Usable Simulink Embedded Coder Target for Linux

Michal Sojka, Pavel Píša
Czech Technical University in Prague
Technická 2, 121 35 Praha 6, Czech Republic
{sojkam1,pisa}@fel.cvut.cz

Abstract

Matlab/Simulink is a commercial tool used by many engineers and researchers worldwide to design and develop various systems, usually containing a lot of mathematical computations. Initially, Simulink was intended for performing simulations of dynamic systems (hence the name), but nowadays it also allows to create their prototypes or even final implementations. The system (for example a motor controller) is first designed in a graphical way in the form of a data-flow graph and then the Embedded Coder tool (a part of Simulink) is used to generate the C code directly from the graphical model.

Simulink Embedded Coder already contains support for several popular embedded boards running Linux such as Raspberry Pi or BeagleBoard but for unknown reason, this support can only be installed on Windows hosts. Moreover, the code generated for these targets has problems with precise timing when run on Linux with real-time (preempt_rt) patches.

In this paper we describe a custom developed Embedded Coder target ERT_LINUX that does not suffer from the above mentioned shortcomings and is freely available for use. We also describe a few applications developed with this target, for example a simple motor controller with the Raspberry Pi that can be used for education. The other applications show that ERT_LINUX-based solutions can be used even for more demanding applications with sampling frequencies around 20 kHz.

1 Introduction

Matlab/Simulink is a commercial tool used by many engineers and researchers worldwide to design and develop various systems, usually containing a lot of mathematical computations. Initially, Simulink was intended for performing simulations of dynamic systems (hence the name), but nowadays it also allows to create their prototypes or even final implementations. The system (for example a motor controller) is first designed in a graphical way in the form of a data-flow graph (such as in Figure 4) and then the Embedded Coder [1] (a part of Simulink) is used to generate the C code directly from the graphical model. The code is then compiled and optionally also downloaded to the target device where it is run, all at just one click.

Simulink Embedded Coder already contains support for several popular embedded boards running Linux such as Raspberry Pi or BeagleBoard but for unknown reason, this support can only be installed on Windows hosts. Support for Linux targets on Linux hosts is very problematic. According to Matlab documentation (and other sources [2]), this combination seems to be only supported via integration with Eclipse IDE [3]. Eclipse is essentially used to compile the code and download the resulting binary to the target. While using Eclipse might be beneficial for some users, there is a simpler and faster way how to achieve the same without the need for gigabytes of memory required by Eclipse: using the `make` tool. Given that Simulink already uses *GNU Make* for other targets, it is not clear to us why it is not possible to use it for Linux targets instead of Eclipse. Moreover, the code generated for Linux targets does not work well with preempt_rt patches [4] that extend Linux with hard real-time capabilities.

In this paper we describe a custom developed Embedded Coder target that does not suffer from the above mentioned shortcomings and is freely available for use. The structure of the paper is as follows: The next section introduces the Embedded Coder tool and explains in more detail the problems with the available Linux targets. Section 3 then describes our ERT_LINUX target, while Section 4 mentions some applications developed with the target. Finally, we conclude in Section 5.

2 Embedded Coder in a nutshell

In this section, we explain the basics of the Embedded Coder and related tools.

We start our description with the tool called *Simulink Coder*, formerly known as *Real-Time Workshop* (hence the RTW acronym). It is a tool for translating Simulink models to C code. *Embedded Coder* is an extension to the Simulink Coder. The principle of operation (described below) is the same but Embedded Coder produces more optimized code and offers additional features such as traceability reports to support development according to several safety standards.

The principle of Embedded Coder's operation is as follows. A Simulink model is first transformed to a so called `.rtw` file, which is a structured description of the model and it contains all the information necessary for code generation. This file is consumed by the *Target Language Compiler* (TLC), which produces C or C++ code. The TLC has a (weird) programming language, in which one can specify how the individual blocks in the Simulink model get translated to the target language. In essence, each Simulink block has a corresponding "TLC template" and these templates are combined together to form the resulting code. Similarly, the structure of the main program (the `main()` function etc.) is also determined by a template. After all code is generated, it gets compiled. This is typically accomplished by generating a Makefile from another template and calling the `make` utility. The code generation process can be extended by custom hooks executed at different phases of the process. For example, a hook can be used to download the produced binary to the target system and run it there.

