ISSN 0280-5316
ISRN LUTFD2/TFRT--5692--SE

Rapid Prototyping with Matlab/Simulink -A case study

Håkan Nilsson

Department of Automatic Control
Lund Institute of Technology
August 2002

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
August 2002

Document Number
ISRN LUTFD2/TFRT--5692--SE

Author(s)
Håkan Nilsson

Supervisor
Anders Wallenborg, Gambro
Rolf Johansson, LTH

Sponsoring organization

Title and subtitle
Rapid prototyping with Matlab/Simulink. –A case study (Prototyputveckling med Matlab/Simulink som utvecklingsverktyg)

Abstract
In order to minimise development time it is important to be able to implement and test control functions at an early stage in a project, even before the electronic hardware of the new product is available. For this purpose, a PC based rapid prototyping system including a graphical modelling/simulation tool and automatic C code generation for real-time simulations with hardware in the loop may be used.
In this thesis an evaluation of Matlab/Simulink as a tool for rapid prototyping of control functions in dialysis machines has been done. A comparison has also been done to the tool that is in use today at Gambro Lundia AB, MatrixX/SystemBuild.
The sections that have been investigated are block diagram modelling including translation from SystemBuild block diagram, state machine implementation, Graphical User Interface, data acquisition, simulation with hardware-in-the-loop, and code generation.
The Matlab/Simulink tool has been tested on a hydraulic system prototype during the evaluation and advantages and disadvantages have been noted.
The report gives an overview of how Matlab/Simulink meets the requirements that can be expected of a rapid prototyping tool, for control functions and design tips to avoid problems during the development phase.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
43

Recipient's notes

Security classification

Rapid prototyping with Matlab/Simulink

- A case study

Master's Thesis

Håkan Nilsson

Department of Automatic Control
Lund Institute of Technology
August 2002

Sammanfattning

För att korta utvecklingstiden i ett projekt så är det viktigt att kunna implementera och utvärdera reglerfunktionerna i ett tidigt skede medan övrig hård- och mjukvara fortfarande är under utveckling. För detta ändamål kan ett PC-baserat grafiskt modellerings och simuleringsverktyg med C-kodsgenerering för inbyggda system användas.

I denna rapport utvärderas Matlab/Simulink för detta användningsområde. En jämförelse görs med MarixX/Systembuild som är ett motsvarande verktyg och som används idag hos Gambro Lundia AB.

De delar som utvärderats är modelleringsverktyget för blockschemadesign inklusive översättning av SystemBuild till Simulink modell, användande av tillståndsmaskin i blockschemamodellen, grafiskt användargränssnitt, dataloggning, simulering av regler funktionerna med hårdvara som återkoppling samt kodgenerering.

Implementering av en reglermodell m.h.a. Matlab/Simulink har testats på en prototyp av en dialysmaskins vätskesystem och för och nackdelar har noterats.

Rapporten ger svar på hur Matlab/Simulink motsvarar kraven som kan ställas på ett prototypverktyg för reglertillämpningar och designtips för att undvika problem under utvecklingsfasen.

Acknowledgements

First of all I like to thank my supervisors prof. Rolf Johansson at the Department of Automatic Control, Lund Institute of Technology, and tech. lic. Anders Wallenborg at Gambro Lundia AB for valuable comments and help during the master's project.

I would also like to thank Johan Severson and Magnus Ericsson at Gambro Lundia AB for the help during simulation and code generation, and Hossein Mousavi at Comsol AB who has helped me with Matlab/Simulink support during the work.

Contents

1 Introduction

An important function in a dialysis machine is the preparation of the dialysis fluid, a saline solution produced from liquid concentrates by continuous dilution with purified water. The water entering the dialysis machine is heated to a temperature between 36°C and 40°C before it is mixed with the dialysis concentrate. The incoming water contains large amounts of dissolved air that must be removed. In the degassing system, the fluid is exposed to a large negative pressure by using a flow pump to pull the flow through a restriction in the flow path. The released air is then removed in a degassing chamber ("bubble trap") after the flow pump.

In order to minimise development time it is important to be able to implement and test control functions at an early stage in a project, even before the electronic hardware of the new product is available. For this purpose, a PC based rapid prototyping system including a graphical modelling/simulation tool and automatic C code generation for real-time simulations with hardware in the loop may be used.

1.1 Background and purpose

At Gambro, a system called MatrixX/SystemBuild is already in use. MatrixX/Systembuild was originally developed by Integrated Systems, Inc. in USA. After repeated changes in ownership, further development of MatrixX/SystemBuild has now been discontinued. One of the biggest competitors is Matlab/Simulink and therefore is it natural to make an investigation and find out the advantages/disadvantages of the Matlab/Simulink software tool as an alternative to MatrixX/SystemBuild.

The purpose of this master's project is to test and evaluate the feasibility of using the Matlab/Simulink Real-time Workshop and Embedded Coder as a rapid prototyping and code generation tool.

1.2 Outline of the report

The outline of the report follows the development process and includes a description of the physical process prototype, design and implementation of control loops, building a Graphical User Interface, simulation with data acquisition and code generation of the final system. In each section a comparison to MatrixX/SystemBuild is made and at the end of the report a summary of the Matlab/Simulink software tool is given.

The reader should have some knowledge of how Matlab/Simulink or MatrixX/SystemBuild works, for a better understanding.

2 The Physical system

2.1 The Hydraulic prototype

In Figure 1 the physical process can be seen. The degassing system exposes the fluid to a large negative pressure by using a flow pump, Pu_{Dg}, to pull the flow through a restriction in the flow path. The released air is then removed in a bubble trap, BT_{Dg}, after the flow pump. A heater loop is running to maintain the correct temperature, T_{Dg}, and a second flow pump, Pu_o, controls the flow through the fluid path. In the bubble trap chamber, a level detector is mounted that detects air. If air is detected the V_{BTDg} is opened until the air is removed.

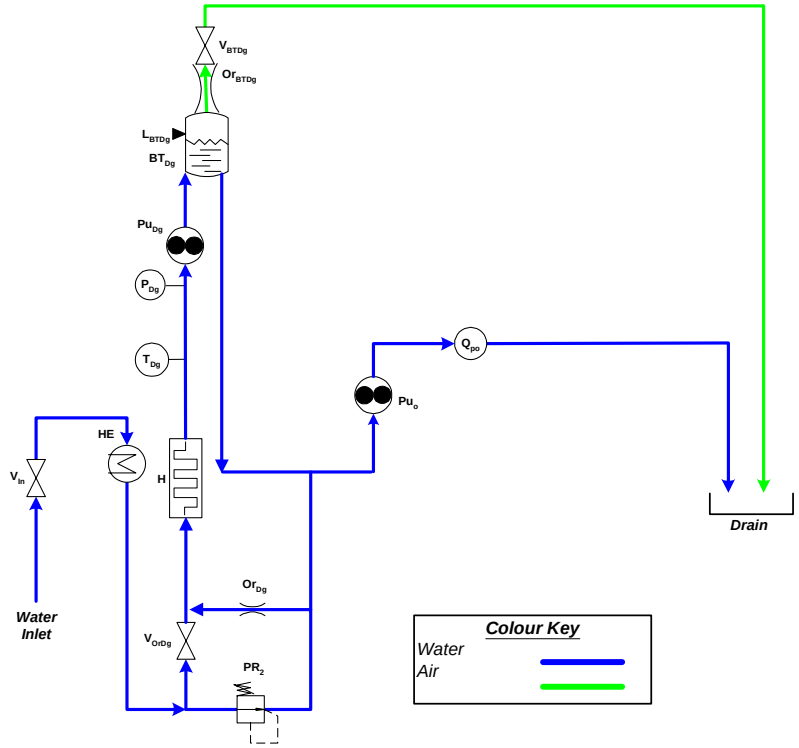

Figure 1 A schematic view of the hydraulic system prototype.

2.2 Connection between control model and prototype

In Figure 2 an overview of the prototype system can be viewed. At the host PC the Matlab/Simulink software is running. Through the Ethernet a connection is established with the target PC and in the target PC the I/O cards are mounted and wired to the hydraulic system prototype.

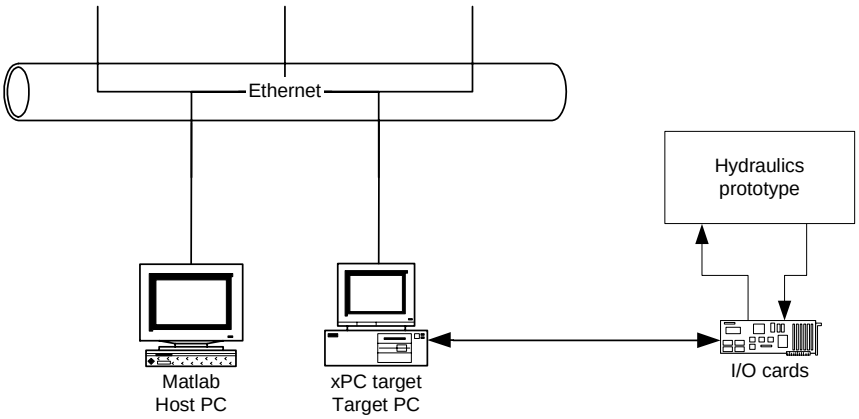

Figure 2 An overview of the rapid prototyping system.

The MatrixX/SystemBuild configuration is similar. The only difference is that MatrixX/SystemBuild uses an AC104, a proprietary PC type computer, as target system instead of a standard desktop PC with standard I/O cards.

3 Conversion from SystemBuild to Simulink

From the existing SystemBuild model an equivalent Simulink model should be made. The translation of the existing model to a Simulink model can be done in two ways:

1. With a translation tool, SB2SL (SystemBuild to Simulink), that Matlab provides.
2. Conversion by hand, block-by-block.

Important questions to be answered are:

1. Which way is the best, do it by hand, with a translation tool, or a mix of the both? Time consumption?
2. Can the translation tool translate all SystemBuild blocks?
3. Is it necessary to do some cleaning up in the block diagram after the translation with the tool is finished?
4. How should variables be managed? How should SystemBuild's partitions be translated?
5. How should the assigning of sample intervals to sub blocks be done?
6. Naming conflicts?
7. How does the translation tool handle a sub block that is used in several sub systems?

3.1 Automatic translation using SB2SL

SB2SL is a translation tool that translates a SystemBuild model to an equivalent Simulink model. SB2SL translation is performed on a block-by-block basis. Except for a few blocks, all SystemBuild blocks are translated into either a Simulink counterpart or a masked subsystem[1] block containing the computational equivalent if no Simulink counterpart exists.

A few blocks are not translated, but appropriate blank placeholder blocks are created in the resulting Simulink model. The User Code Block and the State Transition Diagram block in SystemBuild are examples of blocks that are not translated. For a detailed list of the limitations according to the translation tool, see [1].

After the translation it is necessary to perform clean up, optimisation and validation of the model.

3.1.1 Configuration of the translation tool

Before a translation some translation and report generation options must be enabled/disabled. The options pane has three different categories, Build, Report Layout and Report Format. Below is a summary of the options that are of interest for our application.

[1] A masked subsystem in Simulink is a way to customise and simplify the use of the model by replacing many dialog boxes in a subsystem with a single one. It is also possible to create a more descriptive and helpful user interface for the subsystem. The masking also prevents unintended modifications in the subsystem by hiding their contents.

