ISSN 0280-5316
ISRN LUTFD2/TFRT--5793--SE

Evaluation of Automatic Code Generation Tools

Nora Ajwad

Department of Automatic Control
Lund University
April 2007

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* April 2007 *Document Number* ISRNLUTFD2/TFRT--5793--SE
Author(s) Nora Ajwad	*Supervisor* Jacco Koppenaal at Haldex Traction AB in Landskrona Karl-Erik Årzén at Automatic Control in Lund (examiner) *Sponsoring organization*

Title and subtitle
Evaluation of Automatic Code Generation Tools. TargetLink from dSPACE Real-Time Workshop Embedded Coder from The MathWorks. (Utvärdering av automatiskt kodgenereringsverktyg TargetLink från dSPACE. Real-Time workshop embedded coder från the MathWorks)

Abstract
There are different tools for automatic code generation today. A comparison between different tools is necessary in order to highlight the differences between the tools according to specific aspects. TargetLink from dSPACE and Real-Time Workshop Embedded Coder from The MathWorks are two tools that have been evaluated and tested in order to point out the differences between the performance of those tools regarding, among others, user friendliness, generated code interface, code readability and traceability, execution time and memory usage. Even though both tools differ slightly in their performance they both show good, sometimes even better performance when compared with hand written code.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 74	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/

Acknowledgements

During this work many people have helped and supported me and I would like to thank all of them.

First I want to express my gratitude to my supervisors: Prof. Karl-Erik Årzén at The Department of Control, Lund University, Sweden and Jacco Koppenaal at Haldex Traction AB in Landskrona, Sweden. I want to thank Prof. Karl-Erik Årzén for recommending me for this master thesis. I also want to thank Jacco Koppenaal and Staffan Nilsson at Haldex Traction AB for giving me the opportunity to perform my master thesis at Haldex Traction AB.

Also a special thank to all Haldex employees who have helped me along the way with any questions that I had.

I would like to thank Roger Aarenstrup at The MathWorks who has been very helpful during my work with Real-Time Workshop Embedded Coder. I also want to thank Henry Feng and Jonas Cornelsen at Fengco for their support with TargetLink.

Finally I would like to thank my family for their patience and support during this time. A special thank goes to my sister for being there for me all the way from New York, to my brothers Nawar, Naofal and Nawras for motivating me all the way. I also want to thank my friends for being there for me when I needed them.

For my mother who has always believed in me and supported me throughout my journey. For my father who has already shown me the real model of being a successful engineer that I would be proud to be one day. Thank you.

April 2007

Nora Ajwad

Content

1 Introduction

1.1 Background

Haldex Traction AB is part of the Haldex Group, which consists of four product divisions: Haldex Brake Systems, Haldex Traction Systems, Haldex Hydraulic Systems and Haldex Garphyttan Wire. Haldex Traction AB is responsible for the manufacture and worldwide marketing of the advanced and flexible All-Wheel-Drive-system, AWD-system.

Haldex Traction designs and develops a coupling called Haldex Limited Slip Coupling, HLSC, which is a complete AWD-system. The coupling consists of two main parts: torque transfer part and control system. The latter contains a microprocessor with the needed software. The control system part is responsible for controlling the coupling function with the desired properties, depending on several factors e.g. the driving situation. The idea behind the coupling is that when the front and rear shafts of the vehicle start to rotate with different speeds, e.g., due to slip, a hydraulic piston pump will start and generate an oil flow through an open throttle valve. When the throttle valve is partially closed an oil pressure will start to build up in the clutch piston. The clutch piston will then force discs in the multi-plate clutch together leading to a torque capacity from the front to the rear shaft, see Figure 1. The torque transferred in the coupling is thus a function of the pressure. The idea here is to control the hydraulic pressure in the clutch in order to be able to control the torque transfer. Today there are four generations of Haldex Limited Slip Coupling and a fifth generation is under the development phase. The tested HLSC in this work belongs to the second generation.

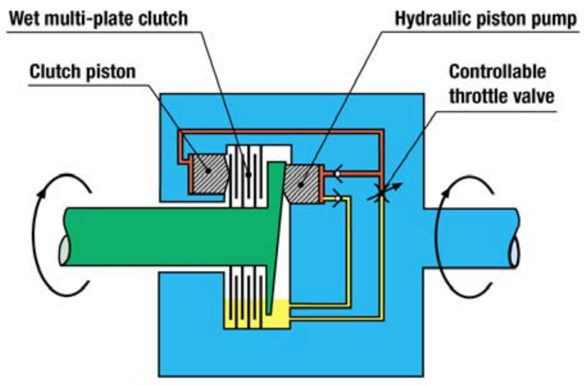


Figure 1: Schematic sketch of Haldex Limited Slip Coupling [1]

Today Matlab/Simulink is widely used to develop and test control models. At Haldex, rapid prototyping is used, as well, to test control algorithms designed in Simulink. A tool from dSPACE, called MicroAutoBox, is used for this purpose. MicroAutoBox makes it possible to test the model on the target processor directly.

A tool for automatic generation of code is desirable to save time and avoid programming errors that can be made by the programmer.

1.2 Purpose

The purpose of this work is to evaluate and compare two automatic code generation tools: TargetLink from dSPACE and Real-Time Workshop Embedded Coder from The MathWorks. The tools will be evaluated according to Haldex Traction specific requirements and needs. A model of a controller from Haldex Limited Slip Coupling Generation II is used during this evaluation.

1.3 Target Group

The target groups for this work are the employees of Haldex Traction AB and MSc. students. The readers are assumed to be familiar with Matlab/Simulink and C programming language.

1.4 Automatic Code Generation Tools

There are several tools for automatic code generation used in industrial companies today. In this work two tools are chosen to be evaluated: **TargetLink** and **Real-Time Workshop Embedded Coder**. The evaluated versions are TargetLink 2.1.6 and Real-Time Embedded Coder R2006b.

TargetLink is a development tool that uses Simulink as a base environment. TargetLink uses its own blocks in addition to several Simulink supported blocks. The tool generates C code straight from the graphical model specification made in Matlab/Simulink/Stateflow. The model can be designed directly in TargetLink or in Simulink, but in the latter case, a conversion of the model is required.

A TargetLink model can look like Figure 2 below.

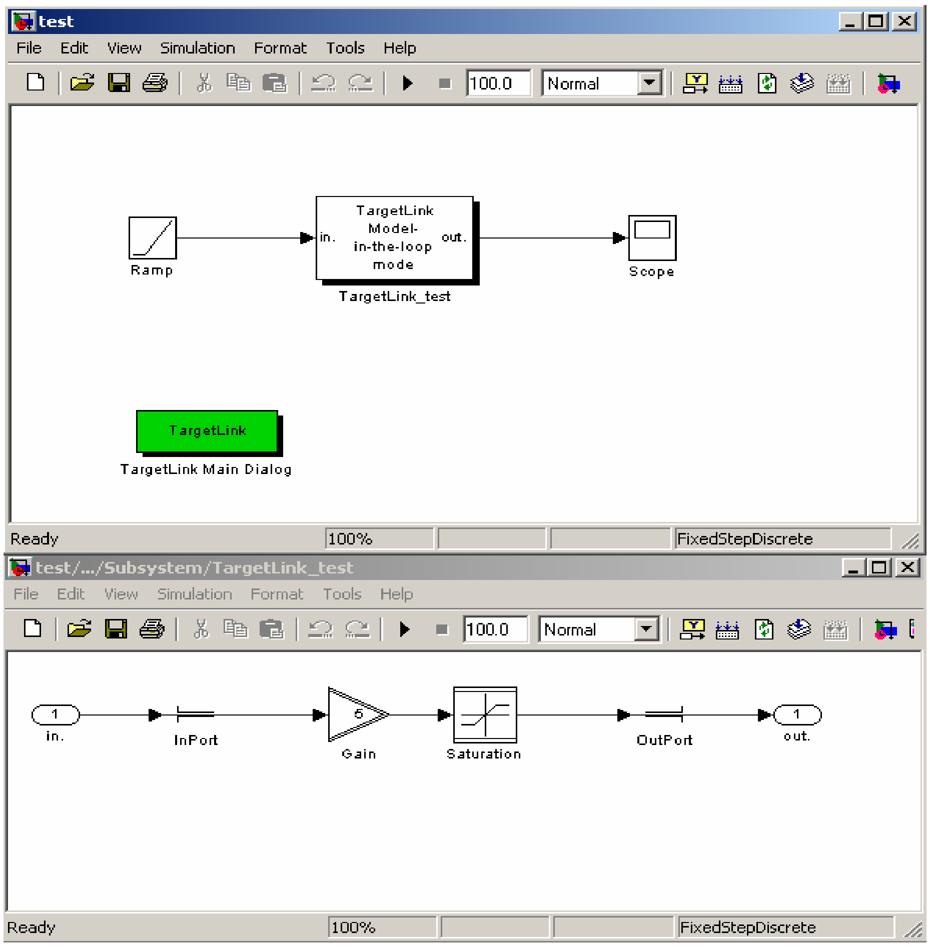


Figure 2: A simple model built in TargetLink

TargetLink generates C code for the model shown above in just one click giving the C code shown below. Note that most generated code comments have been deleted from this code due to lack of space:

```
#ifndef _TARGETLINK_TEST_C_
#define _TARGETLINK_TEST_C_
#include "tl_defines_a.h"
#include "TargetLink_test.h"

CONTINUOUS Sa1_InPort;
CONTINUOUS Sa1_OutPort;

Void TargetLink_test(Void)
{
   /* SLLocal: Default storage class for local variables |
Width: 32 */
   CONTINUOUS Sa1_Gain;

   /* Gain: TargetLink_test/Gain */
   Sa1_Gain = Sa1_InPort * 5.F;

   /* Saturation: TargetLink_test/Saturation */
   if (Sa1_Gain > 100.F) {
      /* # combined # TargetLink outport:
TargetLink_test/OutPort */
      Sa1_OutPort = 100.F;
   }
   else {
      if (Sa1_Gain < 0.F) {
         /* # combined # TargetLink outport:
TargetLink_test/OutPort */
         Sa1_OutPort = 0.F;
      }
      else {
         /* # combined # TargetLink outport:
TargetLink_test/OutPort */
         Sa1_OutPort = Sa1_Gain;
      }
   }
}
#endif/*_TARGETLINK_TEST_C_ */
```

Other C- and h-files are generated for the model to specify a header-file, the base types, e.g. Int32, and the data types, e.g. CONTINUOUS in the example code above.

TargetLink also provides powerful simulation capabilities to analyse fixed-point quantization and overflow effects which is done by choosing to simulate using the generated code using Software In the Loop, SIL, simulation. The code can also be run on the target directly using Process In the Loop, PIL, to validate the controller designs directly on the target.

Real-Time Workshop Embedded Coder is a separate, additional product for use in Matlab/Simulink environment. RTW EC generates code from Simulink and Stateflow models. For fixed-point simulation and code generation an additional tool is required, that is the Fixed-Point Toolbox. Code can be generated by a simple click for both fixed-point and floating-point. To run the code directly on the target some modifications are required to the Embedded Real-Time target files provided by RTW EC.

RTW EC generates code directly from Simulink model and no conversion is necessary.

2 Automatic Code Generation Tools Manufacturers

2.1 dSPACE – Solutions for Control

dSPACE develops and distributes integrated electronics and software systems for electronic control units. The systems are widely used in the automotive industry and also employed in electrical, aerospace, and robotics. dSPACE was established in 1988 and has its headquarter in Germany and subsidiaries in France, the USA, the UK and Japan.

TargetLink is a main product from dSPACE that was launched in 1999. Many companies use TargetLink today for automatic code generation. Haldex Brake AB, in Landskrona, uses TargetLink and the tool is highly recommended by the users from Haldex Brake AB. TargetLink has been developed throughout the years and new versions that have better performance are launched successfully.

2.1.1 Support

Fengco Real Time Control AB is the only representative for dSPACE products in Sweden. During testing and evaluation of TargetLink, Fengco has been contacted for evaluation license and technical support. A contact person was assigned for support during the work. Fengco offers courses and seminars for dSPACE products. These courses and seminars can be given at the location of the company or locally at Fengco in Stockholm [6].

2.2 The MathWorks – Accelerating the Pace of Engineering and Science

The MathWorks develops, sells and supports Matlab and Simulink as well as a broad range of products in the areas of technical computing, data analysis, signal and image processing, control design and dynamic systems simulation. The company is well established and has a dominant place in the market. It has Matlab and Simulink as its main products. This does not imply that RTW EC is a side product since The MathWorks is well involved in custom needs and requirements regarding this product. According to The MathWorks production code generation is a top priority.

Real-Time Workshop Embedded Coder is one of many in the Simulink product family which has been developed by The MathWorks. It is considered as a rather new product and has been developed a great deal since 2002 when Haldex Traction did their first study about it. New versions of RTW EC are coming during 2007.

2.2.1 Support

The MathWorks in Stockholm has been very supportive during this work. A contact person was assigned to help throughout testing and evaluation of the tool.

Online support is useful too where the user can find all the documentation for any of The MathWorks products.

3 Code generation

3.1 TargetLink

TargetLink makes it easy to generate code since everything related to the code generation is gathered in one place. It is one tool for both scaling and code generation. For fixed-point in Simulink environment the user can specify scaling by choosing the type of the signal, or the parameter, and the maximum and minimum ranges or by simply choosing the type of the signal/parameter, the scaling factor and the offset. The maximum and minimum of the block will then be automatically calculated.

3.1.1 Automatic Scaling

Automatic scaling can be performed on the whole model. This is done by feeding the model with as much information as possible. The user should enter maximum and minimum ranges of all known signals and parameters. Since all inputs and outputs ranges are known, these are propagated to adjacent blocks following the signal lines forward and backward. The automatic scaling tool, called autoscaling in TargetLink, can then be chosen and a worst case scaling is determined. A limitation for TargetLink automatic scaling is that closed loops in the model will stop the automatic scaling process unless at least one block ranges are specified by the user.

Simulation based scaling is also supported in TargetLink. This gives the user the responsibility to enter the worst case simulation inputs to obtain useful results.

3.1.2 Generating Code for One Subsystem

TargetLink makes it possible to generate code for one subsystem by choosing to generate a separate file for the subsystem during code generation. This is done by adding a Function block which is a TargetLink block that makes it possible to generate separate code for the chosen subsystem. Code can also be generated for the subsystem only, without needing to generate code for the whole model. All of these are possible via the Function block [2].

3.1.3 Code Interface

The user of TargetLink has the freedom to influence the interface of the generated code in an easy and practical way. The user can choose the output value for any subsystem in the model to be a reference variable, a return value, a pointer parameter or a global variable, see Table 1 [3]. There is also flexibility in choosing the generated function parameters since every signal and parameter can be chosen to fulfil certain functionality in the generated code, see Table 1.

Variable Class Object	Description	Memory
GLOBAL	Normal global variables	RAM
CAL	Global calibratable parameters	ROM
DISP	Global observable variables	RAM
MACRO	Global pre-processor macros	-
CONST	Global constant	ROM
FCN_ARG	Function arguments (non-constant, call-by-value)	STACK
FCN_PARAM	Function parameters (constant, call-by-value)	STACK
FCN_RETURN	Function return value	STACK
EXTERN_GLOBAL_ALIAS	Global variables, which are declared in a user-written header file	-
EXTERN_CAL_ALIAS	Calibratable global variables, which are declared in a user-written header file	-
RTOS_ACCESS_FUNCTION	Apply an access function to read/write the variable	-

Table 1: Categories of variable classes

3.1.4 Simulation Using the Generated Code

The user can check and test whether the model behaves in the same way when using the generated code. TargetLink has three options for running the model in Simulink, Model In the Loop, MIL, Software In the Loop, SIL, and Process In the Loop, PIL. By choosing MIL, the program runs the model as it is, using floating point. If the user wants to run the model with the generated code then SIL should be chosen and the program runs using the generated code. Process In the Loop mode is used to verify simulation results from the MIL and SIL simulation modes by executing the code on the actual target hardware. Process In the Loop was not tested in this work.

By just clicking on the desired option: MIL, SIL or PIL, the user can easily and without changing the model, control how to run in Simulink and the comparison is, in that way, easily done between floating point and fixed point results.