All the templates and hooks described above are together called a *code generation target* or just target in short. The targets for the Embedded Coder are denoted as ERT (Embedded Real-Time), whereas targets for the plain Simulink Coder are known as GRT (Generic Real-Time). When the user wants to generate the code from a Simulink model, the most important configuration option is the target selection.

2.1 What's wrong with MathWorks Linux targets

Simulink already contains several code generation targets. Some of them can be used out of the box

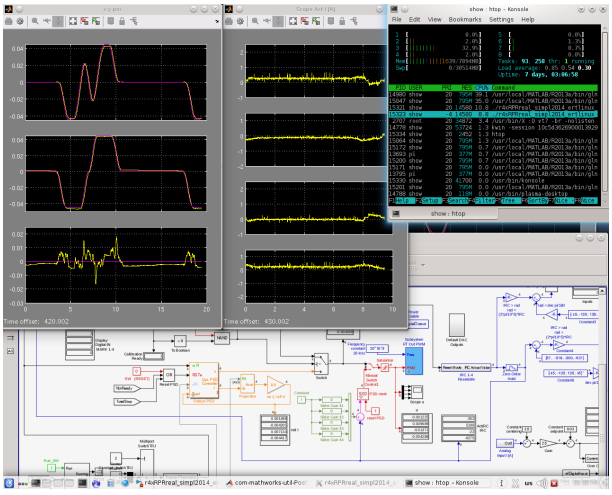

Figure 1: ERT_LINUX in action – robot control system

(e.g. the VxWorks OS target), others can be downloaded and installed on request. As it was already mentioned in the introduction, all Linux targets can only be installed on Windows hosts, or they have to be used via Eclipse IDE.

Besides the above, the code generated for the Linux targets uses POSIX timers. The problem with POSIX timers is that their current implementation under PREEMPT_RT Linux [4] (Linux with patches that turn it into a real-time OS) cannot guarantee real-time properties. POSIX timers use signals to notify the thread about timer expiration. Due to implementation details, the notification has to be deferred to the softirq [5]. This basically means that the notification code runs at random (very likely non-real-time) priority [6], because most often it will be run in the context of `ksoftirqd`. The result is that although the thread that controls the timing of the Simulink model execution has real-time priority, the execution can be delayed as if the thread ran at non-real-time priority.

These shortcomings led us to the development of a new code generation target, which is described in the next section.

3 Welcome to ERT_LINUX

The ERT_LINUX target [7] is a code generation target for Embedded Coder tailored specifically for preempt_rt versions of Linux. Compared to MathWorks provided targets, it is easily usable on Linux hosts and uses `clock_nanosleep` rather than POSIX timers to provide better real-time performance under

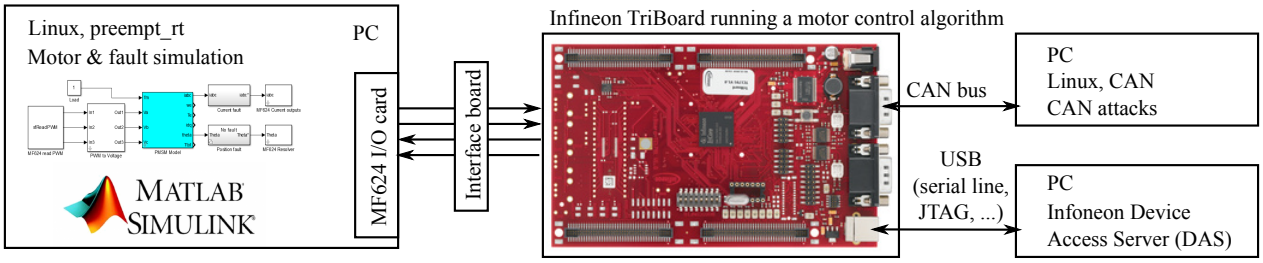

Figure 2: PMSM simulation block diagram

preempt_rt. The target supports the so called *external mode*, which allows the Simulink GUI to communicate with the generated application running on the target system via TCP in order to visualize data from the target or to tune certain parameters during run-time. Figure 1 shows an example of external mode in action.