Build

1. Fit System to view.
   Select this option to scale the model to fit the window size or deselect if using the original size (default=on).
2. Add terminator and Ground blocks.
   With this option on the unconnected block inputs and outputs are terminated with Simulink Terminator or ground blocks (default=on).
3. Avoid block crossing
   Selecting this option minimises signal lines crossing blocks in the Simulink model resulting from SB2SL translation (default=on).
4. Optimise the resulting model
   With this option on, SB2SL maximises the use of standard Simulink blocks when translating the following SystemBuild blocks (default=on):
   - Data store blocks
   - Algebraic/logical expression blocks
   - Integrator blocks

Report Layout

1. Catalogue of SuperBlocks (default=on)
2. Detailed SuperBlock information (default=off)
3. Partitions and parameters (default=off)
4. List of BlockScript blocks (default=off)
5. List of unconverted blocks (default=on)

Good choices are also to include partitions and parameters, detailed SuperBlock information and a list of BlockScript blocks in the report.

Report Format

1. Output format.
   HTML, RTF or TeX (default=HTML)
2. Style (default=single page)
   - Single or multipage output for the HTML format
   - Standard, simple, or large type print for the RTF or TeX formats
3. View report after conversion (default=on).
   The report will be displayed after the report is created.

For more details, see [1].

3.1.2 Translation with results

A translation was made of the whole control system. This gives the possibility to see differences to the SystemBuild model, how parameters, sample time, variables and partitions are handled, and the optimisation of the translated model.

When the translation was started the first thing to notice was the translation time. The host computer was a PC with Windows NT4 running on an Intel Pentium 1 GHz CPU and the translation of the model takes three to four minutes to perform.

To get an overview of the translation, the translation report is a good start. The translation report says:

1. The partitions were translated to data structures with same name and the variables have same name and type.
2. All Blocks were converted except the User Code blocks, State Transition Diagram blocks and IF-THEN-ELSE blocks. This was the expected result.
3. Signals and block names are the same.
4. The block structure is the same.

The list of missing blocks in the translation report makes it easy to see where the blocks are and of which type. Missing blocks are replaced with blank placeholder blocks with appropriate number of inputs and outputs.

The Simulink block library contains a SystemBuild block library and during the translation the translator finds a SystemBuild block and replaces it with a Simulink implementation of the block. This type of translation makes the model look like SystemBuild and some rebuilding must be done to get an efficient model for simulation with Simulink. For example, an Algebraic Expression block in SystemBuild can be used in many different ways. When it is converted by SB2SL to a masked Simulink block it can be a bad implementation depending on how the Algebraic Expression block is used. If the Algebraic Expression Block is used to implement a constant, the translation gives a masked Simulink block with a Constant block inside. In this case the natural way to implement would be using only the block Constant.

At subsystem level, i.e. the different control loops, the translation gives a good result, see Figure 3 and Figure 4 for an example. Things to notice are that the block structure layout is preserved, BlockScript[2] blocks are translated to Simulink S-functions and the parameters are stored in a Matlab data structure that corresponds to the partition for the SuperBlock in SystemBuild.

When looking at the top level there are major differences. In Figure 5 and Figure 6 the differences are obvious. The Simulink model has a poorly structured layout and it will take hours to rearrange all blocks to a user-friendly layout. The option “avoid block crossing” seems to work badly when the system has many blocks and connection lines.

A subsystem in SystemBuild runs with the assigned sample time in the SuperBlock. After translation to a Simulink model the sample time is assigned only in the Simulink blocks that needs this for their functionality, i.e. delay blocks, integrator blocks, and filter implementations. This raises the discussion of how Simulink handles sample times. In section 3.3 this is discussed in more detail.

The translator does not find subsystems of which there are multiple instances in the SystemBuild model, so a clean up and optimisation must be done after the translation is finished. An estimation of the time to do this clean up and optimisation of the model is about 2 weeks, not including the following testing and validation of the translated model.

The tool also gives the possibility to translate parts of a model. In manual translation this can be used as a starting point, the subsystems are translated by SB2SL and the top level is translated manually.

[2] BlockScript is a block type in SystemBuild that makes it possible to write small pieces of code in a textual “Block script ” language instead of building with blocks.

In the SB2SL Using Guide, [1], a complete description of the translation procedure can be found.

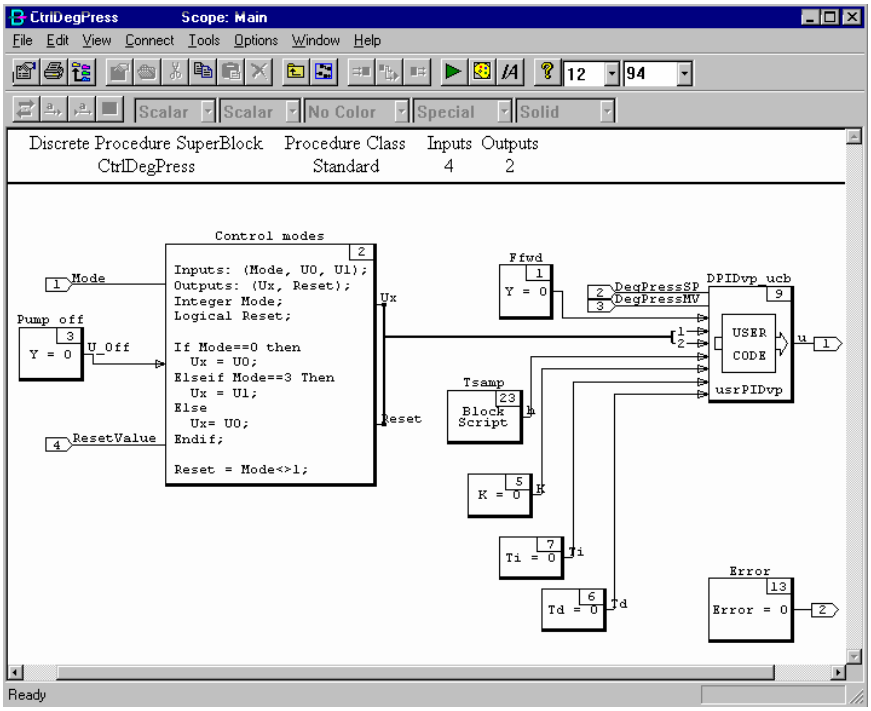

Figure 3 The degassing pressure controller in SystemBuild.

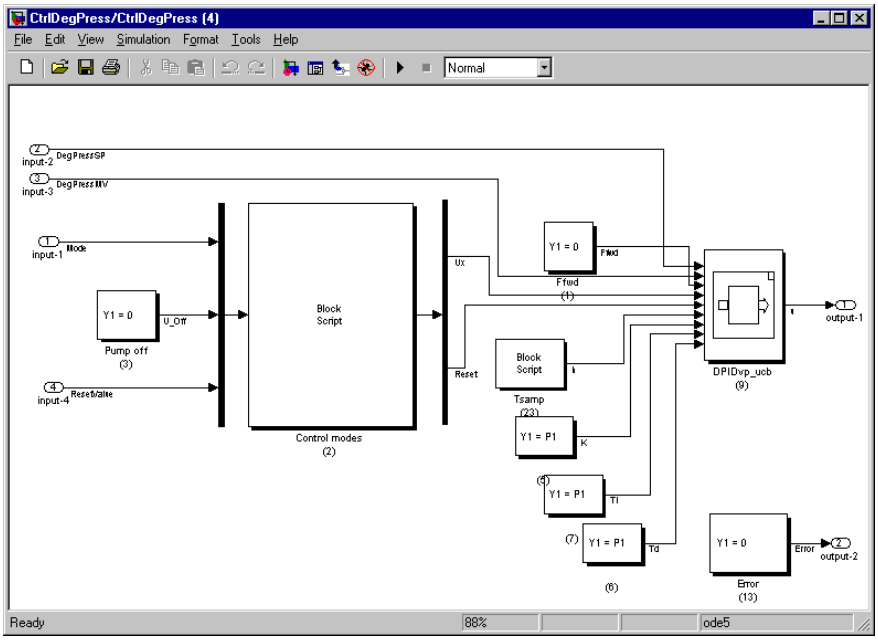

Figure 4 The translated degassing pressure controller. The similarities with the SystemBuild block in figure 3 can be seen. The DPIDvp_ucb is not implemented and must be done afterwards. The Control modes block is implemented as a S-function.

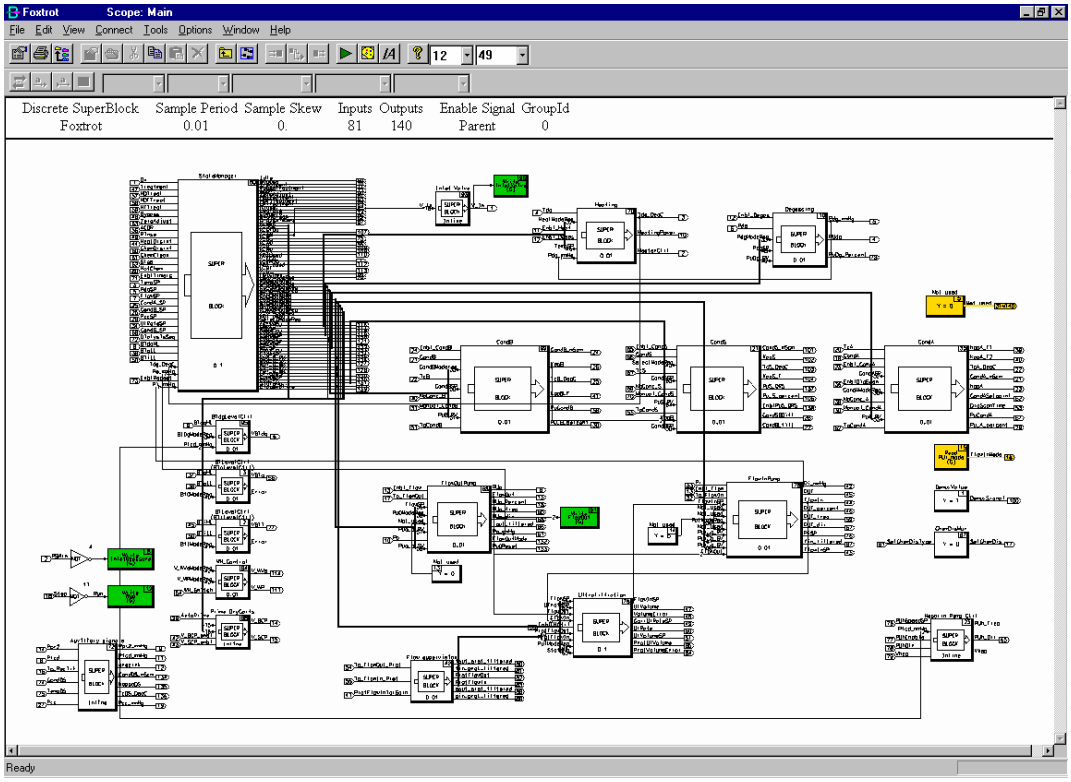

Figure 5 The top-level view of the complete system in SystemBuild.

