EINDHOVEN UNIVERSITY OF TECHNOLOGY

# Using custom controllers on the H-drive

*Citation for published version (APA):*
Koot, M. W. T. (2000). *Using custom controllers on the H-drive*. (DCT rapporten; Vol. 2000.029). Technische Universiteit Eindhoven.

*Document status and date:*
Published: 01/01/2000

*Document Version:*
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

*Please check the document version of this publication:*

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

*General rights*
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

*Take down policy*
If you believe that this document breaches copyright please contact us at:

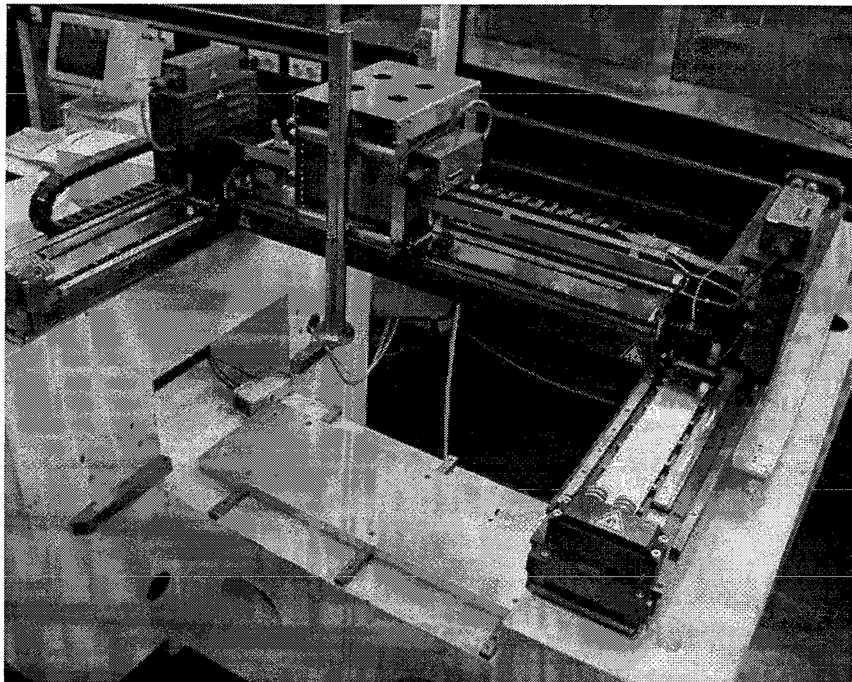
openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 04. Oct. 2023

# Using custom controllers on the H-Drive

M.W.T. Koot
September 2000
D&C Report 2000.29



This report is the result of a trainee assignment.


Student: Michiel Koot 421747
Coach: René van de Molengraft
Professor: Maarten Steinbuch

# Abstract

The H-Drive is an industrial positioning system with a built-in controller that communicates with a program running on a PC. To increase the performance, one wants to use other controllers that are designed in Matlab/Simulink and that can be easily implemented on a Digital Signal Processor. Therefore the H-Drive must be connected to this Rapid Control Prototyping system (RCP). It is desired to establish an interface where the default controller of the H-Drive is disabled, but where other features, such as initialization can still be used.

It is chosen to use a program on the PC, the LAPI Server. This program communicates with both the H-Drive and the RCP system. It can initialize and shut down the H-Drive properly and can disable the built-in controller to give control access to the DSP. A Simulink model has been made that captures the position measurement, outputs the controller signal and has a protection system against collision of the carriage against it's boundaries. The implementation has been done only for the x-axis.

The performance of the interface has been tested by rebuilding the default controller in Simulink. The performance of the controller on the RCP system is worse than that of the built-in controller. This is probably caused by bad synchronization and delay times between the RCP system and the H-Drive.

# Contents


1 Introduction 3

2 Description of the hardware 4
2.1 The H-Drive . . . . . . . . . . 4
2.2 The Digital Signal Processor . . . . . . . . . . 5

3 Design of the interface 6
3.1 Task specification . . . . . . . . . . 6
3.2 Approach . . . . . . . . . . 6
3.3 Communication . . . . . . . . . . 7

4 Implementation 8
4.1 Procedure . . . . . . . . . . 8
4.2 Serial port communication . . . . . . . . . . 8

5 The LAPI Server 10
5.1 Task specification . . . . . . . . . . 10
5.2 Design . . . . . . . . . . 10
5.3 Implementation . . . . . . . . . . 10

6 The Simulink model 12
6.1 Task specification . . . . . . . . . . 12
6.2 The protection system . . . . . . . . . . 12
6.3 Implementation . . . . . . . . . . 13
6.3.1 Layout . . . . . . . . . . 13
6.3.2 The communication block . . . . . . . . . . 14
6.3.3 The protection function . . . . . . . . . . 15
6.3.4 The timer clock . . . . . . . . . . 16
6.3.5 Other S-functions . . . . . . . . . . 17

7 Test experiments 19
7.1 Testing the performance . . . . . . . . . . 19
7.2 Testing the protection system . . . . . . . . . . 20

8 Conclusions and recommendations 22

A Block scheme of the controller 23

B LAPI Server function descriptions 24

C LAPI Server source code 29

# Chapter 1

# Introduction

The H-Drive is an industrial positioning system with a built-in controller, consisting of a PID-type feedback and a model-based feedforward. The built-in electronics can communicate with a program running on a PC. With this program, the H-Drive can be initialized, set-points can be given and controller parameters can be changed.

The performance of the H-Drive may be improved, by using custom controllers based on other theories. This can be done by connecting the H-Drive to a Rapid Control Prototyping environment (RCP), which in this case will be a dSPACE system with a dedicated Digital Signal Processor (DSP) in combination with the software package Matlab/Simulink. This way, custom controller designs can be easily implemented on the H-Drive.

To establish this connection, several problems have to be dealt with. One of them is the communication between the RCP system and the H-Drive. Another problem is whether it is possible to disable the control algorithm of the H-Drive, but still make use of some other features, such as initialization and protection against collision. When an interface is made, it's performance will be tested and compared with the default controller.

This report is build up as follows. In Chapter 2, the features of the H-Drive and the RCP system are studied. In Chapter 3, the required tasks of the connection are described and the best approach to fulfil those requirements is analyzed. In Chapter 4, 5 and 6, an implementation of the interface is made. In Chapter 7, the interface is tested in practice. In Chapter 8, conclusions are made and suggestions for further research are given.

# Chapter 2

# Description of the hardware

## 2.1 The H-Drive

The H-Drive is an industrial robot made by Philips CFT, used for fast and accurate positioning. It consists of three LiMMS axes (Linear Motor Motion System), one for movements in X-direction and two for movements in Y-direction. Movements are made with a maximum acceleration of 20 $m/s^2$ and maximum velocity of 2 m/s. This report will be restricted to the x-axis, which has a working range of 0.65 m.

Each of the three axes is driven by a Single Axis Controller (SAC) which contains a $3^{rd}$ order trajectory generator and a controller which consists of a PID-type feedback controller and a model-based feedforward. The power is provided by an external power amplifier. A setup containing one LiMMS axis is displayed in figure 2.1.