In order to simplify porting to different Simulink versions, we tried to reuse as much existing TLC code as possible. Therefore, ERT_LINUX consist only of the main program template, Makefile template and a few necessary boilerplate scripts.

For the target to be useful, it is also necessary to support I/Os. In this area, we cannot really compete with MathWorks targets, which typically support most of the peripherals of the target platform. Currently supported IO options are outlined below:

MF624 data acquisition card MF624 [8] is a PCI-based IO card. Simulink blocks using the UIO driver from the mainline Linux can be obtained from our Git repository[1]. The available blocks are: analog inputs/outputs, digital inputs/outputs, rotary encoder (IRC) inputs and PWM inputs/outputs.

CAN bus communication is supported via Linux PF_CAN (a.k.a. SocketCAN) subsystem. The Simulink blocks are compatible with standard simulink CAN Pack/Unpack blocks and can be downloaded from another Git repository[2].

Raspberry Pi The Raspberry Pi is a credit-card sized computer [9] that is well supported by standard Simulink, but with all the limitations described in Section 2.1. We developed a few IO blocks that are described in more detail in Section 4.2.

[1] `https://rtime.felk.cvut.cz/gitweb/mf624-simulink.git`

[2] `https://rtime.felk.cvut.cz/gitweb/socketcan-simulink.git`

4 The ERT_LINUX target in practice

The ERT_LINUX has been successfully used in several applications. These are briefly described in this section.

4.1 PMSM motor simulator

For Michal Kreč master thesis [10, 11] we needed to simulate a permanent magnet synchronous motor (PMSM) in order to evaluate properties of its control software that runs on an a microcontroller. The control software runs with sampling frequency of 20 kHz, so it was essential to run the motor simulation at the same frequency. The motor simulation was created in Simulink, compiled with ERT_LINUX and ran on an ordinary PC with preempt_rt Linux. This allowed us to achieve the needed sampling frequency without any problems. The block diagram of the resulting test bed can be seen in Figure 2.

One interesting property of this application is that the used I/O card (MF624) is able to read the PWM signals directly, without the need of low-pass filters and ADC converters.

4.2 Motor control with Raspberry Pi

In the real-time programming course taught at our university, one task for the students is to write a controller of a DC motor connected to an "industrial" PowerPC MPC5200-based board running VxWorks [12]. We were interested, whether it is possible to control the same motor with preempt_rt Linux and Raspberry Pi (RPi) – a popular low cost embedded board [9]. The initial implementation was done by Radek Mečiar in his bachelor thesis [13, in Czech]. Later, we improved it and used Simulink with ERT_LINUX to implement the control algorithm. This is described in the following paragraphs.

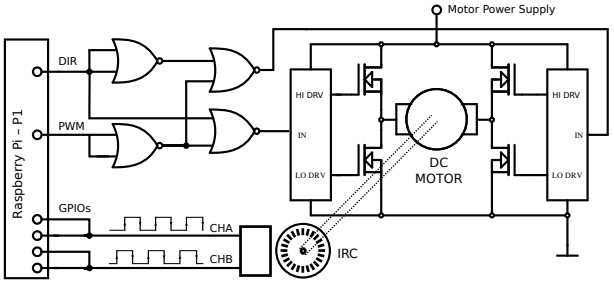

Figure 3: Connection of a DC motor to the Raspberry Pi

Hardware Since the RPi (and its BCM2835 SoC) is not intended for motion control applications, there is no hardware support for incremental encoder inputs (IRC) and the RPi connector has only one pulse width modulation (PWM) output. We wanted to have the interface hardware as simple as possible so we ended up with the connection shown in Figure 3. In order to be able to rotate the motor in both directions, we demultiplex the one available PWM signal by using four NOR gates (SN74HCT02) and use an additional GPIO output (DIR) to control the direction. The incremental encoder signals are connected directly to the RPi and processed only in software.