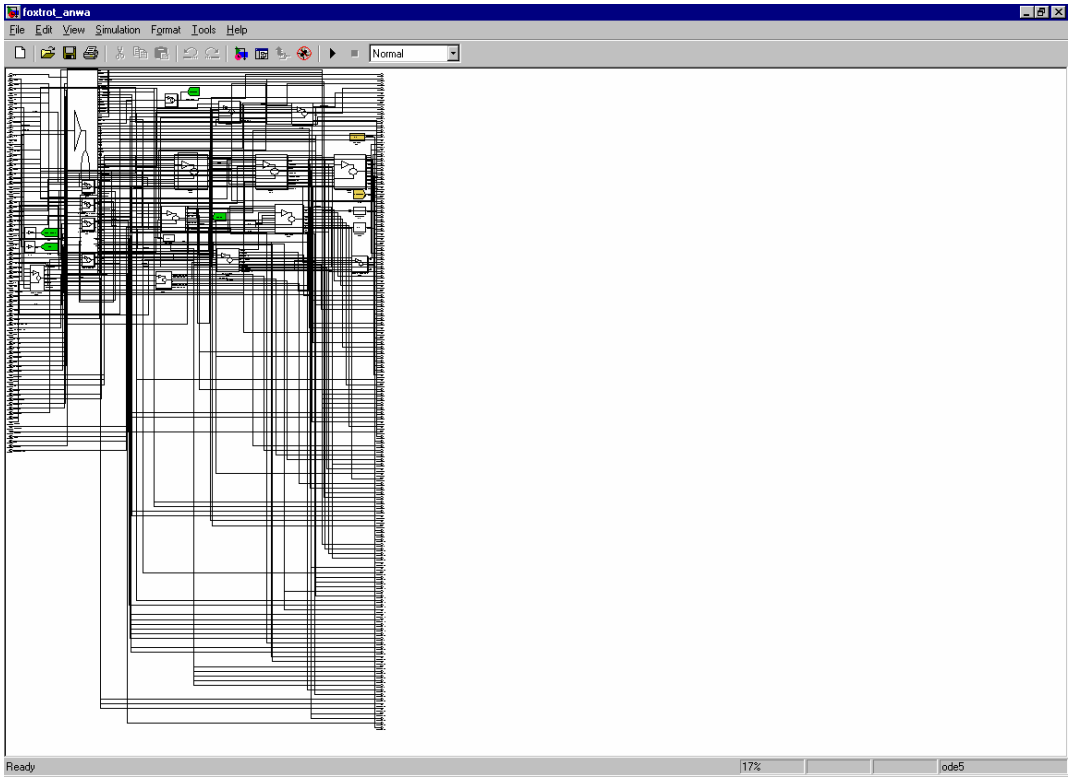

Figure 6 The top-level view of the translated system in Figure 5.

3.2 Manual translation

The translation tool, SB2SL, translates the model but the model is not optimised as a Simulink model and some of the blocks are not translated. The PID algorithm that is a User Code Block, the State Machine and the IF-THEN-ELSE blocks in the SystemBuild model must be translated by hand to Simulink blocks.

To get a feeling for how long it would take to translate the SystemBuild model by hand to a Simulink model, a part of the total model was translated. The reason to only make a partial translation is that when the PID algorithm is translated and the structure of the State Machine is translated, the other logic is not a large part and does not bring much more information to the evaluation. The size of the translated model is that three of eight control loops are translated, and four states of nearly a hundred in the state machine are implemented.

When translating the SystemBuild model, the model is opened up and a Simulink model is made with the same logical structure. MatrixX partitions can be translated into Matlab data structures. The partitions are stored in a MatrixX script and the contents of this script can be copied to the Matlab script for easy setting of the data structures. This speeds up the translation process and there will be no missing information.

The PID algorithm is implemented as a block structure and saved as a library block. Subsystems of which there are multiple instances should be implemented and saved as a library block.

After this the surrounding logic can be translated quite fast. The state machine has matrices that can be reused from MatrixX and the Stateflow layout is a manual copy from SystemBuild's State Transition Diagram, see section 4 for more details about state machine implementation. In Figure 7 the top level view of the handmade model can be seen and in the left pane, the Model Browser, the tree structure of the model is shown.

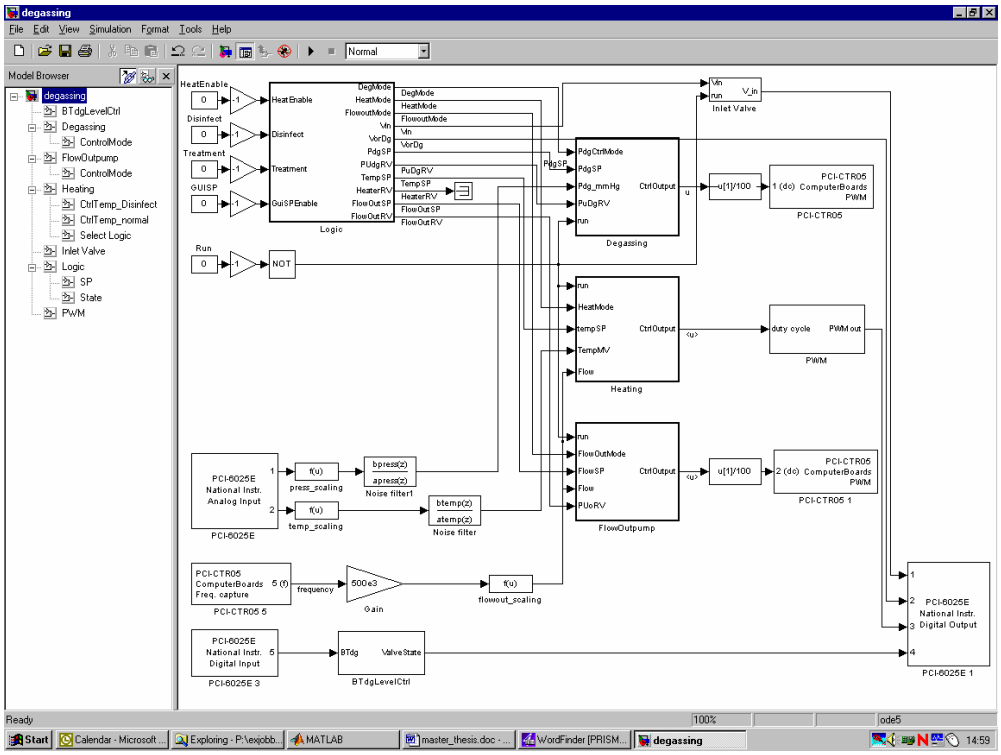

Figure 7 An overview of the control model that is implemented in Simulink. The Logic block contains a state machine. The control loops are implemented in Degassing, Heating, FlowOutpump and BTdgLevelCtrl blocks.

An estimation of the translating time for the whole system is 2 - 3 weeks, not including the following testing and validation of the translated model.

3.3 Important differences between Matlab/Simulink and MatrixX/SystemBuild

In many parts the two programs works in the same way. However there are differences between the two software tools that need to be considered when translating a model. These are presented in the following list:

Partitions and name scoping

A partition in MatrixX is analogous to a directory or folder. It is used to store variables and organise data. The variable names in one partition can be the same as those in another partition, even though they represent totally different pieces of data. Partitions are not hierarchical and therefore one partition cannot contain another but there are no limitations in the number of partitions. In Matlab this concept does not exists. The most equivalent way to handle this problem is to create a data structure containing the variables for a partition. When a SuperBlock has a reference to a partition, all children of the SuperBlock can use the variables in the specified partition. In Simulink all variables are accessible everywhere and therefore all variables must be referenced with full name, e.g. the gain value of the degassing controller is written like DgsCtrl.K, where DgsCtrl is the name of the data structure containing all degassing variables. This handling of the variables seems to work well and do not raise any problems.

Assignment of sampling intervals

Assigning a sampling rate to a sub system has different implementations in the two programs. In SystemBuild the assignment of the sample rate is made for the SuperBlock and all children blocks inherit the sampling rate. For Simulink the sample rate is inherited from the previous block, i.e. the sample time of each block is inherited from the block producing the input signal to that block. To ensure that a sub system is running with a specific sample rate, Sample & Hold blocks with the specified sample rate must be placed on all incoming signals to the sub system. Simulink has a useful option that shows the sample time of different blocks and signals with a colour code.

Representation of parameters and signals

All connections between the SystemBuild model and the target application are handled with an interface for outgoing and incoming signals. This interface is on the top level and as long there are no changes in the number of outputs and inputs it is possible to make changes in the model without any changes in the interface.
In Simulink all parameters and signals are referenced by a number in a parameter/signal list. If a block is added, removed or moved to a new position in the model this change will affect the individual numbering in the parameter/signal list. After every change scripts that handle parameters or signals must be updated with the new numbering. This representation creates a lot of extra work during a development process when the model changes layout and structure frequently so it is a serious drawback.

Naming convention

Simulink’s way of changing the parameter and signal list makes it necessary to use a good

naming convention of the blocks. Even if a subsystem is masked, the output signal from the subsystem is referenced by name to the signal-producing block in the subsystem. An example, the block that calculates the output of the PID controller is placed at third sub level from the top level in the subsystem and is preferably named Control_Output or Output. This name should appear in the signal list that is used for data acquisition.

It is also possible to add a label to a signal. This can be done for increased readability of the Simulink model but the label gives no extra readability in the signal list.

In the naming of sub blocks special precautions must be taken if you are planning to use code generation, see section 8 for details.

Using S-function builder in Simulink

SystemBuild has a block type called Block Script that makes it possible to write small pieces of code in a textual “BlockScript” language instead of building it with blocks. A similar block in Simulink is called S-function builder block. If a copy of an S-function Builder block is done, both the copy and the original block use the same s-function file. If a change is done in one of the block it will affect both blocks. If code generation shall be used special precautions will be necessary, see section 8.1.

Environment variables in SystemBuild translated to parameters in Simulink

SystemBuild and AutoCode environment provides predefined system variables that can be imported into the Block-Script code through an environment list. These variables are implemented as parameters in the Simulink model.

3.4 Summary

If you should use a manual, or automatic translation, or a mix of both translation techniques depends on the size and type of blocks in the SystemBuild model.

If the model has a subsystem of which there are multiple instances a manual translation is preferred. If an automatic translation is done, these instances will become multiple separate subsystems. A subsystem with multiple instances must therefore be implemented as library component.

Translating a model automatically takes a couple of minutes but it takes a lot of time to cleanup and optimise. A manual translation takes longer time but the structure will be good from the beginning and subsystems with multiple instances are implemented in the beginning for later use. The two techniques therefore take almost the same time.

The drawbacks in Simulink are sample time handling and the varying order of entries in the signal list.

4 State Machines

A State machine is useful for the design of complex discrete event systems. In a dialysis system there are many different states and modes and it has been found that a state machine is a good way to implement the system behaviour.

In Simulink, state machines are represented and implemented with the Stateflow tool that is based on the Statechart formalism.

4.1 Stateflow

Stateflow is a graphical design and development tool for control and supervisory logic problems. When using Stateflow it is possible to:

1. Model and simulate systems based on finite state machine theory.
2. Design and develop control systems with many different operating modes
3. Easily modify the design, evaluate the results and verify the system's behaviour at any stage of the design.
4. Generate code directly from the design.
5. Use as a building block together with Matlab and Simulink to model, simulate and analyse the complete system.

Inputs and outputs to Stateflow are managed in a data dictionary. The Data dictionary is Stateflow's graphical editor for maintaining inputs and outputs. An advantage is the possibility to define name and type declaration of inputs and outputs and these can be used in the transition conditions. For further information, read Stateflow User's guide, [2].

4.2 Implementation and comparison with SystemBuild

The implementation structure in MatrixX/SystemBuild can be viewed in Figure 8. This structure is recommended in MatrixX programming guidelines, [4].

To handle the different modes and steps in a dialysis machine the state machine is used to define the different states. The transition logic (f) maps the input signals (u) to transition conditions, the state transition diagram (g) goes from current state and transition conditions to next state, and the control action (h) maps the current state to current output signals (z), e.g. valve positions and control loop setpoints and modes.

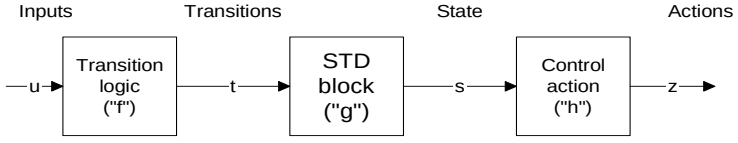

Figure 8 Recommended state machine implementation structure when using SystemBuild.