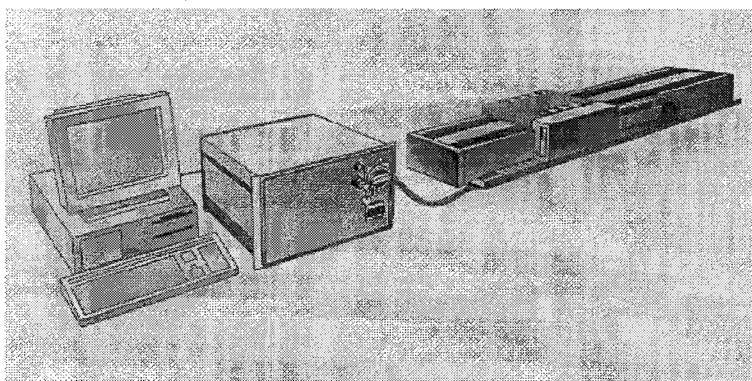


Figure 2.1: Single LiMMS axis setup

The three axes can communicate with each other, the power amplifier and a PC using the CAN protocol (Controller Area Network). On this PC, a program is running that makes use of a function set called LAPI (LiMMS Application Programming Interface), which is written in the C programming language. Using this LAPI program, commands can be given to the SAC, e.g. to initialize the H-Drive and move the carriage to it's home position. After that, set-points can be submitted and the controller parameters can be changed. It is also possible to monitor two signals and inject one signal at several positions in the closed loop system. This feature is intended to be used for measuring transfer functions, but maybe it is also suitable for injection of a signal generated by a different controller. The block scheme of the

controller with it's monitor and injection points can be seen in Appendix A.

The position can be measured by monitoring it, but this is a noisy analog signal which is not accurate enough to use it for a feedback controller. Besides, it has a delay of one sample time. Therefore, the measurement from the incremental encoder will be used. This encoder has an accuracy of 1 $\mu$m. To safely use the signal of the incremental encoder for both the SAC and the DSP, an electronic circuit is used which duplicates the signal.

More information on the H-Drive can be found in [1, 2, 3].

## 2.2 The Digital Signal Processor

The RCP real-time controller board used here, is the dSPACE DS1103. This board is based on the Motorola PowerPC 604e processor. It contains also a DSP subsystem based on the Texas Instruments TMS320F240 DSP and a CAN subsystem based on the Siemens 80C164 microcontroller, which is used for connection to a CAN bus. The board is an ISA-card, which can be mounted in a PC.

The board has (among others) the following connections:

- 4 parallel A/D converters multiplexed to 4 channels each with 16-bit resolution and 4 $\mu$s sampling time.
- 4 parallel A/D converters with 1 channel each, 12-bit resolution and 800 ns sampling time.
- 2 D/A converters with 4 channels each and 14-bit resolution.
- an incremental encoder interface comprising 1 analog channel with 22/38-bit counter range, 1 digital channel with 16/24/32-bit counter range and 5 digital channels with 24-bit counter range, switchable between differential RS-422 and single-ended TTL.
- CAN support fulfilling CAN specifications 2.0A and 2.0B
- a serial port that can be configured as an RS-232 or RS-422 interface.

The board is placed in a Pentium II 400 PC with 64 MB RAM. The software used is Windows 95 and Matlab 5.3 (Release 11) with Simulink 3 and Real-Time Workshop (RTW). The PC is also equipped with a CAN card to communicate with the three SAC's on the H-Drive.

The dSPACE board comes with a Simulink block-set and the software packages Real-Time Interface (RTI) and Control Desk. The Simulink blocks give access to the various in- and output ports of the dSPACE board. With RTW in combination with RTI, it is possible to convert a Simulink model to a program which can run on the PowerPC. With Control Desk, it is possible to capture measurements and to change the parameters of the real-time implementation of the Simulink model.

# Chapter 3

# Design of the interface

To use a controller build in Simulink, it must be implemented on the RCP system. The RCP system must be connected to the H-Drive. Therefor an interface is required, which may consist of both hardware and software.

## 3.1 Task specification

The interface that is going to be created, has to perform the following tasks to use a custom controller on the H-Drive.

Firstly, the H-Drive must be initialized. The initialization routine is normally done by the SAC and can be started by calling a LAPI function. This routine checks if all the hardware connected to the CAN-bus, is functioning properly. If this is the case, the three carriages are moved to find the markers of the incremental encoders. When the initialization is successfully completed, the carriage of the x-axis has to be moved to a predefined home position.

When one wants to use the custom controller running on the RCP system, the built-in controller must be disabled and the signal generated by the RCP system must be used instead.

When the H-Drive is driven by the custom controller, a protection system must be used, which prevents the carriage from collision with it's boundaries. The SAC has a built-in protection system, but when the controller is disabled, the protection system is probably disabled too. Therefore, a new protection system must be made and implemented on the RCP system.

When one wants to stop using the custom controller on the RCP system, the built-in controller must be re-enabled. Finally, the H-Drive must be shut down safely. This is normally done by calling a LAPI function.

## 3.2 Approach

There are several possible solutions for connecting the RCP system to the H-Drive. The most rigorous option is to remove the SAC and connect the RCP system directly to the power amplifier of the H-Drive. In this case, no LAPI functions can be used, so an initialization and a shut down routine must be created in Simulink. This can be very difficult, since not much documentation on the LiMMS system is available. Therefor it is desired to still make use of

some features of the SAC. This can be achieved in several ways.

One possibility is to implement the LAPI functions on the RCP system, since the dSPACE board is equipped with a CAN bus. This has the advantage that there is no need for a LAPI program on the PC anymore, since everything is controlled from within the DSP. It is probably very complicated or even impossible to cross compile the LAPI source code (which is written in C) for the PowerPC. Therefor it is decided not to choose for this option.

Another possibility is to still use a program on the PC, a so called LAPI Server. This program initializes the H-Drive, moves it to it's home position and then disables the built-in controller and gives control access to the DSP. Afterwards, the built-in controller can be re-enabled to shut down the H-Drive properly. Because the LAPI Server is probably easy to create by modifying an existing LAPI program, it is decided to choose for this option.

## 3.3 Communication

Now there are four items that have to cooperate:

1. The Single Axis Controller on the H-Drive
2. The LAPI Server running on the PC
3. Control Desk running on the PC
4. The Simulink model running on the RCP system

The SAC and the LAPI Server communicate with each other using the CAN bus. The LAPI Server and the Simulink model can communicate with each other using a null-modem cable between the serial ports of the PC and the dSPACE board. The Simulink model reads the position of the carriage from the incremental encoder and can inject the control output directly in the SAC at the controller output point by using the DAC on the dSPACE board. Control Desk communicates with the dSPACE board using the ISA-bus of the PC. The communication is schematically displayed in figure 3.1.

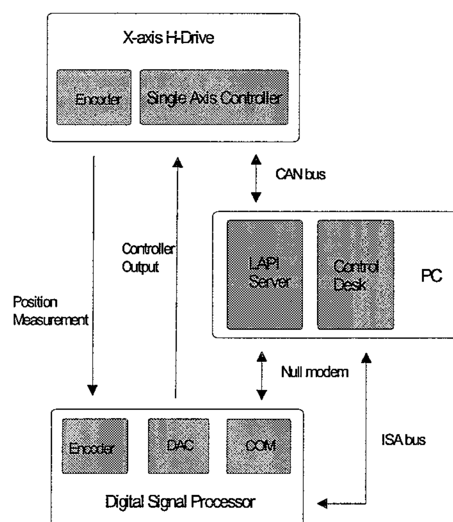



Figure 3.1: Communication scheme

# Chapter 4

# Implementation

The interface that is going to be created, consists of a LAPI Server program and a Simulink model. Before they can be designed, the procedure of using the H-Drive and the communication protocol must be defined.

