BENCHMARKING THE PERFORMANCE OF A DSPIC CONTROLLER PROGRAMED WITH AUTOMATICALLY GENERATED CODE

V. Lamberský, J. Vejlupek

Abstract

This paper benchmarks a midrange MCU from Microchip (dsPIC33FJ family) used as a controller for a magnetic levitation plant. Basic control algorithms (slightly modified PID and State Space) are tested to present a rough idea of how long it takes to compute one iteration of the control algorithm loop. Beside this, the significance of various optimisation parameters for the code generation process is demonstrated. The measured results demonstrate how important the proper setting of data type arithmetic and Matlab optimisation parameters is.

1 Introduction

Rapid prototyping tools are not new, however their use is becoming more frequent as it reflects the market need for shortening the development cycle together with increasing the computing power of embedded processors [1, 2, 3]. In this area, Matlab becomes the industrial standard tool for designing controllers for various systems [4]. Matlab offers several packages [5] and third party tools [6], which allow direct code generation from a simulation scheme. Once the controller is designed in the Simulink environment, the code for the embedded processor is generated almost automatically, without further need to modify the simulation model.

The Simulink was originally designed for simulating systems behaviour on a desktop computer. The Simulink language is very distinct from low-level programming languages, as it uses high accuracy floating-point variables (usually 64 bits) and does not treat system resources efficiently (when the simulation is run without compilation). Therefore to be able to generate efficient C-code from the Simulation scheme, additional toolboxes are needed to overcome this problem [5].

The quality of the code generated from Simulink can be affected by changing various parameters. At the simulation level, it is advised to use data types appropriate for the target platform. The code generation process is affected by modifying the simulation and code generation parameters. At the last step, the C compiler translates the C code to CPU instructions. In the last stage, C30 compiler properties allow generating code optimised for speed or low memory usage.

The code generation process is quite complex and consists of several steps as mentioned above. This complexity makes it very difficult to estimate the computing power required for various algorithms implemented in the embedded processor with a limited set of instructions and variable length [7]. Beside this, it is generally not known how important optimisations set as parameters are for code generation. To test these properties, the experimental plant was controlled with various algorithms generated with different optimisation settings.

2 Benchmarked plant and controller description

The control algorithm performance is demonstrated on a magnetic levitation plant. The primary focus is not controller quality (e.g. control precision, response to disturbances) but the computing time needed to perform one step of the control task. Accordingly, the controllers used were benchmarked without evaluating their accuracy and response to disturbances.

The controller is a dsPIC33FJ128MC804 with the core running at 20 MHz clock frequency. This controller is connected to the plant input (analogue output from MCU) and sensors (analogue input to MCU) integrated into the controlled plant. The micro-controller is also connected to the computer via an RS232 interface. The testing apparatus is illustrated in Fig.1.