The reason for this state machine implementation is the notation of the input signals in SystemBuild's State Transition Diagram. These are denoted u_1, u_2, …, u_n and this

limitation restricts the readability and therefore are the state transition conditions placed in Transition Logic (f). The Control action (h) contains matrices with all actions and gives a good overview of set point values, control modes, and valve status for the different states

The State Flow tool in Simulink can handle the same functionality as the State Transition Diagram tool in SystemBuild. Advantages in Stateflow are the possibility to use user-selected names and type declaration of input and output signals. This functionality raises the question of implementation structure. A possible change in the used structure could be that simple transition conditions are implemented directly in the Stateflow diagram and complex ones are handled outside as today. Today a state vector is the output from the state transition diagram. This can be changed to an integer that selects the state vector of set points and actions from the Control action matrix. A more detailed case study would be needed to define guidelines for the best use of the Stateflow tool.

5 Graphical User Interface (GUI)

A Graphical User Interface (GUI) is necessary to have a good overview of what is happening in the application and an easy way to control the application. The demands on the GUI are:

1. It should be easy to build and maintain.
2. When the application becomes large and complex some kind of process picture hierarchy is needed, i.e. a top level containing an overview and lower levels with more details.
3. It is desirable that the GUI does not change the layout of the application. This means that the GUI should only use references to the model, i.e. there should be no explicit connection blocks in the model and the references to the model in the GUI should be invisible.
4. It should be possible to change between different levels in the process picture hierarchy during runtime.
5. Library functions for GUI objects, i.e. buttons, charts and indicators. A library of predefined GUI objects saves time during the design of the process pictures and the different types of objects should have the same look.
6. It should be easy to start the GUI application, i.e. everybody should be able to start and run the application without a large checklist.
7. Copy/paste between process pictures. If process pictures have similar layout a copy/paste between them of common objects should be possible for quicker building.

5.1 Creating a GUI

Simulink has a library called Dials & Gauges. This is a library that contains GUI objects for monitoring signals and controlling the Simulink model. These blocks use ActiveX technology and run only on Microsoft Windows platforms. There are three different ways to use this block set:

1. Include the Dials & Gauges blocks directly in the model.
2. A subsystem containing the Dials & Gauges blocks is created in the application model.
3. The GUI containing the Dials & Gauges blocks is created in a separate model and from this model references to the application model are made.

Solution number 3 has been chosen for this implementation. This gives a stand alone GUI and the connections are references to the Simulink model.

Creating a GUI with Dials & Gauges is the same as building a model in Simulink. xPC-target reference blocks maintain the connection between model and GUI. The different GUI views can be found in app. B and the GUI hierarchy can be found in Figure 9.

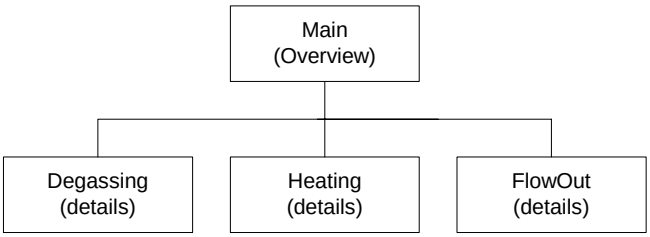

Figure 9 Hierarchy overview of the GUI for the model in Figure 7.

5.2 Problems and solutions

During the evaluation process it has been found that Simulink does not have a GUI tool well suited to rapid prototyping of complex control systems. Some problems have been found and the solutions/workarounds for these are presented.

Communication failure.

If the GUI model is started without the corresponding application downloaded to xPC-target some problems with the communication can be noticed. To avoid this just make sure that there is a target application running on the xPC target.

Visible connection lines

When connecting a dials & gauges object with the xPC-target reference block a connection line is visible. After a few blocks there will be a lot of xPC-target reference blocks and connection lines and these blocks and lines do not contribute any information to the user during runtime. If the foreground colour of connection lines and xPC-target reference blocks are the same as the background colour they will be invisible. For maintenance of the model just change the background colour temporarily and the objects become visible.

Process picture hierarchy

When a system becomes large the GUI often has an overview picture and a couple of pictures with process details. The best way is to place the process pictures in some kind of hierarchy, see Figure 9. Simulink's Model Browser Pane can be compared with Windows Explorer and makes it possible to navigate between different model levels. This would be the optimal tool to use, but during runtime the ActiveX controller of the objects will send errors with communication failure. The workaround to the problem is that all process pictures are opened in different Simulink windows and Windows taskbar is used as navigation tool.

No update function for input objects

Dials & gauges slider objects are used as input objects. A disadvantage is that if an input value is changed, the change is not propagated to other objects that refer to the same parameter. This gives an uncertainty of the actual value. A workaround for the problem is to add a display that shows the actual value for the input. This is a cumbersome and inefficient solution that results in cluttered process pictures.

5.3 Summary and comparison to SystemBuild

To build a GUI with the dials & gauges toolbox is easy and quite a quick work. It is easy to change the properties for an object to a desirable look and behaviour and the user has the

possibility to save objects in libraries. The copy/paste functionality was useful during the creation of the detailed process pictures for each control loop. The common objects were easily copied to all process pictures.

Connections to the target application with target reference blocks create a lot of unnecessary connection lines and blocks. A more elegant way should be an extra pane in the object's property dialog box where it is possible to select target signal references.

The handling of process picture hierarchy does not work very well due to the communication problem described in sec. 5.2. The solution that has been used here works for smaller systems, but when the whole system is implemented there will be around 15 – 20 process pictures. All these process pictures cannot be open at the same time in the Windows taskbar. It will be impossible to have a good overview and easily change between them.

During the testing of the complete system it also has been noticed that the system has communication problems. If there has been a communication failure it is necessary to close the whole session, i.e. GUI, application and the Matlab program, and start everything again. This has been reported to Comsol for further investigation.

When comparing to SystemBuild the problem mentioned above does not exist. SystemBuild has a function that provides the possibility to switch between the different process pictures with a click on a button. Inputs of values in one process picture are updated in the other screens and no problems have been found with communication failure.

The big difference between Simulink and SystemBuild is the connection between the GUI and the model. Simulink uses xPC-target reference blocks for access of signals and parameters directly to the model. SystemBuild has an interface and all signals to/from the model must go through the top level of the model, see Figure 10.

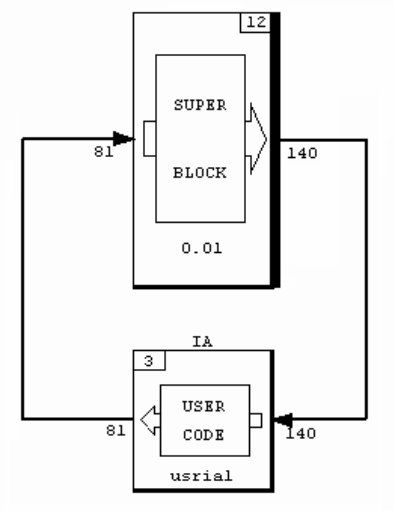

Figure 10 The top level in the SystemBuild system. The SuperBlock contains the model and the User Code Block contains the GUI. All signals between model and GUI must be connected to the outputs and inputs of these two blocks.

Simulink does not fulfil all the demands. Demands number 1, 2, 5, 6, and 7 can be considered as fulfilled but number 3 and 4 cannot. The workaround with changing colour of the connection lines and xPC reference blocks in the GUI is a temporary solution and switching between process pictures works but with limitations.

6 Data Acquisition

During the development phase it is important to have an easy way to select and log data from the test work. To make the data acquisition easy, the following requirements should be fulfilled:

1. Be able to select the signals, preferably by name, from a list of all available signals.
2. Save the logged data to file for post-processing and plotting.
3. Be able to save a collection of logging setups for reuse.
4. Manual start/stop of data logging + automatic triggering when selected I/O signals match user specified conditions.

6.1 The different techniques for data acquisition

In Matlab there are two ways to save data for further data processing.

1. Using the graphical scopes that can be found in the xPC Target library in Simulink. These are added in the model and from these scopes the selection of signals and saving collected data to disk can be made.
2. Create, start and stop scope objects in the xPC target object from Matlab's Command line (the commands can be saved in scripts for reuse). After the scope is stopped the logged data is saved to a .mat-file (Matlab specific file saving format).

In the first alternative, graphical scope objects are added to the model block diagram and it is possible to view the logged signals during runtime. From a signal label list it is possible to add or remove signals and choose the number of samples that should be saved.

The second way is easier to maintain and preferable. The commands are placed in logging scripts that can be used during the execution of the application, see example in Appendix C. This solution makes it possible to have a collection of different logging setups saved and leaves the Simulink model block diagram untouched. The disadvantage is that the references to signals must be updated every time a change has been done in the Simulink model block diagram, e.g. if a block is removed, added or moved in the Simulink model the signal list will be rearranged and as a consequence the script must be updated with the new signal indices, cf. section 3.3.

6.2 Summary and comparison to MatrixX/SystemBuild

When using scripts there is full freedom to design scripts with a functionality that suits the application. The big disadvantage is the need to update the signal number list after every new download of the model, to the xPC-target, if the model has been changed. If the naming of the blocks is made according to the design tips in section 3.3 it should be quite easy to pick out the wanted signals from the list. An example of how a script can work can be seen in Appendix C, Logging scripts.

The scripts can handle manual start and stop of the data acquisition. The logg_stop script also stops the logg after predefined number of data samples. If a logg should start with trigger conditions on a specific signal the Logg_start script has to be modified. The scope object that is used for data acquisition has object properties for this type of problem and

there should be no problem to set the scope object properties for a desirable behaviour. After the data logging is finished the data must be saved to a file (.mat-file) on a disk for later pre-processing and plotting of the measured results.

The requirements are fulfilled except the list function of the available signals. If there are changes in the block model diagram between two builds & downloads the logging scripts must be revised. If the model is large the signal list are very long and it can take some time to revise the different logging scripts. This is a serious drawback.

The only and major difference is that signals are selected from the input/output interface in MatrixX/SystemBuild. MatrixX/SystemBuild's handling of signals makes the data acquisition much easier to maintain.

The commands that are needed to produce plots of the logged data are saved in .m-files so it is easy to reproduce results. The way to handle this is the same for Matlab/Simulink and MatrixX/SystemBuild.

7 Simulation with hardware-in-the-loop

Hardware-in-the-loop simulation with Matlab/Simulink requires the following products:

1. Matlab
2. Simulink
3. Real-Time Workshop
4. xPC Target
5. C-compiler (MS Visual C/C++ or Watcom)

For this evaluation Microsoft Visual C/C++ has been used as C compiler. For more details, see [3].

7.1 xPC target

xPC target is a host-target solution for prototyping, testing and development of real-time systems using standard PC hardware. The advantage of xPC target compared to MatrixX/RealSim is that it uses a standard PC and standard I/O cards. This solution eliminates the need to install software or modify existing software configurations. After or during a development phase it is possibly to use the target PC as a desktop computer as usual.

7.2 Building and controlling a target application

How to build and download the application can be found in [3]. To avoid problems during the simulations prepare the simulation in the following steps

1. Load parameters for the model block diagram to Matlab's workspace, i.e. control loops, filters and other model parameters. Parameter values should be set with scripts.
2. Open the Simulink model, build and download to the target PC. The Real Time Workshop option settings for the system target file should be xPC Target (xpctarget.tlc) and the option "Generate Code Only" should be unselected.
3. Start the target application from Command window.
4. Open the GUI model and start it.

A start up in this order avoids trouble with initiations of the target reference blocks in the GUI and communication problems.

During the building and downloading process a target object (tg) is created. The object is created and downloaded with one command either from command line or a button in Simulink. The downloaded object can be controlled both from host and target. During runtime it is possible to change parameters in the target object. This can be done both from the Matlab command line and from xPC- target's command line.