## 4.1 Procedure

To use a custom controller on the H-Drive, the following procedure will be used:

- When the LAPI Server is started, the H-Drive will be initialized and the carriage is moved to it's home position.
- When the LAPI Server receives a START command from the RCP, the built-in controller will be disabled and the signal from the DSP is injected in the controller output point of the SAC.
- When the LAPI Server receives a STOP command from the RCP, the signal from the DSP is no longer injected and the built-in controller is re-enabled.
- When the LAPI Server receives a HOME command from the RCP, the STOP routine is used and then the carriage is moved back to it's home position by the SAC. Afterwards, the controller on the RCP can be re-enabled by the START routine. The HOME routine can be used after the protection system has had to stop the carriage.
- The LAPI Server has a button to shut down the H-Drive. If necessary, the STOP routine will be executed first.

The LAPI Server is described in chapter 5, the Simulink model in chapter 6.

## 4.2 Serial port communication

The real-time implementation of the Simulink model can send the following codes to the LAPI Server by pressing buttons in Control Desk: START to start using the custom controller, STOP to stop using the custom controller and HOME to go back to the home position.

The LAPI Server can send the following codes to the Simulink model: READY, which means that the controller in Simulink has access to the H-Drive, ERROR which means that the controller in Simulink doesn't have access to the H-Drive and HOME READY which means that the carriage has returned to it's home position.

Both the LAPI Server and the real-time implementation of the Simulink model can send and receive 8 bit ASCII values using the null-modem cable. For the codes described above, the following characters and corresponding ASCII values will be used:

| code | character | value |
|---|---|---|
| START | G | 71 |
| STOP | S | 83 |
| HOME | H | 72 |
| READY | R | 82 |
| ERROR | E | 69 |
| HOME READY | P | 80 |

The serial ports will be configured as RS-232 with a baudrate of 9600 bps and no parity check.

# Chapter 5

# The LAPI Server

## 5.1 Task specification

The LAPI Server is a program running under Windows 95, that makes use of several LAPI functions. It must communicate with the with the SAC and with the Simulink model.

The LAPI Server communicates with the Simulink model running on the RCP system using the serial port. It can receive requests for using the H-Drive, i.e. to start and stop injecting a signal in the controller and for homing. Besides it must send codes defining the status of the H-Drive, i.e. whether the Simulink controller has access to the H-Drive.

The LAPI Server communicates with the SAC using the CAN bus. This can be done by calling LAPI functions. These will be used for initialization, homing, shutting down and start and stop injecting a signal in the controller.

## 5.2 Design

The LAPI Server will have two buttons, one to start the initialization procedure and one to shut down the H-Drive and close the program. The following procedure will be used:

When the LAPI Server is started, the serial port will be initialized. When the INIT button in the LAPI Server window is used, it will call a LAPI function to initialize the H-Drive. When the H-Drive is initialized, two parallel threads are used. The first thread checks for commands coming from the Simulink model on the RCP system. These commands are START, STOP and HOME. The second thread checks whether the EXIT button in the LAPI Server window is used and calls the corresponding function to shut down the H-Drive and to exit the program.

This procedure is visualized in a flow chart, which is displayed in figure 5.1.

## 5.3 Implementation

The Windows layout of the LAPI Server is displayed in figure 5.2. The window has two buttons, one to start the initialization procedure and one to shut down the H-Drive and close the program. In the status window, messages from functions are displayed.

The program is written in ANSI C. The C-code of the functions used in the LAPI Server is described in Appendix B, the complete source code of the LAPI program can be found in Appendix C.

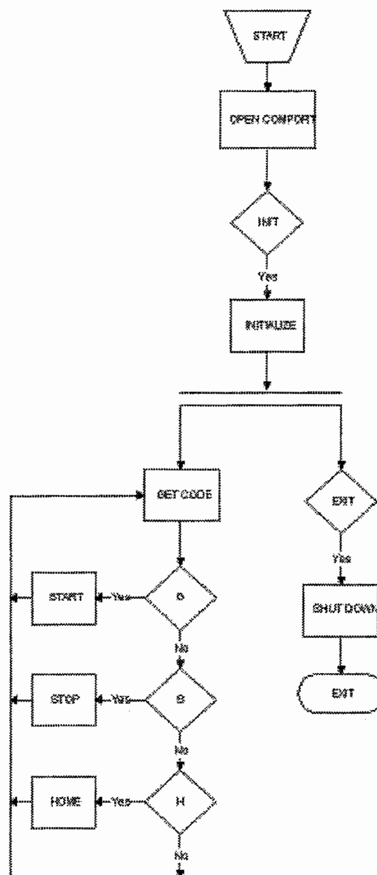



Figure 5.1: Flow-chart LAPI Server

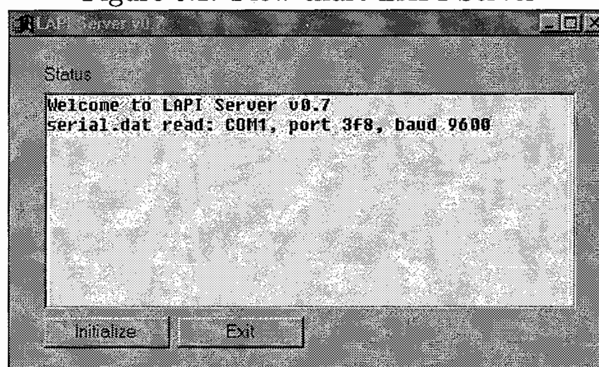



Figure 5.2: LAPI Server

# Chapter 6

# The Simulink model

## 6.1 Task specification

To run a real-time application on the RCP system, a Simulink model has to be built, which has to perform the following tasks:

- Communicate with the LAPI Server running on the PC using the serial port. This can be done by using a Simulink block that came with the dSPACE software.
- Capture the position measurement of the carriage from the incremental encoder. This can also be done by using a Simulink block that came with the dSPACE software.
- Compute the control signal. This is done by the controllers that will be designed.
- Send the control signal to the DAC of the DSP. This can be done by using another Simulink block that came with the dSPACE software.
- Prevent the carriage from hitting the boundaries. This requires designing a new protection system, which will be done in the following section.

## 6.2 The protection system

When the controller of the SAC is disabled, it's protection system is disabled too. To prevent the carriage from hitting the boundaries, a new protection system has to be made in Simulink. This system has to take it over from the controller built in Simulink when the carriage gets outside a certain area and slow down the carriage as fast as possible. When the protection system is enabled, the carriage has to be homed again, before control is given back to the controller built in Simulink.

The carriage has a range from 0.03 m to 0.67 m. The working range will be limited from 0.15 m to 0.55 m, which leaves 0.12 m on both sides for the emergency stop.