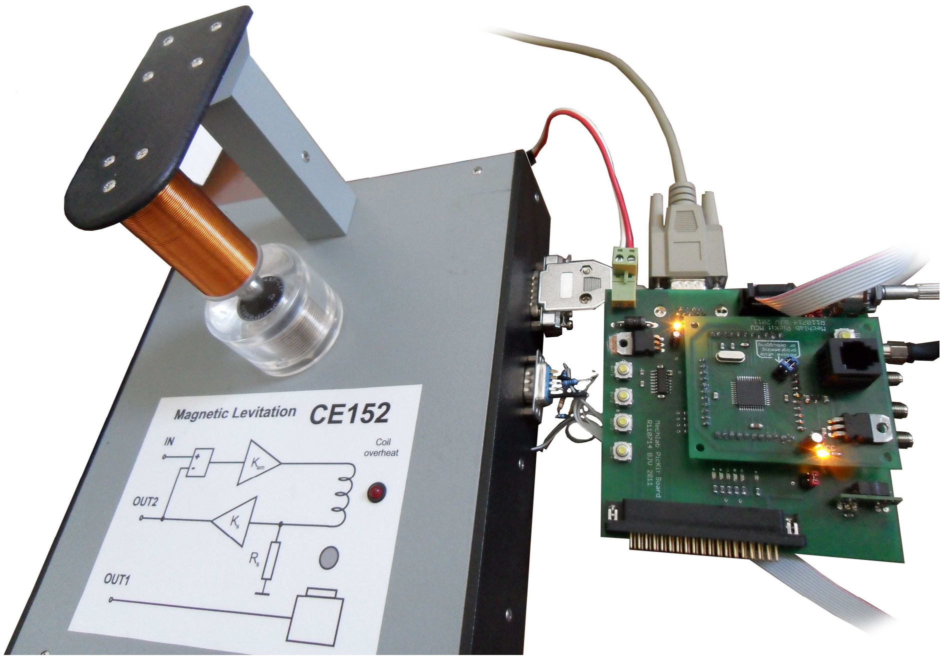


Figure 1: Controller connected to magnetic levitation plant.

The position and current are measured (inputs for the micro-controller). Since the micro-controller does not have an integrated DAC converter, a separate device for generating analogue output was attached via SPI to the main micro-controller. This DAC7715 device converts output from the micro-controller (digital value) to voltage suitable for the current controller integrated in the magnetic levitation plant.

The MCU has to handle a control loop and serving peripherals. The block scheme of the implemented algorithm is illustrated in Fig. 2.

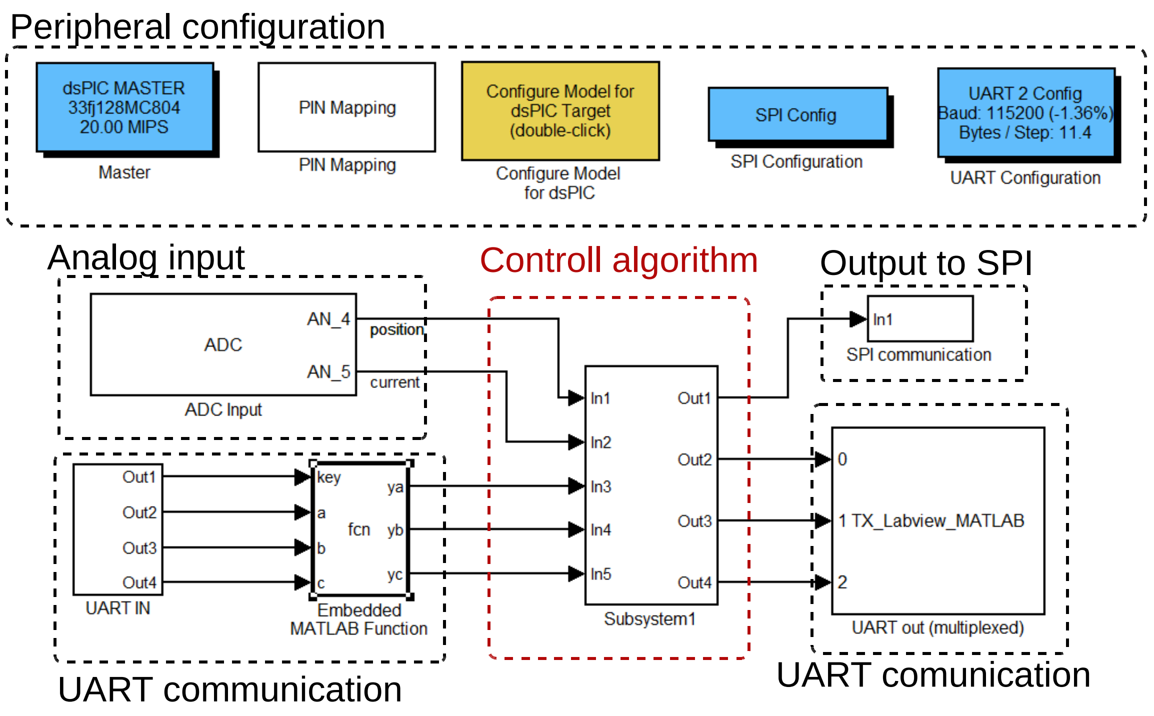
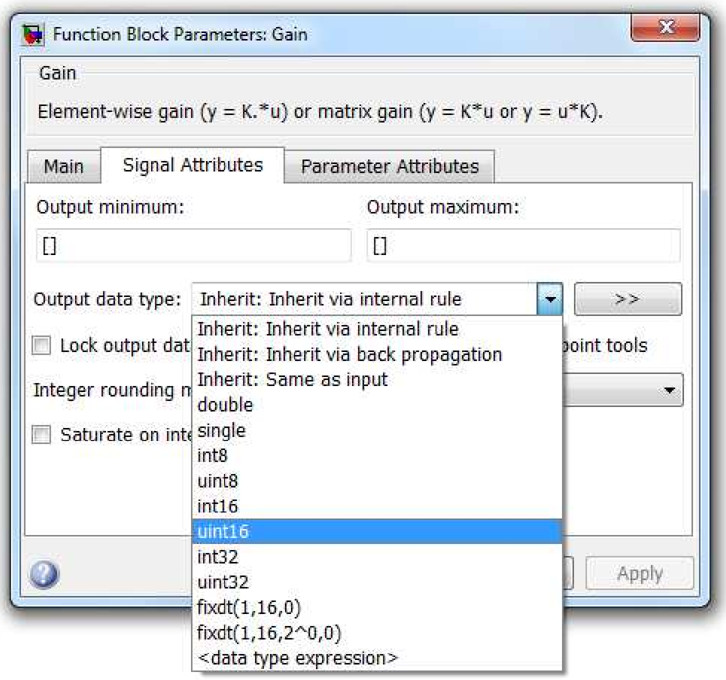