It is possible to save an xPC target object to disk. Just use the Save command on the object. When loading it to workspace the Load command is used. The Load command is used once again to download the target object to xPC target.

7.3 Running the translated model

To verify that the model is correctly translated, a couple of simulations have been done to see if the behaviour of the two implementations is the same. The test suite contains:

1. Step response for Degassing Pressure control loop, see Figure 11 and Figure 12.
2. Step response for Flow Out control loop, see Figure 13 and Figure 14.
3. Step response for Degassing temperature control loop, see Figure 15 and Figure 16.
4. Load disturbance for Degassing temperature, see Figure 17 and Figure 18.

In practice the test was done first on the SystemBuild model and after that the same test was made on the Simulink model. All parameters are the same in the two systems.

7.4 Test result

The results of the tests are shown in Figures 11 to 18. When comparing the two systems, the behaviour is the same but a few things can be noticed.

1. The steady state level of the control output in the Degassing and Flow control loops is different. The reason for this is that the PWM output from SystemBuild's PWM card gives a little higher voltage at the pump compared to Simulink's PWM card. The differences between the two PWM card output duty cycles are of the same magnitude as the differences between the two control outputs from the model.
2. An oscillation in the Flow Out signal has been found. This is a control signal quantisation problem. In the Flow Out Step response it appears clearly, see Figure 13. When the set point is 500 ml/min the problems is visible but when the set point is 600 ml/min this phenomena has disappeared. If the lower set point is changed to 510 ml/min the problem also disappears.
3. The noise level in temperature signal is different for the two systems. The Matlab signal has higher noise level than the SystemBuild model has. An explanation is that the analog I/O cards have different noise sensitivity.

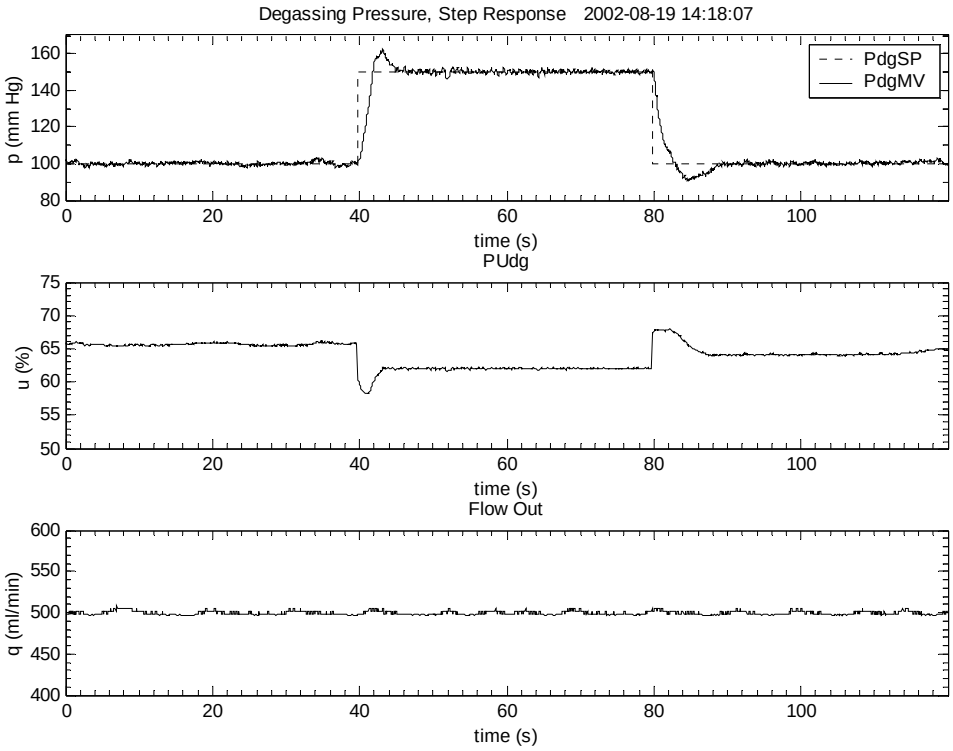

Figure 11 Step response for the Degassing Pressure with Matlab/Simulink environment.

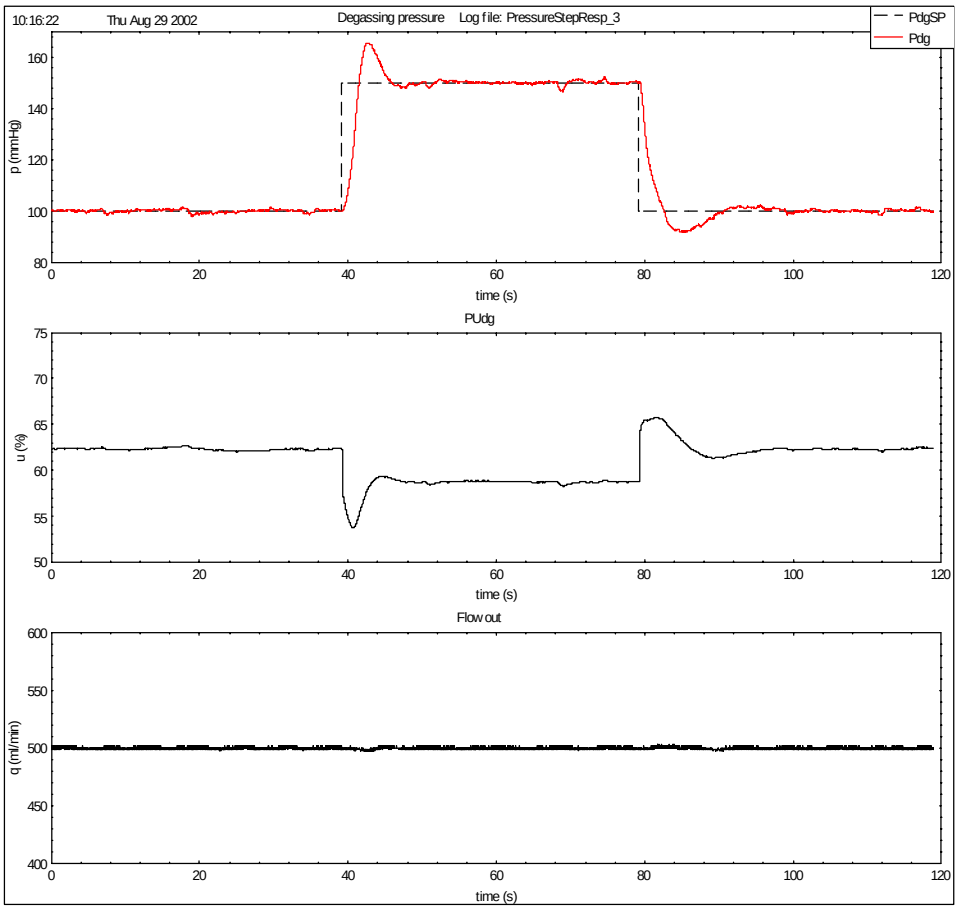

Figure 12 Step response for the Degassing Pressure with MatrixX/SystemBuild environment.

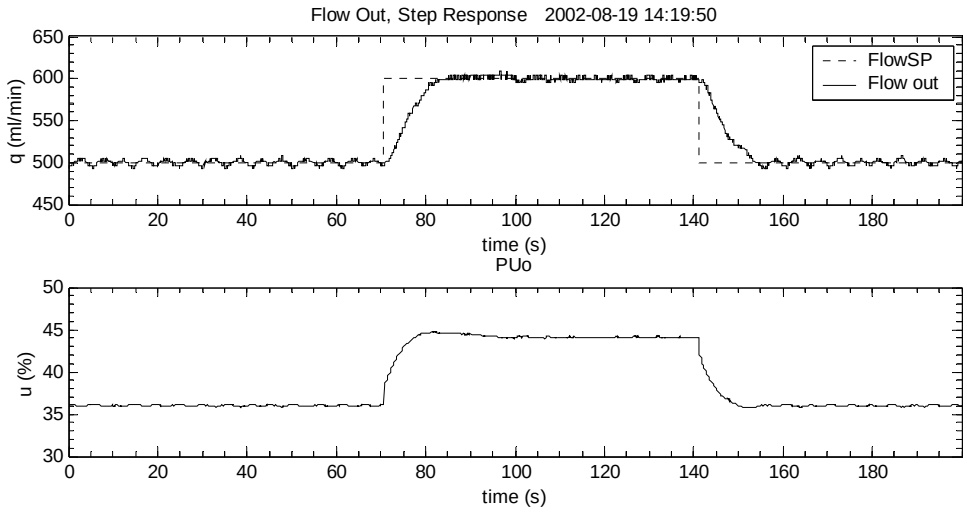

Figure 13 Step response for the Flow Out with Matlab/Simulink environment.

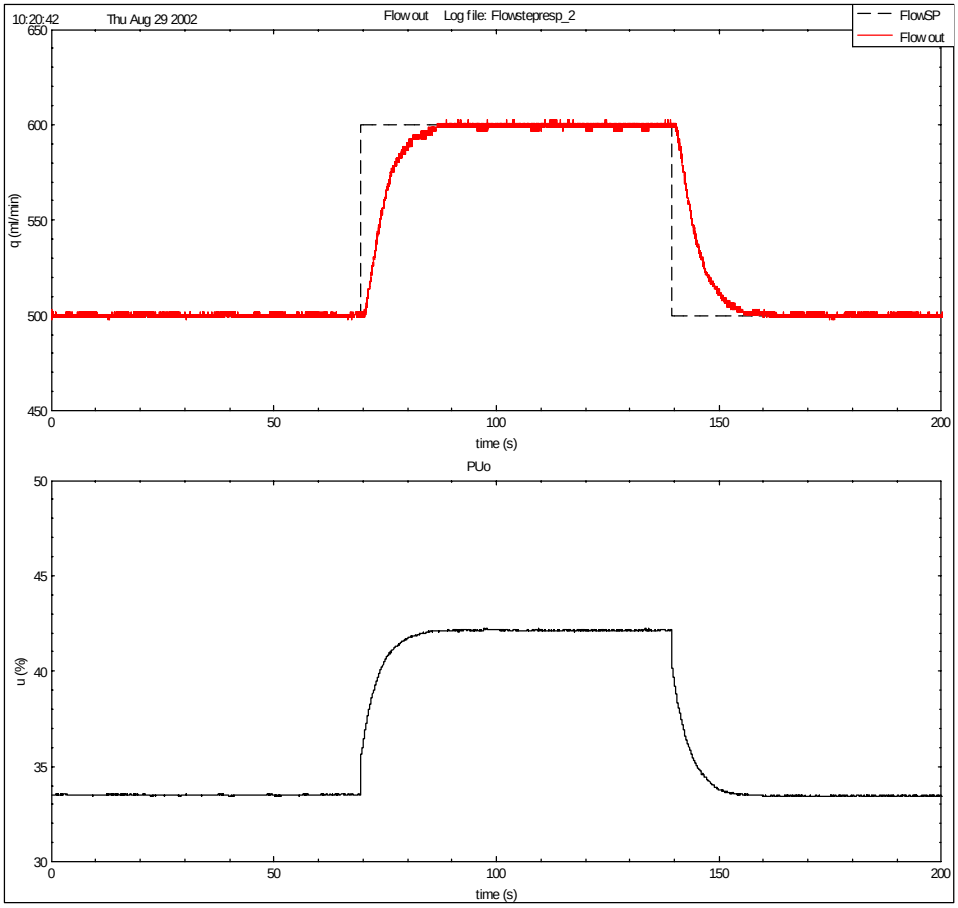

Figure 14 Step response for the Flow Out with MatrixX/SystemBuild environment.