Because the working range is larger than the safety margin, the maximum force used by the controller has to be less than the force used for the emergency stop. If the carriage behaves like a pure mass system and it is accelerated over 0.4 m with 100 N, it has to be decelerated with 400 N to make it stop within 0.1 m. If viscous and dry friction are taken into account, maybe a smaller force will satisfy, but this has to be tested by experiments. The SAC has a

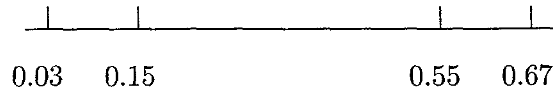



Figure 6.1: Working range

default value for the maximum force of 750 N. This will be used for the emergency stop. The maximum force used by the controller will be limited to 200 N. Since the mass of the carriage lies around 30 kg, movements with a maximum acceleration of 5 m/s$^2$ can be made, which is sufficient for most education purposes. When a controller has been tested thoroughly, the protection could be weakened, so more power is available to the controller.

When a boundary is passed, the protection function acts like a PD controller:

$$\begin{array}{ll} if(x < LB) & u = -U_{max} \cdot (P \cdot (x - LB) + D \cdot \dot{x}) \\ if(x > UB) & u = -U_{max} \cdot (P \cdot (x - UB) + D \cdot \dot{x}) \end{array}$$

with LB=0.15 m, UB=0.55 m, P=20 N/m, D=2 Ns/m and $U_{max}$=750 N. The values of P and D are chosen in such way, that the controlled mass system becomes a critically damped second order system, so $P = \frac{m\omega_n^2}{U_{max}}$ and $D = \frac{2m\zeta\omega_n}{U_{max}}$ with relative damping $\zeta = 1$ and bandwidth $\omega_n \approx 22\ rad/s$. With these values for P and D, the maximum force is reached when either the position is 0.05 m behind the boundary or the velocity is greater than 0.5 m/s.

## 6.3 Implementation

The x-axis of the H-Drive is represented by a subsystem with the computed control signal as input and the position, the time and the status as outputs. When a simulation model is made and tested in Simulink, the model of the plant can be replaced by this block to implement the designed controller on the real plant. This block can be seen in figure 6.2.

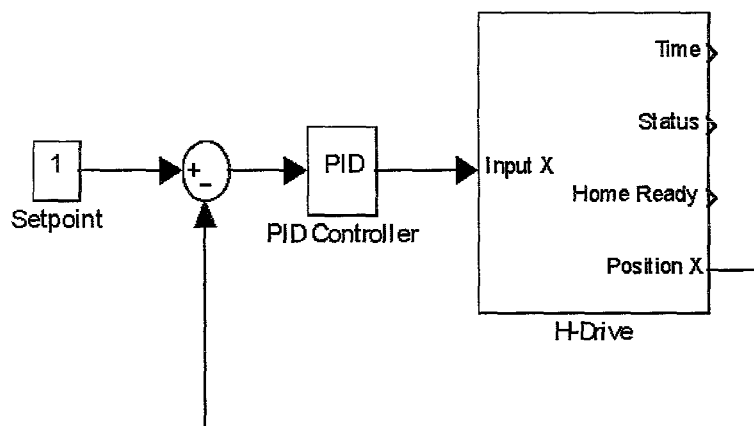



Figure 6.2: The controlled system

### 6.3.1 Layout

As can be seen in figure 6.3, this block contains three subsystems:

- the serial port communication
- the protection system
- the timer clock

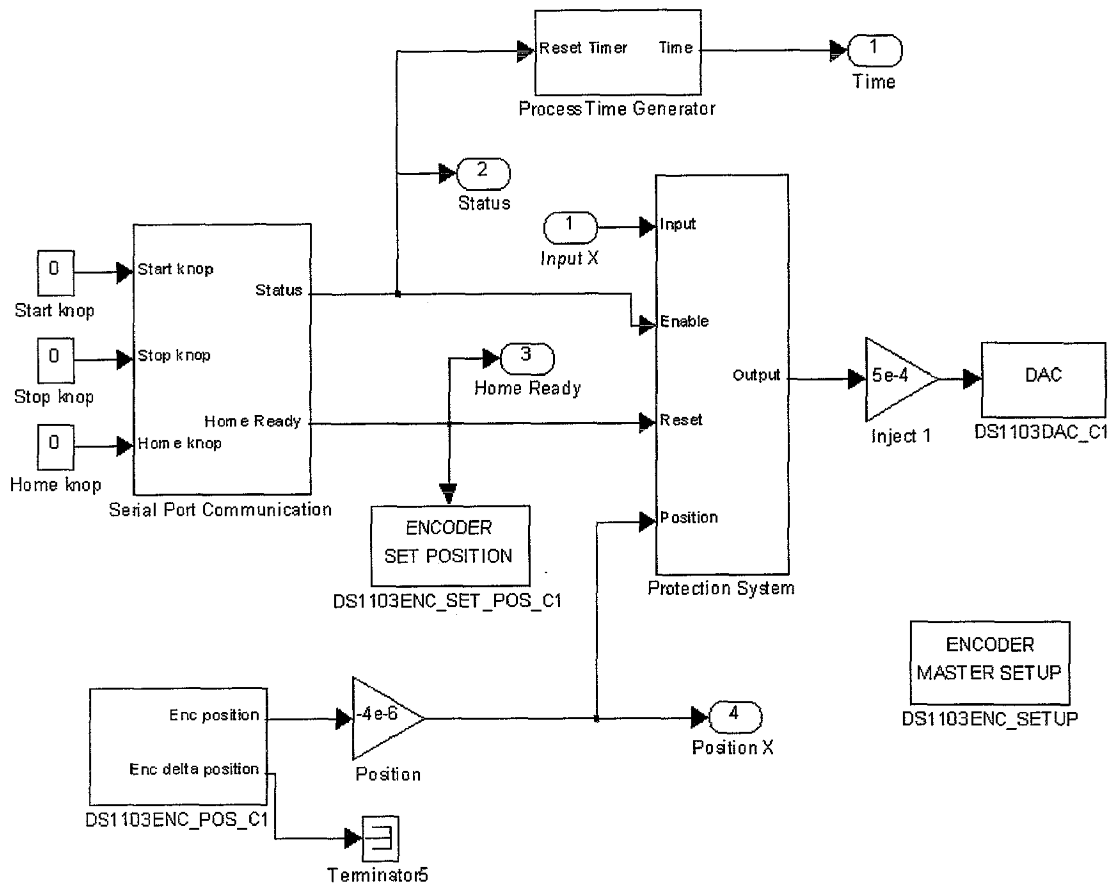



Figure 6.3: H-Drive

### 6.3.2 The communication block

This subsystem is used to communicate with the LAPI Server using the serial port. It can be seen in figure 6.4.

The Simulink model can send the codes START, STOP and HOME to the LAPI Server. These commands are given by pressing buttons on the instrument panel of Control Desk. Outgoing codes are sent to the Serial Out block. To prevent the Serial Out block from sending codes every sample time, it is placed in an subsystem, which is enabled with a pulse function (see section 6.3.5) whenever a button is pressed.