Figure 2: Implemented control algorithm scheme.

3 Variables affecting the code generation process

The code generation process from the Simulink model can be divided into two main parts. In the first part, the C-code is generated based on the Simulink model. In the second part, the C-code is translated into an MCU instructions sequence. There are a lot of parameters which are set in Matlab and C compiler environments. The main groups of these parameters are described below.

The performance of the embedded CPU without FPU is strongly influenced by the arithmetic and precision used. However, converting the code to fixed-point arithmetic might be in some cases very difficult or even impossible [8]. The data type arithmetic used is set in block parameters (Fig. 3).



Function Block Parameters: Gain

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Main | Signal Attributes | Parameter Attributes

Output minimum: []

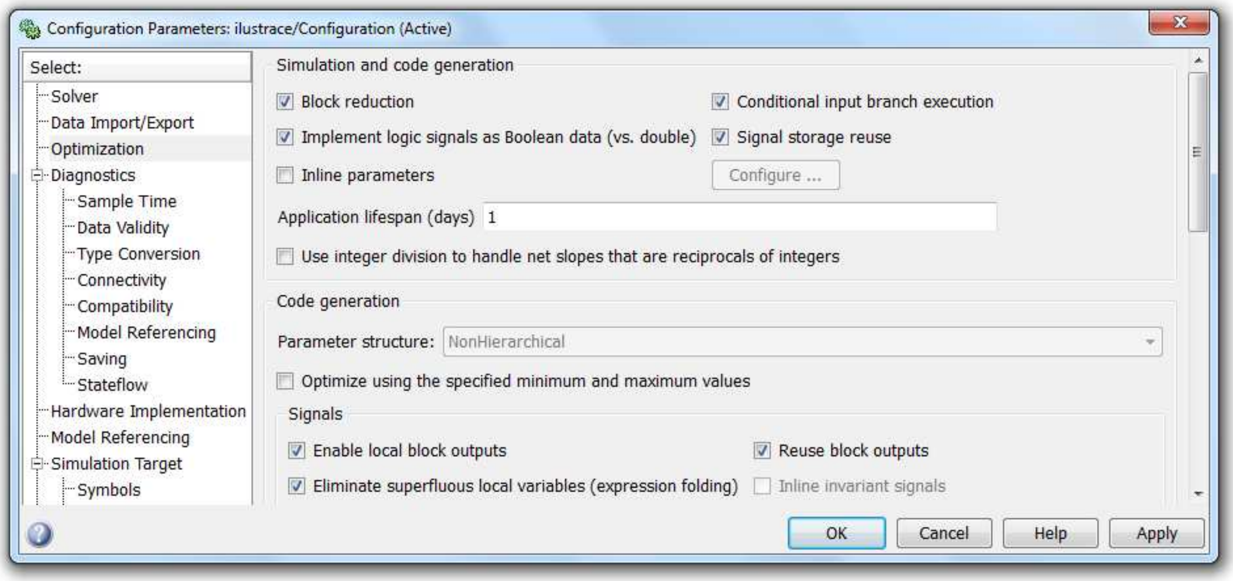
Output maximum: []

