AN AUTOMATED TRANSFORMATION TRAJECTORY FROM A MODEL OF A CONTROLLED SYSTEM TO THE CONTROL CODE *)

Dusko S. Jovanovic, Bojan E. Orlic and Jan F. Broenink

Twente Embedded Systems Initiative, Drebbel Institute for Mechatronics, dept. of Control Engineering, University of Twente
P.O.Box 217, 7500AE Enschede, the Netherlands
Phone: +31 53 489 2788 Fax: +31 53 489 2223
e-mail: d.jovanovic@utwente.nl

Abstract – This paper reports a design trajectory for rapid prototyping embedded software that is motivated by an obvious fact that nowadays it is impossible to separate control engineering from the software production. Besides these two are found in definitions of mechatronics, this approach deals with refinement and a thorough exploitation of their strong natural interdependency. It is another truism that in an engineering cycle of a mechatronic or, more general, automatically controlled product, a time span from a concept to the first prototype is too large. An idea of an immediate real-world validation of a modelled and simulated control system is inarguably attractive.

The paper provides a detailed description of the framework that makes migration from a modelled and simulated controller design to an implementation on a chosen processing unit a few mouse clicks far-off. Moreover, a debugging facility by off-line monitoring the execution of the automatically generated code is documented. Described trajectory is based on our 20-sim modelling and simulation PC software package in combination with various hardware targets, with an Analog Devices Inc. DSP board as an instance.

1. INTRODUCTION

In all modern reactive systems, what all mechatronic systems intrinsically are, one always finds one or more embedded computers. The functionality of these computers, and in turn performance of the controlled systems, is powered by embedded software. The essential properties of computers to be embedded in mechatronic products is that they usually have to be small compared to the size of the product and they must not contribute significantly in the price of the product. These design constraints imply that computer systems with scarce resources are used, hence a number of software development trouble-shooting instruments stay beyond an embedded software developer disposal. Additionally, an ultimate demand is that embedded software has to be absolutely reliable, thus preferably observable.

A specific advantage in the field of control embedded software that is to be exploited lays in the fact that computer code stems from the model of the system that is usually simulateable, thus freed of a number of prospective error-sources. If a modelling and simulation environment possesses a capability of automatic code generation, than the most error-prone phase in embedded control software production is circumvented: a manual transformation from the modelled control strategy to the target executable code. Furthermore, a significant feature of this approach is that burden of programming peculiarities of a concrete embedded computer are hidden from the end-users. They can concentrate on the control problems and not be concerned with the implementation details.

A leading idea for the framework is that an end user (an engineer or a student) confronting an embedded control implementation avoids discontinuities in the design stretch when migrating from a viable modelled solution to coding for chosen hardware platform. Once provided with a comprehensive modelling and simulation package, the user can refine a prototype setup from a coarse sketch to a level of fine-tuned plant dynamics and control strategy by means of simulations. Moreover, on the course towards the final design, as soon as a successful simulation of the control part of the prototype is performed, it is possible to deploy the calculations directly on the attached hardware controller! Subsequently, an immediate validation gives an opportunity to compare collected real-situation signals from the prototype setup with the signals predicted by simulation as soon as something is changed in the design.

This conceptive idea is captured both by academia and commercial solutions providers, but there are just a few recognized implementations, of which the most famous is a hardware extension to the MathWorks’ MATLAB/Simulink Real-Time Workshop, released by dSPACE GmbH, and very recently the MathWorks proprietary prototyping framework called xPC Target.

Use of the framework described here is facilitated by an automatic C/C++ code-generation feature of 20-sim, resulting in a time costless implementation on a targeted platform. The main virtue of this framework is ease for adaptation to virtually any decent hardware platform for a price, in the case of Analog Devices DSP platform, two magnitudes less compared to the mentioned dSPACE realization.

2. THE FRAMEWORK FRONT ENDS AND REQUIREMENTS

A. Software user’s interface

20-sim is a modelling and simulation tool developed by ControlLab Products B.V. [1], a spin-off company of the Control Engineering group of the University of Twente. It is a standard MS Windows application consisting of several
integrated modules that support modelling and design of mechatronics products in many aspects. The actual version is 3.3.

Users’ inputs to the modelling module can be performed by means of one or more 20-sim Editors: bond-graph, block diagram, iconic diagram, equation or by importing MATLAB models. Transfer functions of closed loop control systems are being entered either as numerator/denominator or zeros-poles-gain or through state spaces. Discretization is doable with various discretization methods (Tustin bilinear, Backward difference etc).

The tool complies well with the demand of offering a time-efficient and elaborate feedback to the user on the modelling/design decisions. By means of a flexible simulation module and visualization modules as animated graphs and 3-D animations of modelled object, the 20-sim Simulator allows for reliable verification of built models. A Fast Fourier Transform engine allows yet another insight into the system behaviour. This engine is often used for identification purposes, since 20-sim Simulator can import and visualize data stored in files. To the same end, it is often combined with Multiple Run simulation module.