The Simulink model can receive the following codes from the LAPI Server: READY, ERROR and HOME READY. Incoming codes are coming from the Serial In block. HOME READY

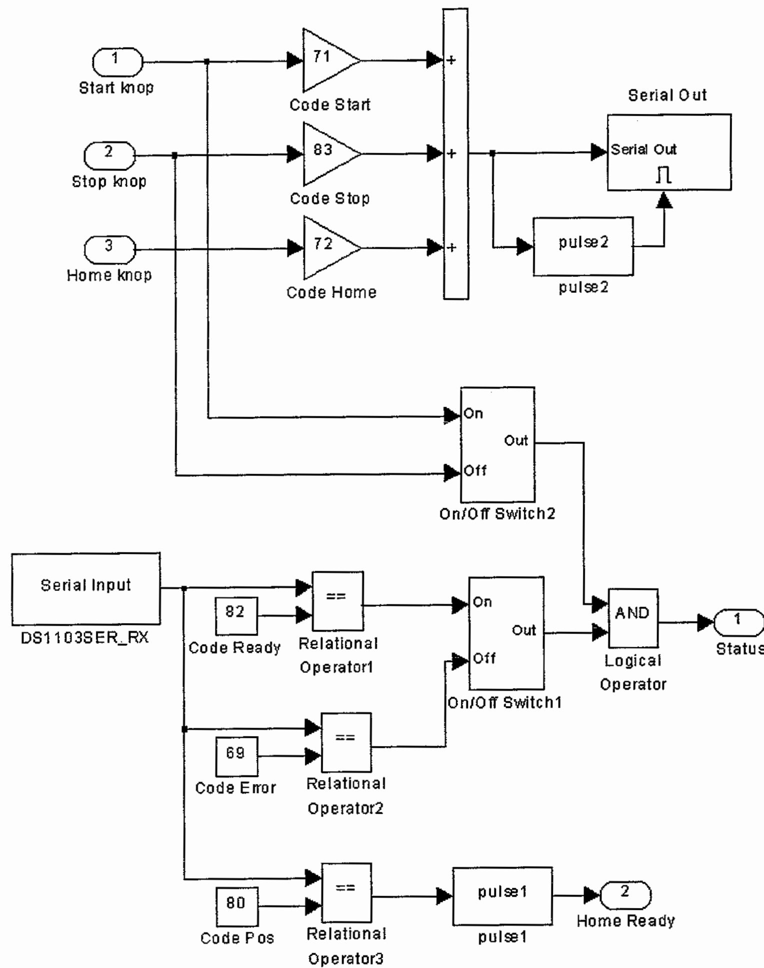



Figure 6.4: The communication block

sets the incremental encoder to a value corresponding with 0.2 m. READY and ERROR are connected to a switch (see section 6.3.5). The signals from the START and STOP buttons are connected to another switch. Both switches are connected to an AND port, which changes the status between enabled and disabled. The second switch is an extra protection in the case that the LAPI Server should fail to stop inject the signal from the DSP when the STOP command is given.

### 6.3.3 The protection function

The protection function, which is designed in section 6.2, is written as an Simulink S-function in C.

```
     if ( U(3)==1 ) { k = 0; }

     if ( U(1)<LB && k==0 ) { k = -1; }
else if ( U(1)>UB && k==0 ) { k =  1; }

     if ( k== 0 )
      {  y[0] = U(0); }
else if ( k==-1 )
```

```
      {  y[0] = -UMAX*(PS*(U(1)-LB) + VS*U(2)); }
else if ( k== 1 )
      {  y[0] = -UMAX*(PS*(U(1)-UB) + VS*U(2)); }
```

The in- and outputs are:

- U(0) : original controller output
- U(1) : carriage position
- U(2) : carriage velocity
- U(3) : reset
- y[0] : protected controller output

k is a static integer variable, i.e. it keeps it's value between every time the function is called. It's initial value is 0.

The protection function is placed in a subsystem, which can be seen in figure 6.5.

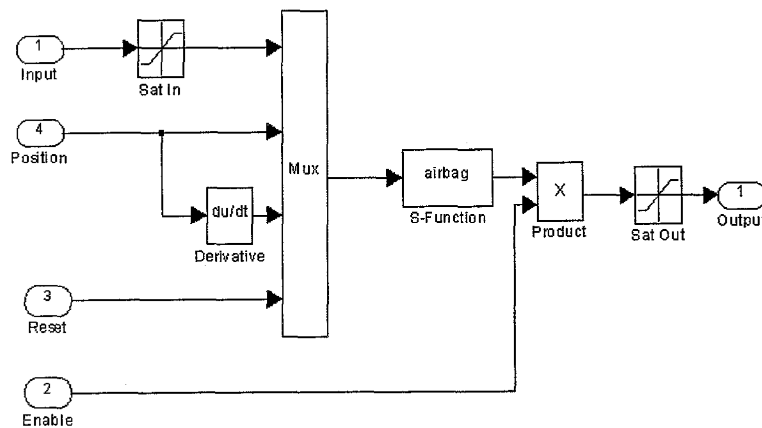



Figure 6.5: The protection system

The original controller output is first limited between $\pm 200$ $N$ before it goes into the protection function. The protected controller output is limited between $\pm 750$ $N$. The protection function can be reset by the HOME READY signal. The protected controller output is only sent to the DAC if the status is READY. This is done as an extra protection in the case that the LAPI Server should fail to stop inject the signal from the DSP when the STOP command is given.

### 6.3.4 The timer clock

To use a look-up table for e.g. a reference trajectory or feed forward signal, it is necessary to have a timer clock which can be reset every time the START command is given. Therefore an S-function with the following routine is used:

```
if ( U(1)==1 )
```

```
 { k=1; t0 = U(0); }

if ( U(2)==1 )
 { k=0; }

if ( k==1 )
 { y[0] = U(0)-t0; }
else
 { y[0] = 0; }
```

The in- and outputs are:

- U(0) : absolute time
- U(1) : timer start
- U(2) : timer reset
- y[0] : simulation time

k is a static integer variable, it's initial value is 0. t0 is a static float variable, it's initial value is 0.

The time delay function is placed in a subsystem, which can be seen in figure 6.6.

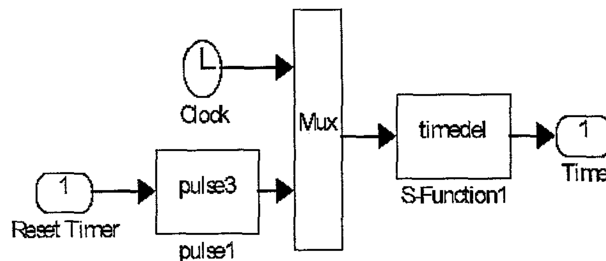



Figure 6.6: The timer clock

### 6.3.5 Other S-functions

In the Simulink model two other S-functions are used. These functions are both written in C and are described here.

#### The switch function

The switch function is a function whose output is switchable between 0 and 1 by 2 input signals ON and OFF. It's output value is initially 0, becomes 1 whenever the ON signal equals 1 and becomes 0 whenever the OFF signal equals 1. After switching it keeps it's value till one of the input signals becomes 1 again. When both inputs are 1, the output becomes 0 so the OFF signal has priority.

The function is used to start and stop the controller built in Simulink. The following routine is used:

```
     if ( U(1)==1 )
      { y[0]=0.0; k=0; }
else if ( U(0)==1 && U(1)==0 )
      { y[0]=1.0; k=1; }
else if ( U(0)==0 && U(1)==0 && k==0 )
      { y[0]=0.0; }
else if ( U(0)==0 && U(1)==0 && k==1 )
      { y[0]=1.0; }
```

The in- and outputs are:

- U(0) : ON signal
- U(1) : OFF signal
- y[0] : output signal

k is a static integer variable, it's initial value is 0.

