready to run will be called form the main loop.

For porting the E-machine we use the GNU C compiler for compiling both the E-machine and the application for the ARM processor. For debugging we use GDB together with the OpenOCD software, *openocd.berlios.de*, and an Amontec JTAG key *www.amontec.com*.

References

[1] *ARM7TDMI Technical Reference Manual.*

[2] J. Corbet, A. Rubni, and G. Groah-Hartman. *Linux Device Drivers.* O'Reilly Media, 2005.

[3] Emilia Farcas, Claudiu Farcas, Wolfgang Pree, and Josef Templ. Transparent distribution of real-time components based on logical execution time. *SIGPLAN Not.*, 40(7):31–39, 2005.

[4] Freescale Semiconductor. *Serial Periferal Interface (SPIV3) Block Descpiption.*

[5] Thomas A. Henzinger and Christoph M. Kirsch. The embedded machine: predictable, portable real-time code. In *PLDI '02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 315–326, New York, NY, USA, 2002. ACM.

[6] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME – Journal of Basic Engineering*, 82:35–45, 1960.

[7] Naderhirn and Hackl. Personal Communication.

[8] Josef Templ. Timing definition language (TDL) 1.5 specification. Technical report, University of Salzburg, 2007. Available at http://www.softwareresearch.net.

[9] David H. Titterton and John L. Weston. *Strapdown Inertial Navigation Technology.* IET, The Institution of Engineering and Technology, 2005.