IRC processing The IRC signal processing is the most demanding part of our solution since the frequency of the IRC signal can go up to 21 kHz, when maximal voltage is applied to the motor. To achieve reasonable performance, a kernel driver has been implemented for this purpose [13]. It uses four GPIO pins – two for each IRC channel. One of these two pins is configured to generate interrupts for rising edges and the other for falling edges. Motor position is derived from the order of interrupts and applications can read it via `/dev/irc0`.

This solution has an interesting property that it works even when the processing of one (or few more) interrupt(s) is delayed due to other activities in the system. In preempt_rt, the interrupt handlers run in dedicated threads, which can be preempted by threads with higher priority. Thanks to the fact that the scheduler run queue is a FIFO queue, the IRQ threads are activated in the same order as the original interrupts and our position calculation works even in the case of a delay. The alternative approach, where the position is calculated from the actual IRC signal levels read in the interrupt handler would fail, because the level read in a delayed handler might be different from the level at the time when the corresponding IRQ was generated.

Even better performance could be probably

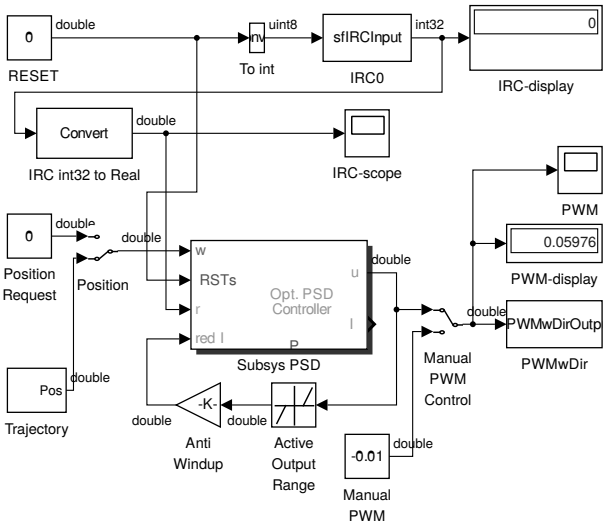

Figure 4: Simulink model of the motor controller for Raspberry Pi

achieved by processing the IRC signals in ARM's fast interrupt requests (FIQ), but this solutions would not be portable to other architectures.

Simulink model Simulink model of the motor controller is depicted in Figure 4. The *IRC0* and *PWMwDir* blocks are so called C MEX S-functions. The former reads the motor position from `/dev/irc0`, the latter controls the PWM and DIR signals by directly accessing the GPIO registers via `mmap()`ed `/dev/mem`. The actual motor control is performed by a proportional-sum-derivative (PSD) controller enhanced with an anti-windup technique. Setpoint w is generated either manually or by a simple trajectory generator. Sampling period of the whole model was set to 1 ms. The source code of the S-functions as well as the model are available from our Github repository[3].

The ERT_LINUX target was configured to use an ARM C cross-compiler. The generated binary was copied to the target RPi board by the `scp` command and ran there. Simulink *external mode* was used to tune certain model parameters on-line as well as to see the actual signal values in the scope windows.