### The pulse function

The pulse function is a function whose output is initially 0 and becomes 1 for exactly one sample time whenever the input changes from 0 to 1. The function is used to reset the timer clock and to send exactly one code over the serial port. Therefore the following routine is used:

```
     if ( U(0)==0 )
      { k=0; y[0]=0.0; }
else if ( U(0)==0 && k==0 )
      { k=1; y[0]=1.0; }
else  { y[0]=0.0; }
```

The in- and outputs are:

- U(0) : continuous signal
- y[0] : pulse signal

k is a static integer variable, it's initial value is 0.

# Chapter 7

# Test experiments

Now the interface is built, it has to be tested in practice. One point of interest is the performance of the interface. Another important issue is, whether the protection system works properly.

## 7.1 Testing the performance

To test the performance of the interface, the built-in controller of the SAC is rebuild in Simulink, with the same values for the parameters. Now the error signals of a point-to-point movement can be compared. A third order trajectory is used with a maximum jerk of 100 $m/s^3$, a maximum acceleration of 2 $m/s^2$ and a maximum velocity of 0.5 $m/s$. The set-point position can be seen in the upper left plot of figure 7.1.

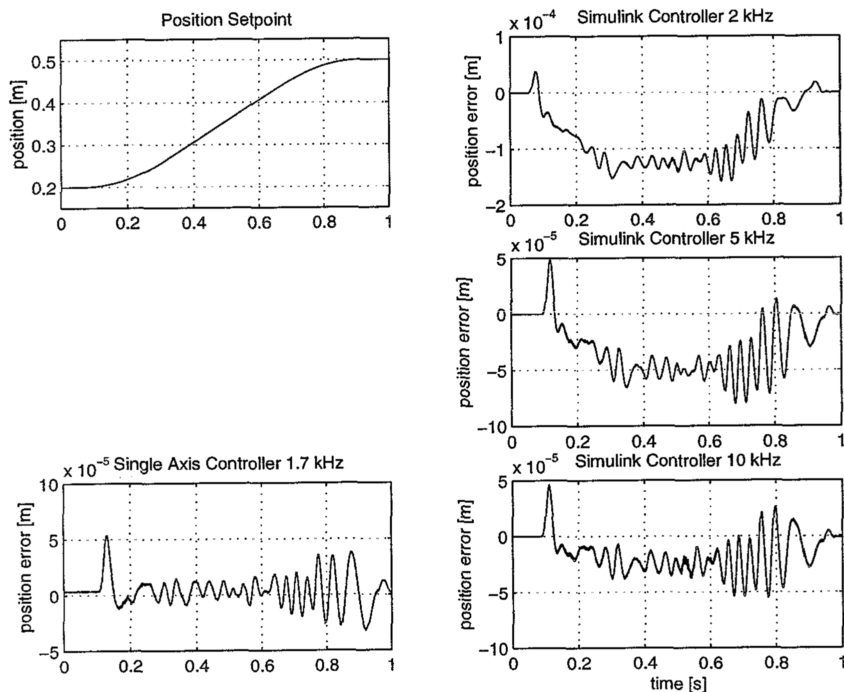



Figure 7.1: Comparison of controllers

The controller on the SAC has a sampling rate of 600 $\mu$s, which corresponds with a sampling rate of approx. 1.7 kHz. The position error achieved with this controller can be seen in the lower left plot of figure 7.1.

When the controller on the RCP system is used, the controller on the SAC is disabled, but the signal from the RCP system is injected in the SAC, which is also done at 1.7 kHz. The controller on the RCP system is firstly used with a sampling rate of 2 kHz. As can be seen in the upper right plot of figure 7.1, the tracking error of the Simulink controller is much larger than that of the built-in controller. This decrease in performance cold be caused by bad synchronization between the DSP and the SAC. To decrease this effect, the Simulink controller is also used with a sampling rate of 5 kHz and 10 kHz. These are shown in the middle and lower right plots of figure 7.1. As can be seen, the error becomes smaller for a higher sampling rate, but the error at 10 kHz is still larger than that of the default controller on the SAC at 1.7 kHz. The decrease in performance is probably caused by both bad synchronization and additional delay times between the DSP and the SAC.

## 7.2 Testing the protection system

Another important issue is, whether the protection system works properly. This is tested firstly by using a constant force of 200 N as input force for the robot. The carriage starts at 0.2 m, so it can move for 0.35 m before the protection system interferes. The results of this experiment can b seen in the left plots in figure 7.2. When the carriage excites the 0.55 m boundary, the protection system takes over and the input forces changes immediately to -750 N and then goes to zero. The carriage stops at 0.65 m, which is still before the boundary at 0.67 m, without overshoot.

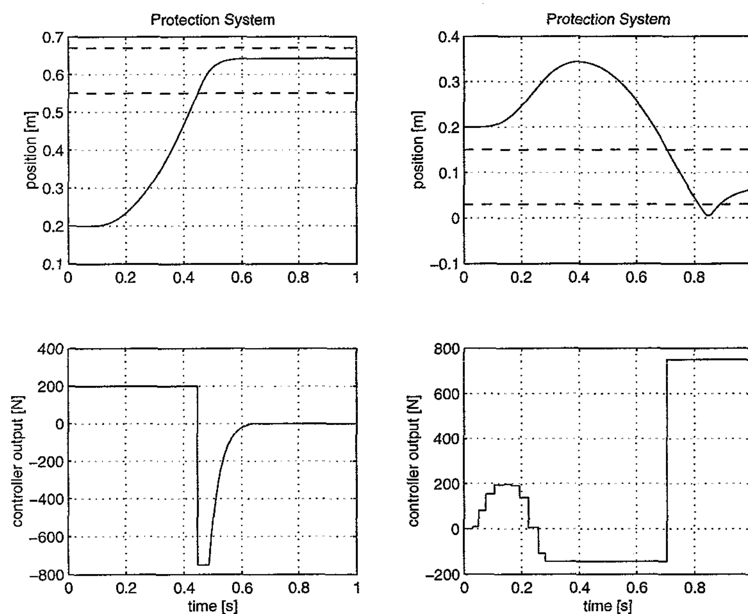



Figure 7.2: Testing the protection system

Another test has been done with an input signal which is varied by hand between +200 and -200 N. This test was less succesful, as can be seen in the right plots of figure 7.2. When the carriage excites the 0.15 m boundary, the protection system takes over. Although the carriage

is slowed down, it still hits the boundary at 0.03 m, which is a very stiff spring. This causes the power amplifier to be turned off. This is done by a switch mounted on the carriage that gets in contact with the spring.

In this experiments a sampling rate of only 1 kHz was used. The fact that the carriage does not stop in time is probably caused by the bad synchronization and additional delay times between the DSP and the SAC as discussed in the previous section. Besides using a higher sampling rate, collision could be avoided by decreasing the maximum force allowed to the Simulink controller, or increasing the control parameters P and D of the protection system. This has not yet been tested.

# Chapter 8

# Conclusions and recommendations

With the interface created, it is possible to implement controllers built in Simulink on the H-Drive and still make use of some features of the SAC. The implementation has been done only for the x-axis. The two y-axes are more difficult, because they are not allowed to differ in position too much. This requires extra security measures.