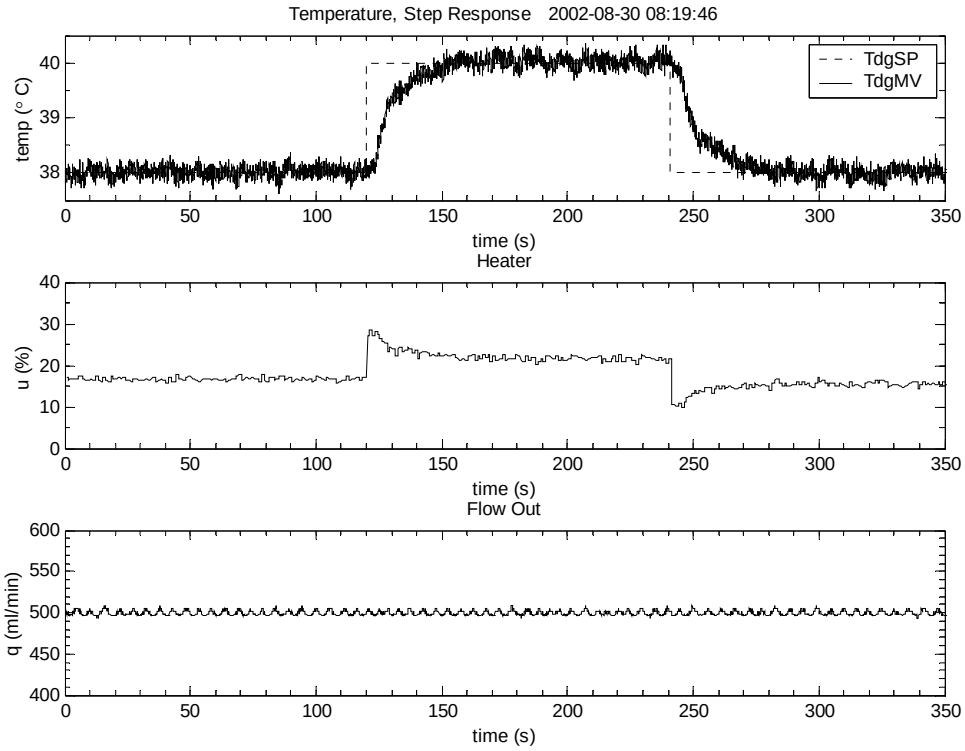

Figure 15 Step response for the Degassing Temperature with Matlab/Simulink environment.

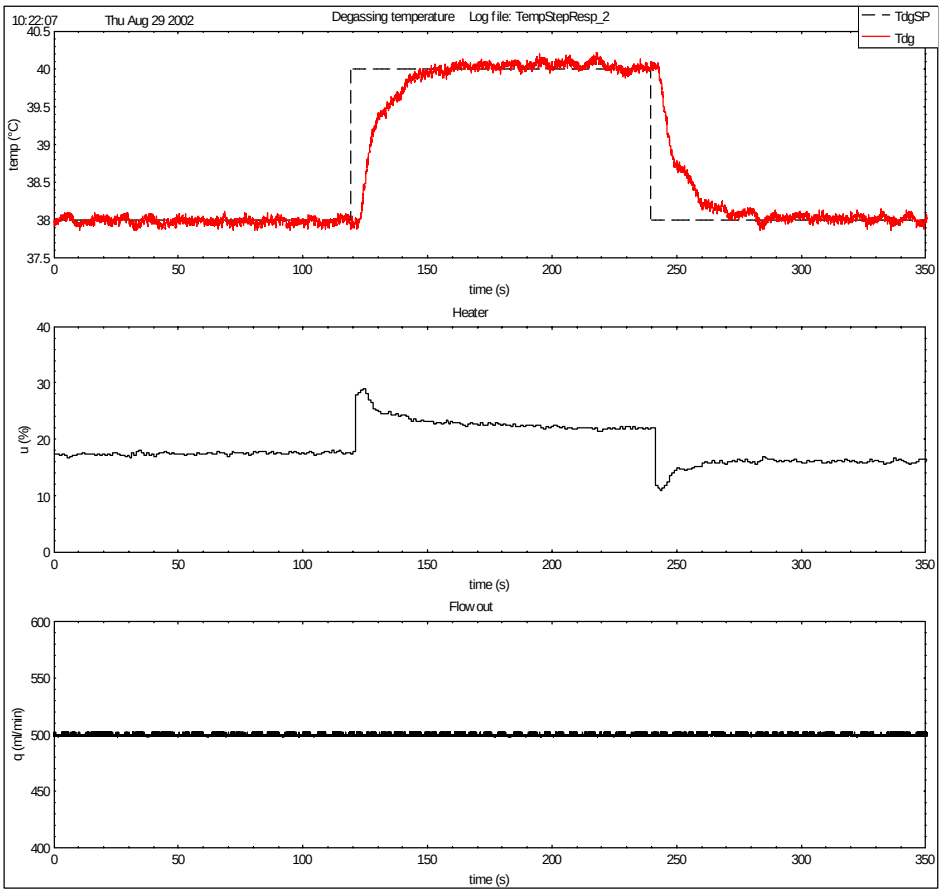

Figure 16 Step response for the Degassing temperature with MatrixX/SystemBuild environment.

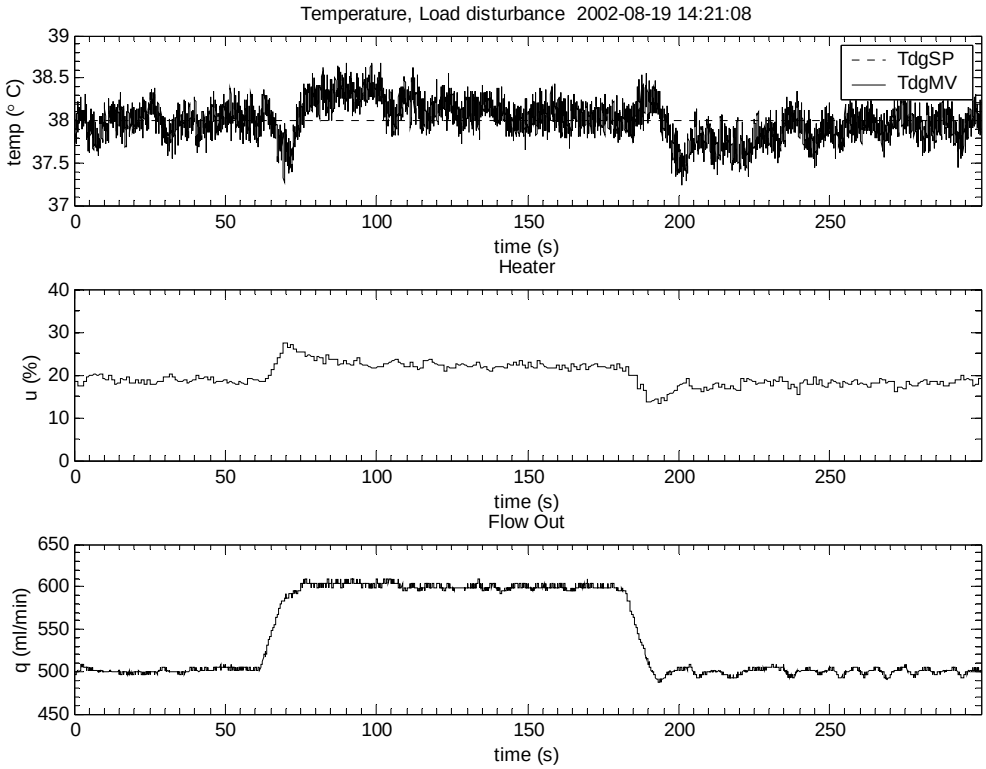

Figure 17 Load disturbance for the Degassing temperature with Matlab/Simulink environment.

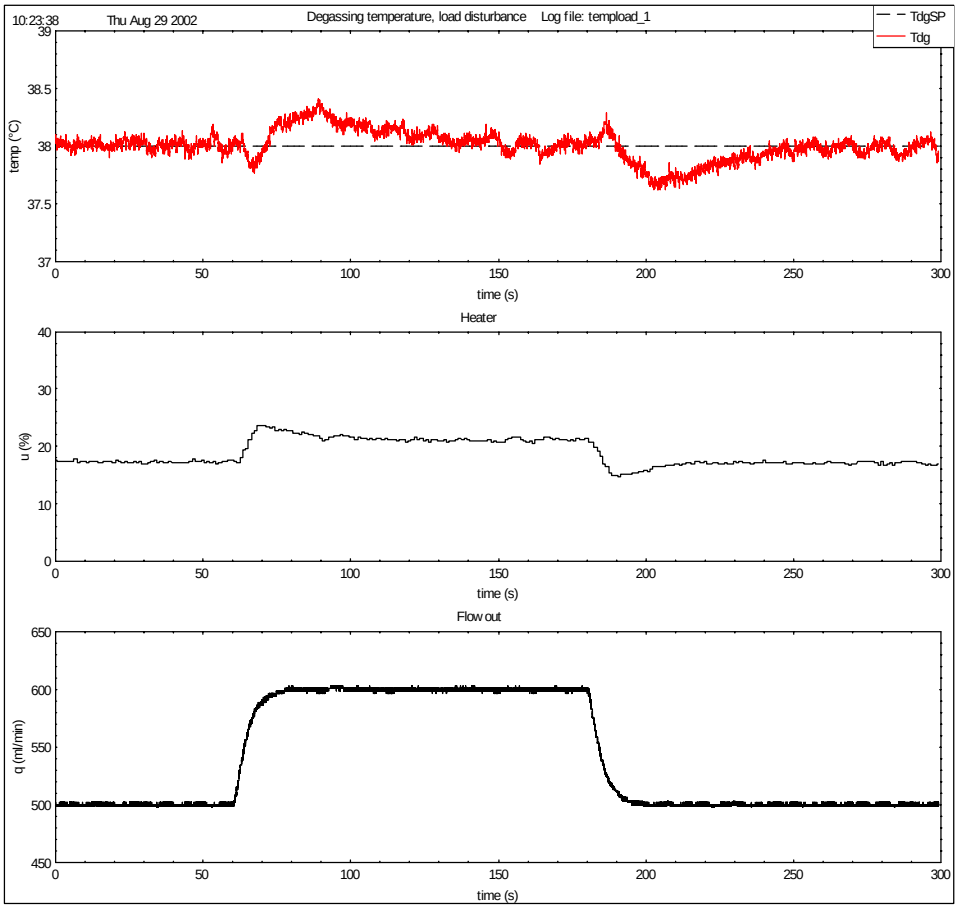

Figure 18 Load disturbance for the Degassing temperature with MatrixX/SystemBuild environment.

7.5 Summary and comparison

Because of the use of standard components for the hardware system, i.e. target computer and I/O and network cards, the hardware in Matlab/Simulink RealTime Workshop with xPC target is easy to maintain. The compiling and downloading of the target object is done in one command. The only way to follow the building and downloading process is the log in Matlab's command window. Using a script with separate commands for analysis, compiling, and downloading to the target system would be preferred. The use of a script gives the user full control of the different steps in the process.

The big problem that has caused a lot of work without a solution is the use of a counter/timer card for frequency input from flow transducers and generation of PWM signals to drive flow pumps. The used counter/timer card was selected from a list of cards supported by Matlab (cf. Appendix A). In the Flow Out control loop, see Figure 7, the control loops reads a value from a frequency input and the control output feeds a PWM output. A handful of times the control loop has worked as expected and the test result has been verified. But for the major part of downloads the control loop has not worked properly. The different failures that have appeared are

1. The frequency measuring does not work properly.
2. The PWM output does not produce a signal.
3. The control loop does not execute properly, control output $\equiv 0$.