Performance evaluation The Linux kernel used for our experiments was the official Raspberry Pi rpi-3.12.y Linux kernel version 3.12.28 patched with Steven Rostedt's stable preempt_rt patch (patch-3.12.26-rt40) and with Junjiro R. Okajima's aufs3.12.x 20140512. The latencies measured by

[3] `https://github.com/ppisa/rpi-rt-control`

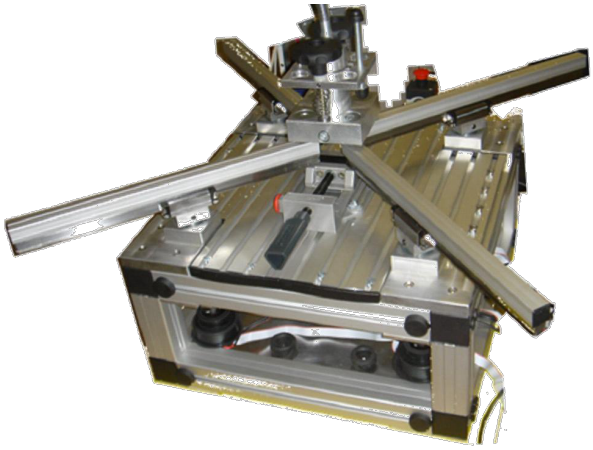

Figure 5: Experimental robot with parallel kinematic structure

the `cyclictest` tool when the system was loaded by TCP communication and SD card accesses are as follows. The maximal latency was 170 µs, average about 40 µs. When graphical desktop was run the maximal latencies increased to 280 µs (probably caused by contention on the system bus generated by the VideoCore GPU).

System load caused by running an unoptimized (-O0) model was about 2%. The USB connected Ethernet controller (a part of RPi) generated load of about 10%. The highest load was generated by the software IRC signal processing – up to 95%, i.e. the limit for RT tasks. This happened during fast motor rotation (e.g. when maximal input voltage was applied) and the interrupt frequency was 8 kHz per each channel (32 kHz in total). As can be seen, processing of the IRC signal at full speed (21 kHz) is above capabilities of the Linux kernel on this hardware. As mentioned above, the use of FIQ (or raw interrupts) could help here. For lower speeds, our simple solution works flawlessly and can be used as a great tool for control education and experiments.

4.3 Parallel kinematic robot control

The ERT_LINUX target was also used to control an experimental robot with parallel kinematic structure shown in Figure 5. The robot has more actuators (4) than degrees of freedom (3), which brings some benefits such as higher stiffness, but more complex control algorithms are required. The robot was developed at Adaptive Systems Department of Institute of Information Theory and Automation Academy of Sciences of the Czech Republic. The robot is equipped with four DC motors and four incremental encoders

N	Signal	MF624
4×	Motor position	IRC input (with index)
4×	Homing mark	Digital input
4×	Motor voltage	PWM output
4×	Motor direction	Digital output
4×	Current sensor	ADC converter
3×	Buttons	Digital input
2×	LED indicators	Digital output

Table 1: Interface between the MF624 IO card and the robot

(IRC). We used the MF624 IO card to control the robot from Simulink. The interface between the robot and Simulink is summarized in Table 1.

The MF624 card is equipped only with four pulse width modulation (PWM) outputs so we used the same concept to control motor direction as in the Raspberry Pi case.

The control algorithm for this robot was the most complex application of ERT_LINUX – see the screenshot in Figure 1. Even if the control algorithm for the redundant parallel kinematic structure is based on simple proportional-sum-derivative (PSD) control, it requires reduction of antagonistic forces based on the robot kinematics model [14]. This ensures that the integral sum of small deviations caused by discrepancies in the model does not cause the motors to "fight" against each other, resulting in the overload of the power stage. Antagonistic force reduction requires complex iterative computation in each sampling interval. Similarly, robot homing requires a complex multi-phase procedure at the startup [15]. Because of the lack of the Stateflow[4] license, an application specific homing block has been implemented in C language as a C MEX S-function.

The performance of the control algorithm with ERT_LINUX was more than satisfactory. It ran together with Simulink on a PC with Intel i7-2600 CPU and 64-bit Linux kernel 3.12-rt. The sampling frequency of the controller was 1 ms. Despite the complex computations, the load caused by the control tasks was about 20%. Their timing was neither affected by a lot of scope windows (see Figure 1) nor by loading the system with different types of loads (multiple busy loops, parallel execution of `find` commands, execution of other applications).