3.1.5 External Function Calls and S-function Support

Function calls to hardware are sometimes replaced with a subsystem in the simulation model which will do something different during the simulation. This is to avoid having to be connected to the hardware during the simulation and code generation stage. In TargetLink the user can have, as mentioned earlier, a subsystem for which separate code is generated in a separate file. The user can specify the name of that specific file and the name of the function call. In that way the user can replace the generated file with the original file that contains the function call to the hardware. No changes in the generated code are then required since the user can choose the same name and interface for the function call in question. In that way the simulation will run smoothly and when running against the target the user does not need to change anything in the code but only replaces certain files. External code can be added easily to the model as well, this is done by using a specific block to add custom code to the model or using S-functions which are fully supported in TargetLink.

3.1.6 MISRA C, Platform Dependency and Different Microprocessors Support

The Motor Industry Software Reliability Association, MISRA, has developed a standard for C programming language for guidelines in safety critical systems. TargetLink respects the majority of MISRA C rules. Deviations from MISRA C rules are technical necessity and when they occur they are identified and well documented [10]. For interested users there is specific documentation for TargetLink support of MISRA C provided by TargetLink.

TargetLink generates ANSI-C code which is platform independent and can therefore be compiled with any ANSI-C C-compiler on any platform.

TargetLink supports a list of microprocessors if the program Target Optimisation Module, TOM, is purchased. TOM is an additional program for TargetLink Base Suite, TBS. below is a list of the supported microprocessors in TargetLink 2.2:

Freescale HCS12

Freescale MPC5xx

Freescale MPC55xx

Infineon C16x

Infineon TriCore

Renesas M32R

Renesas SH-2

The optimised generated code is still effective ANSI C code which is platform independent. TOM code can also be target specific, i.e. insertion of assembly code and usage of compiler specific extensions to further improvement of code efficiency is available if the user allows TargetLink to do so.

3.2 Real-Time Workshop Embedded Coder

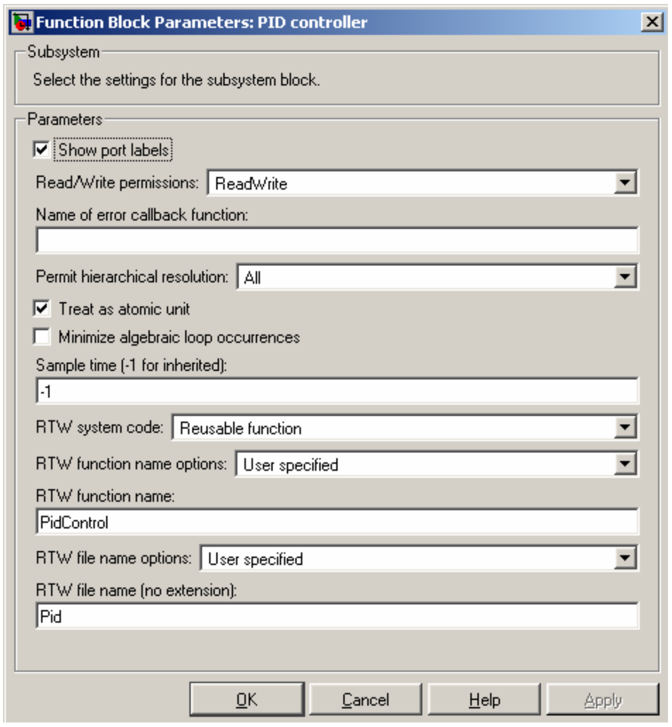
Real-Time Workshop Embedded Coder is a well integrated tool in Simulink. To generate code for fixed-point another Matlab tool, called Fixed-Point Toolbox, has to be used. This tool is used for scaling parameters and signals. The user chooses the signal/parameter type, scaling factor and offset value directly in every block in the model. Some blocks do not need to be scaled, e.g. saturation block which automatically gives an output that has the same scaling as the input.

3.2.1 Automatic Scaling

Automatic scaling is supported in Real-Time Workshop Embedded Coder, but worst case automatic scaling is not. An automatic scaling can be performed based on simulation inputs. A simulation must first be run to log in the maximum and minimum values of every signal in the model. The maximum and minimum simulation values must cover the full intended operating range of the design in order to obtain meaningful results. It is the user's responsibility to run and find the worst case input values for the simulation which guarantee a full coverage of the worst case ranges.

3.2.2 Generating Code for One Subsystem

To make the generation of the code easy, it is recommended to use a model reference for that part of the main model that code will be generated for. In that way, the code will be generated easily and independently from the rest of the Simulink model that may have been added for testing reasons. For traceability reasons every subsystem can have its own function. This is possible if the user chooses the right options in the Subsystem Parameters dialog, a precondition is that the subsystem is atomic. See Figure 3. If a subsystem is atomic it will be treated as having direct feedthrough and thus be sorted to execute before any blocks that depend on it.


Function Block Parameters: PID controller

Subsystem

Select the settings for the subsystem block.

Parameters

[✓] Show port labels

Read/Write permissions: ReadWrite

Name of error callback function:

Permit hierarchical resolution: All

[✓] Treat as atomic unit

[ ] Minimize algebraic loop occurrences

Sample time (-1 for inherited): -1

RTW system code: Reusable function

RTW function name options: User specified

RTW function name: PidControl

RTW file name options: User specified

RTW file name (no extension): Pid

Figure 3: Subsystem parameter dialog pane

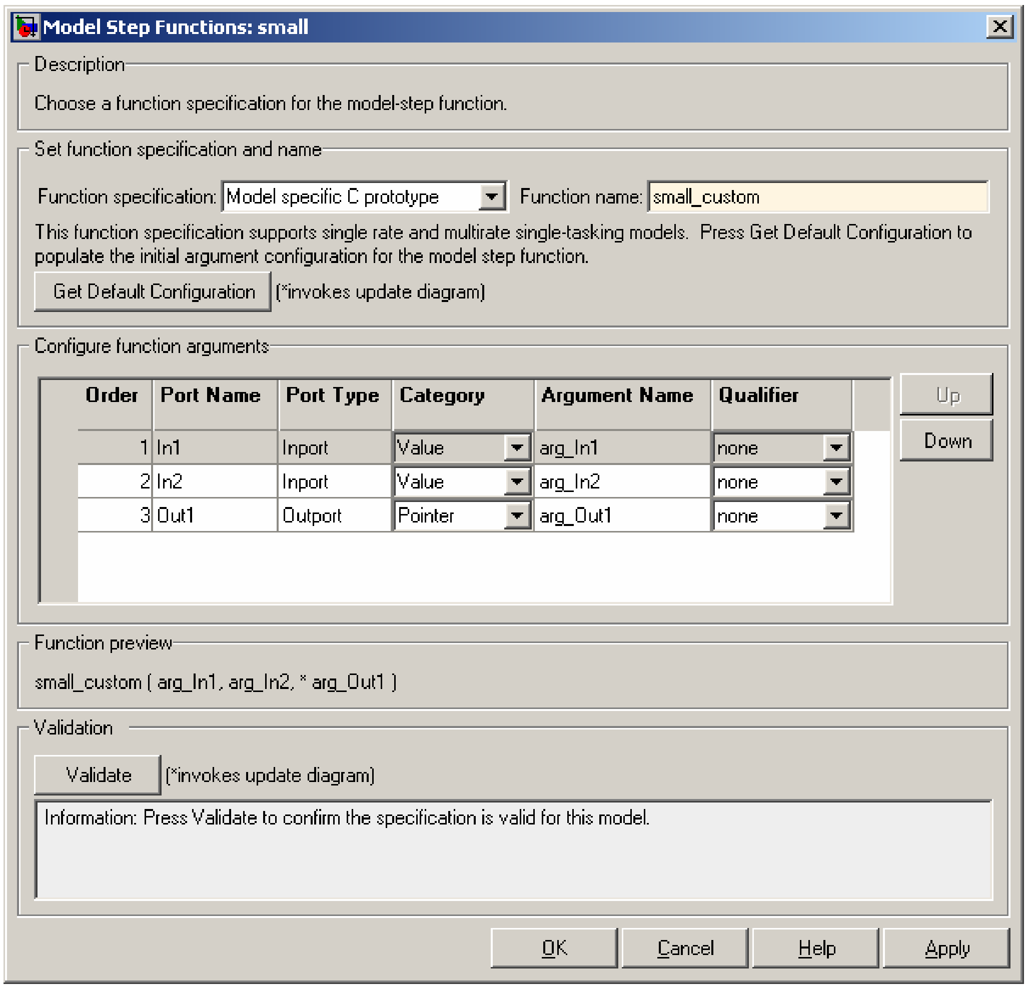
3.2.3 Code Interface

Real-Time Workshop Embedded Coder cannot offer the user full control over the generated code interface. Two options can mainly be selected, reusable code and non-reusable code. The user often needs the code to be reused again in other applications. A PID controller, for example, can be used in several applications and models. When the generated code is reusable all inputs, outputs and tunable parameters are passed on as function arguments. Tunable parameters are saved in a structure which is used as a function argument along with the inputs and outputs of the model.

Function return is not supported yet in Real-Time Workshop Embedded Coder.

Another approach can be used to control the interface of the generated code. Every signal and parameter can be chosen as a global variable and in that way non-reusable code can be generated when the user has access to the global variables making the code reusable. This approach is not preferred by Haldex Traction since the use of global variables is not preferred in the company code.

However in release 2007a there is a user interface to customise any step function, see Figure 4 below.


Model Step Functions: small

Description

Choose a function specification for the model-step function.

Set function specification and name

Function specification: Model specific C prototype   Function name: small_custom

This function specification supports single rate and multirate single-tasking models. Press Get Default Configuration to populate the initial argument configuration for the model step function.

Get Default Configuration (*invokes update diagram)

Configure function arguments

Order	Port Name	Port Type	Category	Argument Name	Qualifier
1	In1	Inport	Value	arg_In1	none
2	In2	Inport	Value	arg_In2	none
3	Out1	Outport	Pointer	arg_Out1	none

Function preview

small_custom (arg_In1, arg_In2, * arg_Out1)

Validation

Validate (*invokes update diagram)

Information: Press Validate to confirm the specification is valid for this model.

Figure 4: RTW EC - non-reusable code options for release 2007a

3.2.4 Simulation Using the Generated Code

To compare between a fixed-point simulation and a simulation with the generated code an S-function of the generated code can be obtained by simply selecting an option to generate an S-function in the code generation pane. The user can test the behaviour of the generated code by replacing the original fixed-point model reference with the generated S-function block.

3.2.5 External Function Calls and S-function Support

Replacing a hardware invocation with another function during simulation stage is not always practical when generating code with Real-Time Workshop Embedded Coder due to the difficulties of shaping the interface of the code. Here the only way to achieve the desired result is to use so called Legacy Code. Legacy code helps integrate external code into the model. In this project an S-function is generated and added to the model. Legacy code is then used so that the S-function block is replaced by the desired hardware function call in the generated code. However legacy code can not be used with reusable code unless the option `SS_OPTION_WORKS_WITH_CODE_REUSE` is set in the S-function.

3.2.6 MISRA C, Platform Dependency and Different Microprocessors Support

Real-Time Workshop Embedded Coder supports most of MISRA-C code standard rules. There is always a good motivation for those parts of MISRA-C rules that RTW EC does not support[1].

The generated code is ANSI-C or ISO-C by default which is platform independent. The generated code is therefore applicable in all kinds of platform and no changes are necessary if the platform is changed.

RTW EC supports many microprocessors, a list of which is:

xPC Target

Real-Time Windows Target

Target for Infineon C166®

Target for Freescale (tm) MPC5xx

Target for TI C2000 (tm)

Target for TI C6000 (tm)

The optimised generated code is platform dependent and all data types and storage classes that can be influenced by different platforms should be specified in different configuration sets as some targets have optimised blocks with target specific implementations for code generation.

[1] Information from Roger Aarenstrup, Senior Application Engineer at The MathWorks, Stockholm.

4 Analysis of the Generated Code

4.1 Memory Usage RAM, ROM and Execution Time

To make a comparison between the generated code and the hand written code, the system is built for host and target with the generated code from both TargetLink and the code generated from Real-Time Workshop Embedded Coder. Memory usage and execution time are calculated. The tables below show the execution times, RAM and ROM memory usage. The tested program code is the code for the pressure controller including a PID controller. See Appendix IV The Model.

The hand written code in Haldex base software was replaced by the generated code and a model was built and run on a C166 microprocessor as a target. Many tests were carried out while running on the target to see if the controller behaves as expected. ROM and RAM memory usage were calculated after the whole model was built. ROM and RAM memory usage were saved in a map file.

The generated code that was used in the tests is ANSI-C code when generated with TargetLink and optimised for Infineon C16x, XC16x when generated with RTW EC.

Pressure Controller & PID	**TargetLink (bytes)**	**Real-Time Workshop Embedded Coder (bytes)**	**Hand-written Code (bytes)**
ROM	1594	2676	2282
RAM	31	2	52

Table 2: RAM and ROM memory usage

It should be mentioned here that calibrated parameters were not chosen in RTW EC. The use of calibrated parameters in RTW EC was not investigated in this work due to lack of time. This can be why RTW EC gives less RAM memory usage than TargetLink as shown in Table 2 above. However, for future studies the option Inline Parameters should be investigated.

The generated code from TargetLink uses functions for all saturation checks in the whole code while RTW EC uses if-sets in every saturation checking everywhere in the code. This makes the RTW EC code less efficient and that is one of the reasons for the memory usage differences between TargetLink and RTW EC as shown in Table 2 above. An example of this is the following code which is generated for the same block:

TargetLink:

```
/* Sum: PressureController/PID controller/I-part/TrackingError */
Aux_I32 = (LONG) (((ULONG) (LONG) (I32UIn >> 1)) - ((ULONG) (LONG)
(X_Sa2_VOld >> 14)));
I16TrackingError = C__I16FITI32_SAT(Aux_I32, 32767
/*0.0004882663488 */);
```

`C__I16FITI32_SAT` is a function called in more than one place in the code.

Real-Time Workshop Embedded Coder:

```
/* Sum: '<S4>/TrackingError' incorporates:
     *  UnitDelay: '<S1>/VOld'
     */
    tmp_1 = (localDW->VOld_DSTATE >> 14);
    if (tmp_1 > 32767L) {
      tmp_0 = MAX_int16_T;
    } else if (tmp_1 <= -32768L) {
      tmp_0 = MIN_int16_T;
    } else {
      tmp_0 = (INT)tmp_1;
    }

    tmp_1 = (LONG)(rtu_U >> 1) - (LONG)tmp_0;
    if (tmp_1 > 32767L) {
      tmp_0 = MAX_int16_T;
    } else if (tmp_1 <= -32768L) {
      tmp_0 = MIN_int16_T;
    } else {
      tmp_0 = (INT)tmp_1;
    }
```

In the process one thread usually runs for the pressure controller. This thread is called CtrlTorqueUpdate and the execution time for that thread is saved in the vector RedThreadMaxTimes. Execution time is shown in the table below. The test was carried out for different inputs of the actual pressure and the results are summarised in Table 3 below.

Actual Pressure	**TargetLink (µs)**	**Real-Time Workshop Embedded Coder (µs)**	**Hand-written Code (µs)**
6 Bar	596	692	648
12 Bar	597	694	638
40 Bar	568	694	602
Fluctuating between 10-20 Bar	592	694	640

Table 3: Execution times

The table shows that TargetLink has the smallest execution time in all ranges of input.

4.2 Readability and Traceability

4.2.1 TargetLink

The generated code is well documented and follows the model designed in TargetLink. Any block in the model can be tracked in the code by either name or by following the code and comparing with the model.

The names of variables, blocks outputs, states or parameters in the generated code can be defined manually in every block or automatically generated by TargetLink using name macros. Every name macro starts with $ followed by a letter which is replaced by the string it represents during code generation. E.g. $M$B is replaced by <model name><block name> in the generated code. TargetLink has a total of 20 name macros.

TargetLink gives full freedom to change the code interface as desired. Any subsystem in the model can be chosen to be generated as a function with certain parameters and function returns if desired.

4.2.2 Real-Time Workshop Embedded Coder

RTW EC generates a code that is readable and can be traced if the Simulink model is followed carefully.