Output data type: Inherit: Inherit via internal rule

- Inherit: Inherit via internal rule
- Inherit: Inherit via back propagation
- Inherit: Same as input
- double
- single
- int8
- uint8
- int16
- uint16
- int32
- uint32
- fixdt(1,16,0)
- fixdt(1,16,2^0,0)
- <data type expression>

Figure 3: Settings for various data types.

The parameters for C code generation from the Simulink model are set in a Configuration Parameters dialog (Fig. 4). These parameters affect generating loop cycles, variables overflow protection, bitwise boolean representation, variables initialization and other properties.



Configuration Parameters: ilustrace/Configuration (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Validity
  - Type Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
  - Saving
  - Stateflow
- Hardware Implementation
- Model Referencing
- Simulation Target
  - Symbols

Simulation and code generation

- [✓] Block reduction
- [✓] Conditional input branch execution
- [✓] Implement logic signals as Boolean data (vs. double)
- [✓] Signal storage reuse
- [ ] Inline parameters
- Configure ...

Application lifespan (days) 1

- [ ] Use integer division to handle net slopes that are reciprocals of integers

Code generation

Parameter structure: NonHierarchical

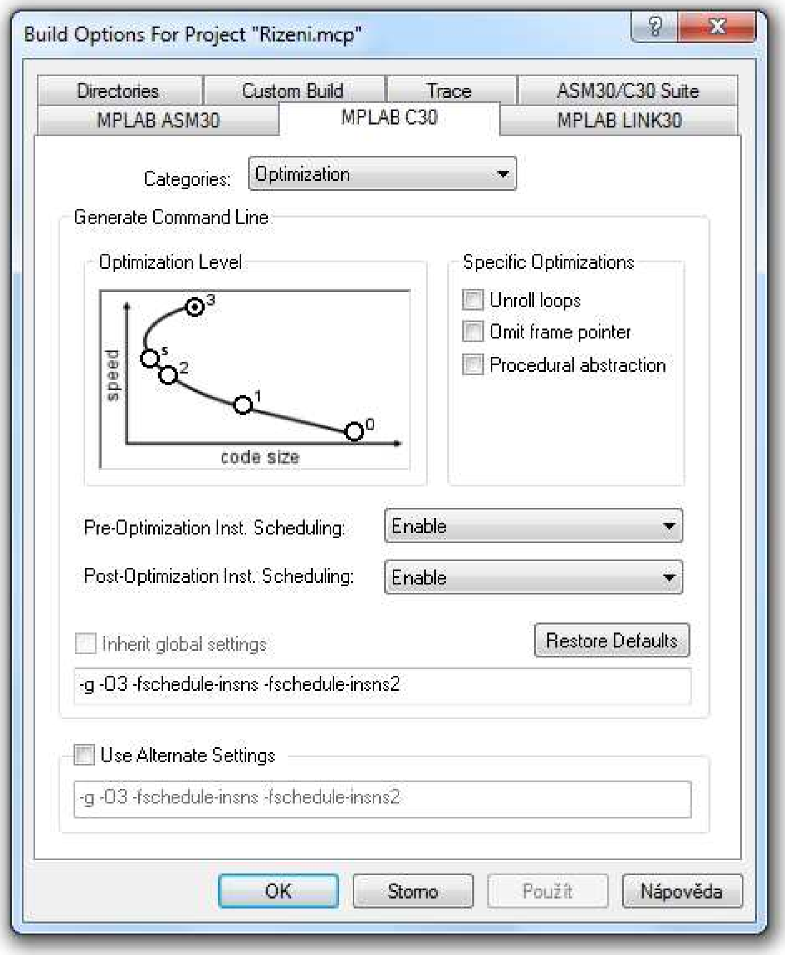
- [ ] Optimize using the specified minimum and maximum values