[4]Stateflow is a tool for designing state machines in Simulink

5 Conclusion

In this paper, we introduced the ERT_LINUX – a code generation target for Simulink with Embedded Coder. We argued that our target does not suffer from several shortcomings present in targets distributed with Matlab/Simulink. The target was successfully used in several applications, which are also briefly described in the paper.

As for the future work, we would like to develop Simulink blocks compatible with Comedi interface [16]. This would allow using ERT_LINUX with many data acquisition cards without the need for developing special Simulink blocks for each of them.

Acknowledgment

This work was supported by the Grant Agency of the Czech Republic under the Project GACR P103/12/1994.

We would like to thank Rostislav Lisový for helping us with manuscript preparation.

References

[1] MathWorks. Simulink – Embedded Coder. [Online]. Available: http://www.mathworks.com/products/embedded-coder/

[2] MathWorks, "Targeting embedded Linux systems from Matlab and Simulink." [Online]. Available: http://www.mathworks.com/videos/targeting-embedded-linux-systems-from-matlab-simulink-68903.html

[3] "Eclipse Luna." [Online]. Available: https://www.eclipse.org/

[4] "RTwiki." [Online]. Available: https://rt.wiki.kernel.org/index.php/Main_Page

[5] T. Gleixner, "hrtimer: fixup hrtimer callback changes for preempt-rt." [Online]. Available: https://git.kernel.org/cgit/linux/kernel/git/rt/linux-stable-rt.git/commit/?h=v3.12-rt&id=d630b68cc073a4f582f870b8bd82550d5dc95169

[6] T. Gleixner, "softirq: Split softirq locks." [Online]. Available: https://git.kernel.org/cgit/linux/kernel/git/rt/linux-stable-rt.git/commit/?h=v3.12-rt&id=ba9bf2368cb80fa329cf6513b8fd9cf8d729e00b

[7] M. Sojka, P. Píša *et al.*, "Homepage of Linux target for Simulink Embedded Coder." [Online]. Available: http://lintarget.sourceforge.net/

[8] Humusoft, "MF624 data acquisition board." [Online]. Available: http://www.humusoft.cz/produkty/datacq/mf624/

[9] "Raspberry Pi." [Online]. Available: http://www.raspberrypi.org/

[10] M. Kreč, "Evaluation of safety and security properties of automotive motor control software," Master's thesis, Czech Technical University in Prague, 2014. [Online]. Available: http://cyber.felk.cvut.cz/research/theses/papers/461.pdf

[11] M. Sojka, M. Krec, and Z. Hanzalek, "Case study on combined validation of safety & security requirements," in *Industrial Embedded Systems (SIES), 2014 9th IEEE International Symposium on*, June 2014, pp. 244–251.

[12] M. Sojka, "PSR course semestral work assignment." [Online]. Available: http://support.dce.felk.cvut.cz/psr/cviceni/semestralka/

[13] R. Mečiar, "Řízení motoru s deskou Raspberry Pi a Linuxem," bachelor thesis, České vysoké učení technické v Praze, 2014. [Online]. Available: https://support.dce.felk.cvut.cz/mediawiki/images/1/10/Bp_2014_meciar_radek.pdf

[14] M. Valášek, V. Bauma, Z. Šika, K. Belda, and P. Píša, "Design-by-optimization and control of redundantly actuated parallel kinematics sliding star," in *Eccomas – Multibody Dynamics 2003, International Conference on Advances in Computational Multibody Dynamics*, vol. 1. Lisboa: Instituto Suparior Téchnico Av. Rovisco Pais, 2003, pp. 98–104.

[15] K. Belda and P. Píša, "Homing, Calibration and Model-Based Predictive Control for Planar Parallel Robots," in *UKACC International Conference on Control 2008 Proceedings*. Manchester: University of Manchester, 2008, pp. –.

[16] "Comedi – control and measurement interface." [Online]. Available: http://www.comedi.org/