The problem has been reported to Comsol for further investigation. New driver software for the counter/timer card has been tested without any improvement in the behaviour. When the control loop is fed with an analog signal it works perfect. This behaviour raises serious concerns about the stability and reliability of the RealTime Workshop system.

8 Code Generation

The code generation tool is used when the xPC target model is intended to produce code for production. In embedded systems, memory usage and execution time are of big interest. The generated code should be

1. Easy to read, i.e. traceability to the block diagram and a good code structure
2. Efficient code (with possibility to optimise the code).
3. Modular, e.g. one function generated for each control loop.

If code is generated for subsystems every subsystem must have a unique name to avoid naming conflicts and make the code easier to read. Also a block structure that is easy to follow is preferable.

8.1 Code generation procedure

Real Time Workshop makes it possible to generate code for different target systems. The target system that has been used in this case is the embedded C code generation. This code format should give the most efficient code for use with other software. During the code generation it is only the subsystems, i.e. the control loops that are code generated. The reason for this is that other software modules handle reading and setting of I/O signals and time scheduling in the final target system.

During the work with code generation following problems have been found:

1. When using the S-Function builder blocks they must be preceded with a Signal Specification block. The reason for this is a bug in S-function builder block. The Signal Specification Block specifies the input port dimensions of the builder block explicitly. If not an error will occur and the code generation is aborted.
2. No code generation can be made together with xPC target I/O blocks. This is a software bug and the work around is to copy the subsystem that should be code generated, paste it in a new Simulink model window and then generate code.

The code generation has been done with the default values and the code generation procedure is the following:

1. Copy the subsystem to be generated to a separate Simulink model window.
2. Connect Inport and Outport blocks to the subsystem inputs and outputs. Set up Real-Time Workshop Options as below:
    a. Target Configuration parameters:
    System target file: RTW Embedded Coder (ert.tlc)
    Select "Generate Code Only".
    b. ERT code generation options:
    Use defaults. (HTML report option might be useful)
    c. Solver: Fixed-step (Fixed Step Size = auto, Mode = auto)
3. Generate code with the command Tools - Real-Time Workshop - Build Subsystem. The generated code will be placed in a separate folder with the same name as the subsystem appended with "_ert_rtw".

The HTML report contains information about the code generation and all source and header files that have been produced.

Using default values of the code generation options gives an idea of what type of further optimisation that is needed. As a final test the generated code for the degassing loop was implemented in the final target system that is running on a Hitachi H8S/2633 with a CPU speed of 25MHz.

8.2 Results of the code generation

The generated code is placed in a subfolder to the root folder of the model. For a subsystem there is one source file and a couple of header files created plus reusing of the wrapper source files created from the S-Function builder blocks that are used in the model. The file structure with the dependence between the different files can be seen in Figure 19. The beginning of the filenames is the code generated subsystem name.

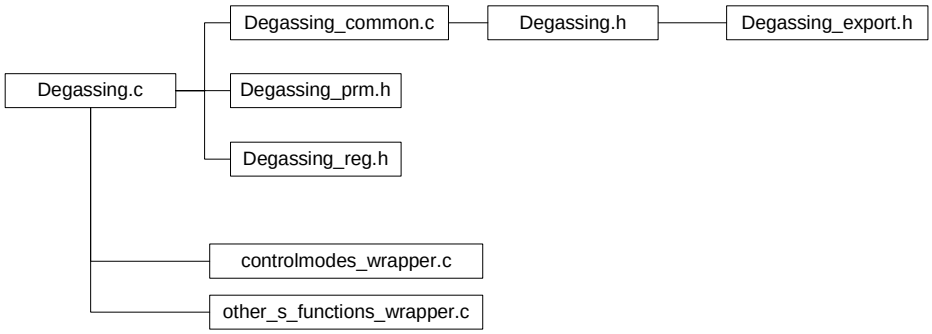

Figure 19 The files starting with degassing is the code generated ones. Wrapper files are reused from the model structure.

When the generated code is compiled together with the software for the final target system, the following has been noted:

1. Declarations of inputs and outputs have the same name in the different subsystems. This will cause name conflicts then the other sub systems are implemented in the final target system.
2. Parameters are declared in a header file. This should be done in the source file instead.
3. Includes of a number of header files that belong to the Matlab system.
4. Global variables had to be moved to the source file manually.

When the code was compiled and linked with the surrounding software in the final target system RAM, ROM and execution time for the loop was measured. The same was done for the SystemBuild code. The result of this showed that the Simulink code used a few bytes more memory than the SystemBuild code, but runs twenty micro seconds faster. The execution time was measured with a Hitachi E6000 emulator. The difference in performance is not significant.

The generated code from the two systems has been compared. SystemBuild generates only two files, one source file and one header file, to be compared with Simulink's six files except the reused ones.

The readability of SystemBuild's code is better. One reason for this is that the MatrixX-code is generated with the PID-controller implemented as a C function in a User Code

block and Simulink's code is generated from a block model structure. If the PID-controller was implemented as a S-function the Simulink code would be easier to read. However, the comments in the code give a good traceability to the block model structure in both systems.

The MatrixX code is implemented as a function with input arguments, but Simulink uses void functions and global variables to pass information to/from the function. The use of global variables puts an extra attention to the programming of the surrounding software.

The code generation tool in Simulink produces code that can be used in a final target system. Optimisations that have to be done are better type declarations of input and output signals for the subsystem, minimise of the amount of produced files and the PID-controller should be implemented as a S-function to reduce the number of reused files.

It should be pointed out that the MatrixX/SystemBuild code was generated with a modified code generation template developed at Gambro, see [4]. When the default code generation template is used, only a source file is produced and naming conflicts similar to Matlab/Simulink appear.

9 Summary and conclusions

This summary will give an overview of the two development tools and point at the differences that have been found.

Model Building

Parameters & signals

Matlab/Simulink	MatrixX/SystemBuild
Parameters can be organised in data structures. All values can be accessed everywhere. Signals are stored in a list when the system is built. Changes in the model between builds change the signal indices of the list.	Parameters are organised in Partitions. In a SuperBlock the partition is assigned and all children to that SuperBlock can access the assigned partition.

Sample time

Matlab/Simulink	MatrixX/SystemBuild
A block inherits the sampling time from the previous block's output. Without the use of colour codes it is confusing. To ensure that a subsystem runs with specific sample time all input ports should have sample & hold blocks with the specified sample time.	The sample time is set in the SuperBlock and all children blocks inherit the sample time. This is a straight-forward and easy-to-use method

Names on blocks and signals

Matlab/Simulink	MatrixX/SystemBuild
The signal name is inherited from the block producing the signal. Even if a subsystem is masked, the output signal from the subsystem is referenced by name to the signal-producing block in the subsystem. If the signal should be logged the block that produce the desired signal must have an appropriate name. Signal labels are used in a model if scope object blocks are added to the model and for increased readability.	The signal name is the name of the output of the block that produces the signal value. The signal name appears in the interface between Block diagram model and GUI.

State machine

Matlab/Simulink	MatrixX/SystemBuild
The user can define names of inputs and outputs. These names can be used in the transition conditions.	Inputs are denoted u_1, u_2, …, u_n. This limitation restricts the readability of the state transition conditions.

GUI

Matlab/Simulink	MatrixX/SystemBuild
A GUI block that makes it possible to type in values to the model is missing. Only input sliders and knobs are provided. If a value is changed in one input slider, there will be no propagation to other sliders. This creates confusion about the actual value. Cannot handle hierarchy based GUIs very well (Software bug). Sensitive to communication problems. Cluttered process pictures with redundant objects. Workaround: Connection lines and xPC target reference blocks must have the same foreground color as the background. Otherwise the blocks and lines will be visible.	A simple and well-working tool with the necessary control and visualisation blocks. Handles hierarchy based GUIs very well. Drawback: No copy/paste function between pictures, only within picture.

Data acquisition

Matlab/Simulink	MatrixX/SystemBuild
No limit in the amount of scopes. Write your own m-files that handle setup, signal list and start and stop of the scopes. Drawback: If block diagram is modified the signal list must be revised.	Has a built-in function for data acquisition and handles up to 10 different log setups.

Simulation with hardware-in-the-loop

Matlab/Simulink	MatrixX/SystemBuild
Sensitive to communication problems. It is possible to control the target application from both the target and host command lines. The xPC target object must be saved to a .mat file if the xPC target object should be used again without a new build & download.	A GUI guides the user through the build and download process. Possible to download an existing executable file. No communication problems noted.

Code generation

Matlab/Simulink	MatrixX/SystemBuild
Provides code generation for a couple of different target systems. Embedded-C has been used. Also ADA code generation is provided.	Generates functions or stand-alone executable. C or ADA code generation.

A first impression, Matlab/Simulink seems equivalent to, and in some respects even better than MatrixX/SystemBuild. However, there are several serious bugs and functional deficiencies that must be fixed before Matlab/Simulink can be considered as really useful for the studied type of application.

A better GUI tool is necessary and the data acquisition must be easier to maintain. If the signals can be referenced by their name it would simplify the maintenance of logging scripts significantly. Also the handling of the sampling times is confusing and causes problems during hardware-in-the-loop simulations.

To convert a system that is optimised for one development environment to another with other possibilities is a lot of work and the development project can lose a lot of valuable time. The best time to change a rapid prototyping tool is between two projects. If doing it in this way the developers have time to “grow up” with the new tool and can find out how the tools can be used in the best way.

Suggested subjects for future work are optimisation of the code from the code generation and a design study of how to best use the Stateflow tool.

10 References

1. The MathWorks Inc., SB2SL -- User's Guide, Version 2, September 2000
2. The MathWorks Inc., Stateflow – User's Guide, Version 4, June 2001
3. The MathWorks Inc., xPC Target For Use with Real-Time Workshop – Getting Started Guide, Version 1, November 2000
4. Anders Wallenborg, MatrixX Programming Guidelines, Gambro Lundia AB, issue 6, 2001-03-27

11 Appendix A, Program versions and I/O cards

Program versions

MATLAB Version 6.1.0.450 (R12.1) on PCWIN

MATLAB Toolbox	Version 6.1	(R12.1)	18-May-2001
Simulink	Version 4.1	(R12.1)	06-Apr-2001
Stateflow	Version 4.1	(R12.1)	21-May-2001
Stateflow Coder	Version 4.1	(R12.1)	21-May-2001
Real-Time Workshop	Version 4.1	(R12.1)	18-May-2001
Real-Time Windows Target	Version 2.1	(R12.1)	02-Feb-2001
Dials & Gauges Blockset	Version 1.1.1	(R12.1)	02-Feb-2001
Real-Time Workshop Embedded Coder	Version 2.0	(R12.1)	01-Mar-2001
Simulink Performance Tools	Version 1.1	(R12.1)	18-May-2001
SB2SL (converts SystemBuild to Simulink)	Version 2.5	(R12.1)	30-Apr-2002
xPC Target	Version 1.2	(R12.1)	09-Apr-2001
xPC Target Embedded Option	Version 1.2	(R12.1)	09-Apr-2001

MatrixX Product Family Version 6.2.2 Build 62mx1118 (win32)