Signals

- [✓] Enable local block outputs
- [✓] Reuse block outputs
- [✓] Eliminate superfluous local variables (expression folding)
- [ ] Inline invariant signals

Figure 4: Dialog for changing parameters for the Matlab code generation process.

C compiler has a huge number of optimisation parameters [9]. Therefore, only the main groups will be tested. These optimisations are enabled via setting the compilation parameters passed to the compiler. Mplab has a graphical interface for setting the main optimisation levels (Fig. 5).


Build Options For Project "Rizeni.mcp"

Categories: Optimization

Generate Command Line

Specific Optimizations: Unroll loops [ ], Omit frame pointer [ ], Procedural abstraction [ ]

Pre-Optimization Inst. Scheduling: Enable

Post-Optimization Inst. Scheduling: Enable

Inherit global settings [ ]

-g -O3 -fschedule-insns -fschedule-insns2

Use Alternate Settings [ ]

-g -O3 -fschedule-insns -fschedule-insns2

Figure 5: Graphical dialog for C compiler settings.

A computing time of one cycle was measured. This time includes all operations performed in the control loop, reading values from peripherals and sending data to a computer. The computing time is measured using a timer synchronized with the simulation. (The counter is reset when the simulation starts and its value is read when the iteration is completed.)

The algorithms described in the next section were tested with various optimisations to demonstrate the computing power of the dsPIC33FJ family and the importance of optimisation parameters.

4 Tested algorithms

One of the most popular and used control algorithms is PID. This algorithm uses one input (measured position) and provides one output (required current for the coil). Since the regulated plant is highly non-linear, a simple compensation was introduced in the controller scheme (see Fig. 6).