The performance of the interface has been tested by rebuilding the built-in controller in Simulink. It turned out that the performance of the controller on the RCP system is worse than that of the built-in controller. This is probably caused by delay times in the interface and/or synchronization problems. A remedy has not yet been found. This decrease of performance makes the interface less suitable for it's attended purpose, which is trying to increasing the performance of the H-Drive by using other controllers. However the performance of other controllers can still be compared with the Simulink version of the default controller running on the RCP system.

When the controller of the SAC is disabled, it's protection system is disabled too, so care must be taken. An alternative protection system has been made, although it does not work sufficient for all cases yet. Besides, it is not resistant to failure of the dSPACE board. Therefore it is preferable to add an extra hardware protection against collision.

To prevent that other people working with the H-Drive can modify the interface, it's better to rewrite the Simulink model to one compiled S-function. This is possible but more difficult, because standard Simulink blocks have to be replaced by C-code.

Using the CAN bus on the dSPACE board to communicate with the SAC is much more difficult and probably doesn't have much advantages, but it can be an interesting subject of investigation because CAN is a widely used protocol in modern industrial applications.

# Appendix A

# Block scheme of the controller

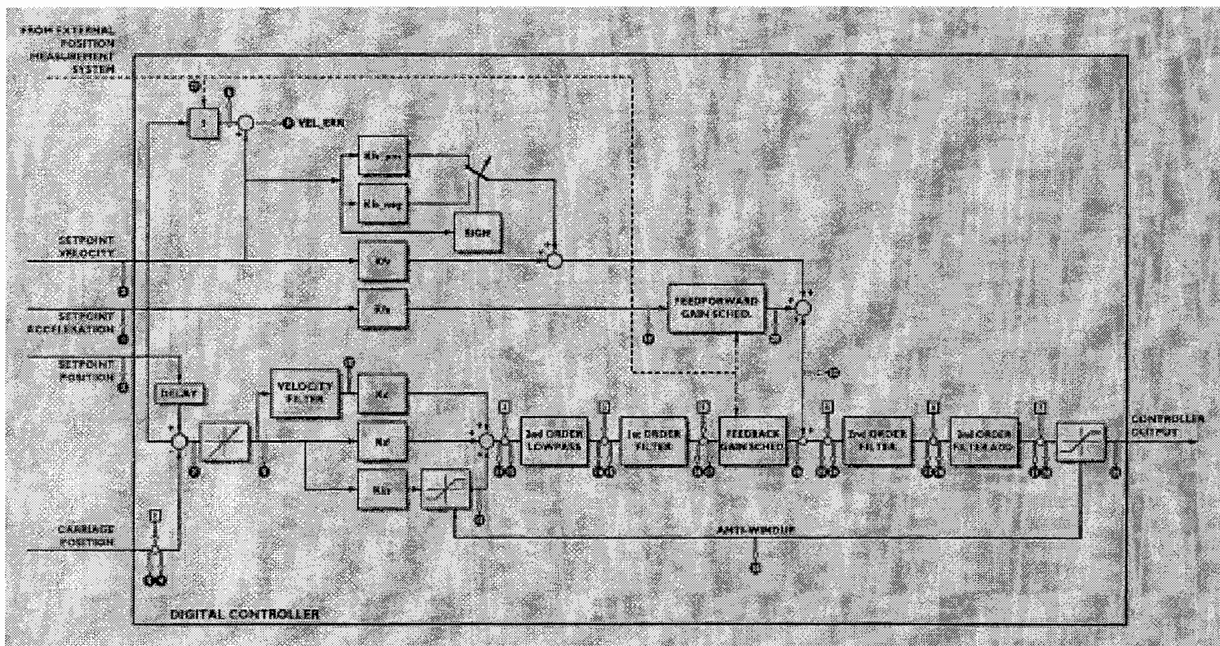



Figure A.1: Block scheme of the controller

# Appendix B

# LAPI Server function descriptions

The program written in ANSI C using the Salford C++ Compiler [4]. It makes use of the following header files:

| | |
|---|---|
| `limms.h` : | This file is written by Philips and contains the LAPI function set. |
| `limmslib.c` : | This file is written by René van de Molengraft and contains LAPI-based functions. |
| `serial.h` : | This file is written by René van de Molengraft and contains functions for serial port communication under Win32. |

## The main function

The `main` function contains the windows layout, the configuration of the serial port and a routine to start functions desired by the DSP. The Windows layout is created using the ClearWin library shipped with the Salford C++ Compiler [4]. All messages from functions made with `printf` are displayed in the status window.

```
main()
{
code c,*pc;
pc=&c;

winio("%sp&",0,0);
winio("%ww[topmost]&");
winio("%sy[3d_thin]&");
winio("%ca[LAPI Server v1.0]&");
winio("  Status%ff&");
winio("%50.10cw%ff&",NULL);
winio("  %^tt[Initialize]&",cbf_init);
winio("  %^tt[Exit]&",cbf_exit);
winio("%lw",&WinCtrl);

printf("Welcome to LAPI Server v1.0\n");

OpenComPort();
```

```
do
  {
  get_code(pc);

       if ( c==CG )
        { go_start(); c=0; }
  else if ( c==CS )
        { go_stop(); c=0; }
  else if ( c==CH )
        { go_home(); c=0; }

  Sleep(100);
  yield_program_control(Y_TEMPORARILY);
  }
while (exit==0);

CloseComPort();

WinCtrl=0; window_update(&WinCtrl);

return 0;
}
```

## Call-back functions

Call-back functions are function that are called when a button in the windows layout is pressed.

```
int cbf_exit()
{
code c,*pc;
pc=&c;
int close=-1;

if (init_ok==-1)
  {
  printf("LiMMS was not initialized.\n");
  exit=1;
  return 1;
  }

printf("Closing LiMMS...\n");

c=CE; send_code(pc);

set_inject(0,0.0); unfree_x();

close=shutdown_limms();
```

```
while (close==-1)
  {
  Sleep(100);
  }

if (close==1)
  {
  printf("Closing LiMMS succeeded.\n");
  exit=1;
  }
else
  {
  printf("Closing LiMMS failed.\n");
  }

return 2;
}
```

## Functions called by the DSP

```
int go_start()
{
code c,*pc; pc=&c;

if (init_ok==1)
  {
  free_x();
  set_inject(2,-mult);
  free=1;
  c=CR;
  send_code(pc);
  }
else
  {
  printf("Initialize LiMMS first.\n");
  }

return 1;
}
```

The built-in controller is disabled and the signal from the DSP will be injected in the controller output point of the SAC. The code READY will be send back to the Simulink model.