Windows NT Version 4.0

I/O Cards

National Instruments, PCI-6025E multi-function card

Measurement Computing, PCI-CTR05 Counter/Timer card

12 Appendix B, GUI Pictures

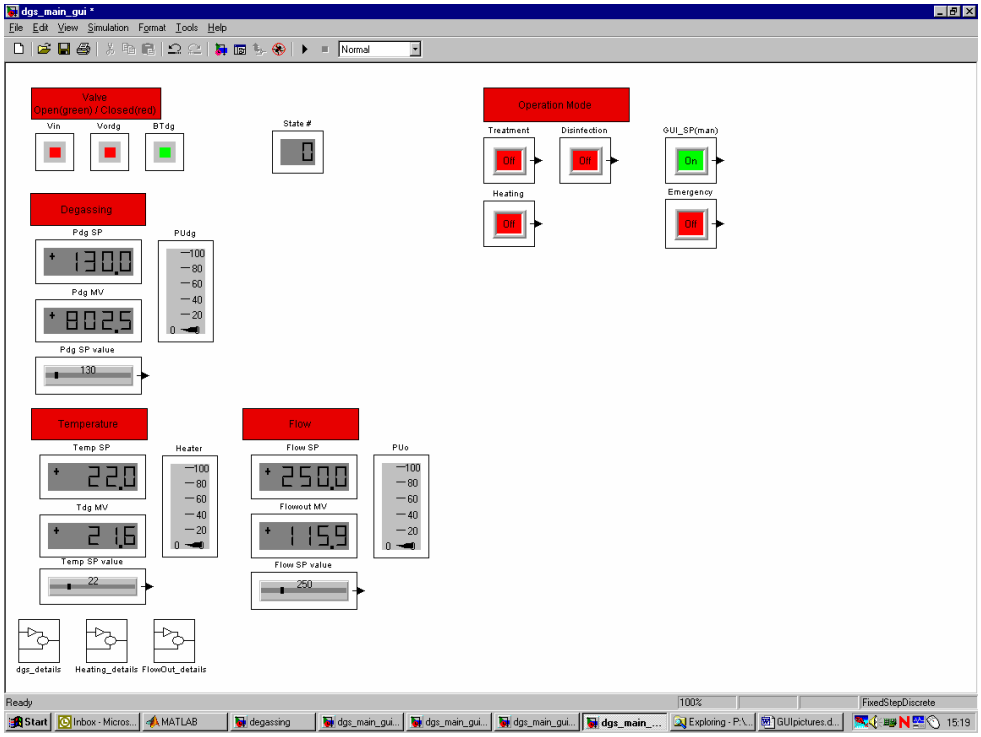

Figure 20 The Main GUI window. This GUI window gives an overview of the whole system. Set points, measured values and control outputs for the different loops, valve status and system state are presented here. From this window the operational mode is selected.

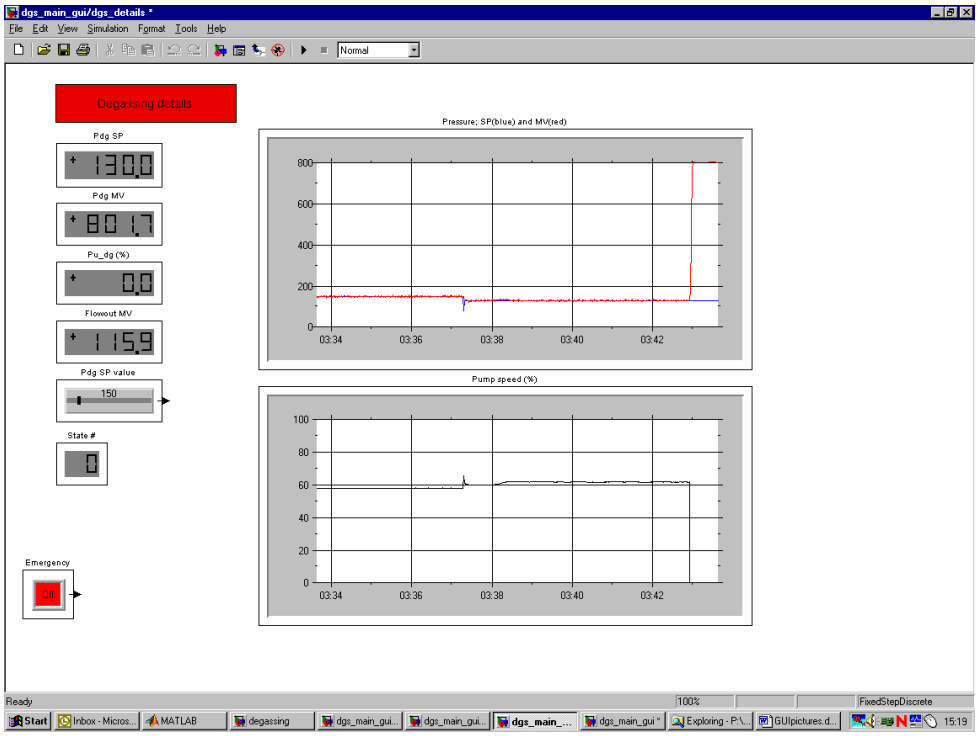

Figure 21 Details for degassing loop. Plot showing set point (SP), measured value (MV) and control output are presented. Flow Out MV and system state can be viewed. It is also possible to adjust degassing SP.

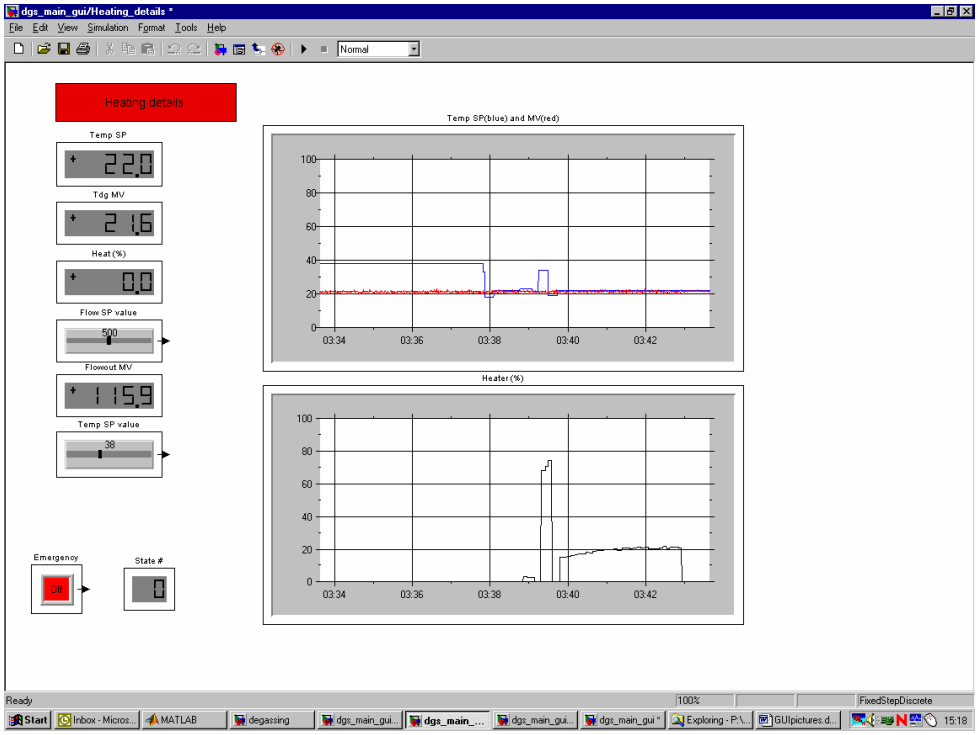

Figure 22 Details for heating loop. Plot over set point (SP), measured value (MV) and control output are presented. Flow Out MV and system state can be viewed. Flow SP and Temperature SP can be adjusted.

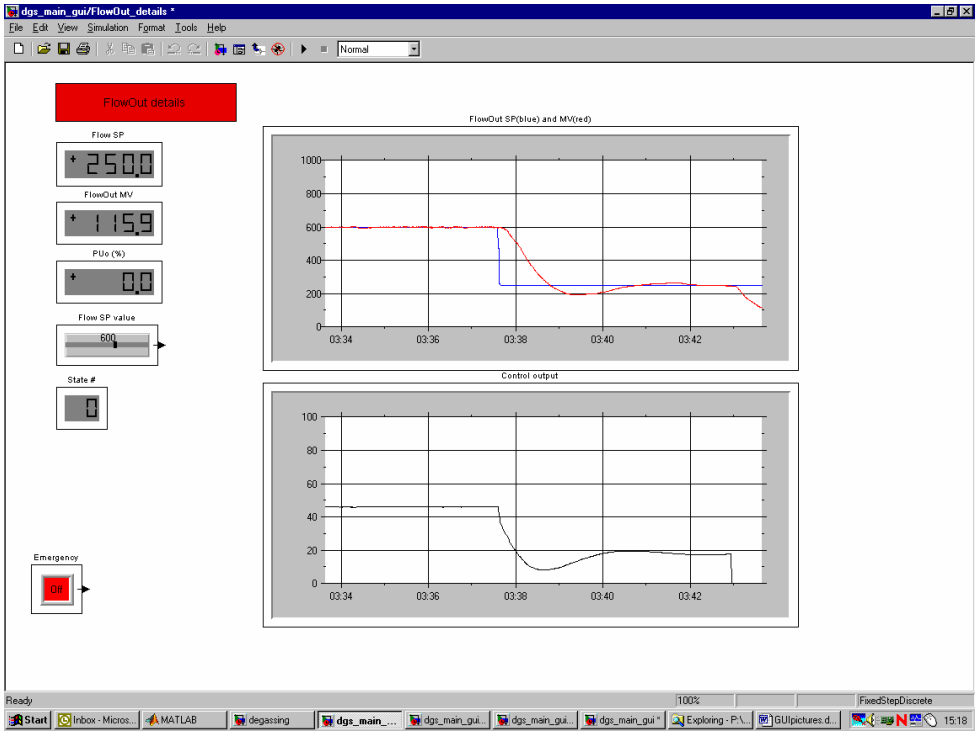

Figure 23 Details for FlowOut loop. Plot over set point (SP), measured value (MV) and control output are presented. System state can be viewed and Flow SP can be adjusted.

13 Appendix C, Logging scripts

The logg_start script creates an xPC target scope and starts it after finished set up. The logg_stop script returns a matrix with the collected data. If the scope is running the script stops the scope before returning the collected data.

Example:

```
>> LOGG_START(tg, 1, [59 239 270 225 247 343 116 237 307], 20000);
>> data=LOGG_STOP(tg, 1);
```

```
function logg_start(xpcobj, id, signals, numsamples)
%   LOGG_START(XPCOBJ, ID, SIGNALS, NUMSAMPLES)
%
%   Logg_start starts a logging session with a xPC target
%   scope object.
%
%   XPCOBJ is the xPC target object running on the target PC.
%   ID is a numeric index unique for each scope.
%   SIGNALS is a vector of the signals to logg, use the signal
%   indices from the xPC target object
%   NUMSAMPLES is the number of samples to collect.
%
%   Note! If an already existing scope with ID is used the
%   old ID is removed.
%
%   Author: Håkan Nilsson
%
%   Revision record:
%
%   Date                Sign.   Comment
%   ----------    -----    --------------------------------
%   2002-06-17  HANL    First version.
%
%   2002-09-04  HANL    A more detailed help added.
%
%
%

scope=get(xpcobj,'scopes');
for i=1:length(scope)
   if scope(i)==id;
        remscope(xpcobj,id);
   end;
end;

addscope(xpcobj,'host',id);
sc=getscope(xpcobj, id);
addsignal(sc, signals);
sc.NumSamples=numsamples;
sc.start;
```

```
function data=logg_stop(xpcobj, id)
%   LOGG_STOP(XPCOBJ, ID)
%
%   Logg_stop stops the acquisition if it is not
%   finished and returns data. It is an error to
%   access a nonexistant scope.
%
%   XPCOBJ is the xPC target object running on the target PC.
%   ID is a numeric index unique for each scope.
%
%   Author: Håkan Nilsson
%
%   Revision record:
%
%   Date                Sign.   Comment
%   ----------    -----    --------------------------------
%   2002-06-17  HANL    First version.
%
%   2002-09-04  HANL    A more detailed help added.
%
%

sc=getscope(xpcobj, id);
sc.stop;
data=[sc.time' sc.data];
```