Embedded coder generates, if selected, a html report of the generated code where the code can be followed. In the code comments in the html report there are links to the blocks in the model. By just pressing on the block name in the code the corresponding block in the model is highlighted.

Any signal can be named and accessed in the code. The user can for example name a signal between two blocks and make it defined as a global variable in the generated code. Variables, blocks outputs, signals, states and parameters are automatically named in the same way as for TargetLink. RTW EC has less name macros to choose from, it has totally 6 name macros. These name macros apply to the whole model simultaneously and the user can not specify certain name macros for every block separately.

RTW EC is limited in the choice of code interface. Non-reusable and reusable are the only possible options here.

Function return is not supported in RTW EC.

5 Haldex Traction Specific Requirements

In this chapter specific Haldex Traction requirements are investigated in both TargetLink and Real-Time Workshop Embedded Coder.

Haldex Traction uses a practical approach to have several parameter sets which the user can change before driving and during runtime. These parameters are saved in a so called pdc-file which is programmed by Haldex Traction to access the parameters during runtime. The parameters sets are saved in vectors in the pdc-file. By calling the right index in the vectors, the desired parameter set is chosen.

Example:

If parameter sets are saved in the vectors ref1= [5 6 7] and ref2= [3 9 5] then parameter ref1 can have the value 5, 6 or 7 while ref2 can have the value 3, 9 or 5.

The user can access the parameters by a call for param1 = ref1[nbr] and param2 = ref2[nbr] where nbr is the index of the desired parameter set , 0, 1 or 2 in this example. By simply changing the value of nbr in one place in the code, the whole set is changed everywhere the parameters are called in the entire code.

It is also desired to make variables calibratable in order to be able to change the parameters via a calibration system. Calibration systems access variables directly by their addresses in the ECU memory.

5.1 TargetLink

TargetLink offers so called variant coding to handle parameter sets. There are two types of variant coding: data variants and code variants. The user can assign different values to a variable during runtime while code variant is to assign different data types or scaling parameter to a variable during code generation time. For Haldex Traction data variant is the interesting kind of variant coding. In the generated code all variables belonging to one data variant item, one parameter set, are collected in a struct. An example is shown below [3]:

```
typedef struct Vca DV tag {
      const volatile UInt8 param1;
      const volatile UInt8 param2;
      const volatile UInt8 param3;
} Vca_DV_tp;/*Struct data type for data variant
configuration DV*/
```

One instance of the structure above is generated for each variant item, that is one structure for each parameter set. There are two variant coding styles that are supported by TargetLink: the user can either generate a separate structure for every parameter set or generate an array of structures containing all instances. Those structures can have as many parameters as the memory allows and no limitation is set for how large the structures can be. The following example shows three parameters and two parameter sets in two separate structures:

```
Vca_DV_tp Vca_DVA = {
     5
     3
     1
}; /* for data variant item A */
Vca_DV_tp Vca_DVB = {
     6
     9
     1
}; /* for data variant item B */
```

In the generated code the parameter sets are accessed by a pointer which refers to the selected parameter set structure. The following code example shows how the parameter set structures are used in the generated code:

```
switch(varID) {
      case 1: {
            pVca_DV = &Vca_DVA;
            break;
      }
      case 2: {
            pVca_DV = &Vca_DVB;
            break;
      }
      default: {
            pVca_DV = &Vca_DVA;
            break;
      }
}
```

It is a simple task for the user of TargetLink to make a parameter calibratable. The right class should be chosen for the calibratable parameter in question. TargetLink has predefined classes that make variable calibration possible. The variables are written into a ASAM-MCD 2MC file. ASAM-MCD 2MC, formerly known as ASAP2, is a commonly used standard for describing ECU-internal data. The standard was developed by ASAM e.V. which stands for Association for Standardization of Automation and Measuring Systems e.V. ASAM-MCD 2MC files contain the information required by an MC system to access an ECU for parameter calibration.

5.2 Real-Time Workshop Embedded Coder

No parameter set approach is available in RTW EC yet. After discussing the idea with Roger Aarenstrup, senior application engineer from The Mathworks, a solution was discussed with The MathWorks development.

The solution that The Mathworks offers in the coming releases is to have a parameter vector in the Simulink model in Matlab workspace. This vector can be defined in an external file and accessed by a pointer, external code can therefore have access to the vector. The example below shows the idea, this has not been tested in this work but it is a recommendation from The MathWorks.

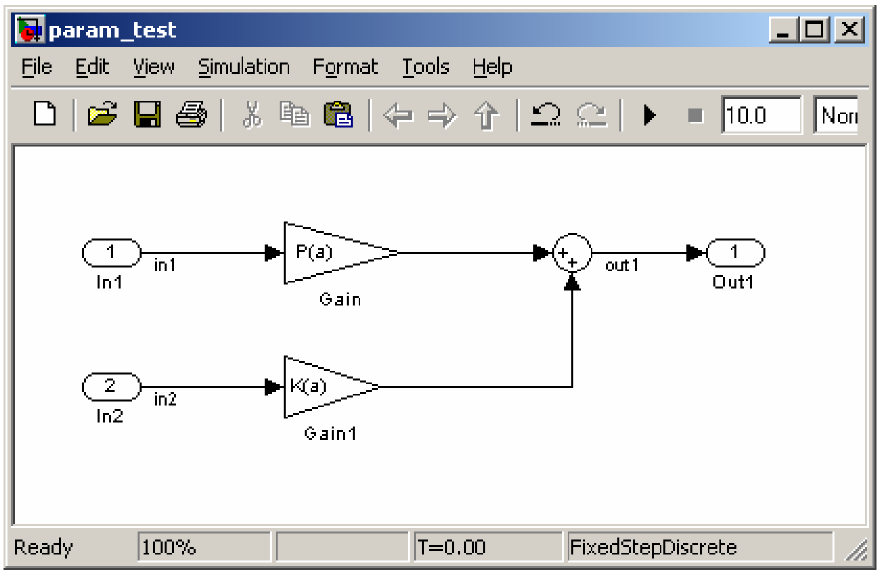


Figure 5: Example for the parameter set solution

The model in Figure 5 gives the following code:

Param_test.c

```
void param_test_step(void)
{
  /* Sum: '<Root>/Sum' incorporates:
   *  Gain: '<Root>/Gain'
   *  Gain: '<Root>/Gain1'
   *  Inport: '<Root>/In1'
   *  Inport: '<Root>/In2'
   */
  out1 = P[(int32_T)a - 1] * in1 + K[(int32_T)a - 1] * in2;
}
```

Param_test_private.h

```
/* Imported (extern) block parameters */
extern real_T a;                                    /* Expression: P(a)
                                                     * Referenced by blocks:
                                                     * '<Root>/Gain'
                                                     * '<Root>/Gain1'
                                                     */

/* Imported (extern) pointer block parameters */
extern real_T *K;                                   /* Expression: K(a)
                                                     * '<Root>/Gain1'
                                                           */
extern real_T *P;                                   /* Expression: P(a)
                                                     * '<Root>/Gain'
                                                     */
```

In another file, which is not generated from the model above, the following code can be used to specify the parameter sets:

```
Int a = 1; /* Specify parameter set */

/* Parameter sets */
Real_T *P = {0.3, 0.5, 0.2…}; /* P – constant */
Real_T *K = {0.3, 0.5, 0.2…}; /* K – constant */
```

Calibratable parameter can be chosen by GetSet function that can be chosen for any parameter or signal in the model. The tool for calibration can get access to the parameter/signals in that way. ASAP2 files can be generated too.

6 Simulation and Modelling

6.1 Usability

How easy and how practical it is to use the code generation tools is investigated in this chapter. Simulink is the common main environment in both TargetLink and RTW EC. The model can be built and tested in Simulink first and then converted to the code generation tool since both TargetLink and Real-Time Worskhop Embedded Coder use Simulink. The model can also be built directly in TargetLink. Both TargetLink and RTW EC have tools that help the user along the way while building and generating the code.

6.1.1 TargetLink

The user can directly use TargetLink blocks to build the model. It is also possible to easily convert an already existing Simulink model to a TargetLink model. TargetLink has a conversion tool which can be used to convert any model from Simulink model to TargetLink model and vice versa. After conversion the user will be able to enter scaling, data type and offset for every block in the model.

The user can easily specify which subsystem in the model the code will be generated for. TargetLink generates code for only the subsystems that contain TargetLink blocks. TargetLink subsystem can be part of a Simulink model. In that way the user can use Simulink blocks connecting to the TargetLink block system which makes it easy to test and handle in general.

TargetLink offers Data Dictionary, DD, where all central data can be saved. The user can manage all variables and the corresponding properties from the Data Dictionary Manager. A library for data can be created in the Data Dictionary and a central data library is then available and can be accessed independently by any project TargetLink model. This makes managing data practical and helps keep data consistent, especially in large and complicated models. TargetLink gives the user full freedom to save the desired properties as subjects in the Data Dictionary. The user can, for example, choose to create a data object for scaling which the user can refer to anywhere in a model. In that way the user will save time by just referring to the created scaling object in several blocks in the model. If a change in the data object occurs then the change will apply to all blocks that are connected to that data object and the user will only need to make the desired change once. There are different kinds of Data Dictionary objects and Table 4 shows the important kinds of these objects.

Data Dictionary Object	**Description**
Variable	Defines the properties of a variable in the C code. All kinds of variables are supported: plain variables, pointers, macros, structs and bitfields. Relevant variable properties are Type, Class, Scaling, Value, Min, Max. Further properties exist to describe how a variable can be accessed in a calibration system, for example, ByteOrder, Deposit, Format and MaxRefreshRate. The latter mentioned properties were not tested in this work.
Typedef	Defines a C data type, including pointers, structs and bitfields. The user can create user-defined types. Each typedef is mapped to one of TargetLink basetypes like Bool, Int8, or Int16. If needed, the user can also associate a typedef with certain constraints, e.g., a scaling data object or min/max values.
Scaling	Describes how to convert the fixed-point representation of a variable in the ECU memory into the corresponding physical value.

Table 4: TargetLink Data Dictionary object types

Another useful tool is the Property Manager. It is a graphical user interface that displays the properties of TargetLink blocks in the model. The Property Manager allows the user to view and modify several blocks properties simultaneously.

To make the model easier to follow, the blocks properties are preferred to be visible in the model. Properties like data type, offset and scaling are practical information to have visible in the model to make it easier for the user to see, for example, if scaling mismatch occurs. TargetLink makes this possible but only during the use of the automatic scaling tool. The information disappears as soon as the automatic scaling tool is closed. According to dSPACE a script can be written that will make the desired information visible constantly[2].

To have default block properties is not fully supported by TargetLink. The only default property that TargetLink blocks can have is the data type e.g. Int16, Int32 or Bool[3].

[2] Jonas Cornelsen from Fengco.
[3] Has not been tested in this work but Jonas Cornelsen from Fengco has confirmed it.

If a TargetLink license is not available a model can still be run in Model In the Loop mode, which means that the model runs with floating-point. This allows co-workers who do not have a license to have access to any model built in TargetLink. These users can use TargetLink stand-alone-mode for which no license is necessary.

6.1.2 Real-Time Workshop Embedded Coder

The user starts to build the model in Simulink using fixed-point and no model conversion is necessary. It is recommended to use model reference for the subsystem that the code will be generated for. This makes it easier to generate the code.

In Simulink the user can easily visualise important information for every block in the model. The user can choose to visualise port data types, storage class and sorted order of every block in the model. These options are available in any Simulink model under Format. In that way the user can easily follow the scaling flow in the model.

RTW EC does not have the possibility to change the properties for several blocks at the same time. The user has to change the properties of every block separately. Matlab/Simulink offers Model Explorer which contains all data from the model. Model Explorer allow the user to easily change the properties of every variable, block or signal from the model without having to go back to the model interface. Creating and saving a central data library that can be used in many models from different projects is possible in RTW EC by using storage class objects. This has not been investigated in this work.

Carrying out code modifications in RTW EC is not always straight forward. In order to define a parameter as a macro in the generated code, two steps at least are required. The user has to change properties for the parameter in two different places in RTW EC tools. This is what makes it sometimes confusing for the user, especially for first time users.

Most kinds of variables are supported in RTW EC: plain variables, pointers, structs and macros with the restriction of the earlier mentioned Constant block.

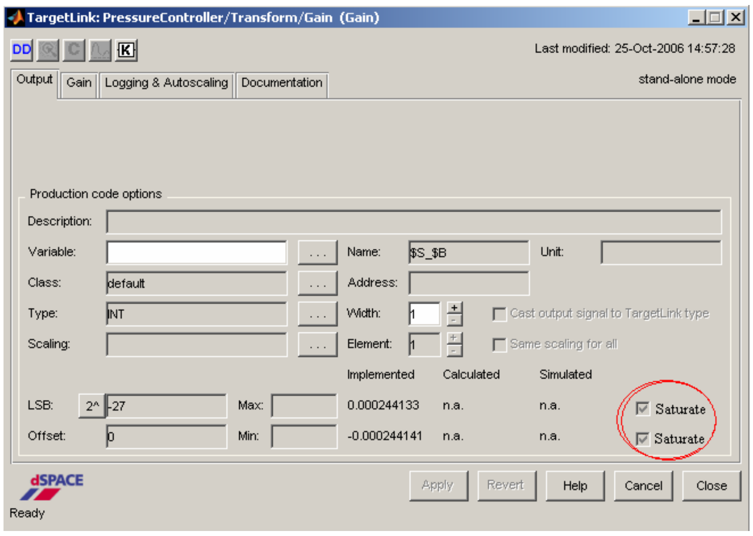
Users who do not have a license for RTW EC and Fixed-Point Toolbox can also run the model. These users must first change some properties in the Fixed-Point Settings from the model Tools menu. These changes will enable users without license to simulate with floating-point.

6.2 Supported Simulink Blocks and Their Properties

6.2.1 TargetLink

In a TargetLink subsystem the user can use TargetLink simulation blocks, TargetLink utility blocks and supported Simulink blocks. Appendix II shows all available blocks that the user can use to build a model in TargetLink.

If the user knows what blocks to use from the beginning then the implementation of any control model can be done. Almost all TargetLink blocks have overflow protection available. There are two separate alternatives for overflow protection, one for the maximum value and one for the minimum value. The user can choose protection against upper overflow, lower underflow or both. Figure 6 shows an example of a Gain block and how overflow protection can be chosen easily.


TargetLink: PressureController/Transform/Gain (Gain)

Last modified: 25-Oct-2006 14:57:28

stand-alone mode

Output | Gain | Logging & Autoscaling | Documentation

Production code options

Description:					
Variable:		Name:	$S_$B	Unit:	
Class:	default	Address:			
Type:	INT	Width:	1	☐ Cast output signal to TargetLink type	
Scaling:		Element:	1	☐ Same scaling for all	

				Implemented	Calculated	Simulated	
LSB:	2^ -27	Max:		0.000244133	n.a.	n.a.	✓ Saturate
Offset:	0	Min:		-0.000244141	n.a.	n.a.	✓ Saturate

Figure 6: TargetLink Gain block dialog

TargetLink provides the user with Addfile block that makes it possible to add custom code to the generated code. Another block called Custom Code can also be used for this purpose. This block uses S-function when simulating the model in Simulink and in the generated code added in the Custom Code block is used instead of the S-function in the model. All TargetLink blocks and Simulink supported blocks are listed in Appendix II.

6.2.2 Real-Time Workshop Embedded Coder

Real-Time Workshop Embedded Coder supports all Simulink blocks except Matlab Fcn and M-file and Fortran S-function which are not inlined with TLC, Target Language Compiler [4]. Some blocks have restrictions, for example some blocks cannot be used inside a triggered subsystem hierarchy as a triggered subsystem can not include blocks that are dependant on a sample time since the trigger makes it non-sample time based. On the other hand not all Simulink blocks support fixed-point implementation and that should be taken into consideration. Appendix III shows the fixed-point blocks that are supported by Simulink and code generation tool [5].

Some blocks have overflow protection. Blocks like Gain block can saturate the output if the latter increases or decreases the desired maximum or minimum values. This is not preferred for code protection since it gives an extra amount of code lines which takes extra time from the CPU. At the same time it is practical for critical blocks in the model where the user expect overflow.