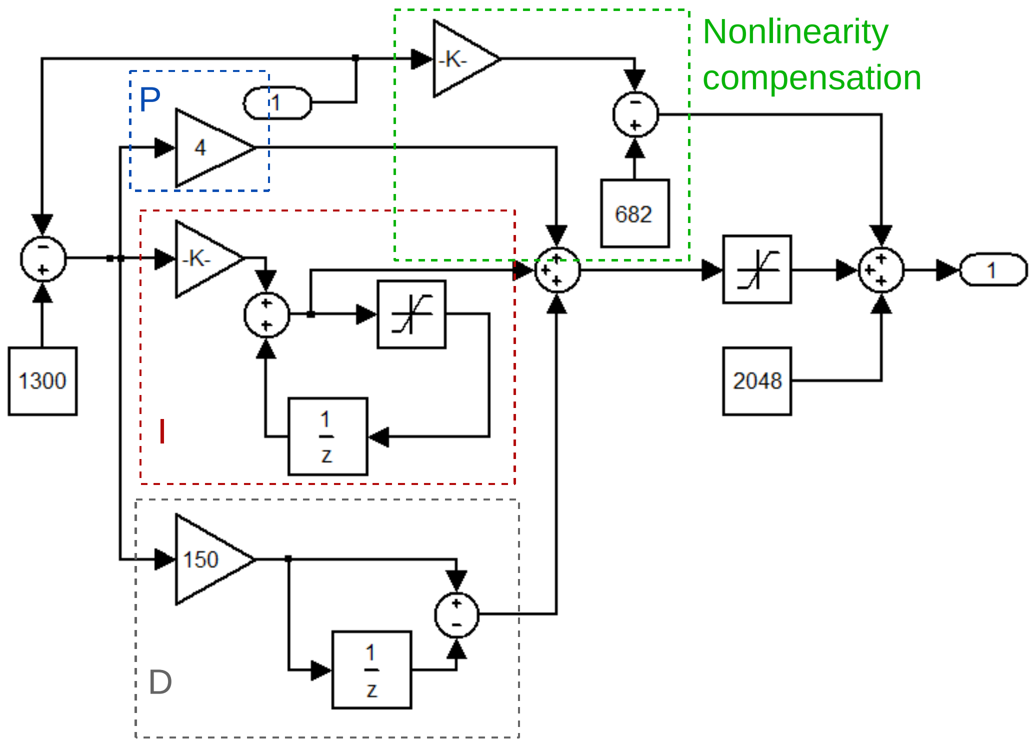


Figure 6: Simulink PID controller scheme.

The LQI controller was implemented to demonstrate controller performance with an advanced algorithm. The system (LTI with three states) was identified based on the measured input (voltage for the current controller) and output (position). The controller scheme is illustrated in Fig. 7.

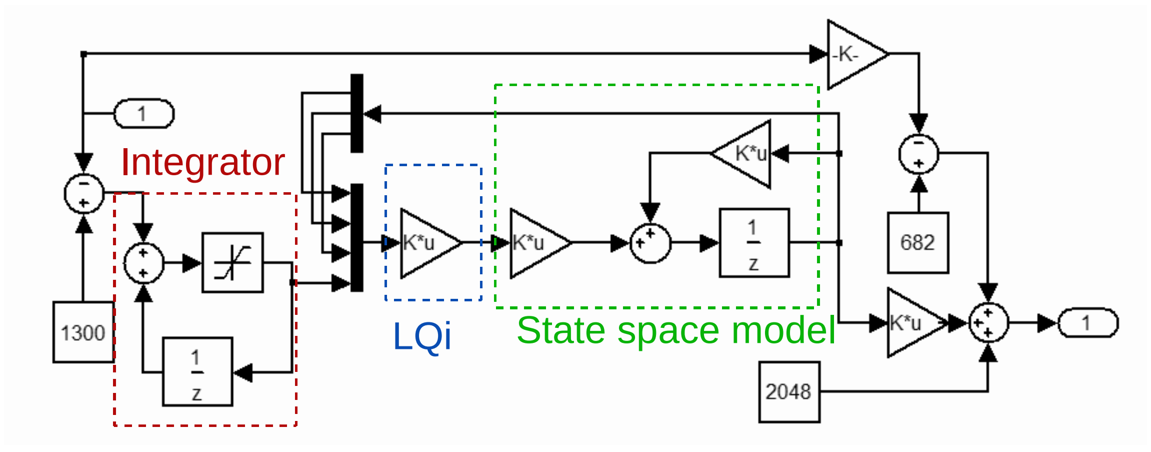


Figure 7: Simulink LQi controller scheme.

For testing purposes, the optimisation parameters were divided into main groups. These groups are presented in the table below (table 1).

Table 1: OPTIMISATION OPTIONS GROUPS FOR VARIOUS CODE GENERATION STEPS.

Used arithmetic	Real data type		Integer data type
Simulink code generation optimizations	None	Default settings	All enabled
Mplab C compiler	None	Default optimisations	Maximize execution speed

5 Results

The measured execution time of a single iteration varies slightly. This is mainly caused by interrupt driven routines for serving MCU peripherals. The modus variable is chosen to represent the time needed to perform one iteration of the control algorithm. The measured times related to various Matlab optimisations settings are summarized in Fig. 8.