When the LAPI Server receives a STOP command from the Simulink model, the function go_stop is called:

```
int go_stop()
{
code c,*pc;
pc=&c;
```

```
if (init_ok==1)
  {
  c=CE;
  send_code(pc);
  set_inject(0,0.0);
  unfree_x();
  free=0;
  }
else
  {
  printf("Initialize LiMMS first.\n");
  }

return 1;
}


int go_home()
{
code c,*pc;
pc=&c;

if (init_ok==1)
  {
  printf("Moving to 0.2 m.\n");
  c=CE;
  send_code(pc);

  if (free==1)
    {
    set_inject(0,0.0);
    unfree_x();
    free=0;
    }

  move_limms_to_sync(0.2);
  c=CP;
  send_code(pc);
  }
else
  {
  printf("Initialize LiMMS first.\n");
  }

return 1; }
```

## Other functions

In the functions described above several new functions are used which are added to `limmslib.c`.

```
int set_inject(int id,float scale)
{
set_test_limms(id,scale);
return 1;
}
```

`free_x` deactivates the built-in controller of the x-axis, so the custom controller can be used:

```
int free_x(void)
{
if (SacDeactivate(logAxisID0)!=LDD_M_SUCCESS)
  {
  printf("FREE_X: Free x-axis failed.\n");
  return 0;
  }

printf("x-axis freed.\n"); return 1; }
```

`unfree_x` reactivates the built-in controller of the x-axis:

```
int unfree_x(void)
{
if (SacActivate(logAxisID0)!=LDD_M_SUCCESS)
  {
  printf("UNFREE_X: UnFree x-axis failed.\n");
  return 0;
  }

printf("x-axis unfreed.\n"); return 1; }
```

For serial port communication, two functions are written that make use of functions from serial.h.

`send_code` sends a code to the DSP:

```
void send_code(char *pc)
{
WriteStringToPort(pc,1);
}
```

`get_code` receives a code from the DSP:

```
void get_code(char *pc)
{
ReadCharFromPort(pc);
}
```

The complete source code of the LAPI program can be found in Appendix C.

# Appendix C

# LAPI Server source code

```
/* Lapi Server v1.0 */

#include "serial.h"

typedef char code;
const code CE='E', CG='G', CH='H', CP='P', CR='R', CS='S';
int home_ok=-1, exit=0, free=0;
int WinCtrl;
int mult=200;

int move_limms_to(float);
int move_limms_to_sync(float);
int free_x(void);
int unfree_x(void);


void send_code(char *pc)
{
WriteStringToPort(pc,1);
}


void get_code(char *pc)
{
ReadCharFromPort(pc);
}


int set_inject(int id,float scale)
{
set_test_limms(id,scale);
return 1;
}


int set_satlev(float satlev)
```

```
{
pxpars=&xpars[0];

get_controller_x(pxpars);

pxpars[27]=satlev;

set_controller_x(pxpars);

printf("SATLEV = %d.\n",(int) satlev);

return 1;
}



int go_start()
{
code c,*pc;
pc=&c;

if (init_ok==1)
  {
  free_x();
  set_inject(2,-mult);
  free=1;
  c=CR;
  send_code(pc);
  }
else
  {
  printf("Initialize LiMMS first.\n");
  }

return 1;
}



int go_stop()
{
code c,*pc;
pc=&c;

if (init_ok==1)
  {
  c=CE;
  send_code(pc);
  set_inject(0,0.0);
  unfree_x();
  free=0;
  }
```

```
else
  {
  printf("Initialize LiMMS first.\n");
  }

return 1;
}



int go_home()
{
code c,*pc;
pc=&c;

if (init_ok==1)
  {
  printf("Moving to 0.2 m.\n");
  c=CE;
  send_code(pc);

  if (free==1)
    {
    set_inject(0,0.0);
    unfree_x();
    free=0;
    }

  move_limms_to_sync(0.2);
  c=CP;
  send_code(pc);
  }
else
  {
  printf("Initialize LiMMS first.\n");
  }

return 1;
}



int cbf_init()
{
code c,*pc;
pc=&c;
double ttt;
int init=-1;

if (init_ok==1)
  {
  printf("LiMMS is already initialized.\n");
```

```
return 1;
}

printf("Initializing...\n");

init=startup_limms();

while (init==-1)
  {
  Sleep(100);
  }

if (init==1)
  {
  home_x_ok=-1;
  home_y1_ok=-1;
  home_y2_ok=-1;

  printf("Homing axes...\n");

  hStartHoming=CreateEvent(NULL,TRUE,FALSE,NULL);
  lpThread05Par=&ttt;

  hThread05=CreateThread(NULL,
                         0,
                         (LPTHREAD_START_ROUTINE) Thread05Proc,
                         lpThread05Par,
                         0,
                         &Thread05Id);

  lpThread06Par=&ttt;

  hThread06=CreateThread(NULL,
                         0,
                         (LPTHREAD_START_ROUTINE) Thread06Proc,
                         lpThread06Par,
                         0,
                         &Thread06Id);

  lpThread07Par=&ttt;

  hThread07=CreateThread(NULL,
                         0,
                         (LPTHREAD_START_ROUTINE) Thread07Proc,
                         lpThread07Par,
                         0,
                         &Thread07Id);

  Sleep(1000);
  SetEvent(hStartHoming);

  printf("Waiting for homing to finish...\n");
```

```
    while (home_x_ok==-1 || home_y1_ok==-1 || home_y2_ok==-1)
      {
      Sleep(100);
      }

    if (home_x_ok==1 && home_y1_ok==1 && home_y2_ok==1)
      {
      move_limms_to_sync(0.2);
      init_ok=1;
      c=CP;
      send_code(pc);
      printf("Initializing LiMMS succeeded.\n");
      }
    else
      {
      init_ok=0;
      c=CE;
      printf("Initializing LiMMS failed.\n");
      }
  }
else
  {
  c=CE; init_ok=0;
  printf("Initialization failed.\n");
  }

return 2;
}



int cbf_exit()
{
code c,*pc;
pc=&c;
int close=-1;

if (init_ok==-1)
  {
  printf("LiMMS was not initialized.\n");
  exit=1;
  return 1;
  }


printf("Closing LiMMS...\n");

c=CE;
send_code(pc);

set_inject(0,0.0);
```

```
unfree_x();

close=shutdown_limms();

while (close==-1)
  {
  Sleep(100);
  }

if (close==1)
  {
  printf("Closing LiMMS succeeded.\n");
  }
else
  {
  printf("Closing LiMMS failed.\n");
  }

return 2;
}



main()
{
code c,*pc;
pc=&c;

winio("%sp&",0,0);
winio("%ww[topmost]&");
winio("%sy[3d_thin]&");
winio("%ca[LAPI Server v0.7]&");
winio("  Status%ff&");
winio("%50.10cw%ff&",NULL);
winio("  %^tt[Initialize]&",cbf_init);
winio("  %^tt[Exit]&",cbf_exit);
winio("%lw",&WinCtrl);

printf("Welcome to LAPI Server v0.7\n");

OpenComPort();

do
  {
  get_code(pc);

      if ( c==CG )
        {
        go_start(); c=0;
        }
else if ( c==CS )
        {
```

```
       go_stop(); c=0;
       }
else if ( c==CH )
       {
       go_home(); c=0;
       }

  Sleep(100);
  yield_program_control(Y_TEMPORARILY);
  }
while (exit==0);

CloseComPort();

WinCtrl=0;
window_update(&WinCtrl);

return 0;
}
```

# Bibliography


[1] *LiMMS: System Description.* Report 8122-968-93523, Philips CFT, Eindhoven, The Netherlands, May 1997.

[2] *LiMMS: LAPI User Guide.* Report 8122-968-93553, Philips CFT, Eindhoven, The Netherlands, May 1997.

[3] *OMC: Single Axis Controller version 3.0 - Reference Manual.* Report CTR-595-97-0031, Philips CFT, Eindhoven, The Netherlands, July 1997.

[4] *Salford C++ Compiler version 3.0 - User Guide.* Salford Software, United Kingdom, 1999.