Real-Time Workshop Embedded Coder has a convenient way for adding custom code to the generated code. The user can write custom code to be included in the header file or the C file of the generated code. The user can write the C-code in the Custom Code pane of the Configuration Parameters dialog. Another way to incorporate existing code in the generated code is to use Legacy Code Tool. This tool transforms an existing C code into C MEX S-function for inclusion in the Simulink model. If the user does not want to build an S-function from scratch, an S-function can be built automatically by converting a subsystem into an S-function. Legacy Code Tool can then be used to include the desired custom code instead of the S-function used in the Simulink model.

A reference model principle can be used in designing a model in Simulink where some subsystems are designed as separate models and then added as a model reference to any kind of design. This results in subsystem saving for reuse in different projects.

6.3 Fixed-Point Support

Fixed-point is used in microprocessors to represent floating values and a loss of accuracy often occurs. Scaling is used to convert the real floating value to a fixed-point value and vice versa.

A value can be, for example, represented as an 8-bit unsigned integer, which correspond to a range between 0 and 255 ($= 2^8 - 1$). If the represented value is between 50 and 150 then almost 60% of the unsigned integer range is unused and all decimals are ignored when they occur, see Figure 7 below.

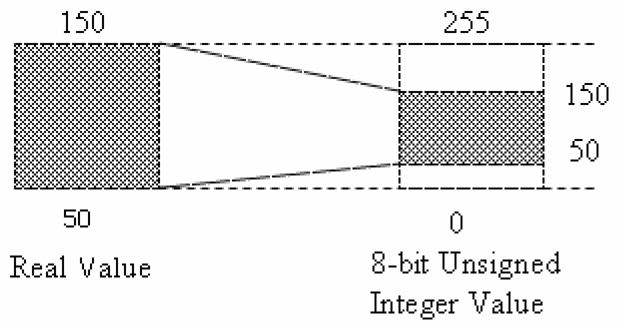


Figure 7: Representing a value without scaling

For example the integer representation of 75.7 is, according to the representation above, 75. A loss of 0.7 can be large depending on the design and the type of the value. To solve this kind of problem a conversion rule is made to use the entire number range provided by an integer and to be able to shift this range. The following equation describes the relation between the real value, x, and its integer representation, x´:

$x = S_x \cdot x' + O_x$, where S_x is the scaling factor and O_x represents an offset that can be used to shift the integer number range.

Using a scaling factor of 0.5, $S_x = 2^{-1}$, and an offset, $O_x = 25$ gives the following conversion of the real value into the 8-bit unsigned integer range:

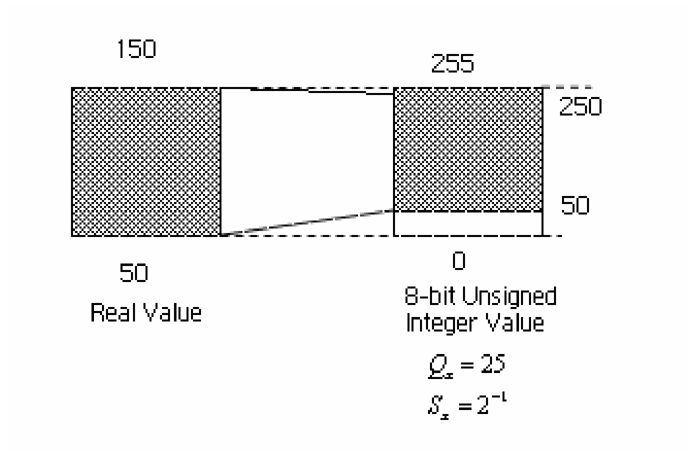


Figure 8: Representing a value with scaling

Using this approach, more than 75% of the 8-bit integer number range is covered and used. This gives a better resolution of the representation of the real value. This kind of scaling is called the power-of-two scaling which uses the scaling factor $S_x = 2^n$, where n is an integer type. Another type of scaling can be used too, called arbitrary scaling where S_x is an arbitrary real number.

6.3.1 TargetLink

Fixed-point is supported in all TargetLink blocks, as mentioned earlier. Both power of two scaling and arbitrary scaling, where the scaling factor is an arbitrary real number, are supported in TargetLink. Scaling can be done with or without offset value. The user can easily choose scaling parameters manually in every block in the model. Blocks can also refer to an already defined scaling objects from the Data Dictionary, see chapter 5.1.1 TargetLink and Table 4.

The user can also perform automatic scaling in the model. TargetLink supports two methods for the determination of the range of a variable: by simulation and by worst case value range propagation. Based on the determined ranges, TargetLink calculates suitable scaling parameters and automatically enters them in the respective block dialog. The user must first enter as many known ranges as possible in the model. If there is a closed loop in the model, at least one block must have known ranges otherwise worst case value range propagation will stop. Every block has a protection option against scaling. The user can choose to leave specific data information in the block untouched when running the automatic scaling tool. Output data type and output offset are examples of block information that can be chosen not to be effected by automatic scaling.

6.3.2 Real-Time Workshop Embedded Coder

To support fixed-point implementations, Matlab must have the Fixed-Point Toolbox installed as well as RTW EC. Fixed-point is supported in many useful blocks, see Appendix III. By knowing which blocks that support fixed-point, the user can build the desired model without any problem. Any control model can be build if the supported blocks are used wisely. The Matlab Fixed-Point Toolbox supports both power of two scaling and arbitrary scaling.

In RTW EC automatic scaling is supported depending on simulation results ranges. No worst case range propagation is supported in Real-Time Workshop Embedded Coder/ Simulink Fixed-Point Toolbox. Propagations depending on the simulation ranges are not preferred. To obtain a useful result the user has to choose the simulation inputs very carefully to cover all the critical ranges. Another problem is that the user cannot have fixed outputs ranges after automatic scaling. The automatic scaling tool propagates ranges depending on the simulation inputs, the outputs of the model will then have specific ranges which are determined by the scaling tool. This results in output scaling factors that do not match those required by the company.

7 Miscellaneous

7.1 Own Experiences

As a beginner working with TargetLink was easier than working with Real-Time Workshop Embedded Coder. TargetLink has a well written user guide that gives the first time users all the information needed to start code generation for any model, whether with floating point or with fixed point. To generate code in RTW EC, two user guides are needed if fixed-point is used which makes it hard for beginners to start code generation. It takes time to know where the desired tools are located in the RTW EC and sometimes it is not clear what steps the user should take in order to give a certain result. For example the user has to make changes in two different panes to make a variable defined as extern global variable in the generated code. However it is not hard to handle once the user gets used to the tool.

In TargetLink all the information needed to generate code is gathered in one place which makes it easy for the user to handle the tool. TargetLink has also a very useful tool for gathering all variables and signals in the model in one place, so called Data Dictionary, which can be saved and accessed from any model built in TargetLink.

RTW EC tools are very well integrated with the rest of the Simulink tools which can be confusing for the user at first. But once the user can find his way around, most desirable code settings can be achieved. The MathWorks recommends to use an excel sheet to save variables for reuse.

One of the problems that were faced with Real-Time Workshop Embedded Coder was to have a function call in the simulation that executes a different code when connected to the hardware. This problem can be caused by quantization errors. These errors should be considered when running with the target, but they are irrelevant in the simulation since no quantization errors occur when the hardware is not connected. To solve this problem, a different code is executed when connected to the hardware, while in the simulation a simpler code is run instead.

The model tested in this work contained a block called Quantisation. This block takes care of the problem described above. Two approaches were tested to solve this. The first is the use of the so called Legacy Code in RTW EC. An S-Function is used in the simulation which just gives an output that is equal to one of the inputs, see Figure 9. Using Legacy Code tool the generated code can have a certain function call instead of the S-function that is used when simulation is run. This solution was not adopted in this work since it was not clear how it can be used with reusable code. If the user chooses to generate the S-function for a subsystem automatically then an extra option must be set manually in the S-function in order to be able to use Legacy Code with reusable code. This was not clear from the beginning so another approach was used in this work.

The other approach was to manually exchange the function call in the generated code. The function call is made in one place only in the whole code. The block Quantisation, Figure 9, has two inputs which are the function parameters, but Real-Time Workshop Embedded Coder always optimises away the declaration of any irrelevant signal. This made it hard to have access to both inputs since one of the signals is terminated inside the block and thus optimised away in the generated code, see Figure 9. The solution to this specific problem was to make a small change inside the Quantisation block in order to have access to both inputs in the code, see Figure 10. This solution did not affect the rest of the generated code since the whole subsystem was chosen to be a function with its own code file. As mentioned earlier this problem could have been solved using Legacy Code Tool.

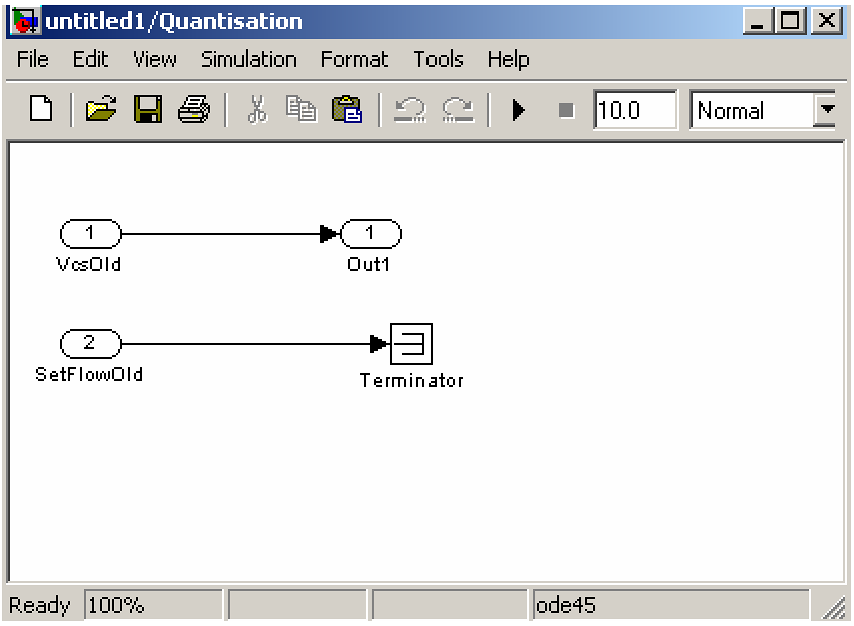


Figure 9: Quantisation subsystem

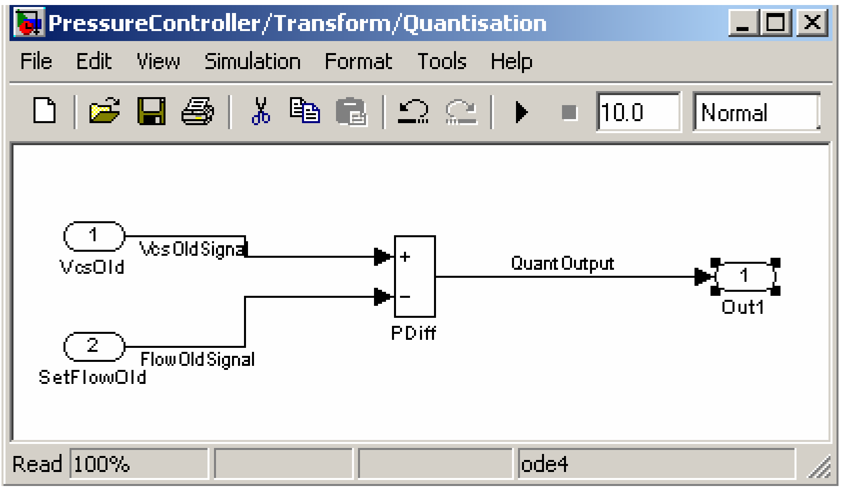


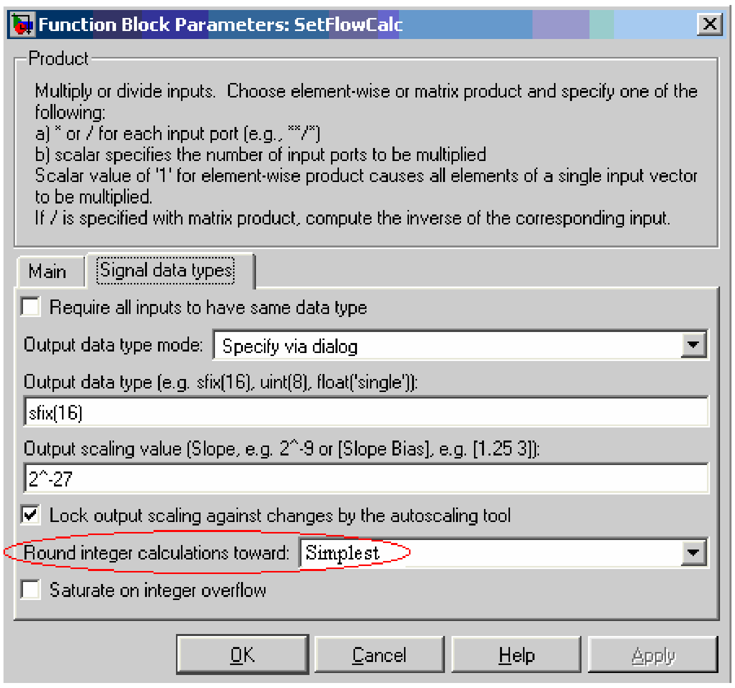
Figure 10: Quantisation subsystem after modifications

Another issue that was a problem in both TargetLink and Real-Time Workshop Embedded Coder was the definition of the base data types. Depending on the target microprocessor every base data type is defined differently, e.g. integers are defined as int or short depending on the type of the microprocessor used in the target. Haldex Traction has its own defined base types. A specific code is developed for the purpose of defining data base types depending on whether the code is running in the PC or on the target.

In TargetLink the base data types are defined in a header file and the user can rename them easily. All base data types in TargetLink were renamed so that they match those of Haldex Traction. TargetLink declaration of those data types were manually removed from the generated code and instead Haldex Traction base data type declarations were used.

Real-Time Workshop Embedded Coder works in the same way as TargetLink. The user can rename the base data types that are declared in a header file that can be modified in the same way described above. The user can also choose to generate code for a specific microprocessor and thus the right declaration of all base data types is achieved. This option is available in TargetLink too but it does not change the declaration of the base data types as required. However the user can change the declaration for all base data types manually in the TargetLink base files and in that way the base data types will be declared as desired. This need to be done only once and the changes made will be applicable every time code is generated.

One of the problems that were faced with RTW EC was choosing the right option in some blocks to obtain the desired simulation results. The simulation gave a graph with a lot of variations in the signal due to sampling. A small change in the properties of the blocks of multiplications, divisions and look-up tables gave the expected result, which was a smooth graph that followed the set points. By simply choosing to round to simplest, see Figure 11, the whole result changed and a smooth graph was obtained. This option also saves 30-40 rows of code for every block.


Function Block Parameters: SetFlowCalc

Product

Multiply or divide inputs. Choose element-wise or matrix product and specify one of the following:
a) * or / for each input port (e.g., **/*)
b) scalar specifies the number of input ports to be multiplied
Scalar value of '1' for element-wise product causes all elements of a single input vector to be multiplied.
If / is specified with matrix product, compute the inverse of the corresponding input.

Main | Signal data types

☐ Require all inputs to have same data type

Output data type mode: Specify via dialog

Output data type (e.g. sfix(16), uint(8), float('single')):
sfix(16)

Output scaling value (Slope, e.g. 2^-9 or [Slope Bias], e.g. [1.25 3]):
2^-27

☑ Lock output scaling against changes by the autoscaling tool

Round integer calculations toward: Simplest

☐ Saturate on integer overflow

Figure 11: Multiplication block in RTW EC with the right options

7.2 Generated Warnings

Both TargetLink and Real-Time Workshop Embedded Coder give a set of warnings regarding overflows and loss of precision. TargetLink generates a text file for these kinds of warnings while RTW EC gives a set of warnings displayed on the Matlab workspace.

7.3 Make-file

Both TL and RTW EC generate a make-file. None of these make-files was used in this work. Haldex Traction general make-file was modified to include the generated files instead.