In version 3.1 20-sim introduced a step forward in covering the design cycle of a mechatronic product, towards software implementation of control laws [2]. In that version (December 2000) automatic code generation for submodels became one of the features. Stemming from internal simulation model, generation of C-code in a few variants (Stand-alone ANSI-C code, ANSI-C function, Simulink S-function) was available. Equally important to these PC (x86) targeted variants was the opportunity of extending C-code generation module to generate code for virtually any processing platform with a C compiler and the standard C libraries. Next to the handy simulation, visualisation and animation capabilities, real-situation prototyping became feasible.

Table 1. ADSP-21992 peripherals building-blocks with its default names

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In short, the principles of automatic code generation are as following. A model of a subsystem for which code is to be generated ("on-board code submodule" in the remainder) is firstly transformed into a form convenient for numerical computation. This is done anyway for simulation purposes. The code generation feature is implemented in 20-sim via use of so called template files and keywords. Template files are C/C++ source files with a certain number of special keywords that serve as placeholders for actual strings and values (tokens) representing model specific constants, variables, parameters, states and rates. Generated target-specific source files are made according to the template files by replacing keywords with actual model tokens and put into a target directory. A template directory contains template files that correspond to every supported hardware platform. For each model that contains an on-board code submodule the framework generates code in a separate target directory. In a configuration file (Targets.ini) target and template directories are defined as well as DOS console commands to be executed during pre-processing and post-processing phases. Before target-specific code is generated, pre-processing phase can take place to adjust template files to specific needs. In a post-processing phase generated target C/C++ code files can be further transformed to more suitable form, compiled, linked, downloaded and executed.

B. Hardware user’s interface

The second part of the framework is supposed to be a decent processing unit suitable for embedding in wide range of mechatronic products. By an academic donation from Analog Devices Inc. examples of mixed-signal processor ADSP-21992 were available for investigation. ADSP-21992 contains a 16-bits fixed point CPU’s up to 160 MIPS sustained performance, 32K+16K program and data memory words, 8-channel Analogue-to-Digital (20M samples/s) converter with 14 bits resolution, one three phase and two mono phase PWM 16 bits outputs, three programmable 32-bit interval timers, 32-bit incremental encoder interface unit with companion encoder event timer and so forth. In order to ease prototyping with the ADSP family, Analog Devices Inc. supplies the processors mounted on evaluation boards [3] with additional circuitries: 12-bit Digital to Analogue converters connected to the DSP through its Serial Peripheral Interface, IC’s converting from differential encoder signals to single-ended signals and external memory interface.

The evaluation boards are accompanied with VisualDSP++ integrated development environment, a code composer that is reminiscent of MS VisualC++. It allows for building C/C++ code and/or ADSP-219x assembly code as well. The communication (status obtainment, code download etc.) between a PC and the board goes via an USB interface.

C. Exploitation requirements

In order to cover new operational qualities according to the chosen hardware platform, the existing 20-sim front end and functionality is extended to allow for:

1. Automated code generation for the ADSP-21992 processor
2. Monitoring run-time variables from the generated code
3. Minimal use of the VisualDSP++ IDE

The quality of this first version of the framework was to be assessed against realistic design challenges and requirements. A proposal for a second-year practically-oriented subject at the Electrical Engineering of the University of Twente included use of the framework with standard requirements that can be found in any industrial setting as well: precise control over sampling period of digital processing, sampling frequency above 1 kHz, a
variety of inputs and outputs to the processing unit, reliability of logs of any vital variable, safety measures for a user as well as for the equipment and relatively low price. In order to alleviate exploitation under conditions of timely scarce student practical exercises and to ensure prospective industry acceptance, it was required to simplify interaction with the tools and to shelter the end user of any burden of implementation details. The process had to be kept intuitive (in terms of the 20-sim modelling paradigms) and in line with common use of 20-sim, as students or industrial users get trained in courses of 20 or 30 hours.

3. IMPLEMENTATION

Generic 20-sim template files contain only iterative numerical computations of a given on-board code submodel. Therefore, for the purpose of this project, target-specific template files containing drivers for supported peripheral units (Analogue-to-Digital and Digital-to-Analogue converters, encoder and PWM interface and digital I/O) are designed to map its occurrences in the submodels to actual I/O peripherals. To enable simulations of the on-board code submodels, specific building-blocks were designed to capture functionality and define behaviour of those peripherals (Table 1). Simulations of those hardware mapping submodels are realized using dynamic link library (dll) calls, thus a digital control system is completely tested in simulations. Moreover, when the final target-specific code is being generated, calls to dll functions are interpreted as calls to actual drivers of the same signature that exist in the template files. In other words, on all places where simulation flow goes through symbols of the DSP peripherals, calls to the drivers functions are inserted in the target code.