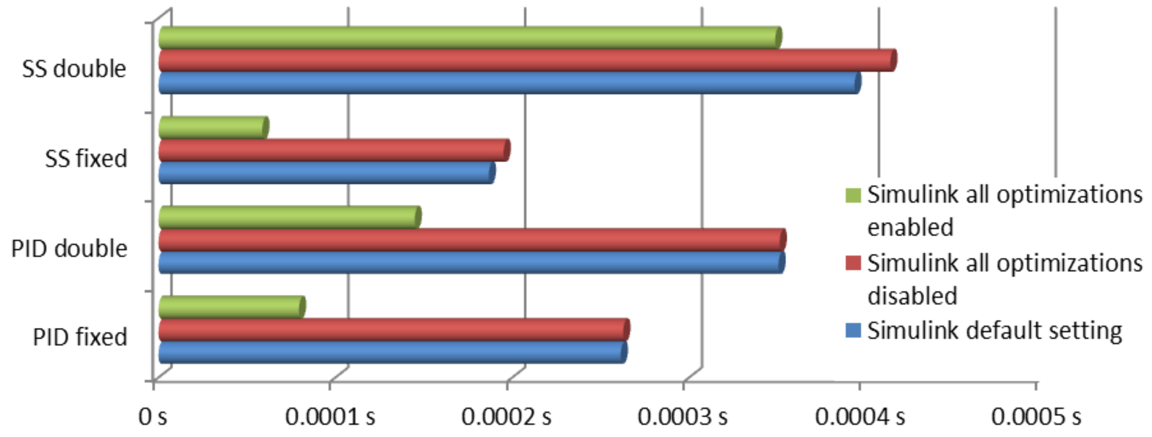


Figure 8: Computing time affected by setting different optimisations for code generation.

The measurements of C30 optimisations efficiency are provided in figure 9. The C-code generated in the previous step from Matlab was translated with a C30 compiler several times, each time with different optimisation levels (none, default and maximum execution speed).

The measured results are very surprising. Almost no computing time improvement was measured when the optimisations were set to prefer execution speed compared to code without any optimisations. Since the Simulink and C30 compiler provide similar optimisation parameters (unrolling loop, in-line parameters) it seems natural to expect similar results. However the Simulink performs some optimisations which change the algorithm behaviour (e.g. protecting variables from overflow) and these are not available in the C30 compiler. Removing these types of “unnecessary” protection leads to a tremendous rise of computing speed. However, some unwanted situations might happen, e.g. in the case of variables overflow.