7.4 The Generated Files

TargetLink has a practical way to gather all the generated files into one place which makes it easier for the user to see which files are relevant. With just one click all generated files are exported into a map. Real-Time Workshop Embedded Coder does not have this property.

By default TargetLink generates the following files:

<subsystem>.c	Contains the production code that is generated for the subsystem named in the angle brackets.

<subsystem>.h	Provides the declerations of global variables and functions defined in <subsystem>.c. The variables and the functions in this header file describe the interface of the generated code.
<subsystem>_udt.h	Contains user-defined types. This file is not generated if no user-defined types are included in the model.
tl_basetype.h and tl_types.h	Contains TargetLink defined types, for example, Int8 or UInt32.
tl_defines_<subsystem_ID>.h	Contains TargetLink-defined pre-processor macros like FX_GROUND, macros of variable classes with the use name attribute and the log macro. This file was not generated in this work.

More files are generated for simulation purposes only, those files do not affect the production code in any way and are not mentioned here.

Real-Time Workshop Embedded Coder generates the following files:

<subsystem>.c	Contains entry points for all code implementing the model algorithm.
<subsystem>_private.h	Contains local macros and local data that are required by the model and subsystems. This file is included by the generated source files in the model.
<subsystem>.h	Declares model data structures and a public interface to the model entry points and data structures.
<subsystem>_data.c	Conditionally generated. It contains the declarations for the parameters data structure, the constant block I/O data structure, and any zero representations used for the model’s structure data types.
<subsystem>_types.h	Provides forward declarations for the real-time model data structure and the parameters data structure. These may be needed by function declarations of reusable functions. Also provides type definitions for user-defined types used by the model.

rtwtypes.h	Defines data types, structures and macros required by Real-Time Workshop Embedded Coder generated code.
ert_main.c	This file is generated only if the **Generate an example main program** option is on. This option is on by default. autobuild.h is generated only if the **Generate an example main program** option is off. autobuild.h contains #include directives required by the static version of the ert_main.c main program module. Since the static ert_main.c is not created at code generation time, it includes autobuild.h to access model-specific data structures and entry points.
<subsystem>_capi.c and <subsystem>_capi.h	Optional files, provide data structures that enable a running program to access model parameters and signals without use of external mode.

There is a pack-and-go capability in RTW EC to give a zip-file with all the generated files.

8 Results

A comparison between the simulation results in Simulink and the simulation results in the respective tool was done. Four different simulations were carried out for different inputs to test the behaviour of the controller in different situations:

Test case 1 - constant pressure set points and constant differential speed
Test case 2 - varying pressure set point and constant differential speed
Test case 3 - constant pressure set point and varying differential speed
Test case 4 - Extern VCS control with constant pressure set points and constant differential speed

Figures 13-16 show results for Simulink, TargetLink and Real-Time Workshop Embedded Coder respectively. Note that the results from the simulation in both tools are the results of running software in the loop, which means that the generated code is used in the simulation:

Test case 1:

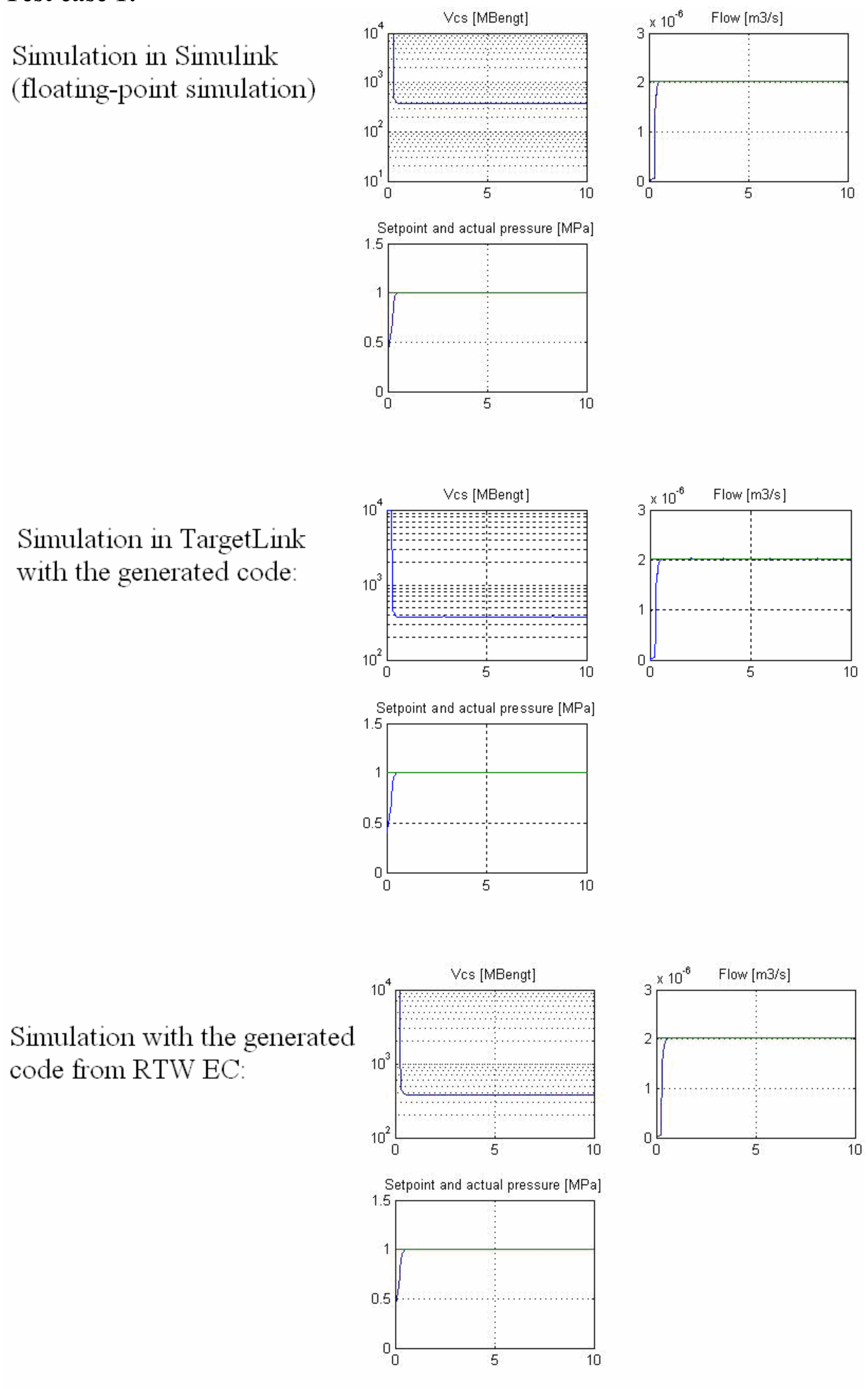


Figure 12: Test case 1- constant pressure set points and constant differential speed

Test case 2:

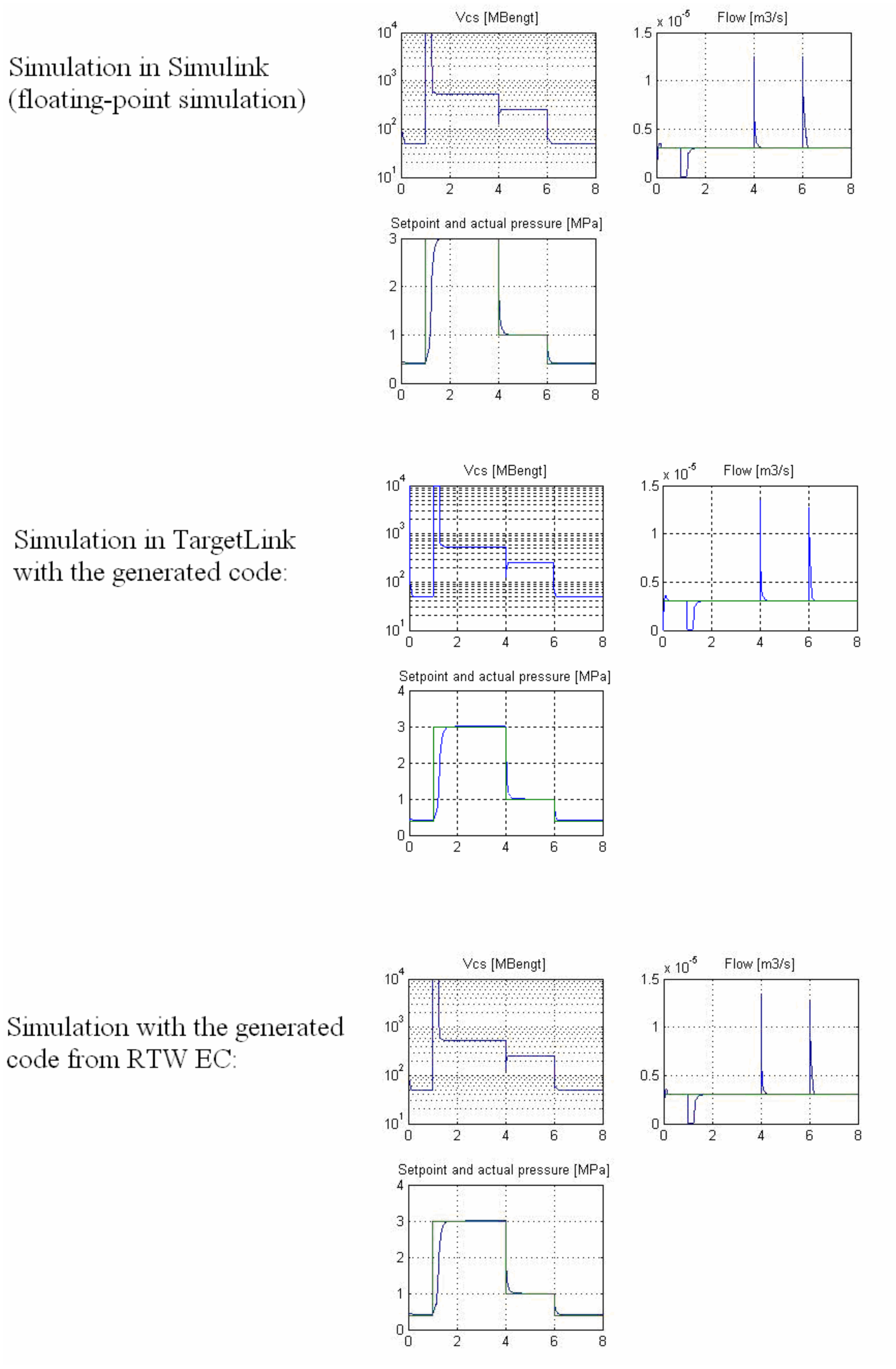
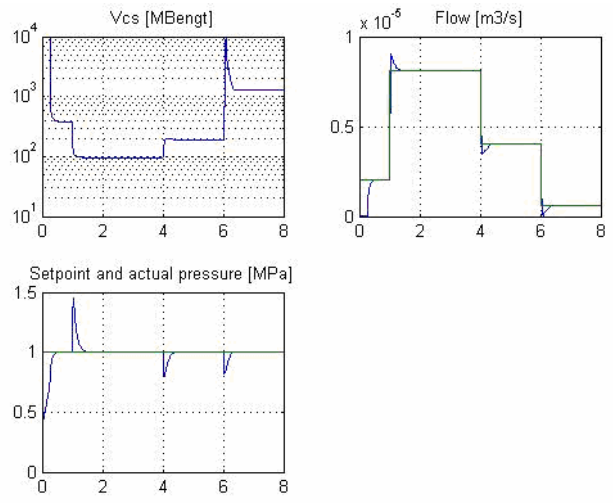


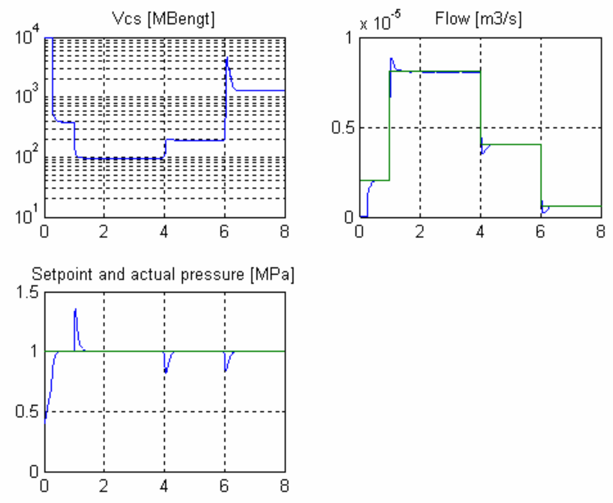
Figure 13: Test case 2- varying pressure set point and constant differential speed

Test case 3:

Simulation in Simulink (floating-point simulation)



Simulation in TargetLink with the generated code:



Simulation with the generated code from RTW EC:

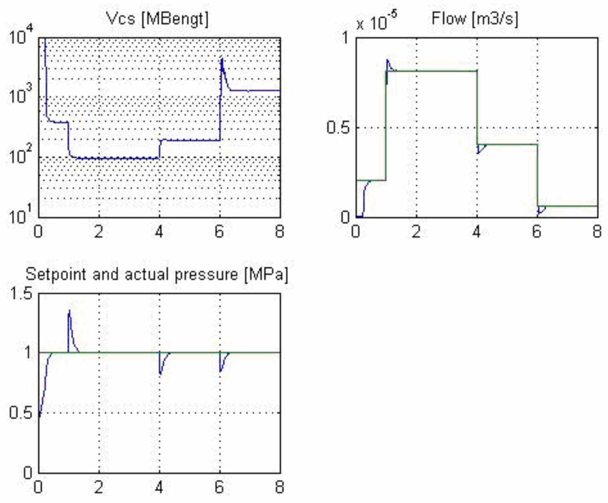
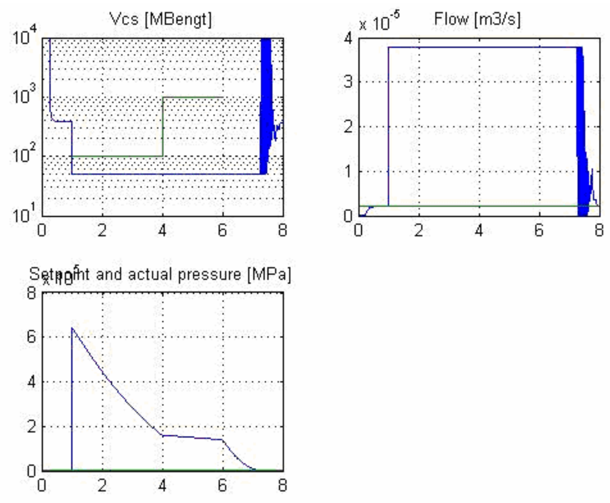


Figure 14: Test case 3- constant pressure set point and varying differential speed

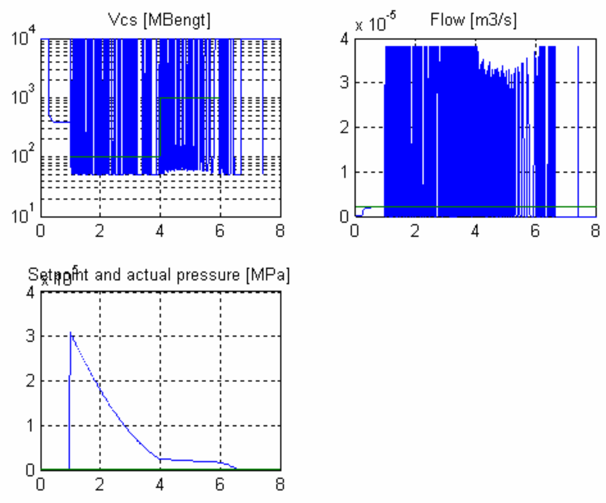
The above figures show the simulations for different inputs for the system. It is clear that the controller acts in the same in all simulations, whether it is a Simulink floating-point simulation or a simulation with the generated code from TargetLink or Real-Time Workshop Embedded Coder.