As it was already stated, the 20-sim software package has, among the others, a rich simulation utility that is capable of plotting data from files. The format of files should have time stamps in zero column and values of signals in following columns. The main idea during the design of the monitoring utility was to exploit those 20-sim features in order to use existing graphical user interface to plot both signals resulting from simulation and their counterparts obtained during execution of programs on the target. This is the simplest way to validate conformance of real signals on the target to behaviour specified in the model. Special template files for this logging utility had been built to enable this feature.

While generated code is being executed on the target, data is sent using UART protocol via Serial Peripheral Interface, since the USB port is claimed for the board commands. PC side part of the logger consist of two threads: first one with higher priority is responsible to wait on communication events, receive data, and put it in memory buffers after discarding data with wrong parity. Second thread takes data from buffers and feed it into a file. Data is read from this file by the 20-sim Simulator during logged data validation for comparison with simulated variables.

A submodel for the logging utility had been built to define parameters that allow customizing the task of logging according to the execution conditions. Those parameters include: name of the file where data will be logged, number of COM port used, baudrate for serial communication and skip-parameter logStep. This parameter specifies how many times is frequency of logging lower then sampling frequency. Logger is constructed in a way that prevents influence of logging on the sampling period. This is accomplished by using memory buffers and sending data only when the processor is idle. To avoid the periods when the logging engine doesn’t keep up with sending data simultaneously with the code execution, the user adapts skip-parameter logStep in accordance with complexity of the on-board code submodel and sampling frequency. The experience from exploitation shows that reliable logging of a few variables can be done accurately up to 2kHz sampling frequency with sufficiently complex algorithms (like in example in the following section).

4. EXAMPLE

In order to illustrate by means of an example the most lucrative quality of the framework, that is the ability of an immediate real-world validation of a control system design, a simple position servo problem is considered. On Figure 1 the closed loop is presented, where the model of the object pictorially describes a rotational load steered by a DC motor. The model of the mechanical setup (called LINIX) is very well known from the previous laboratory experience, hence allows for validation of predicted behaviour of the controlled system.

![Figure 1. Control loop with the LINIX setup](image)

Internals of the controller (on-board code submodel in this example) are depicted on Figure 2.

![Figure 2. The initial controller](image)

Simulation results on Figure 3 suggest redesign of the controller in order to achieve better response to the block reference signal. Dotted line (bottom) represents the shaft position that controller submodel enters via ENCI feedback input, while solid line (upper) renders steering signal fed to the motor through PWM1 interface. After few try-outs, the simulation shows favourable behaviour of the closed loop (Figure 4). As described in the previous sections, here comes the moment the user by selecting the target processor generates code from the 20-sim Simulator and uses VisualDSP++ just to download code and start the on-board code. In order to log
the variables that were already simulated, the user has just to specify the parameters of the serial line communication in the 20-sim Simulator and start the logger on the PC side. Having observed qualitative behaviour of the controlled object for desired time interval, the user stops the execution of the target software and terminates logging. The comparison of the simulated and logged control variables are obtained by running simulation again (Figure 5).

Besides described trajectory for developing control code, the framework can be used in the same way for allowing Hardware-in-the-Loop (HIL) concept, when another controller under examination is interfaced with the programmable processing unit that calculates behaviour of a virtual controlled object. In that case the processor calculates dynamic behaviour of the object instead of the control algorithm. Due to customisable logging utility of the framework, the supported processing platform can also be very well used for parametric identification purposes. The DSP board would be in that case programmed as a testing sequence generator and data acquisition processor.

Experiments showed that the run-time monitoring can work smoothly up to 2 kHz sampling frequency. By excluding the logging feature, sampling frequencies up to 10 kHz were reached. A future advance of the framework aims in logging at such high frequencies by enabling Direct Memory Access (DMA) for monitoring purposes.

Also, implementation of a two-way communication is planned, in order to allow for run-time parameters modification. This will make experimenting with adaptive control strategies feasible.

Another important improvement is recognized in flexible specifying concurrent execution of the control code. In this version generated code runs as a sequential program-flow. The authors together with other members of the Control Engineering group and ControlLab Products B.V. developers work towards unifying the current framework with a dedicated CASE tool (Computer Aided Software Engineering). The tool graphically specifies concurrent execution of the software by simplified expressions in CSP algebra (Communicating Sequential Processes, [2], [4], [5]).

6. ACKNOWLEDGEMENTS

The project benefited a lot from the profundity in real-time computation aspects of our colleague Gerald H. Hilderink and the extensive control engineering experience and devotion of our chairman Job van Amerongen.

Also we want to acknowledge Analog Devices Inc. for donating a few examples of ADSP boards in development phase and twenty complete configurations later on.

REFERENCES