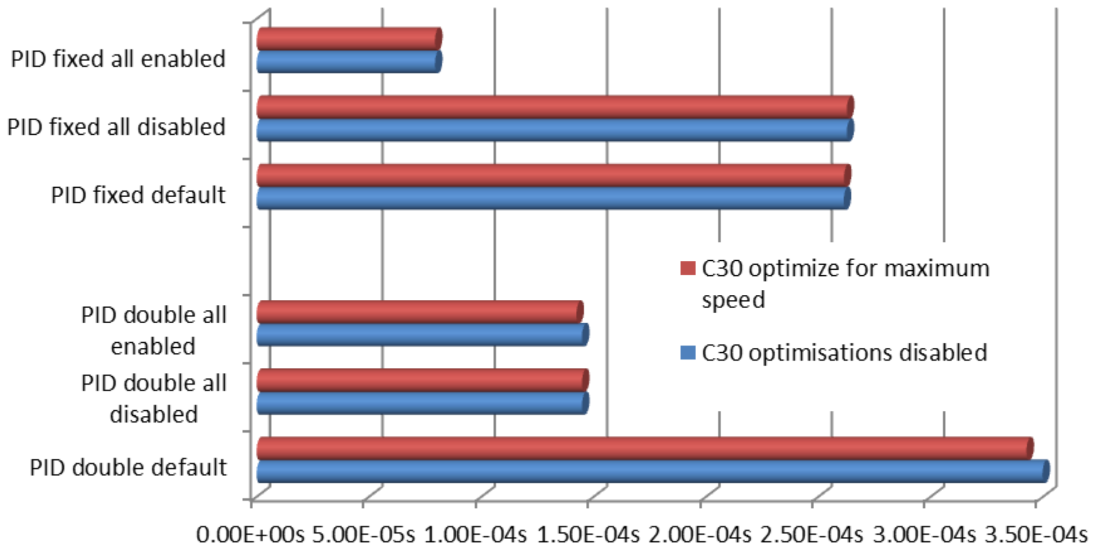


Figure 9: Computing times for C-code generated from the Simulink with default, all optimisations enabled and disabled translated to CPU instructions with different C30 settings.

6 Conclusion

For a simple algorithm it doesn’t seem to be important using optimisation parameters for the compiler. Based on measured data, compiler optimisations do not provide significant improvement in computing time. However optimisations for code generation in the Simulink should be considered, especially in cases where various protective mechanisms are not necessary. Where the conversion is possible, the use of fixed-point arithmetic is recommended, as it significantly increases computing speed on devices without FPU.

Acknowledgement

The work presented in this paper has been supported by research project FSI-S-11-15 "Design, testing and implementation of control algorithms with use of non-linear models of mechatronics systems".

References

[1] Kuhl, M.; Reichmann, C.; Protel, I.; Muller-Glaser, K.D.; , "From object-oriented modeling to code generation for rapid prototyping of embedded electronic systems," *Rapid System Prototyping, 2002. Proceedings. 13th IEEE International Workshop on* , vol., no., pp. 108- 114, 2002

[2] Duma, R.; Dobra, P.; Abrudean, M.; Dobra, M.; , "Rapid prototyping of control systems using embedded target for TI C2000 DSP," *Control & Automation, 2007. MED '07. Mediterranean Conference on* , vol., no., pp.1-5, 27-29 June 2007

[3] Grepl, R. Real-Time Control Prototyping in MATLAB/Simulink: review of tools for research and education in mechatronics IEEE International Conference on Mechatronics (ICM 2011-13-15 April, 2011, Istanbul), 2011.

[4] Bartosinski, R.; Hanzalek, Z.; Waszniowski, L.; Struzka, P.; , "Processor Expert Enhances Matlab Simulink Facilities for Embedded Software Rapid Development," *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on* , vol., no., pp.625-628, 20-22 Sept. 2006

[5] *Matlab Real-Time Workshop, Real-Time Workshop Embedded Coder* Documentation [Online]. Available: http://www.mathworks.com/products/embedded-coder/index.html

[6] L. Kerhuel. (2010) *Simulink block set embedded target for microchip devices*. [Online]. Available: http://www.kerhuel.eu/wiki/Simulink_-_Embedded_Target_for_PIC

[7] Roscoe, A.J.; Blair, S.M.; Burt, G.M.; , "Benchmarking and optimisation of Simulink code using Real-Time Workshop and Embedded Coder for inverter and microgrid control applications," *Universities Power Engineering Conference (UPEC), 2009 Proceedings of the 44th International* , vol., no., pp.1-5, 1-4 Sept. 2009

[8] Osborne, W.G.; Cheung, R.C.C.; Coutinho, J.G.F.; Luk, W.; Mencer, O.; , "Automatic Accuracy-Guaranteed Bit-Width Optimization for Fixed and Floating-Point Systems," *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on* , vol., no., pp.617-620, 27-29 Aug. 2007

[9] Microchip Technology Inc. The Embedded Control Solutions Company$^{®}$ MPLAB$^{®}$ C30 C Compiler User’s Guide [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/C30_Users_Guide_51284f.pdf

---

Vojtěch Lamberský, Josef Vejlupek

Ústav mechaniky těles, mechatroniky a biomechaniky
Fakulta strojního inženýrství
Vysoké učení technické v Brně
Technická 2896/2
616 69 Brno