Test case 4:

Simulation in Simulink (floating-point simulation)



Simulation in TargetLink with the generated code:



Simulation with the generated code from RTW EC:

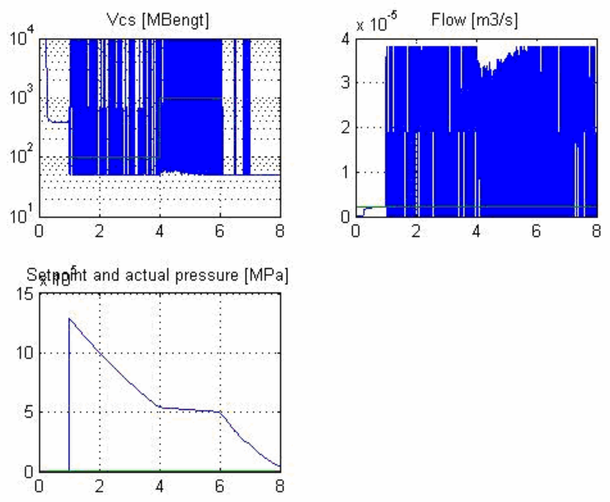


Figure 15: Test case 4- Extern VCS control with constant pressure set points and constant differential speed

TargetLink and RTW EC give the same result in the simulation for test case 4. As shown in Figure 16 for test case 4, the result is a signal which does not follow the set points. This is expected since in this case an external signal is used in the system and the controller signal is ignored and not used.

Another comparison is made, this time between controlling against target with the handwritten code and the generated code from both tools. This is done by integrating the generated code with the base software using Lauterbach to run on the target. Lauterbach is an embedded system that helps to connect and run the software with the hardware which is a microprocessor that controls the valve of the coupling.

Four tests have been carried out with Lauterbach. Different values of the actual pressure in the valve were chosen to see how the controller works to regulate the pressure to follow certain pressure setpoint:

Test case 1: Actual pressure = 0.6 MPa
Test case 2: Actual pressure = 1.2 MPa
Test case 3: Actual pressure = 4 MPa
Test case 4: Actual pressure is varying between 1MPa and 2 MPa

The results are shown in figures 17-23 below for handwritten code, TargetLink generated code and Real-Time Workshop Embedded Coder generated code respectively. In the figures below the green line is a simulation of a driving situation, the red line is the pressure set points, the blue line is the actual pressure which is set by the tester here and it always has a specific value that does not change and does not follow the pressure set point in the figures and finally the purple line is the valve signal which is controlled by the implemented controller.

Test case 1:

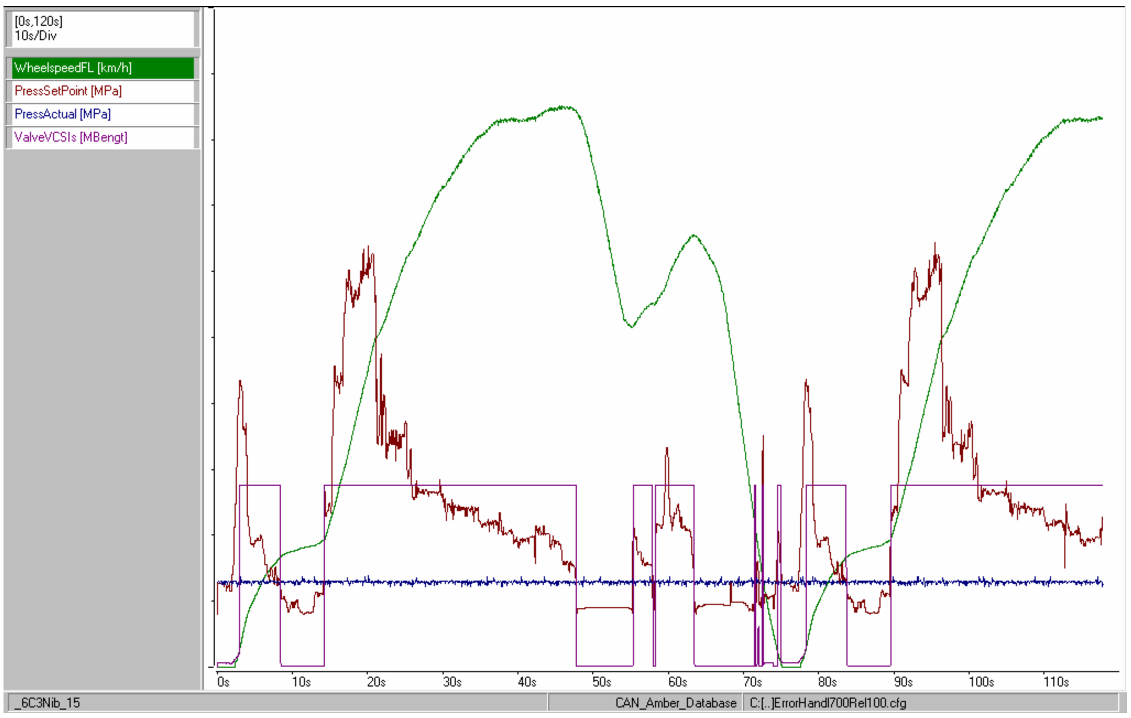


Figure 16: Test case 1 - handwritten code where actual pressure = 0.6MPa

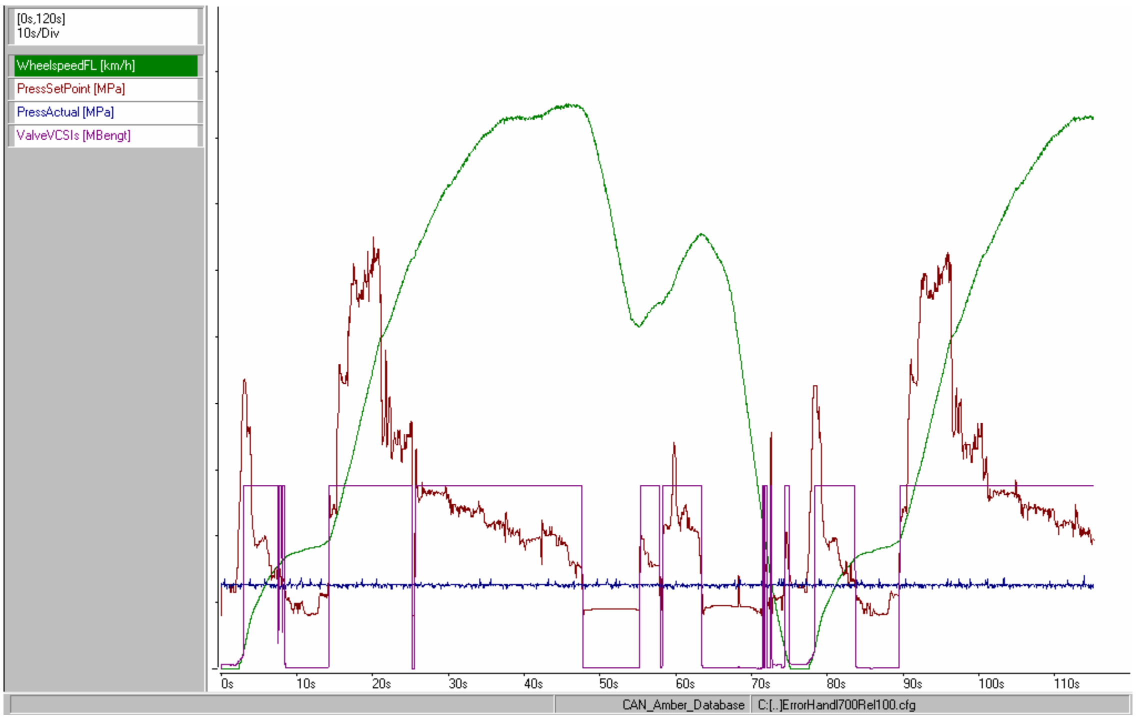


Figure 17: Test case 1 - TargetLink generated code, actual pressure =0.6MPa

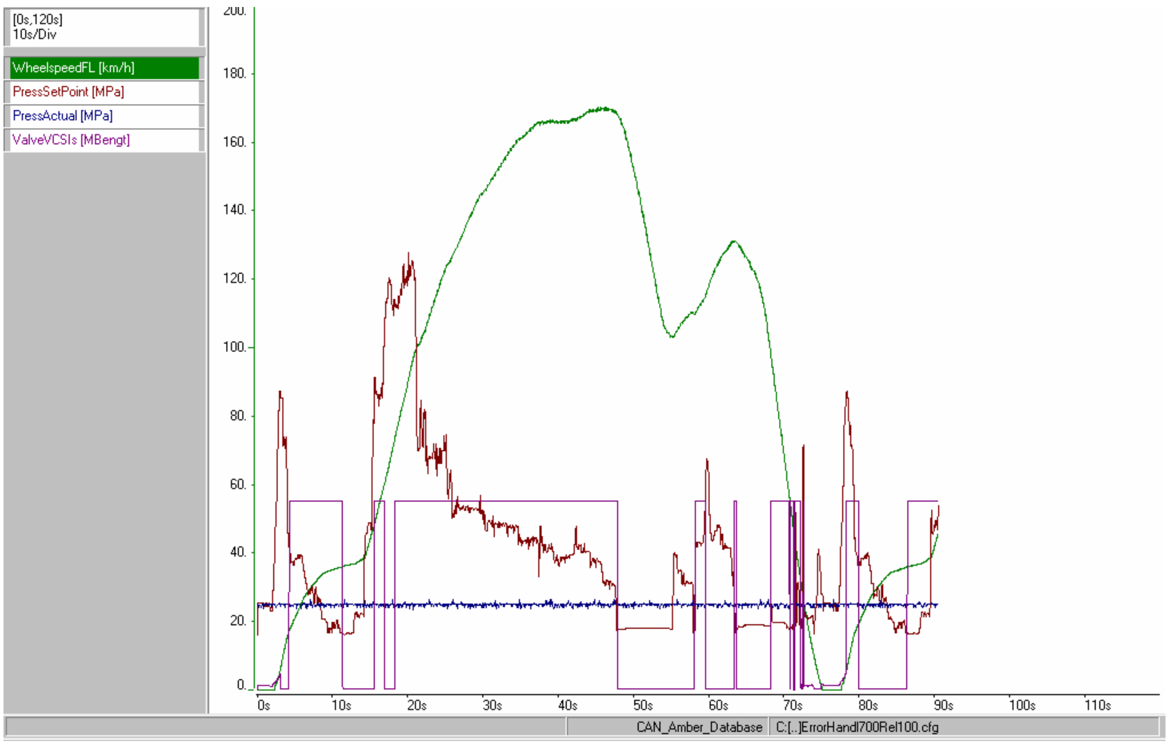


Figure 18: Test case 1- Real-Time Workshop Embedded Coder generated code, actual pressure=0.6MPa

Test case 2:

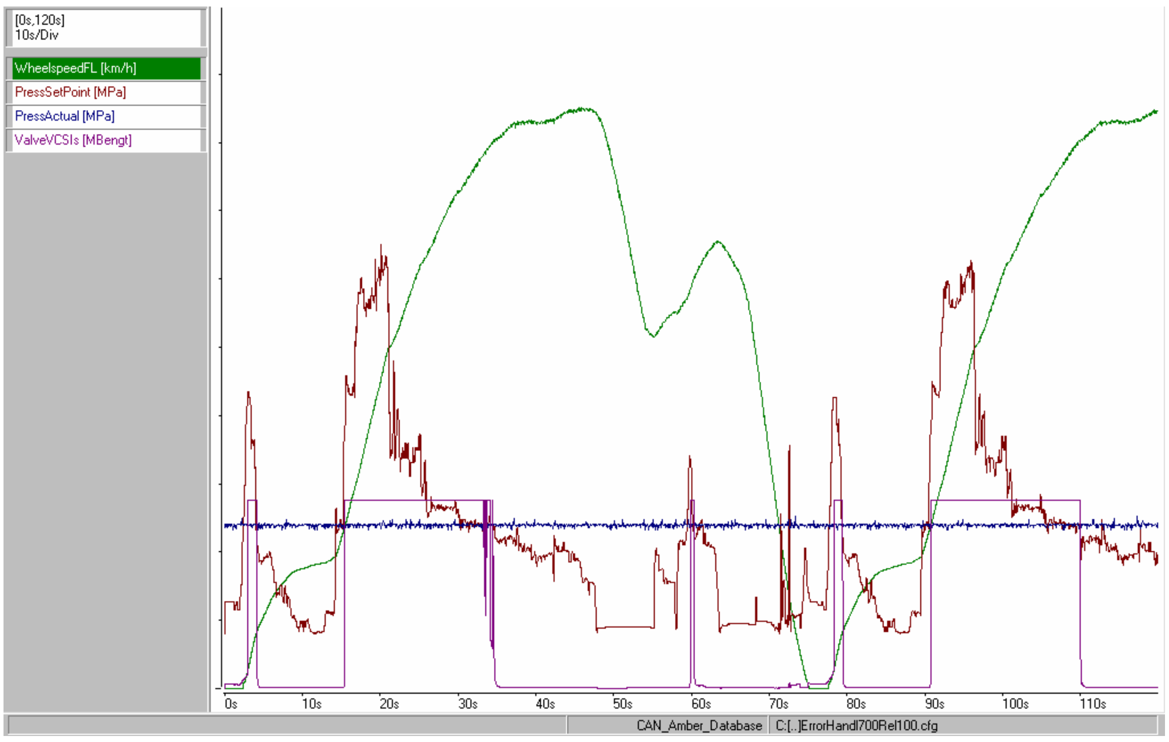


Figure 19: Test case 2 - handwritten code, actual pressure=1.2MPa

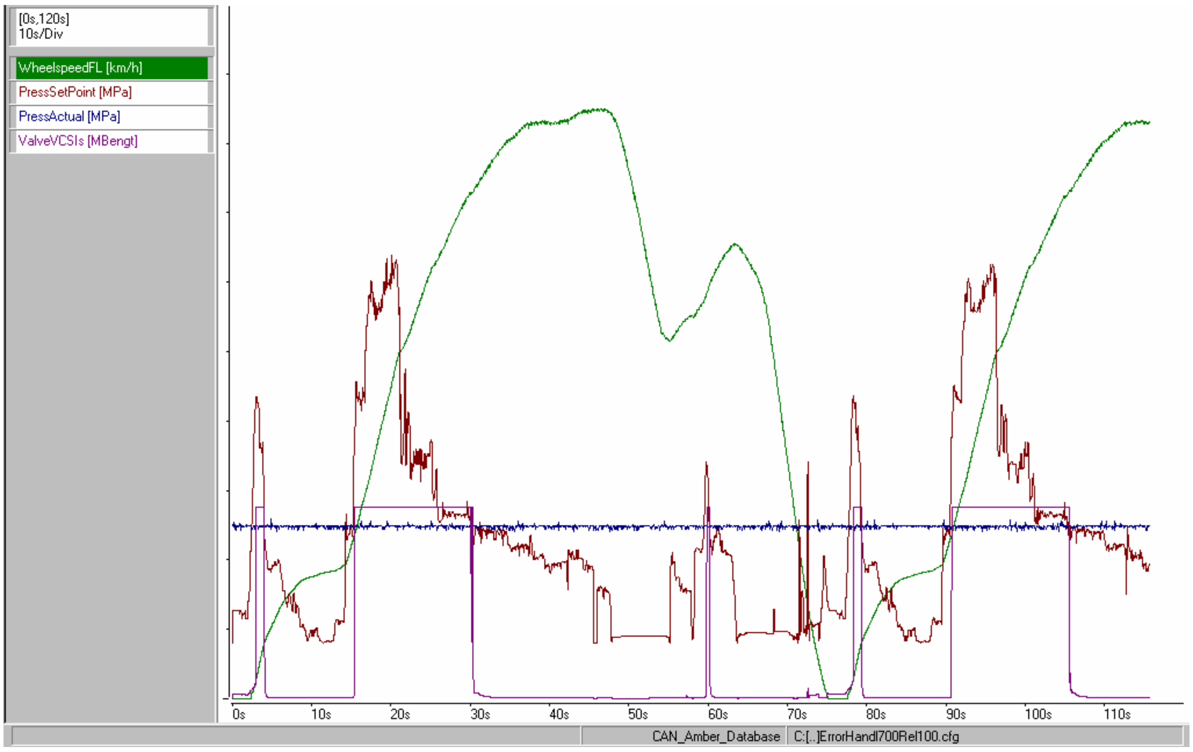


Figure 20: Test case 2 - TargetLink generated code, actual pressure = 1.2MPa

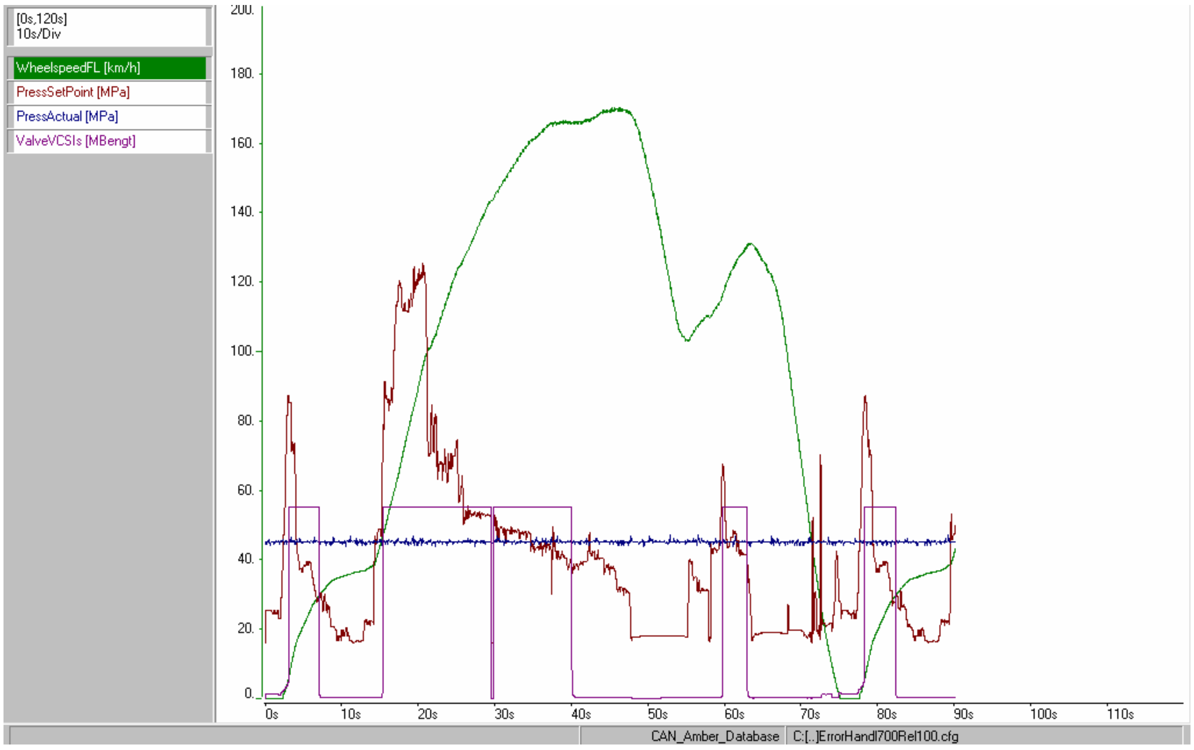


Figure 21: Test case 2 - Real-Time Workshop Embedded Coder code, actual pressure = 1.2MPa

It is clear that in all stages the controller works as it should. It sends a signal to open the valve when the pressure set points is above the actual value so that the actual pressure will follow the set points. The actual pressure is not changed during this test because it is a predefined value that the user sets for testing only. There can be some inconsistency between the graphs because of the manual errors when defining the actual pressure.

Test case 3:

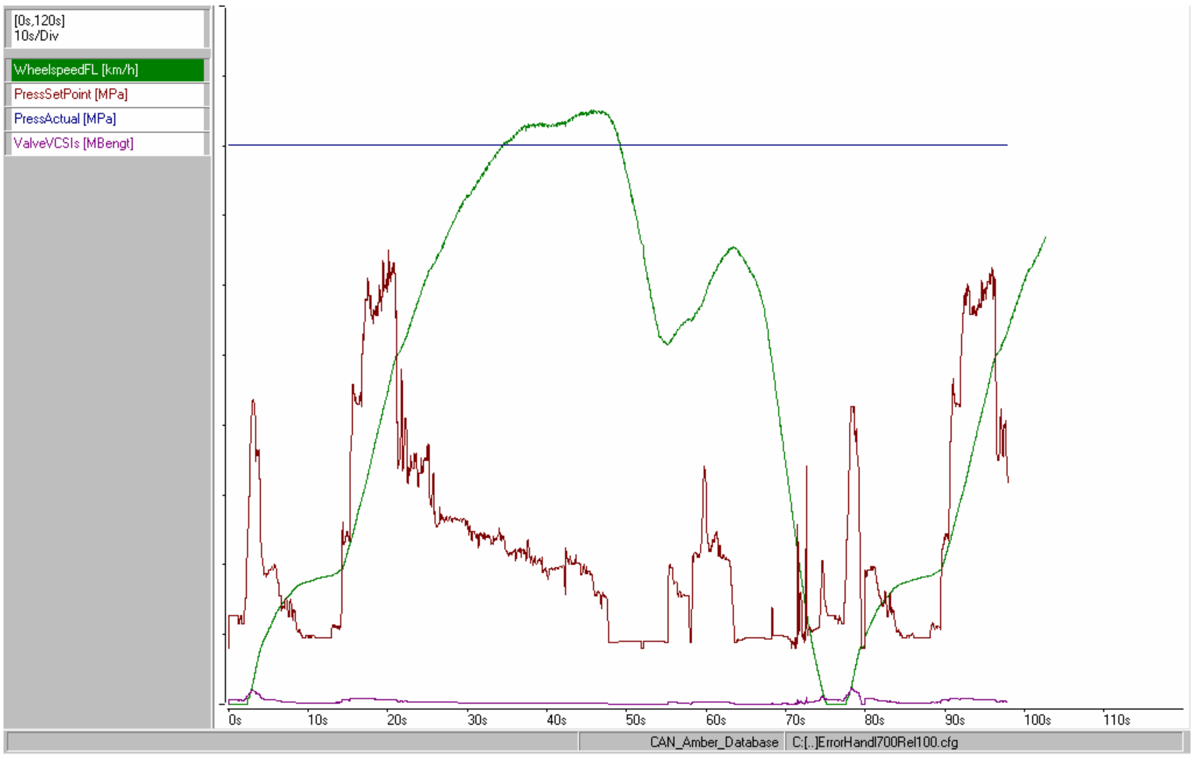


Figure 22: Test case 3 - handwritten code, actual pressure =4MPa

Both TargetLink generated code and RTW EC generated code act exactly in the same way as the handwritten code in the third test case. Here the actual pressure is 4MPa which means that the controller signal will not be used but another special signal is forced on the system and the controller signal is ignored, which is what happens during the testing.

9 Conclusions

This work was aimed at comparing two automatic code generating tools and find which of these tools is more suitable for Haldex Traction. Many aspects were tested and the results were evaluated.

TargetLink and Real-Time Workshop Embedded Coder are two powerful tools for automatic code generation. Most functionalities can be achieved in both tools.

For first time users TargetLink can be found easier to manage because of the simple interface and the practical way in finding all relevant options. RTW EC can be easy to manage after some practice.

Real-Time Workshop Embedded Coder supports almost all blocks for code generation but not all blocks support fixed-point. In TargetLink many Simulink blocks are supported or have corresponding blocks in TargetLink but all these blocks support fixed-point. Both TargetLink and RTW EC support most relevant blocks for generating code for control systems if the blocks are used wisely.

The code generated from both tools give the same overall performance as the handwritten code when running on the target. However there are some differences in execution time and memory requirements where TargetLink gives a faster execution time and better results regarding memory usage.

Both TargetLink and RTW EC generate well documented code that follows the Simulink model flow which makes it easier for the user to understand the generated code.

TargetLink is used by Haldex Brake in Landskrona and RTW EC is used by Haldex Brake in UK. Both partners are happy with their choice but Haldex Traction has its own requirements and priorities that should be taken into consideration.

10 Further Studies

In this work only code generation has been evaluated and tested. However more studies can be done related to integration with a real target and the inclusion of code generation stage in the development process. Studies can also be made about validation and verification.

Parameter set issue can also be studied further with respect to the solution that both TargetLink and Real-Time Workshop Embedded Coder offer.

Reference

[1] Haldex Traction AB (2007). *Technical Information* [www]. Information from <http://www.haldex-traction.com> 19/03/07

[2] dSPACE (2005). *Production Code Generation Guide For TagretLink 2.1.*

[3] dSPACE (2005). *Advanced Practices Guide For TagretLink 2.1.*

[4] The MathWorks (2002-2006). *Real-Time Workshop Embedded Coder User's Guide.*

[5] Matlab/Simulink Help (2006).

[6] Fengco Real Time Control AB (2007). *Kurser* [www]. Information from <http://www.fengco.se/> 20/02/07

[7] Nilsson, Staffan (2004). *Linear Pressure Controller for HLSC Specification.* Reg.No 2002510. Haldex Traction Systems

[8] Jönsson, Bengt (2000). *PID Controller*. Reg.No 2001850. Haldex Traction Systems.

[9] Åström, Karl & Wittenmark, Björn (1997). *Computer-Controller Systems*. 3rd Edition, Prentice Hall.

[10] dSPACE (2007). *TargetLink 2.2.*

Appendix I: Notations

Symbols	**Description**	**Unit**
vcs	Valve characteristic signal	$\frac{\sqrt{Pa} \cdot s}{m^3}$
p	Clutch pressure	Pa
p_b	Base pressure	Pa
p_{sp}	Pressure set point	Pa
Q_v	Valve flow	$\frac{m^3}{s}$
Q_p	Pump flow	$\frac{m^3}{s}$
c	Elasticity	$\frac{Pa}{m^3}$
v	Controller output	$\frac{m^3}{s}$
u	Controller output after limitation	$\frac{m^3}{s}$
V	Accumulated volume pressurising the clutch	m^3
V_{sp}	Set point volume to pressurise the clutch	m^3
η	Set point response factor	-
K	Controller gain	$\frac{m^3}{Pa \cdot s}$
T_i	Controller integration time	s
T_d	Controller derivation time	s
T_{tr}	Controller tracking time	s
N	Controller derivation gain limitation	-

Abbreviations

HLSC	Haldex Limited Slip Coupling
RTW EC	Real-Time Workshop Embedded Coder
TL	TargetLink
MIL	Model In the Loop
SIL	Software In the Loop
PIL	Process In the Loop
TOM	TargetLink Optimisation Modules
TBS	TargetLink Base Suite
MISRA C	Motor Industry Software Reliability Association for C

Appendix II: TargetLink Blocks

TargetLink has its own blocksets and some Simulink supported blocks. All the blocks below are supported in code generation and fixed-point.

Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
	Subsystem	X	X
	Addfile	X	X
TargetLink Simulation Blocks	InPort	X	X
	OutPort	X	X
	Constant	X	X
	Sum	X	X
	Gain	X	X
	Product	X	X
	Logical Operator AND	X	X
	Rational Operator	X	X
	Fcn		
	Look-Up Table	X	X
	Look-Up Table (2D)	X	X
	PreLook-Up Index Search	X	X
	Interpolation (n-D) using PreLook-Up	X	X
	Assignment	X	X
	Saturation	X	X
	MinMax	X	X
	Abs	X	X
	Sign	X	X
	Rate Limiter	X	X
	Relay	X	X
	Trigonometric Function	X	X
	Math	X	X
	Unit Delay	X	X
	Discrete Transfer Fcn	X	X
	Discrete Filter	X	X
	FIR Filter	X	X
	Discrete-Time Integrator	X	X
	Discrete State-Space	X	X

	Data Store Write	X	X
	Data Store Memory	X	X
	Data Store Read	X	X
	Custom Code Block	X	X
	Merge	X	X
	Sink	X	X
	Bus Inport	X	X
	Bus Outport	X	X
	Switch	X	X
	Multiport Switch	X	X
	Unit Delay Reset Enabled	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Extras	D Latch	X	X
	D Flip-Flop	X	X
	S-R Flip-Flop	X	X
	Pre-processor IF	X	X
	Stateflow Logger	X	X
	Stateflow FcnCall Logger	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
RTOS Blocks	Task	X	X
	ISR	X	X
	Crtitical Section	X	X
	Schedule	X	X
	CounterAlarm	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Supported Simulink Blocks	Trigger	X	X
	Enable	X	X
	In1	X	X
	Out1	X	X
	Display	X	X
	Scope	X	X
	For Iterator Subsystem	X	X
	While Iterator Subsystem	X	X
	Function-Call Generator	X	X
	Ground	X	X
	Terminator	X	X
	To File	X	X

	To Workspace	X	X
	From	X	X
	Goto Tag Visibility	X	X
	Goto	X	X
	Data Type Conversion	X	X
	Model Info	X	X
	Configurable Subsystem	X	X
	Switch Case	X	X
	If	X	X
	Mux	X	X
	Demux	X	X
	Bus Selector	X	X
	Bus Creator	X	X
	Selector	X	X
	Zero-Order Hold	X	X
	Rate Transition	X	X
	Switch Case Action Subsystem	X	X
	If Action Subsystem	X	X
	Check Static Lower Bound	X	X
	Check Static upper Bound	X	X
	Check Static Range	X	X
	Check Static Gap	X	X
	Assertion	X	X
	Check Discrete Gradient	X	X
	Check Input Resolution	X	X
	Check Dynamic Lower Bound	X	X
	Check Dynamic Upper Bound	X	X
	Check Dynamic Range	X	X
	Check Dynamic Gap	X	X

Appendix III: Real-Time Workshop Embedded Coder Blocks

All blocks that can generate code and support fixed-point are marked with X under the corresponding column. Some blocks has caveats and/or notes that should be taken into consideration. Caveats and notes are indicated by “C#” and “N#”, respectively, and are described below the table.

Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Sources	Constant	X	X
	Counter Free Running	X	X (N3)
	Counter Limited	X	X (C4)
	Ground	X	X
	Import (In1)	X	X
	Repeating Sequence Interpolation	X	X (C1, C4)
	Repeating Sequence Stair	X	X (C4)
Sink	Display	X	X (N1)
	Floating Scope	X	X (N1)
	Outport (Out1)	X	X
	Scope	X	X (N1)
	Terminator	X	X
	To Workspace	X	X (N1)
	XY Graph	X	X (N1)
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Discrete	Discrete-Time Integrator	X	X (C2, N2)
	Integer Delay	X	X (N2)
	Discrete Tapped Delay	X	X (N2)
	Discrete Derivative	X	X (C2, N2)
	Difference	X	X (C4)
	Transfer Fcn Lead or Lag	X	X (C4)
	Transfer Fcn First Order	X	X (C4)
	Transfer Fcn Real Zero	X	X (C4)
	Weighted Moving Average	X	X
	Memory	X	X
	Unit Delay	X	X (N2)
	Zero-Order Hold	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**

Math Operations	Abs	X	X
	Assignment	X	X (N2)
	Complex to Real-Imag	X	X
	Dot Product	X	X
	Gain	X	X
	Math Function (magnitude^2)	X	X
	Math Function (square)	X	X
	Math Function (conjugate)	X	X
	Math Function (reciprocal)	X (C3)	X
	Math Function (transpose)	X	X
	Math Function (hermitian)	X	X
	Matrix Concatenation	X	X (N2)
	MinMax	X	X
	MinMax Running Resettable	X	X
	Weighted Sample Time Math	X	X
	Bias	X	X
	Unary Minus	X	X
	Product	X	X (N2)
	Real-Imag to Complex	X	X
	Reshape	X	X
	Sign	X	X
	Slider Gain	X	X
	Sum	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Logic and Bit Operations	Logical Operator	X	X
	Relational Operator	X	X
	Interval Test	X	X
	Interval Test Dynamic	X	X
	Compare to Zero	X	X
	Compare To Constant	X	X
	Bit Set	X	X
	Bit Clear	X	X
	Bitwise Operator	X	X
	Shift Arithmetic	X	X
	Extract Bits	X	X
	Detect Increase	X	X (N2)
	Detect Decrease	X	X (N2)
	Detect Change	X	X (N2)
	Detect Rise Positive	X	X (N2)
	Detect Rise Nonnegative	X	X (N2)
	Detect Fall Negative	X	X (N2)
	Detect Fall Nonpositive	X	X (N2)

Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Signal Routing	Bus Creator	X	X
	Bus selector	X	X
	Bus Assignment	X	X
	Data Store Read	X	X
	Data Store Memory	X	X
	Index Vector	X	X
	Data Store Write	X	X
	Demux	X	X
	From	X	X
	Goto	X	X
	Environment Control	X	X
	Goto Tag Visibility	X	X
	Manual Switch	X	X (N3)
	Merge	X	X
	Multiport Switch	X	X (N2)
	Mux	X	X
	Selector	X	X
	Switch	X	X (N2)
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Signal Attributes	Data Type Conversion	X	X
	Probe	X	X
	Weighted Sample Time	X	X
	Signal Conversion	X	X
	Data Type Duplicate	X	X
	Data Type Propagation	X	X
	Data Type Conversion Inherited	X	X
	Data Type Scaling Strip	X	X
	Rate Transition	X	X (C1, N2)
	Signal Specification x	X	
	Width	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Discontinuities	Coulomb & Viscous Friction	X	X (C4)
	Dead Zero	X	X
	Dead Zero Dynamic	X	X (C4)
	Rate Limiter	X	X (C1)
	Rate Limiter Dynamic	X	X (C1, C4)
	Relay	X	X
	Saturation	X	X

	Saturation Dynamic	X	X (C4)
	Wrap To Zero	X	X (C5)
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Lookup Tables	Lookup Table	X	X
	Lookup Table Dynamic	X	X
	Sine	X	X (C4)
	Cosine	X	X (C4)
	Lookup Table (2-D)	X	X
User-Defined Functions	S-Function	X	X (N4)
	S-Function Builder	X	X
Model Verification	Assertion	X	X
	Check Discrete Gradient	X	X
	Check Dynamic Gap	X	X
	Check Dynamic Lower Bound	X	X
	Check Dynamic Range	X	X
	Check Dynamic Upper Bound	X	X
	Check Static Gap	X	X
	Check Static Lower Bound	X	X
	Check Static Range	X	X
	Check Static Upper Bound	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Ports & Subsystems	Atomic Subsystem	X	X
	Configurable Subsystem	X	X
	CodeReuseSubsystem	X	X
	Enabled Subsystem	X	X
	Enabled and Triggered Subsystem	X	X
	For Iterator Subsystem	X	X
	Model	X	X
	Function-Call Subsystem	X	X
	If Action Subsystem	X	X
	Switch Case Action Subsystem	X	X
	Subsystem	X	X
	Triggered Subsystem	X	X
	While Iterator Subsystem	X	X
Sublibrary	**Block**	**Fixed-Point**	**Code**

		Support	Generation Support
Increment/ Decrement	Increment Real World	X	X (C4)
	Decrement real World	X	X (C4)
	Increment Stored Integer	X	X (C4)
	Decrement Stored Integer	X	X (C4)
	Decrement to Zero	X	X (C4)
	Decrement Time To Zero	X	X
Sublibrary	**Block**	**Fixed-Point Support**	**Code Generation Support**
Additional Discrete	Transfer Fcn Direct From ||	X	X (C4, N2)
	Transfer Fcn Direct From || Time Varying	X	X (C4, N2)
	Fixed-Point State-Space	X	X (C4)
	Unit Delay External IC	X	X (C4)
	Unit Delay Resettable	X	X (C4, N2)
	Unit Delay Resettable External IC	X	X (C4, N2)
	Unit Delay Enabled	X	X (C4, N2)
	Unit Delay Enabled External IC	X	X (C4, N2)
	Unit Delay Enabled Resettable	X	X (C4, N2)
	Unit Delay Enabled Resettable External IC	X	X (C4, N2)
	Unit Delay With Preview Enabled	X	X (C4, N2)
	Unit Delay With Preview Enabled Resettable	X	X (C4, N2)
	Unit Delay With Preview Enabled Resettable External RV	X	X (C4, N2)
	Unit Delay With Preview Resettable	X	X (C4, N2)
	Unit Delay With Preview Resettable External RV	X	X (C4, N2)

Caveats:
C1: Cannot be used inside a triggered subsystem hierarchy.
C2: Depends on absolute time when placed inside a triggered subsystem hierarchy.
C3: Currently, the reciprocal supports fixed-point data types only for real inputs.
C4: The primitive blocks that constitute a non-atomic masked subsystem (block) are not explicitly grouped together in the generated code. This flexibility allows

for more optimal code generation. In certain cases, grouping can be achieved by configuring the masked subsystem block to execute as an atomic unit (Subsystem parameter “Treat as atomic unit”).

Notes:

N1: Ignored for code generation.

N2: Generated code relies on memcpy or memset (string.h) under certain conditions.

N3: Not recommended for production code.

N4: M-file S-functions are not supported for the real-time code generation format. S-functions that make calls into MATLAB are not supported for production code.

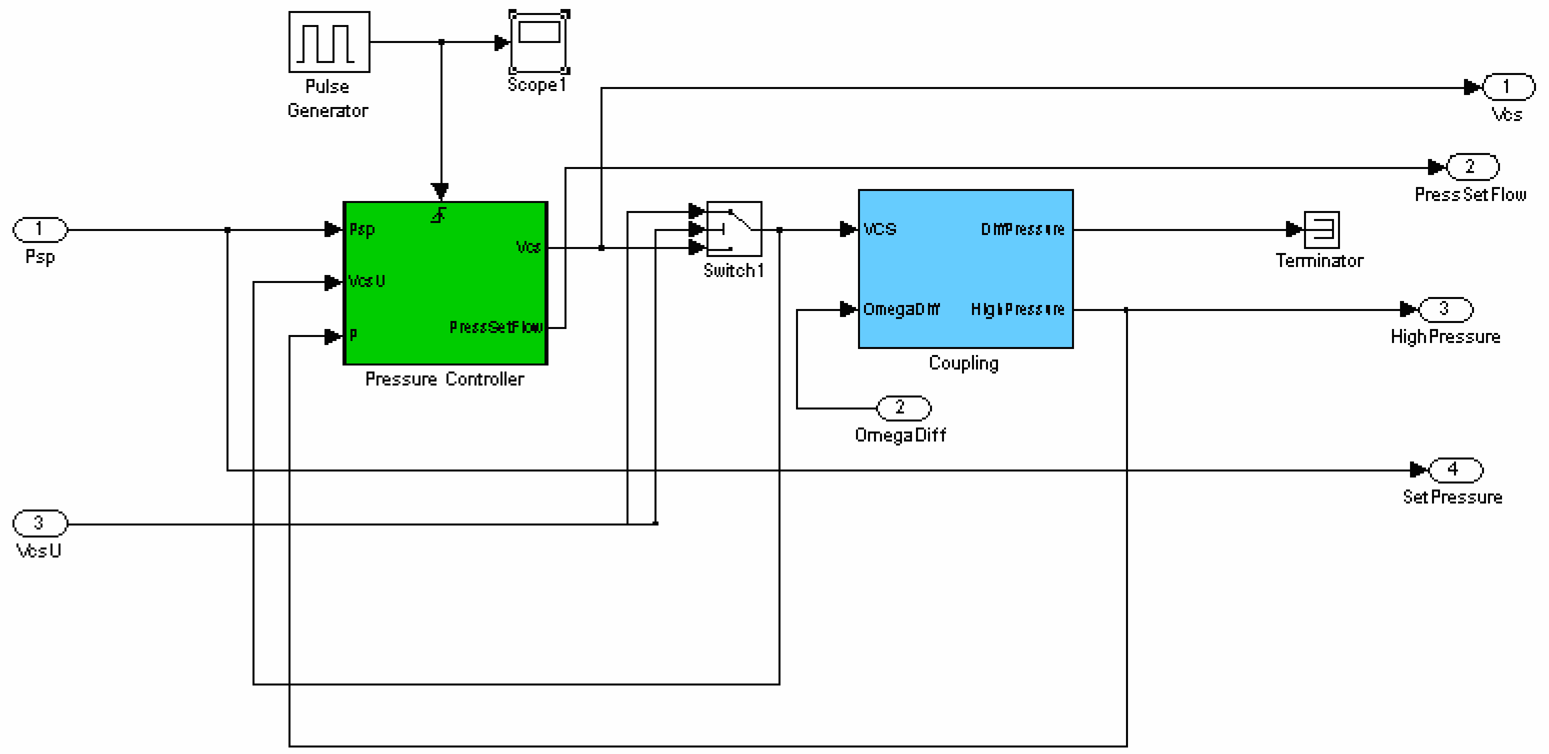
Appendix IV: Pressure Controller – The Design

The tested model is a design of a pressure controller that is used in Haldex Traction base software. The coupling that the pressure controller is designed for is a Haldex coupling generation 2. The model contains a rough approximation of the coupling for test reasons, this subsystem will not be discussed here. The design will not be discussed further here due to company requirements.

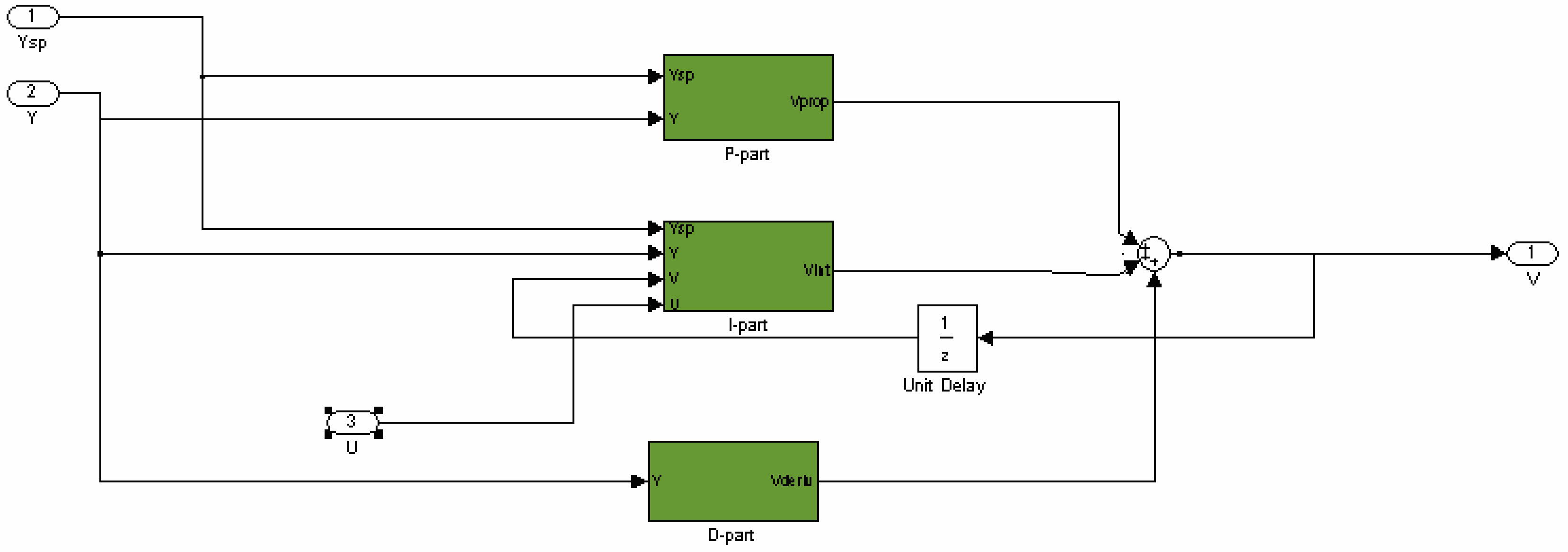
Appendix V: Simulink Model

The PID controller described in Appendix IV is easy to build in Simulink. Every part of the controller is built in a separate subsystem. The coupling subsystem is an approximate design for the actual coupling used in the car, Figure 1. The transform subsystem is used to transform the pressure to VCS signal and vice versa. The subsystem is also used to track quantization and saturation errors. The Quantisation block in the Transform subsystem uses an S-function that in Simulink just let the calculated signal VCS through while in the software and when the code is generated a function call is used instead to return the VCS signal value after being quantised throughout the main program.

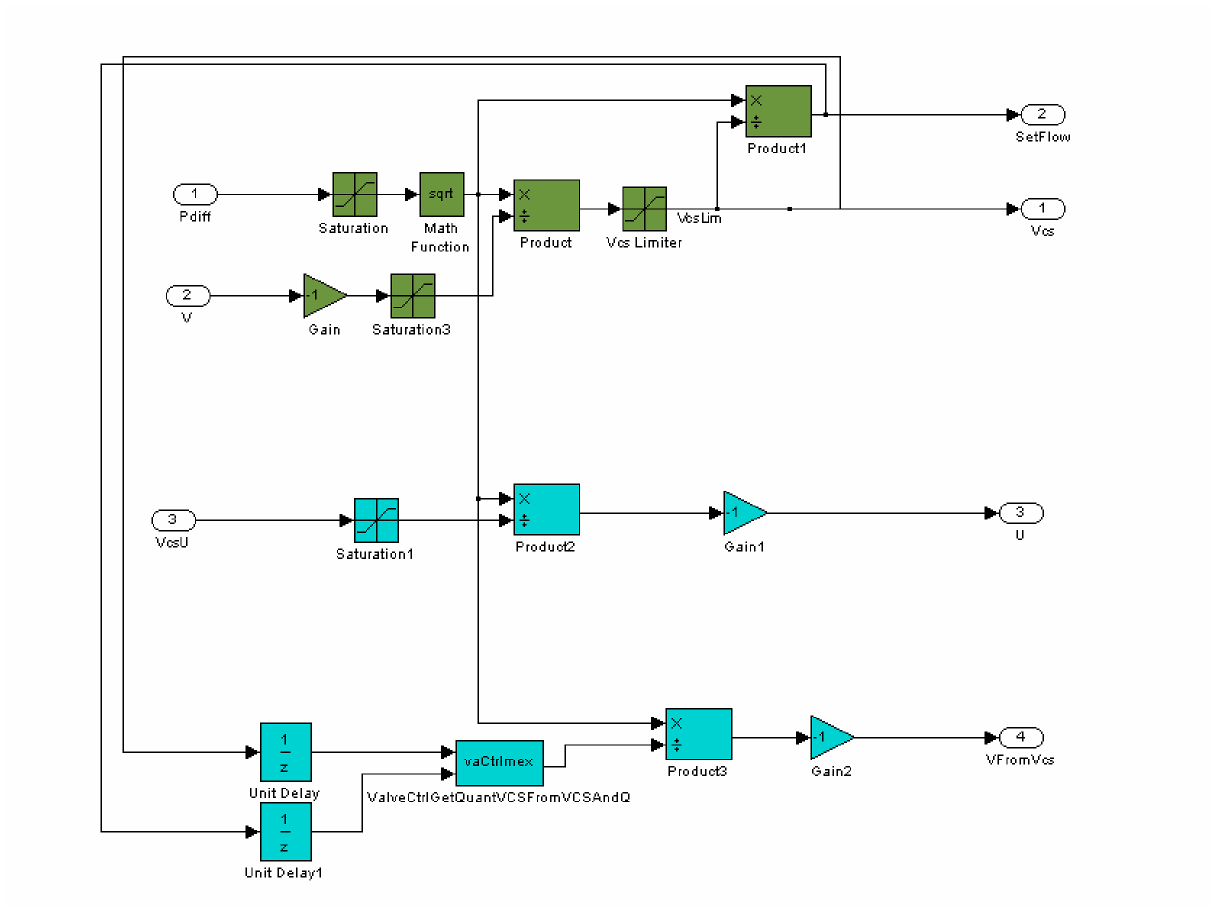
1. Pressure Controller With Coupling



3. PID Controller



Transfrom


1
Pdiff
Saturation
sqrt
Math Function
Product
Vcs Limiter
VcsLim
Product1
2
SetFlow
1
Vcs
2
V
-1
Gain
Saturation3
3
VcsU
Saturation1
Product2
-1
Gain1
3
U
1/z
Unit Delay
1/z
Unit Delay1
vaCtrlmex
ValveCtrlGetQuantVCSFromVCSAndQ
Product3
-1
Gain2
4
VFromVcs